[MS-PROTO]:
Windows Overview

Release: Tuesday, April 22, 2014

Revision Summary

Date	Revision History	Revision Class	Comments
12/18/2006	0.1		MCPP Milestone 2 Initial Availability
03/02/2007	1.0	Major	Monthly release
04/03/2007	1.1	Minor	Monthly release
05/11/2007	1.2	Minor	Monthly release
06/01/2007	2.0	Major	Update to technical content
07/03/2007	2.0.1	Editorial	Revised and edited the technical content.
07/20/2007	2.1	Minor	Updated the technical content.
08/10/2007	3.0	Major	New content added.
09/28/2007	4.0	Major	Updated and revised the technical content.
10/23/2007	5.0	Major	Added new sections and new content.
11/30/2007	5.1	Minor	Updated the technical content.
01/25/2008	5.1.1	Editorial	Revised and edited the technical content.
03/14/2008	6.0	Major	Updated and revised the technical content.
05/16/2008	7.0	Major	Updated and revised the technical content.
06/20/2008	8.0	Major	Updated and revised the technical content.
07/25/2008	8.1	Minor	Updated the technical content.
08/29/2008	8.1.1	Editorial	Revised and edited the technical content.
10/24/2008	8.2	Minor	Updated the technical content.
12/05/2008	8.3	Minor	Updated the technical content.
01/16/2009	8.3.1	Editorial	Revised and edited the technical content.
02/27/2009	8.3.2	Editorial	Revised and edited the technical content.
04/10/2009	8.4	Minor	Updated the technical content.
05/22/2009	8.4.1	Editorial	Revised and edited the technical content.
07/02/2009	8.5	Minor	Updated the technical content.
08/14/2009	8.5.1	Editorial	Revised and edited the technical content.
09/25/2009	9.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
11/06/2009	9.1	Minor	Updated the technical content.
12/18/2009	9.1.1	Editorial	Revised and edited the technical content.
01/29/2010	9.1.2	Editorial	Revised and edited the technical content.
03/12/2010	10.0	Major	Updated and revised the technical content.
04/23/2010	10.0.1	Editorial	Revised and edited the technical content.
06/04/2010	11.0	Major	Updated and revised the technical content.
07/16/2010	11.1	Minor	Clarified the meaning of the technical content.
08/27/2010	11.1	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2010	11.1	No change	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	11.1	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	11.1	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	11.1	No change	No changes to the meaning, language, or formatting of the technical content.
03/25/2011	11.1	No change	No changes to the meaning, language, or formatting of the technical content.
05/06/2011	11.1	No change	No changes to the meaning, language, or formatting of the technical content.
06/17/2011	12.0	Major	Significantly changed the technical content.
09/23/2011	12.0	No change	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	12.0	No change	No changes to the meaning, language, or formatting of the technical content.
03/30/2012	12.1	Minor	Clarified the meaning of the technical content.
07/12/2012	12.1	No change	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	12.1	No change	No changes to the meaning, language, or formatting of the technical content.
01/31/2013	12.1	No change	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
08/08/2013	12.1	No change	No changes to the meaning, language, or formatting of the technical content.
11/14/2013	12.1	No change	No changes to the meaning, language, or formatting of the technical content.
02/13/2014	12.1	No change	No changes to the meaning, language, or formatting of the technical content.
05/15/2014	12.1	No change	No changes to the meaning, language, or formatting of the technical content.

Contents

1 Introduction

The Windows Client-Server Communications Protocols include protocols that are implemented by Windows Server operating systems to interoperate or communicate with Windows clients (Windows NT 3.1 operating system, Windows 95 operating system, Windows NT Workstation operating system, Microsoft Windows 98 operating system, Windows 2000 operating system, Windows Millennium Edition operating system, Windows XP operating system, and Windows Vista operating system). This document describes the technical relationships among these protocols.

Editorial Note: MS-PROTO has no Appendix because it has no behavior notes for specific product versions. Do not remove this comment.

1.1 About This Document

This document is a companion to the protocol and data structure specifications that are included in the Windows client-server communications documentation set. It provides a system overview of how these protocols are related to each other and provides examples for how these protocols could be implemented in order to perform a specific task, such as opening a file or joining a domain.

This document is organized as follows:

- The first section of this document describes basic concepts that are used throughout this document.
- The following sections describe sets of protocols grouped by functionality or technical areas, such as file services and authentication. Each of these sections describe how the protocols are composed and are followed by implementation examples.
- The relationships among these protocols are described using two views:
  - Protocol stack view: This view allows the reader to determine the encapsulation of each protocol as it moves down the stack for eventual transmission on the underlying physical medium. For purposes of this document, this view is called the protocol stack.
  - Logical dependencies view: This view shows how the protocols interrelate, which allows the reader to determine what protocols are used in parallel with the main protocol. This view is useful in cases where other protocols are required for the main protocol to function; however, these other protocols may be logically contained in the main protocol or invoked separately by an implementation of the protocol.

1.2 Basic Protocol Concepts

This section covers basic concepts that are used throughout this document to describe the protocols.

1.2.1 Named Pipes

Common Internet File System (CIFS)/Server Message Block (SMB) serves a special role in the protocols, because CIFS/SMB offers a construct that is known as the **named pipe**. A **named pipe** is a logical connection, similar to a Transmission Control Protocol (TCP) session, between the client and server that are involved in the CIFS/SMB connection. The name of the pipe serves as the endpoint for the communication, in the same manner as a port number serves as the endpoint for TCP sessions. This is called a **named pipe endpoint**.

Many protocols layer on top of named pipes, either directly, as seen in the Common Internet File System (CIFS) Browser Protocol as specified in [MS-BRWS], or indirectly, through the Remote Procedure Call Protocol Extensions as specified in [MS-RPCE]. When named pipes are used, they have the advantage of insulating the higher-layer protocol from what transport was chosen and also of offering the higher-layer protocol the authentication services of the CIFS/SMB connection.

Note When this document refers to SMB, both SMB and SMB2 are applicable, unless otherwise specified.

A share is a local resource that is offered by an SMB server for access by SMB clients over the network. The protocol defines three types of shares: file (or disk) shares, which represent a directory tree and its included files; pipe shares, which expose access to named pipes; and print shares, which provide access to print resources on the server.

A pipe share, as defined by the SMB Protocol, MUST always have the name "IPC$". A pipe share MUST only allow named pipe operations and Distributed File System (DFS) referral requests to itself. The data that is carried over IPC$ is an implementation detail of SMB. This implementation detail is transparent to the Remote Procedure Call Protocol Extensions as specified in [MS-RPCE].

1.2.2 Remote Procedure Calls

1.2.2.1 Remote Procedure Call Model

A Remote Procedure Call (RPC) is a secure interprocess communication (IPC) mechanism that enables data exchange and invocation of functionality residing in a different process. That different process can be on the same machine, on the local area network, or across the Internet. This section explains the RPC programming model and the model for distributed systems that can be implemented by using RPC.

RPC fully supports 64-bit editions of Windows. In Windows XP operating system, there are three types of processes: native 32-bit processes, native 64-bit processes, and 32-bit processes running under the 32-bit process emulator on a 64-bit system (often referred to as WOW6432 processes). Using RPC, developers can transparently communicate between different types of process; RPC automatically manages process differences behind the scenes.

RPC was initially developed as an extension to Open Software Foundation (OSF) RPC. With the exception of some of its advanced features, RPC is interoperable with other vendors' implementations of OSF RPC.

This section also provides an overview of RPC components and their operation. The information is presented in the following topics:

- Programming Model
- Distributed Systems Model
- How RPC Works
- Microsoft RPC Components

1.2.2.2 Programming Model

In the early days of computer programming, each program was written as a large monolithic chunk that was filled with **goto** statements. Each program had to manage its own input and output to different hardware devices. As the programming discipline matured, this monolithic code was

organized into procedures and the most commonly used procedures were packed in libraries for sharing and reuse.

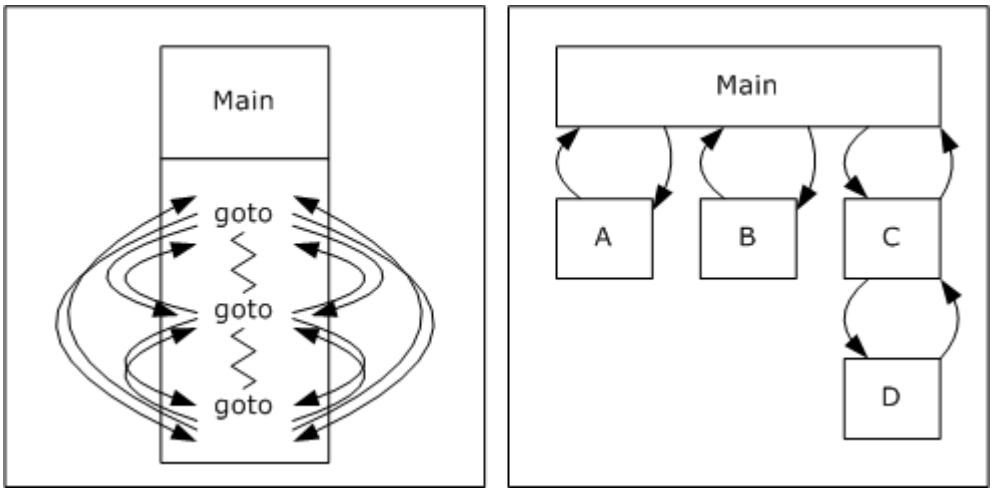

Figure 1: Monolithic vs. procedural implementation model

In procedure-oriented programming languages such as C, the main procedure relates to all other procedures as black boxes. For example, the main procedure cannot find out how procedures A, B, and X do their work. The main procedure only calls another procedure; it has no information about how that procedure is implemented.

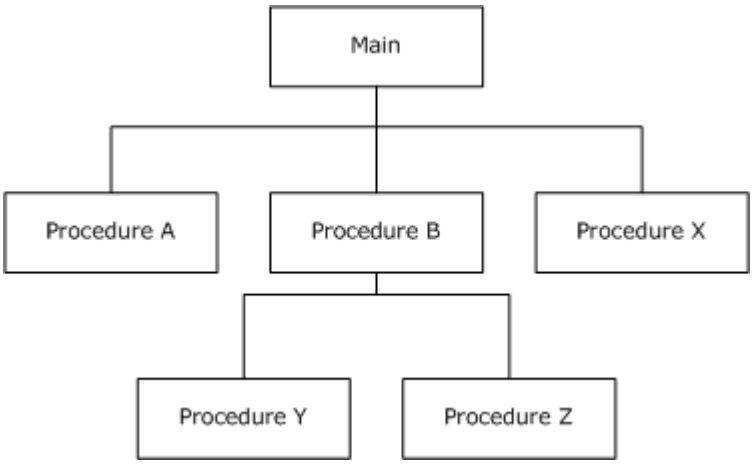

Figure 2: Procedure-oriented code implementation model

Procedure-oriented programming languages provide simple mechanisms for specifying and writing procedures. For example, the ANSI-standard C-function prototype is a construct that is used to specify the name of a procedure, the type of the result that it returns (if any), and the number, sequence, and type of its parameters. Using the function prototype is a formal way to specify an interface between procedures.

Microsoft RPC builds on that programming model by allowing procedures that are grouped together in interfaces to reside in different processes than the caller. Microsoft RPC also adds a more formal approach to procedure definition that allows the caller and the called procedure to adopt a contract

for remotely exchanging data and invoking functionality. In the Microsoft RPC programming model, traditional function calls are supplemented with two additional elements.

- The first element is an .idl/.acf file that precisely describes the data exchange and parameter-passing mechanism between the caller and a called procedure.
- The second element is a set of runtime APIs that provide developers with granular control of the remote procedure call, including security aspects, managing state on the server, specifying which clients can communicate with the server, and so on.

1.2.2.3 Distributed Systems Model

Traditionally, having software run across multiple computers meant splitting the software into separate client and server components. In such systems, the client component handled the user interface and the server provided back-end processing, such as database access, printing, and so on.

As computers proliferated, dropped in cost, and became connected by ever higher bandwidth networks, splitting software systems into multiple components became more convenient, with each component running on a different computer and performing a specialized function. This approach simplified development, management, administration, and often improved performance and robustness because failure in one computer did not necessarily disable the entire system.

In many cases the system appears to the client as an opaque cloud that performs the necessary operations, even though the distributed system is composed of individual nodes, as illustrated in the following figure.

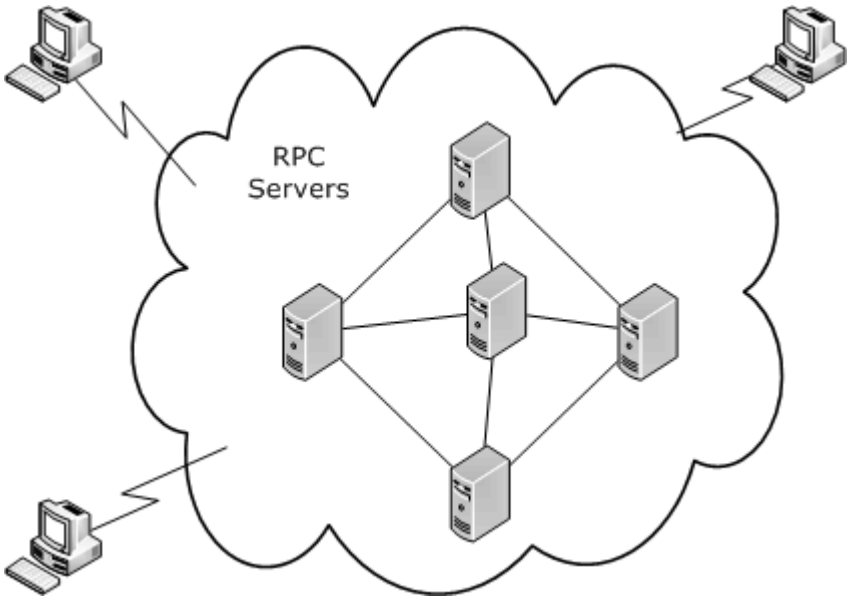

Figure 3: Distributed system paradigm

The opacity of the cloud is maintained because computing operations are invoked on behalf of the client. As such, clients can locate a computer (a *node*) within the cloud and request a particular operation; in performing the operation, that computer can invoke functionality on other computers within the cloud without exposing the additional steps, or the computer on which they were carried out, to the client.

With this paradigm, the mechanics of a distributed, cloud-like system can be broken down into many individual packet exchanges, or conversations between individual nodes.

Traditional client/server systems have two nodes with fixed roles and responsibilities. Modern distributed systems can have more than two nodes, and their roles are often dynamic. In one conversation, a node can be a client; while in another conversation, the node can be the server. In many cases, the ultimate consumer of the exposed functionality is a client with a user sitting at a keyboard and watching the output. In other cases the distributed system functions unattended, performing background operations.

The distributed system may not have dedicated clients and servers for each particular packet exchange, but it is important to remember that there is a caller (or initiator), often referred to as the client. There is also the recipient of the call (often referred to as the server). It is not necessary to have two-way packet exchanges in the request-reply format of a distributed system; often messages are sent only one way.

1.2.2.4 How RPC Works

The RPC tools make it appear to users that a client directly calls a procedure that is located in a remote server program. The client and server each have their own address spaces; that is, each has its own memory resource allocated to data that is used by the procedure. The following figure illustrates the RPC architecture.

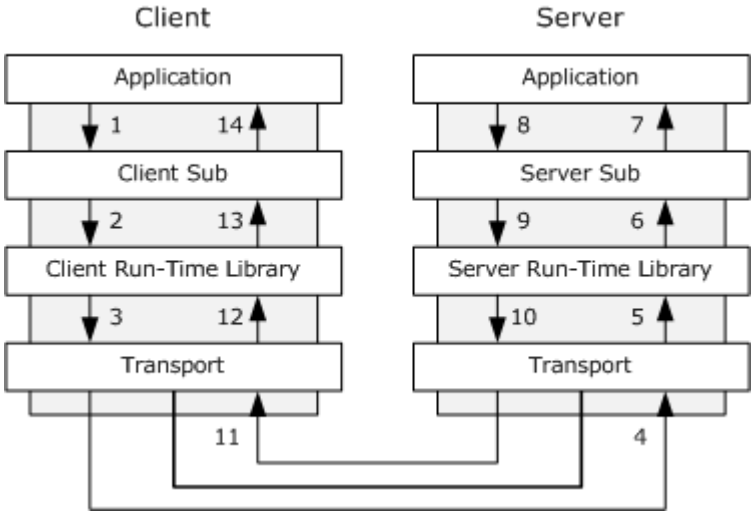

Figure 4: RPC architectural implementation model

As the illustration shows, the client application calls a local stub procedure instead of the actual code that is implementing the procedure. Stubs are compiled and linked with the client application. Instead of containing the actual code that implements the remote procedure, the client stub code:

- Retrieves the required parameters from the client address space.
- Translates the parameters as needed into a standard non-delivery report (NDR) format for transmission over the network.
- Calls functions in the RPC client run-time library to send the request and its parameters to the server.

The server performs the following steps to call the remote procedure.

- The server RPC run-time library functions accept the request and call the server stub procedure.
- The server stub retrieves the parameters from the network buffer and converts them from the network transmission format to the format the server needs.
- The server stub calls the actual procedure on the server.

The remote procedure then runs, possibly generating output parameters and a return value. When the remote procedure is complete, a similar sequence of steps returns the data to the client.

- The remote procedure returns its data to the server stub.
- The server stub converts output parameters to the format required for transmission over the network and returns them to the RPC run-time library functions.
- The server RPC run-time library functions transmit the data on the network to the client computer.

The client completes the process by accepting the data over the network and returning it to the calling function.

- The client RPC run-time library receives the remote-procedure return values and returns them to the client stub.
- The client stub converts the data from its NDR to the format used by the client computer. The stub writes data into the client memory and returns the result to the calling program on the client.
- The calling procedure continues as if the procedure had been called on the same computer.

For Windows, the run-time libraries are provided in two parts: an import library, which is linked with the application; and the RPC run-time library, which is implemented as a dynamic-link library (DLL).

The server application contains calls to the server run-time library functions that register the server's interface and allow the server to accept remote procedure calls. The server application also contains the application-specific remote procedures that are called by the client applications.

1.2.2.5 Microsoft RPC Components

Microsoft RPC includes the following major components:

- The MIDL compiler.
- Run-time libraries and header files.
- The **name service provider** (sometimes referred to as the locator).
- The **endpoint mapper** (sometimes referred to as the port mapper).

In the RPC model, the **Interface Definition Language (IDL)** is used to formally specify an interface to the remote procedures. The Microsoft implementation of this language is called the Microsoft Interface Definition Language, or MIDL.

If you create an interface using the Microsoft development environment, you must pass it through the MIDL compiler. This compiler generates the stubs that translate local procedure calls into remote procedure calls.

Stubs are placeholder functions that make the calls to the run-time library functions, which manage the remote procedure call. The advantage of this approach is that the network becomes almost completely transparent to your distributed application. Your client program calls what appear to be local procedures; the work of turning them into remote calls is done for you automatically.

All the code that translates data, accesses the network, and retrieves results is generated for you by the MIDL compiler and is invisible to your application.

2 Authentication Services Protocols

This section provides an overview of the protocols that provide authentication services. It provides a high-level conceptual overview of the core authentication protocols, classifying them broadly according to how the credentials are passed over a Windows network.

2.1 Authentication Services Overview

The Authentication Services protocols include protocols that provide account authentication, authorization, access control, policy enforcement, data-packets usage accounting, and data-packet auditing services between a client and a server.

These protocols can be used to provide:

- Authentication services substantially similar to how a Windows Server domain controller provides authentication services to Windows clients and users.
- Windows logon and security subsystems services for Windows clients.
- Remote administration functionality to support the authentication services listed above.

2.1.1 Protocols List

The Authentication Services protocols include two sets of protocols: the core protocols that provide the authentication services functionality, and a set of protocols that are required for an implementation of the core protocols (as listed in the Normative References section of the core-protocols technical specifications), or that provide core networking functionality. The latter set of protocols is referred to as the networking protocols group.

The following table lists the core protocols. The list does not include data structures, file structures, or algorithms that these core protocols depend on. The details for these technical dependencies are documented in the technical specifications for each core protocol.

Protocol Name	Protocol Description	Document Short Name
BackupKey Remote Protocol	Provides methods for communicating master-key backup information between the client machine and the domain controller (DC). The Data Protection API (DPAPI) uses this information whenever domain users change their passwords.	[MS-BKRP]
Credential Security Support Provider (CredSSP) Protocol	Enables an application to securely delegate a user's credentials from a client to a target server.	[MS-CSSP]
Directory Replication Service Remote Protocol (drsuapi)	Performs housekeeping and management operations for the Active Directory (AD) on a domain controller. An application uses this protocol through the ntdsapi.dll library.	[MS-DRSR]
Directory Services Setup Remote Protocol (dssetup)	Provides support for retrieving domain information, such as domain names, globally unique identifiers (GUIDs), and a server's configuration and state.	[MS-DSSP]
Kerberos Network Authentication Service (V5)	Extends the Kerberos V5 specification with support for interactive user logon. During an interactive logon, a	[MS-KILE]

Protocol Name	Protocol Description	Document Short Name
Extensions	Kerberos Key Distribution Center (KDC) running as part of a Windows domain controller receives a ticket-granting request from a client for a user logging onto the client. The user is then authenticated, and an account service is queried for account data that corresponds to the user and that is required for logging onto the computer. After the account data is retrieved, the data is inserted into a Kerberos ticket and sent to the requesting computer.	
Local Security Authority (Domain Policy) Remote Protocol	Provides access to local security authority (LSA), a protected subsystem of the Windows NT operating system and the Windows 2000 operating system that authenticates and logs users onto the local system.	[MS-LSAD]
Net Logon Remote Protocol	Provides the complete set of remote procedure call (RPC) methods for Netlogon transactions between client members and DCs.	[MS-NRPC]
NTLAN Manager (NTLM) Authentication Protocol	Enables authentication between clients and server. For Windows 2000 Server operating system, Windows XP operating system, Windows Server 2003 operating system,Windows Vista operating system, and Windows Server 2008 operating system, Kerberos is used instead of NTLM as the default authentication protocol but NTLM is used where Kerberos is not supported such as when a computer is not Kerberos-capable, the server has not joined a domain, or if Kerberos is not configured correctly.	[MS-NLMP]
NT LAN Manager (NTLM) Authentication Protocol; NT LAN Manager Authentication: Network News Transfer Protocol (NNTP) Extension	Extends the common Network News Transfer Protocol (NNTP) extensions documented in [RFC2980], with NTLM authentication. These Microsoft extensions allow exchange of messages in a public forum across the Internet with NTLM authentication.	[MS-NNTP]
NT LAN Manager (NTLM) Authentication Protocol; NT LAN Manager Authentication: Post Office Protocol – Version 3 (POP3) Extension	Are Microsoft security extensions that allow client applications that connect to the Post Office Protocol (POP3) server component included in Windows Server 2003 to use NTLM authentication.	[MS-POP3]]
NT LAN Manager (NTLM) Authentication Protocol; NT LAN Manager Authentication: Simple Mail Transfer Protocol (SMTP) Extension	Are Microsoft extensions to SMTP used to authenticate an SMTP session using NTLM.	[MS-SMTP]
Remote Certificate Mapping Protocol	Is a Microsoft protocol used by servers that authenticate users via X.509 certificates, as specified in [X509]. This protocol allows the server to use a directory, database, or other technology to map a user's X.509 certificate to a security principal.	[MS-RCMP]
Server Side Include (SSI)	Implements one of the Passport authentication methods.	[MS-PASS]

Protocol Name	Protocol Description	Document Short Name
1.4 Protocol	Passport is a trusted third-party identity-management service used from Microsoft websites and Web services. Passport supports a number of Kerberos-based authentication protocols. SSI 1.4 is a Microsoft protocol with client-server semantics to be used in web scenarios over Hypertext Transfer Protocol (HTTP).	
Windows Client Certificate Enrollment Protocol	Consists of a set of DCOM interfaces that allow clients to request various services from a Certificate Authority (CA). These services enable X.509 digital certificate enrollment, issuance, revocation, and property retrieval.	[MS-WCCE]

2.2 Authentication Services Functionality

This section provides a general overview of how the core protocols operate as a group to provide authentication services, followed by an individual view of each core protocol.

The Implementation Scenarios section, later in this document, provides examples that illustrate the interaction of some of the protocols in two sample configurations.

2.2.1 Authentication Services Concepts

This section describes basic authentication protocols concepts and provides some background for the authentication model of Windows.

The main purpose of authentication is to establish the identity of a client and a server that are communicating with each other. Authentication protocols enable this communication in any environment, especially one in which an attacker can inspect or tamper (change, suppress, or replay) messages sent during this communication.

Authentication takes place on the client and on the server. In some cases, server-side authentication, where the server performs authentication of the client, might be sufficient. In others, client-side authentication, where the client authenticates the identity of the server, is sufficient. An example of client-side authentication is the use of Secure Socket Layer (SSL) on the Internet, which is centered on assuring the identity of the server to the client. An example of server-side authentication might be authentication of a client in a protected network environment where all valuable resources reside on a single server, and the server needs to be concerned only about the identity of the client.

On modern networks, however, proving the identity of both the client and the server is critical. The client needs to be assured of the identity of the server to avoid divulging something important to a rogue server; the servers must be assured of the identity of the client to avoid granting the client inappropriate access. This relationship is commonly referred to as mutual authentication.

In tighter-security environments, authentication must be performed using cryptographic operations of some sort, such as encryption or signatures. The two main types of encryption are *symmetric encryption* and *asymmetric encryption*. Symmetric encryption uses the same key to encrypt and decrypt a message. Asymmetric encryption uses one key to encrypt and a different key to decrypt; these keys are linked by mathematical requirements. Signatures can be implemented in a number of ways, through keyed hashes, encrypted hashes, and so on.

2.2.2 Generic Security Support Model

In the early 1990s, John Linn, then of Digital Equipment Corp., proposed that applications not be tied to specific security protocols. This proposal was the genesis of the Generic Security Support Application Programming Interface (GSS API). This interface has driven the model of most authentication protocols that are intended for use within an application protocol. The GSS API is generally referred to as GSS style or the GSS model. Note that there have been a number of channel-based protocols, such as SSL, Transport Layer Security (TLS), and Secure Shell (SSH), which are intended to be below the application protocol layer.

The GSS model, however, has led to a simplified form of interaction between the application protocol and the authentication protocol. In this model, the application protocol is responsible for ferrying discrete, opaque packets produced by the authentication protocol. The application has no visibility into the contents of the message; its responsibility is merely to carry the message. These messages, referred to as *tokens* by the GSS specifications, implement the authentication process.

The application in this model first calls the authentication protocol on the client. The client portion of the authentication protocol creates a token and returns it to the calling application. The application then transmits that token to the server side of its connection, embedded within the application protocol. On the server side, the server application extracts the token and supplies it to the authentication protocol on the server side. The server-side authentication protocol can process the token and possibly generate a response or decide that authentication is complete. If another token is generated, the application must carry the token back to the client, where the process continues.

This exchange of security tokens continues until one or both sides decide that authentication is complete. If authentication fails, the application should drop the connection and indicate the error. If authentication succeeds, the application can then be assured of the identity of the participants, as far as the underlying protocol can accomplish.

When the authentication is complete, session-specific security services can be available. The application can then invoke the authentication protocol to sign or encrypt the messages that are sent as part of the application protocol. These operations are done in much the same way as above, where the application can indicate what portions of the message are to be encrypted, and then it must include a per-message security token. By signing and or encrypting the messages, the application can obtain privacy, resistance to message tampering, and detection of messages dropped, suppressed, or replayed.

Windows generally follows the GSS model of security. In the GSS model, application protocols try to be as agnostic as possible for the specific forms of authentication used in the session. In return, the security protocol (or mechanism) limits its specific behavior to some well-defined interfaces and communicates in opaque messages, referred to as *tokens* in the GSS documents.

Therefore, the application is responsible for calling the security runtime to secure its protocol. Conversely, the application does not need to be aware of any specifics of the security protocol that it has selected. This arrangement leads to the somewhat surprising condition in some instances that an application protocol may not know what security protocol it is using.

2.2.3 NT Local Area Network Manager (LM) Authentication

Windows networking has its roots in Microsoft's Windows Local Area Network (LAN) Manager (LM). LM was designed at a time when client authentication was sufficient for most needs and many computers did not have the capacity to execute complicated algorithms. For example, exhaustively searching Data Encryption Standard (DES) keys was unthinkable except for highly specialized government computers. LM authentication used a straightforward challenge-response style of authentication and was sufficient for most needs at that time.

Windows NT operating system, Windows 2000 operating system, and Windows XP operating system support authentication using LM, NT LAN Manager (NTLM), and NT LAN Manager version 2 (NTLM2) protocols. Windows 2000 and Windows XP use Kerberos v5 as the default authentication protocol. Because few networks use a single operating system (OS) version or use applications that use the same authentication APIs, Windows NT, Windows 2000, and Windows XP support current and legacy authentication methods to preserve compatibility with down-level versions of Windows.

Microsoft's decision to adopt Kerberos for Windows and move away from NTLM required a substantial design change for a number of protocols. This design-change process is still occurring today. Rather than redesign again when circumstances require a new or additional security protocol, Microsoft chose to add a protocol—in this case, Simple and Protected GSS Application Program Interface Negotiation Mechanism (SPNEGO), to allow security protocol selection and extension.

2.2.4 Authentication Methods

A basic definition of an authentication protocol is any protocol used for validating the identity of a user to determine whether to grant the user access to resources on a network. Authentication protocols can be classified according to how the credentials are passed over the network. Some common kinds of authentication protocols include:

- Anonymous access This method allows anonymous users on the Internet access to web content on a server. Some Windows server roles such as Microsoft Internet Information Services (IIS) use this method.
- Basic authentication This method transmits passwords as clear text and is often used in UNIX environments and by File Transfer Protocol (FTP) services. Certain Windows server roles such as IIS also implement this method. This mode can be secured by encrypting the transport, for example, with SSL.
- Digest authentication This method hashes (digests) the user's password before transmitting it, making the password more secure than basic authentication does. Digest authentication is part of the Hypertext Transfer Protocol (HTTP) 1.1 protocol and is also supported by IIS.
- Windows NTLM Challenge /Response This is the standard secure authentication method for Windows NT operating system domains. In certain Windows server components such as IIS, this protocol is referred to as integrated Windows authentication, or plainly as integrated authentication.
- Kerberos v5 This is the standard authentication security protocol for Windows 2000 operating system and Windows Server 2008 operating system domains.
- X.509 Client Certificate authentication This authentication is used in conjunction with SSL and digital certificates; it is also supported by IIS.

2.2.4.1 Anonymous Access

Anonymous access is an authentication method in which a user's identity is not verified. Anonymous access is one of several authentication protocols supported by Microsoft IIS.

Strictly speaking, anonymous access is not an authentication scheme at all because the users' credentials are not requested and, if supplied, they are ignored. Anonymous access is used to allow anonymous users (everyone) to gain access to public content hosted on a web server. Anonymous access is used on low-security, public websites where the identity of the person visiting the site is not important.

Some Windows server applications such as IIS can be configured to allow anonymous users to access the application resources. This access allows distrusted users from unsecured networks, such as the Internet, to access data that is made available to the public at large.

2.2.4.2 Basic Authentication

Basic authentication, also called cleartext or plaintext authentication, is an authentication method that passes users' credentials over a network in an unencrypted form. Basic authentication is defined as part of HTTP version 1 [RFC2617]. Basic authentication is not a secure authentication scheme because anyone who can intercept network traffic and read it using a protocol analyzer can obtain a user's credentials. Although it is sometimes cleartext authentication, basic authentication actually encodes a user's credentials using a well-known public encoding algorithm known as UUencoding or Base64. Because the algorithm for this encoding method is so well known, however, it is easy to decode text and extract a user's credentials from a basic authentication session.

One of the disadvantages of implementing basic authentication is that it is inherently insecure because of the cleartext transmission of the user's password. However, IIS does allow basic authentication to be implemented with SSL encryption, in which case an encrypted session is first established for the user, after with the user's credentials are passed to IIS in encrypted form.

One advantage of implementing basic authentication is that it can be performed through a firewall or proxy server (Integrated Windows authentication or Windows NTLM cannot work in this case).

Basic authentication is often used in a UNIX environment for authenticating remote HTTP users.

2.2.4.3 Digest Authentication

The digest authentication mechanism is used in environments that require users to authenticate to servers to access secure resources. Kerberos and public key-based authentication offer stronger security guarantees, in terms of both initial authentication and subsequent confidentiality and integrity of client-server traffic.

The digest authentication mechanism can be used in environments where these stronger mechanisms are not available, and it can be implemented in interoperability scenarios that involve browsers, web servers, and directory services from multiple vendors.

The following diagram and procedural steps illustrate the function of the Digest Protocol Extensions as specified in [MS-DPSP].

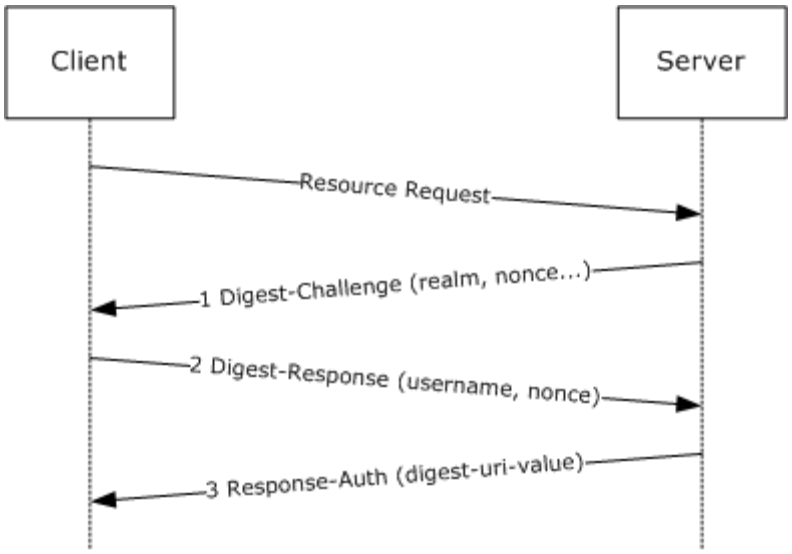

Figure 5: Digest authentication Sequence

After the client attempts to access a protected resource on the server, the server returns a digest-challenge message to the client. Among other fields, the digest-challenge message includes a randomly generated user name, **nonce,** and the host name of the server (via the realm field).

1. The client obtains the user name and password for the user and constructs a response to the server's challenge. In the digest response, the client proves knowledge of the user's password by performing a **keyed hash** over the user name, nonce, and other fields (the password is fed into the hash).

2. The server validates the digest-response message by using the user name that the client sent to look up the user's password, recomputing the **keyed hash** over fields in the digest-response message, and then comparing the resulting value to the value sent by the client. If the server's computed hash and the hash sent by the client match, the client's digest-response message is valid. The server further checks that the client sent the expected nonce value (and not an old, replayed value). Once the digest response is validated, and the client nonce is correct, the client is authenticated to the server. For mutual authentication, the server has the option to send a **keyed hash** over the Uniform Resource Identifier (URI) that the client requested and return it to the client in the Response-Auth message.

2.2.4.4 NTLM Challenge/Response Authentication

NTLM is a family of security protocols in Windows. NTLM uses a challenge/response authentication mechanism that application protocols use to authenticate remote users and optionally to provide session security when the application requests it.

2.2.4.4.1 Message Flow for Basic NTLM Authentication

NTLM is an ongoing extension to the original LM authentication protocol. NTLM is conceptually straightforward and performs only client authentication. NTLM has undergone some revision (known as NTLMv2), which incorporates additional information into the computation of the response, but still follows the same general message flow as shown in the following figure:

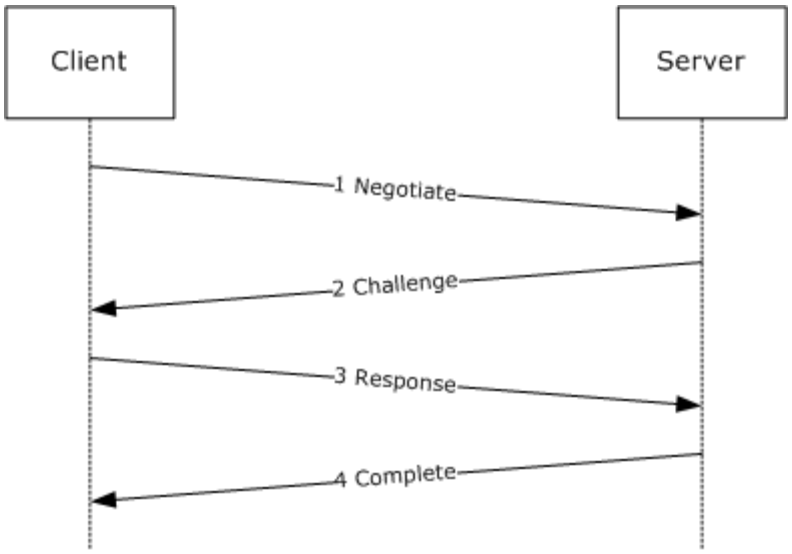

Figure 6: Basic NTLM authentication

These challenge/response messages are in fact carried by an application protocol. The basic flow of the security tokens or messages is as follows:

1. The client sends an initial message to the server, advertising certain options or capabilities such as cryptographic algorithm support (1).
2. The server creates a challenge*, c,* and returns the challenge and the options or capabilities that it can support to the client (2).
3. The client computes a function on the challenge, *resp = f(c, password)*, and sends the results to the server, along with the user's textual name and domain (3).
4. The server looks up the user (by the name passed) and computes the same function, *f(c, user's password).* If the result matches *resp* (that is, what the client sent in step 3), the passwords are presumed to match, and the user is authenticated (4).

The message flow above is for illustrative purposes only. For technical detailed information about NTLM, see [MS-NLMP]. For example, *password* (step 3 above) is (in practice) a hashed, derivative binary form of the actual textual password.

2.2.4.4.2 NTLM Message Flow in a Domain Environment

NTLM authentication becomes more complicated in a domain scenario in which the domain controller hosts the account database, and members of the domain do not have direct access to the account database. In this scenario, the flow from client to server is the same but there is a new interaction with the domain controller, as shown in the following figure:

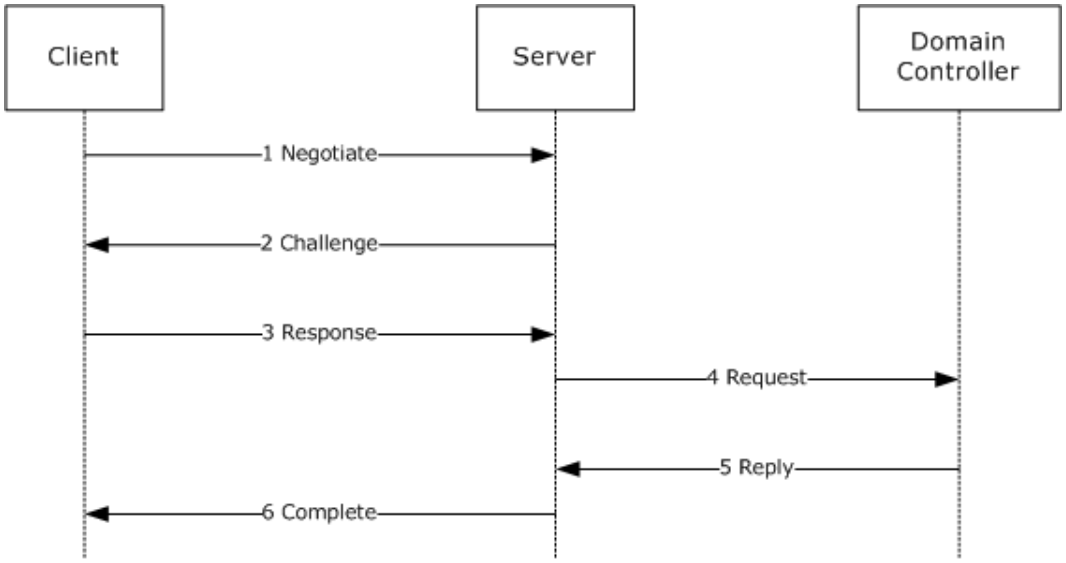

Figure 7: NTLM authentication in a domain environment

In a domain environment, the server adds two additional messages in the flow:

1. Forward challenge, *c*, the response, *resp*, the user name, and the user domain from the client to the domain controller (4).

2. The domain controller looks up the account record for the user specified, computes *f(c, password),* and compares this to the *resp* from the client. If it matches, the domain controller can gather up the user account, group memberships, and so on, and then send this back to the server as the reply (5).

The server can respond to the client that authentication is complete (6).

Adding a server as an intermediary that forwards the authentication request to the domain controller is a powerful point of indirection. Doing this allows different account authorities to be woven into a trust network that is completely invisible to the client. This process of indirection (DC A forwarding a request to domain controller B, which forwards a request to domain controller C, and so on) can be repeated indefinitely.

NTLM cannot provide mutual authentication in all situations; the client can never be assured of the identity of the server in a general manner. The only time NTLM can provide mutual authentication is when the client knows through out-of-band means that the user name used for authentication exists only on the target server. NTLMv2 offers the capability to include domain name and server name information in the authentication exchange. This capability can constrain the authentication to be at least within the extended set of trusted domains.

2.2.4.5 Kerberos Authentication

In 1993, Microsoft made the decision to start working toward adopting Kerberos (Kerberos is an Internet Engineering Task Force [IETF] standard protocol, and a multitude of documents are available that describe how it works, but [RFC4120] can be used as the canonical reference.) Kerberos support for mutual authentication, transitive trust among authorities, extensibility, public inspection and review, and performance and caching all pointed to this as the authentication protocol for enterprise deployment for the foreseeable future. Windows 2000 operating system shipped in 1999 with Kerberos supported natively.

Kerberos is based on a client, a server, and a trusted third party called the Key Distribution Center (KDC). The KDC is associated with an account database and has a key shared with each client or server that it knows about. The management of the account database is explicitly done outside of the Kerberos authentication process. However, each client or server knows its own key through some known method. For example, a human (client) knows his or her own passwords.

Kerberos is based on passing tickets from clients to servers; these tickets are originally created by the KDC that serves the domain in which the account resides. The ticket is useful because the receiving server knows that only the KDC can create that ticket and is trustworthy. The whole process bootstraps by obtaining a ticket to the KDC, which is used to request further tickets. This ticket-granting ticket (TGT) is obtained from a KDC through a slightly different request.

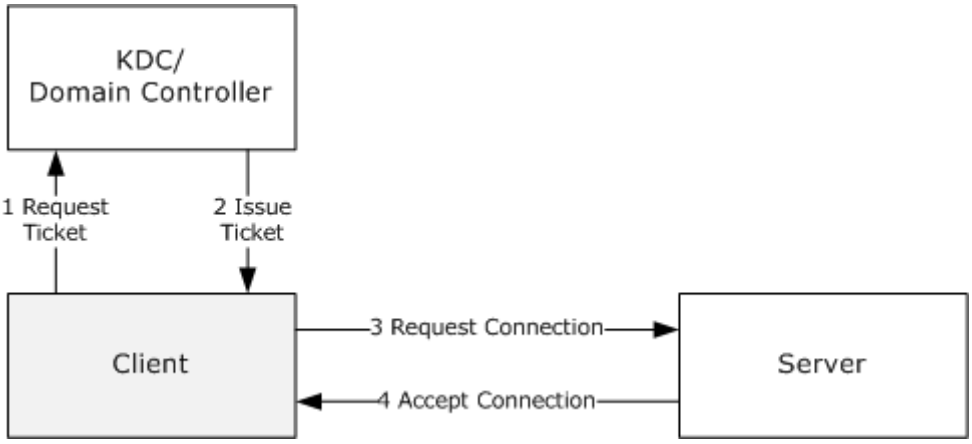

Figure 8: Windows Kerberos Initialization sequence

2.2.4.5.1 Kerberos Message Flow

The basic Kerberos exchange (assuming the client already has the initial TGT) is as follows:

1. The client issues a ticket request message to the KDC on the domain controller requesting a ticket to a named server. The ticket request message contains the TGT and cryptographic proof that the client can use that TGT.

2. The KDC on the domain controller looks up the server and issues a ticket to that server encrypted in the secret key of the server. The new ticket contains all the restrictions of the original TGT and any data that the TGT was carrying. The KDC creates a ticket reply message that contains the ticket and a copy of most of the ticket's contents encrypted for the client. Most important is a unique key that the client and server are to use.

3. The client submits the ticket to the server in an authentication message. The message has the ticket from the KDC and has parts encrypted using the unique key from step 2.

4. The server decodes the authentication message and finds the ticket. Within the ticket, the server finds the unique key from step 2. It can then decrypt and verify the rest of the authentication message. When decryption is successful, the server creates the authentication reply message, which it encrypts with the unique key. On receipt, the client verifies that the unique key is used.

Kerberos as a protocol is strictly an authentication protocol; it is designed to convey only the identity of the principals on each side of the connection. Kerberos does contain extensibility points to allow extra vendor-defined information to be conveyed (along with the ticket) during authentication. This capability is expressed as the *auth_data* field in the Kerberos ticket.

Windows uses this *auth_data* field in the ticket to pass along the account's security identifier (SID), as defined in [MS-DTYP] section 2.4.2, and to group SIDs during authentication. Microsoft Windows encodes authorization information into a structure called Privilege Attribute Certificate (PAC), as specified in [MS-PAC]. The PAC contains all of the account's group memberships, credential information, profile and policy information, and supporting security metadata. The PAC is used to provide sufficient context on the server to make authorization decisions; for example, "can this user open this file?" The encoding of the group information is publicly available from Microsoft. See Utilizing the Windows 2000 Authorization Data in Kerberos Tickets for Access Control to Resources [KERBPAC].

The group membership information is a common use of the *auth_data* field. However, Microsoft also included additional profile and account management information, such as the location of the account's root directory, which greatly aids certain enterprise deployment scenarios.

One strong point of Kerberos is that it is not involved after the ticket is issued until the ticket expires and the client needs a new one. This means that the client and server don't need to involve the KDC for every connection.

Much like other protocols, Kerberos can leverage more than one domain. If the server being requested by a client is in a different domain, the KDC can return a TGT to that other domain. The client can then retry the request on a KDC in the other domain. This process can span multiple domains as long as there is a trusting relationship among the domains.

One issue with a secret key system such as Kerberos is that anyone who knows the key for a principal can create a ticket to that principal. Thus, even though the KDC is the normal creator of tickets, a user, knowing its own password, for example, can create a ticket and send the ticket to the principal. For authentication only, this action is not a threat; the user is fooling itself with its own actions.

When the ticket includes authorization data that is respected and interpreted by something other than code running as that user, however, the potential for problems exists. In the Windows implementation, the authorization data results in a Windows *access token,* which is a system-provided object that encapsulates an account's identity, group memberships, and system privileges. An access token in Windows is used to make authorization decisions by the system—for example, validating access to a file.

A malicious user might construct a ticket to itself that contains a PAC that indicates that the user is a member of an unauthorized group (for example, the Administrators group). If this PAC is processed naively, the system accepts it and allows the user inappropriate access to the system. Windows compensates for this potential by involving the domain controller when a non-privileged server receives a ticket. The Windows Kerberos implementation calls back to the domain controller through the Netlogon channel to verify the contents of the PAC.

2.2.4.6 SSL and TLS

SSL is a protocol first introduced by Netscape to allow browsers to authenticate servers. SSL went through a number of revisions, culminating in an adoption by the IETF and its becoming the TLS protocol (see [RFC2246]).

SSL and TLS are primarily used for authenticating servers and creating a secure pipe between the client and the server, but they do allow for authenticating the client. At almost any point after the channel has been established, the server can demand client authentication. The client signs a challenge from the server with its private, asymmetric key, and then the client sends both the signed data and the certificate for that key to the server.

On Windows, the certificate can be associated with an account in several ways, based on local policy and on what fields from the certificate are interesting to the administrator of the server or domain. Ultimately, however, the SSL/TLS implementation on the server calls the domain controller through the Netlogon channel and asks the SSL instance on the domain controller to determine what account is associated with this certificate.

The account is determined through a number of possible mappings based on the fields present in the certificate. For example, *subjectAltName, commonName*, and other fields can be used to find the account in Windows Active Directory (AD). Once found, the account information (such as account SID and group membership SIDs) is returned to the client. Much like the NTLM case, involving the domain, this process can extend through arbitrary trust relationships. The format for this information is the same as the format of the PAC data from Kerberos.

2.2.4.7 SPNEGO

SPNEGO is an authentication protocol that actually does no authentication. Rather, it is an authentication protocol that allows secure negotiation among other security protocols when the client and server might support more than one protocol. Windows uses SPNEGO instead of a specific protocol to allow for simpler substitutions of additional security protocols in the future.

2.2.4.7.1 SPNEGO Message Flow

SPNEGO fits into the system as a security protocol and shuttles the actual messages of the underlying security protocols back and forth between the client and server. As such, it does not implement any specific security features but performs the very important operation of enforcing policy among security protocols.

SPNEGO is defined as a GSS mechanism, and, as such, it follows the general model of passing security tokens from one machine to another. SPNEGO is a very simple protocol that passes a list of acceptable security protocols (or mechanisms, in GSS parlance) along with an optimistic token from one of these protocols. The response from the server is what protocol has been selected and, optionally (if it matched the optimistic token), a response from that security protocol.

The SPNEGO module asks each protocol available on the system, in turn, whether that protocol can, potentially, authenticate the server named by the component invoking SPNEGO. In the Windows implementation, the first protocol to respond positively is selected as the optimistic choice. Once the protocol has been selected, SPNEGO gets out of the way and lets the application protocol work directly with the selected authentication protocol.

On Windows, SPNEGO has the important task of deciding what protocol is valid for a particular connection. It does this by interpreting a number of input states, and then adjusting the list of security protocols it offers the server. The SPNEGO implementation knows that NTLM cannot implement the same guarantees (because of legacy) that more modern protocols can. Therefore, NTLM is treated in a special manner on the Windows system. To decide the most appropriate protocol, the SPNEGO implementation takes the following information:

- User domain functional level, if known.
- System-wide mutual authentication policy.
- Name of the server.

If the user is a member of a Windows 2000 or later domain, SPNEGO knows that NTLM is a second choice, and that at least Kerberos should be available. Therefore, if Kerberos indicates that it should be used (by returning an optimistic token), SPNEGO does not allow further downgrade to NTLM should the Kerberos authentication fail. Also, if the KDC cannot be contacted, and the user is a

member of a Windows 2000 or later domain, the SPNEGO implementation fails, indicating that it cannot authenticate the connection. The purpose of this failure is to prevent a class of security attacks.

The following sections provide detail for additional authentication services core protocols referenced in section 2.1.1 Protocols List.

2.2.5 Directory Replication Service Remote Protocol

The Directory Replication Service Remote Protocol is an RPC protocol for replication between DCs and management of AD. The protocol consists of one RPC interface, named **drsuapi**.

For a client to establish an RPC connection to an AD Lightweight Directory Services (AD/LDS) domain controller, it needs to know the name of the computer and the number of the LDAP port on which the AD/LDS domain controller is listening. First, the client establishes a connection to the endpoint mapper service on the computer.

Next the client enumerates all endpoints that are registered for the desired interface ID. Finally the client selects the endpoint whose annotation equals the LDAP port number of the desired AD/LDS domain controller.

This protocol is appropriate for the management of objects in a directory, as well as the overall management of the directory service.

2.2.6 Directory Services Setup Remote Protocol

The Directory Services Setup Remote Protocol (drssetup) provides a remote procedure call (RPC) interface for querying domain-related computer state and configuration data. The client end of the Directory Services Setup Remote Protocol is an application that issues method calls on the RPC interface.

The server end of the protocol obtains and replies to the client with the requested data about the computer on which the server is running. If the client connects to and requests information about a domain controller for the directory service, this data includes the status of any pending promotion or demotion of that domain controller.

2.2.7 Local Security Authority Remote Protocol

The Local Security Authority (Domain Policy) Remote Protocol provides a remote procedure call (RPC) interface used for providing remote management for policy settings related to

- Account objects
- Secret objects
- Trusted domain objects (TDOs)
- Other miscellaneous security-related policy settings.

The client end of the Local Security Authority (Domain Policy) Remote Protocol is an application that issues method calls on the RPC interface. The server end of the Local Security Authority (Domain Policy) Remote Protocol is a service that implements support for this RPC interface.

Primary use cases for remote management include

- Creating, deleting, enumerating, and modifying trusts, account objects, and secret objects.

- Querying and modifying policy settings unrelated to trusted domain objects (TDOs), account objects or secret objects, such as lifetimes of Kerberos tickets.

This protocol is used by Windows clients for the "domain join" operation (as specified in [MS-ADTS] as an implementation choice to achieve the end state, as specified in [MS-ADTS]. The specific profile of the Local Security Authority (Domain Policy) Remote Protocol for the "domain join" scenario is specified in section 1.6 as "Windows client-to-server interoperability".

The server end of the Local Security Authority (Domain Policy) Remote Protocol can be in one of two different states—a "domain controller" or a "non-domain controller".

The "Domain controller" state supports all of the Local Security Authority (Domain Policy) Remote Protocol's functionality. However, "non-domain controller" state does not support all of the protocol functionality. This specification contains details on which functionality is not supported in "non-domain controller" state on a case-by-case basis when describing message processing for each method supported by this protocol.

This protocol is a simple request/response-based RPC protocol. Typically, there are no long-lived sessions, although clients can cache the RPC connection and reuse it over time.

This protocol is applicable to the following two high-level scenarios.

1. Remote management of trusted domains, account objects or secret objects, or other miscellaneous machine and domain policy settings controlled by the protocol.

2. Windows client-to-server interoperability.

2.2.8 Net Logon Remote Protocol

The Net Logon Remote Protocol is used for user and machine authentication in domain-based networks. A domain is an administrative collection of machines that share the same user account database and common security policy. The user account database is maintained on special servers called domain controllers (DCs).

Domain controllers are responsible for authenticating access to domain resources by validating supplied user credentials against locally stored shared secrets. The Net Logon Remote Protocol is used for secure communication between machines in a domain (both domain members and DCs) and DCs. The communication is secured by using a shared session key computed between the client and the domain controller that is engaged in the secure communication. The session key is computed by using a preconfigured shared secret that is known to the client and the domain controller.

After establishing a secure channel, a client can call any of the Net Logon Remote Protocol methods that require a secure channel. This requires both the client and the server to have a working RPC implementation, including the security extensions, as specified in [MS-RPCE].

The following section describes a typical scenario in which the Net Logon Remote Protocol is used.

2.2.8.1 Pass-Through Authentication

Consider a scenario in which a user logs in on a client machine with a domain account that connects to a server. The connection needs to be authenticated. The client and the server engage in an authentication protocol, such as NTLM (as specified in [MS-NLMP]), that validates the user credentials and logs on the user on to the server upon successful validation. This type of logon is referred to as network logon because it happens over a network connection from the client to the

server (versus an interactive logon that happens when users enter their credentials interactively at the client machines' consoles).

To authenticate the user, the server needs to pass the user credentials securely to a domain controller in the domain of the user account. (The domain controller is the entity, other than the client machine, that knows the user secret key; that is, the user password.) After the logon request is delivered to the domain controller and the domain controller successfully validates the credentials, the domain controller refers back to the server attributes of the user account that the server can use in authorization decisions (such as granting the user access to a particular file).

It is the responsibility of the Net Logon Remote Protocol to deliver the logon request to the domain controller over a secure channel that is established from the server (acting as the secure channel client) to the domain controller (acting as the secure channel server). The secure channel is achieved by encrypting the communication traffic with a session key computed using a secret key (called a server's machine account password) shared by the server and the domain controller. This is shown in the following diagram.

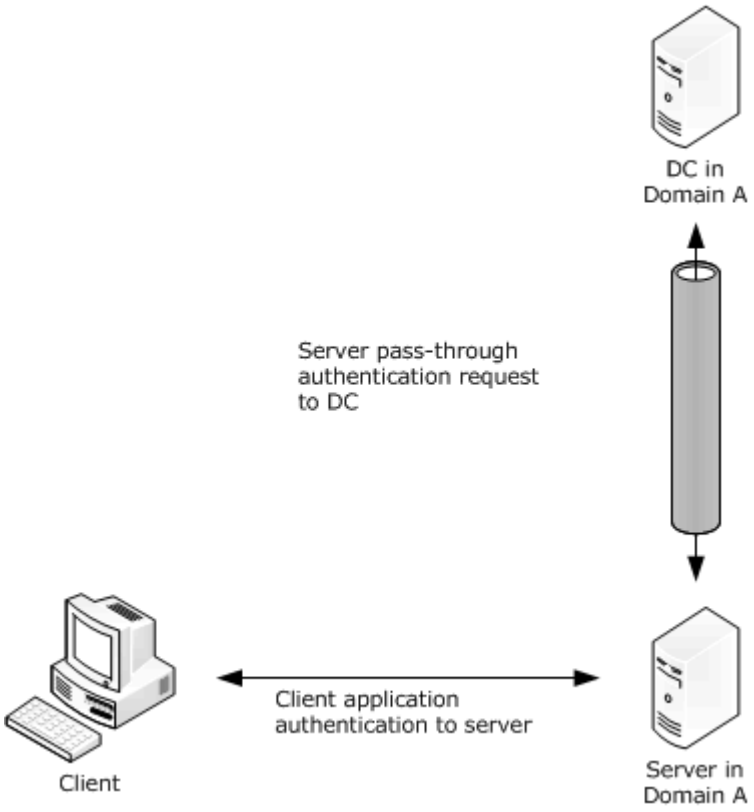

Figure 9: Pass-through authentication

2.2.9 Backup Key Remote Protocol

The Backup Key Remote Protocol provides a way of encrypting a secret value (such as a cryptographic key) so that the encrypted secret can then be backed up on storage to which many parties have read access. In this way, the encrypted secret is secured while being readily available for later recovery. The encrypted form of the secret is called the wrapped secret. The process of restoring the original secret from the wrapped copy is called unwrapping.

Although this protocol was designed specifically to wrap and unwrap 64-byte cryptographic keys, all of its variants will wrap arbitrary secrets, not just cryptographic keys, and only some of its variants limit the caller to 64-byte secrets. This specification and the name of the protocol refer to keys, but the reader is free to assume these key payloads can be any secret value.

The Backup Key Remote Protocol includes four sub-protocols: two for wrapping cryptographic keys and two for unwrapping them. The two options for each of the wrapping and unwrapping protocols depend on whether the wrapping key is a public key or a symmetric key.

In the symmetric key case, the key to be wrapped is sent to the server and the wrapped key is returned to the client. For public key wrapping, the client retrieves the server's public key and then uses it to wrap the key on the client, thus minimizing the exposure of that wrapped key. For unwrapping, the wrapped key is always sent to the server and the original key is returned to the client.

The key wrapping retrieval sequence is depicted in the following diagram.

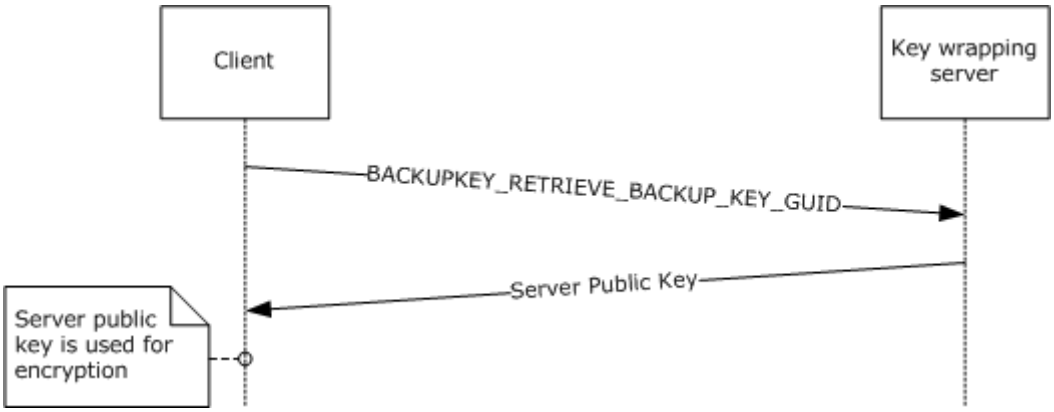

Figure 10: Key wrapping retrieval sequence

The steps are as follows:

1. The client sends a BACKUPKEY_RETRIEVE_BACKUP_KEY_GUID message to the key wrapping server.

2. The key wrapping server sends a DER-encoded X.509 self-signed certificate (as specified in [X509]) containing its public key (as specified in section 2.2.1).

3. The client uses the public key to encrypt a key, wrapping it for later normal backup.

Additional modes of key wrapping and retrieval are described in detail in the Backup Key Remote Protocol specification [MS-BKRP].

2.2.10 NTLM Authentication Protocol; NT LAN Manager Authentication: Network News Transfer Protocol Extension

The NTLM Authentication: Network News Transfer Protocol (NNTP) Extension specifies how an NNTP client and NNTP server can use the NTLM Authentication Protocol, as specified in [MS-NLMP], to allow the NNTP server to authenticate the NNTP client. NTLM is a challenge/response authentication protocol that depends on the application layer protocols to transport NTLM packets from client to server and from server to client.

The NTLM Authentication: NNTP Extension defines how NNTP is extended to perform authentication by using the NTLM Authentication Protocol. The NNTP standard defines an extensibility mechanism for arbitrary authentication protocols to be plugged into the core protocol. The following diagram illustrates the sequence of transformations that are performed on an NTLM message to produce a message that can be sent over NNTP.

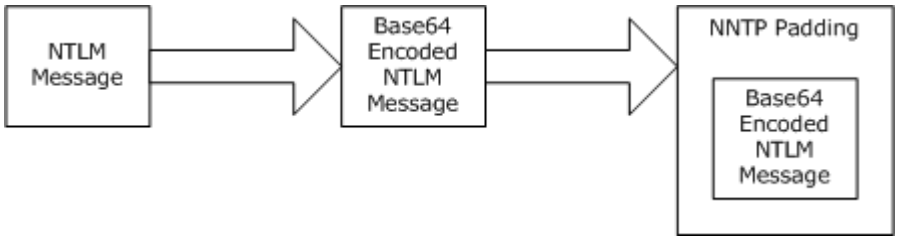

Figure 11: Relationship between NTLM message and NNTP: NTLM Authentication Protocol message

The NTLM Authentication: NNTP Extension defines a server and a client role.

When NNTP wants to perform an NTLM authentication, it needs to interact with the NTLM software appropriately. Below is an overview of this interaction.

2.2.10.1 Client Role

If acting as an NNTP client:

1. The NTLM software returns the first NTLM message to the client, to be sent to the server.
2. The client applies the base64 encoding and NNTP-padding transformations in order to produce an NNTP message and send this message to the server.
3. The client waits for a response from the server. When the response is received, the client checks to see whether it indicates the end of authentication (success or failure), or that authentication is continuing.
4. If the authentication is continuing, the response message is stripped of the NNTP padding, base64 decoded, and passed into the NTLM software, upon which the NTLM software may return another NTLM message that needs to be sent to the server. Steps 2 through 4 are repeated until authentication succeeds or fails.

2.2.10.2 Server Role

If acting as an NNTP server:

The server waits to receive the first NNTP authentication message from the client.

1. When an NNTP message is received from the client, the NNTP padding is removed, the message base64 decoded and the resulting NTLM message is passed into the NTLM software.
2. The NTLM software will return a status that indicates whether authentication completed successfully or failed; or whether more NTLM messages need to be exchanged to complete the authentication.
3. If the authentication is continuing, the NTLM software will return an NTLM message that needs to be sent to the server. This message is base64 encoded, and the NNTP padding is applied and sent to the client. Steps 2 through 4 are repeated until authentication succeeds or fails.

2.2.11 NTLM Authentication Protocol; NT LAN Manager Authentication: Post Office Protocol - Version 3 (POP3) Extension

The NTLM Authentication Protocol; NT LAN Manager Authentication: Post Office Protocol – Version 3 (POP3) Extension specifies how a POP3 client and POP3 server can use the Windows NTLM Authentication Protocol, as specified in [MS-NLMP], to allow the POP3 server to authenticate the POP3 client. NTLM is a challenge-response-style authentication protocol that depends on the application layer protocols to transport NTLM packets from client to server, and from server to client.

The POP3 Extension defines how the POP3 Authentication command [RFC1734] is used to perform authentication using the NTLM authentication protocol. The POP3 Authentication command standard defines an extensibility mechanism for arbitrary authentication protocols to be plugged into the core protocol.

The following diagram illustrates the sequence of transformations performed on an NTLM message to produce a message that can be sent over POP3.

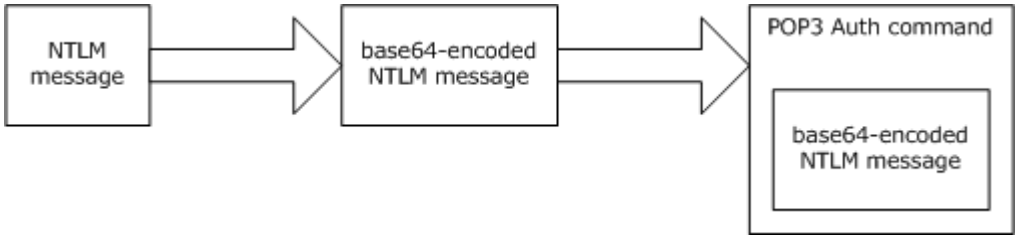

Figure 12: Relationship between NTLM message and POP3: NTLM Authentication Protocol message

The POP3 Extension defines a server and a client role.

When POP3 wishes to perform an NTLM authentication, it needs to interact with the NTLM subsystem appropriately. Below is an overview of this interaction.

2.2.11.1 Client Role

If acting as a POP3 client:

1. The NTLM subsystem returns the first NTLM message to the client, to be sent to the server.
2. The client should apply the base64-encoding and POP3-padding transformations to produce a POP3 message and send this message to the server.
3. The client should wait for a response from the server. When the response is received, it checks to see whether it indicates the end of authentication (success or failure), or that authentication is continuing.
4. If the authentication is continuing, the response message is stripped of the POP3 padding, base64 decoded, and passed into the NTLM subsystem, upon which the NTLM subsystem may return another NTLM message that needs to be sent to the server. Steps 2 through 4 are repeated till authentication succeeds or fails.

2.2.11.2 Server Role

If acting as a POP3 server:

1. The server waits to receive the first POP3 authentication message from the client.

2. When a POP3 message is received from the client, the POP3 padding is removed, the message base64 decoded, and the resulting NTLM message passed into the NTLM subsystem.

3. The NTLM subsystem will return a status indicating whether authentication completed successfully, failed, or whether more NTLM messages need to be exchanged to complete the authentication.

4. If the authentication is continuing, the NTLM subsystem will return an NTLM message that needs to be sent to the server. This message is base64-encoded, the POP3 padding is applied and sent to the client. Steps 2 through 4 are repeated until authentication succeeds or fails.

The NTLM POP3 Extension is used only in the following case: when implementing a POP3 client that needs to authenticate to a POP3 server by using NTLM authentication.

2.2.12 Remote Certificate Mapping Protocol

The Remote Certificate Mapping Protocol is used in deployments where users rely on X.509, as specified in [X509], certificates to gain access to resources. After a client authenticates itself to a server using an X.509 certificate, the server uses the Remote Certificate Mapping Protocol to contact a directory to determine the authorization information to use, such as group memberships. The Remote Certificate Mapping Protocol returns a PAC, as specified in [MS-PAC], which represents the user's identity and group memberships, suitable for making authorization decisions.

There are three steps to associate a certificate with an account for the purposes of authorization.

The **subjectAltName** field of the X.509 certificate should be treated as a user principal name (UPN) and used as the key, in the database sense, to locate the account record and corresponding authorization information.

The issuer and subject names should be taken together as a key, again in the database sense, to locate the account record.

The issuer name alone should be used as the lookup key when locating the account record.

The Remote Certificate Mapping Protocol server uses attributes of this X.509 certificate and the indicated methods by the client to determine the authorization information. An implementer of the Remote Certificate Mapping Protocol must be familiar with X.509 to use the Remote Certificate Mapping Protocol. For more information about X.509, see [GUTMANN].

2.2.13 NTLM Authentication Protocol; NT LAN Manager Authentication: Simple Mail Transfer Protocol Extension

NTLM Authentication: Simple Mail Transfer Protocol (SMTP) Extension specifies how an SMTP client and SMTP server can use the NTLM Authentication Protocol, as specified in [MS-NLMP], to allow the SMTP server to authenticate the SMTP client. The NTLM Authentication Protocol is a challenge/response authentication protocol that depends on the application layer protocols to transport NTLM packets from client to server and from server to client.

The NTLM Authentication: SMTP Extension defines how the SMTP is extended to perform authentication using the NTLM Authentication Protocol. The SMTP standard defines an extensibility mechanism for arbitrary authentication protocols to be plugged into the core protocol. This mechanism is the SMTP-AUTH mechanism.

The following diagram illustrates the sequence of transformations performed on an NTLM message to produce a message that can be sent over SMTP.

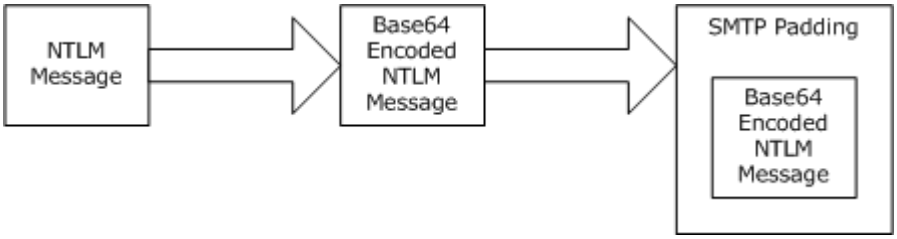

Figure 13: Relationship between NTLM message and SMTP: NTLM Authentication Protocol message

The NTLM Authentication: SMTP Extension is a pass-through protocol that does not specify the structure of NTLM information. Instead, the protocol relies on the software that implements the NTLM Authentication Protocol (as specified in NTLM) to process each NTLM message to be sent or received.

The NTLM Authentication: SMTP Extension defines a server and a client role.

When SMTP wants to perform an NTLM authentication, it needs to interact with the NTLM software appropriately. Below is an overview of this interaction.

2.2.13.1 Client Role

If acting as an SMTP client:

1. The NTLM software returns the first NTLM message to the client to be sent to the server.
2. The client applies the base64 encoding and SMTP padding transformations to produce an SMTP message, and then send this message to the server.
3. The client waits for a response from the server. When the response is received, it checks to see whether it indicates the end of authentication (success or failure) or that authentication is continuing.
4. If the authentication is continuing, the response message is stripped of the SMTP padding, base64 decoded, and passed into the NTLM software, on which the NTLM software may return another NTLM message that needs to be sent to the server. Steps 2 through 4 are repeated until authentication succeeds or fails.

2.2.13.2 Server Role

If acting as an SMTP server:

The server waits to receive the first SMTP authentication message from the client.

1. When an SMTP message is received from the client, the SMTP padding is removed, the message base64 decoded, and the resulting NTLM message is passed into the NTLM software.
2. The NTLM software returns a status indicating whether authentication completed successfully, failed, or more NTLM messages need to be exchanged to complete the authentication.

3. If the authentication is continuing, the NTLM software returns an NTLM message that needs to be sent to the server. This message is base64 encoded, and the SMTP padding is applied and sent to the client. Steps 2 through 4 are repeated until authentication succeeds or fails.

Implementers of the NTLM Authentication: SMTP Extension must possess a working knowledge of:

- SMTP, as specified in [RFC2821] and [RFC2554]
- The Multipurpose Internet Mail Extensions (MIME) base64 encoding method, as specified in [RFC1521].
- The NTLM Authentication Protocol, as specified in [MS-NLMP].

2.2.14 Windows Client Certificate Enrollment Protocol

The Windows Client Certificate Enrollment Protocol allows a client to interact with a Certificate Authority (CA) for enrollment, issuance, and management of X.509 certificates in a public key infrastructure (PKI).

A PKI consists of a system of digital certificate, certification authorities (CAs), and possibly registration authorities (RAs) that verify and authenticate the validity of each party involved in an electronic transaction.

The Windows Client Certificate Enrollment Protocol uses the Distributed Component Object Model (DCOM) as a transport. DCOM extends the Component Object Model (COM) over a network by providing facilities for creating and activating objects, and for managing object references, object lifetimes, and object interface queries.

2.3 Authentication Protocols Logical Dependencies and Protocol Stack Views

This section provides several conceptual authentication areas to describe the logical dependencies among the authentication protocols and protocol stack views as appropriate.

2.3.1 Integrated Authentication Protocols

When discussing authentication, it can be easier to visualize the protocols involved when they are separated into client and server views. The client view of integrated authentication is as follows:

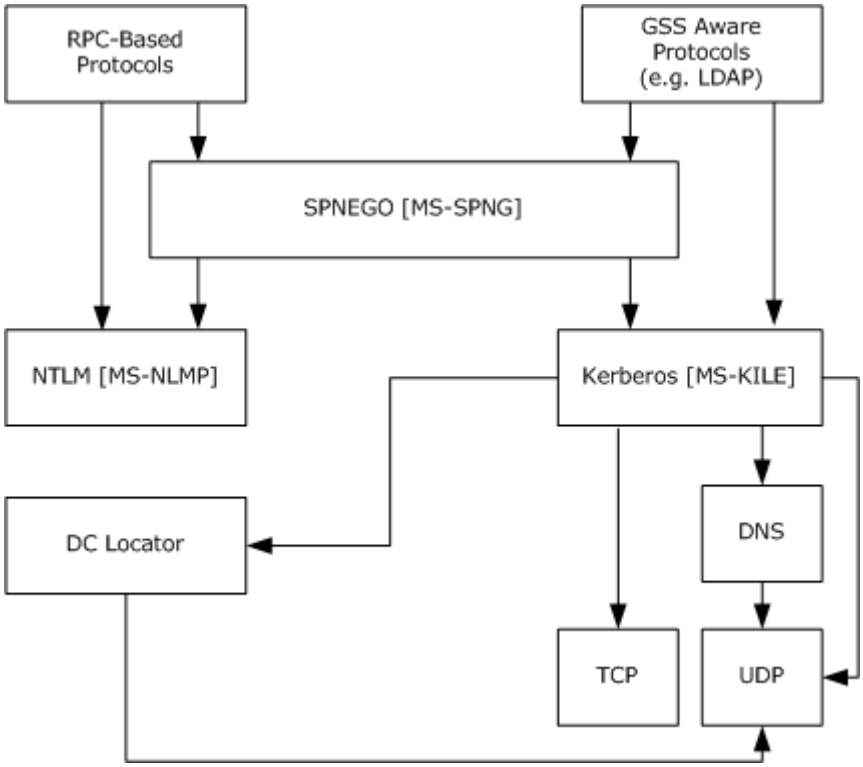

Figure 14: Client and server integrated authentication model

Here, RPC-based protocols use either SPNEGO or NTLM to authenticate to their servers. SPNEGO allows for the selection of Kerberos or NTLM during authentication. Similarly, GSS-aware protocols use either SPNEGO or Kerberos directly; for example, in the case of LDAP (as specified in [RFC2251]).

The server view of integrated authentication is the same as the client view with the addition of domain controller interaction. This is shown below:

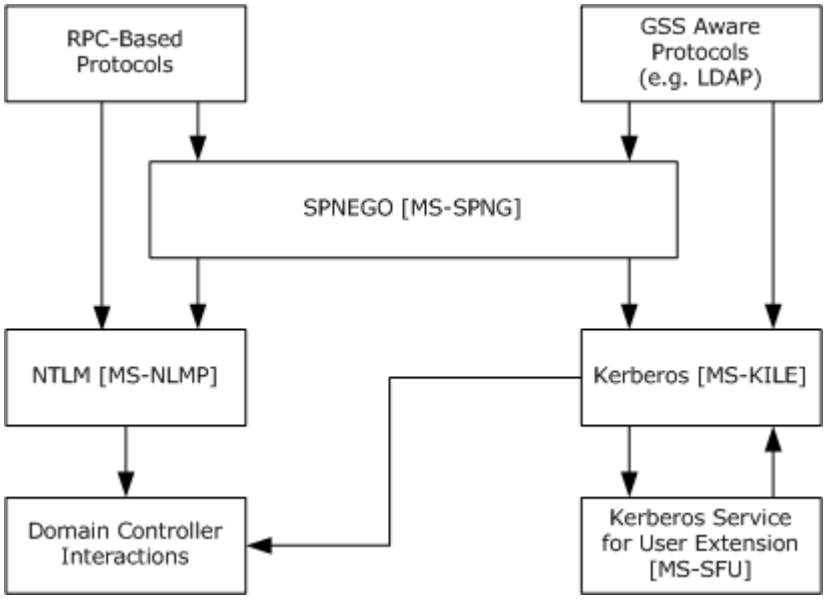

Figure 15: Client and server integrated authentication (with domain controller) model

Here, both NTLM and Kerberos can be seen interacting with the domain controller (more details of that interaction are found in the Certificate Services Server Protocols Overview section). Additionally, Kerberos may invoke the Kerberos Service for User extensions, as specified in [MS-SFU].

2.3.2 Server Side Include 1.4 Protocol

Passport Server Side Include (SSI) Version 1.4 (SSI1.4) protocol is an authentication service that is used by MSN and its partners to authenticate users accessing services that requires that the user be authenticated.

The SSI1.4 protocol is used by online services to authenticate users/clients using the Passport Authentication service. The protocol is also known as Tweener or SSI1.4.

The SSI1.4 protocol is an HTTP-based protocol [RFC2616] for authenticating a client to a partner server, with the assistance of an **authentication server**. It relies on HTTP cookies carried with standard HTTP messages to convey authentication information between an **authentication server** and a partner server, as specified in [MS-PASS].

2.3.2.1 Protocol Stack

There is a simple protocol stack in the SSI1.4 usage by Passport. For example, an application that needs to use a service is asked to authenticate itself (the user actually). The application uses an API to the SSI1.4 protocol. The SSI1.4 protocol uses HTTP or HTTPS as its transport. HTTP(S) in turn uses TCP at ports 80 or 443 over IP.

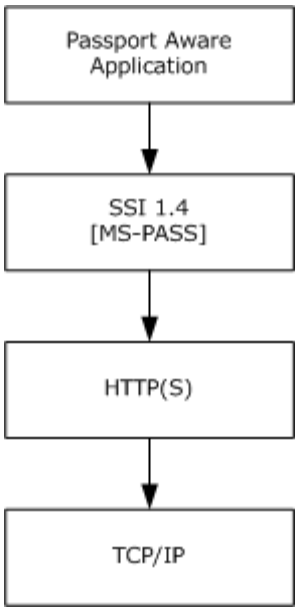

Figure 16: SSI 1.4 protocol stack

2.3.2.2 Logical Dependencies

As specified in [MS-PASS], the SSI1.4 specification, the protocol works between a client, the Partner Service (service being accessed) and the Authentication Service (Passport). These dependencies can be represented as follows.

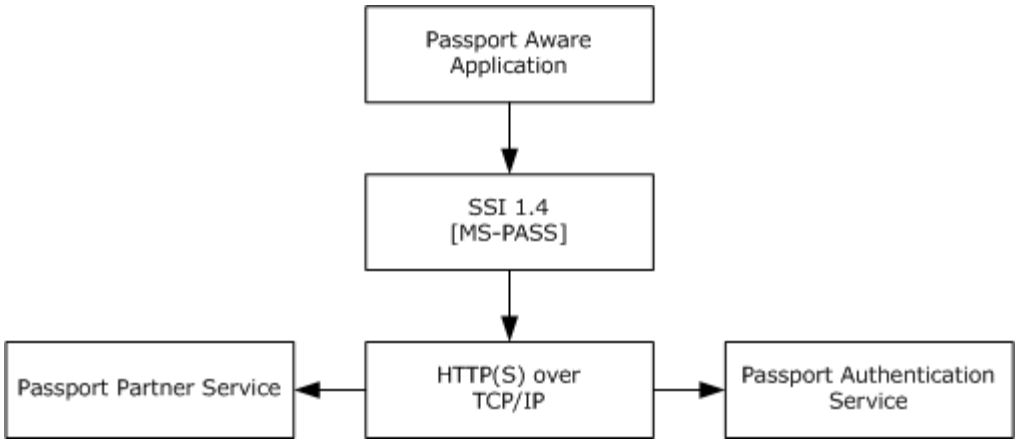

Figure 17: SSI 1.4 model dependencies

During the authentication process, the service, when asked for access to a protected resource, redirects the client to the Passport service; the Passport service prompts the client for his/her credentials; the credentials are submitted to Passport; the Passport service redirects the client to the partner service with successful or failed results of the authentication.

2.3.3 Internet Protocol Security Protocols Extensions

The Internet Protocol Security (IPsec) Protocols Extensions are composed of the following elements:

- A set of extensions to the Internet Key Exchange (IKE) protocol, as specified in [MS-IKEE]
- The Authenticated IP keying protocol [MS-AIPS], which peers may use instead of the IKE protocol when negotiating IP security parameters.

2.3.3.1 Protocol Stack

The IPsec Extensions retain the protocol stack as specified in [RFC2409] and [RFC3947] for the IKE and IKE NAT-Traversal protocols, respectively.

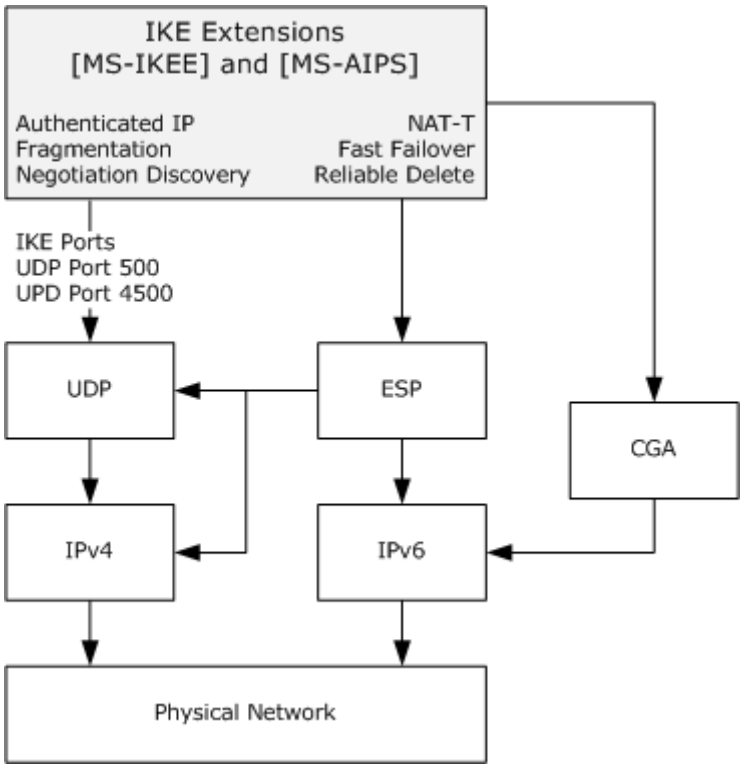

Figure 18: IPSec with IKE and IKE NAT-Traversal Protocols

As shown, all but one of the extensions consists of changes within the IKE protocol itself. This modifies the protocol stack view in one respect only: it supports authentication using a **cryptographically generated address (CGA)** as specified in [RFC3972]. This alteration does not introduce a new layer. Instead, it introduces an additional logical dependency between the IKE Extensions and the CGA functionality of the underlying IPv6 implementation.

2.3.3.2 Logical Dependencies

As noted above, there can be a logical dependency between the IKE Extensions and the IPv6 **CGA** functionality.

The IKE Extensions protocol is a General Security Service (GSS)–aware protocol. As a result, its use of **Authentication Services** results in the logical dependencies specified in section 2.2.4.

2.3.4 Domain Services Interaction Protocols

Domain Services Interaction encompasses both the ability of **domain members** to share the authentication and authorization services of the domain controller, as well as the ability to administer and manage that authorization. Like the previous section on basic authentication, this is best divided into conceptual areas, as shown below:

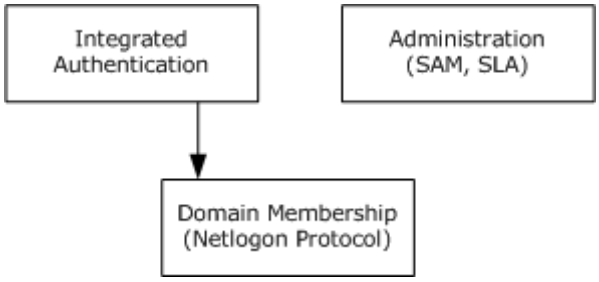

Figure 19: Conceptual areas of domain services interaction

2.3.5 Integrated Authentication

As shown in Authentication Methods (section 2.2.4), integrated authentication provides authentication services to application protocols. For server-side authentication, it is frequently useful to use a domain controller for the authentication and authorization. Authentication Protocols Domain Support [MS-APDS] specifies how various authentication protocols connect to the domain controller and make use of domain resources.

2.3.5.1 Protocol Stack

Various aspects of domain-based authentication are shown below:

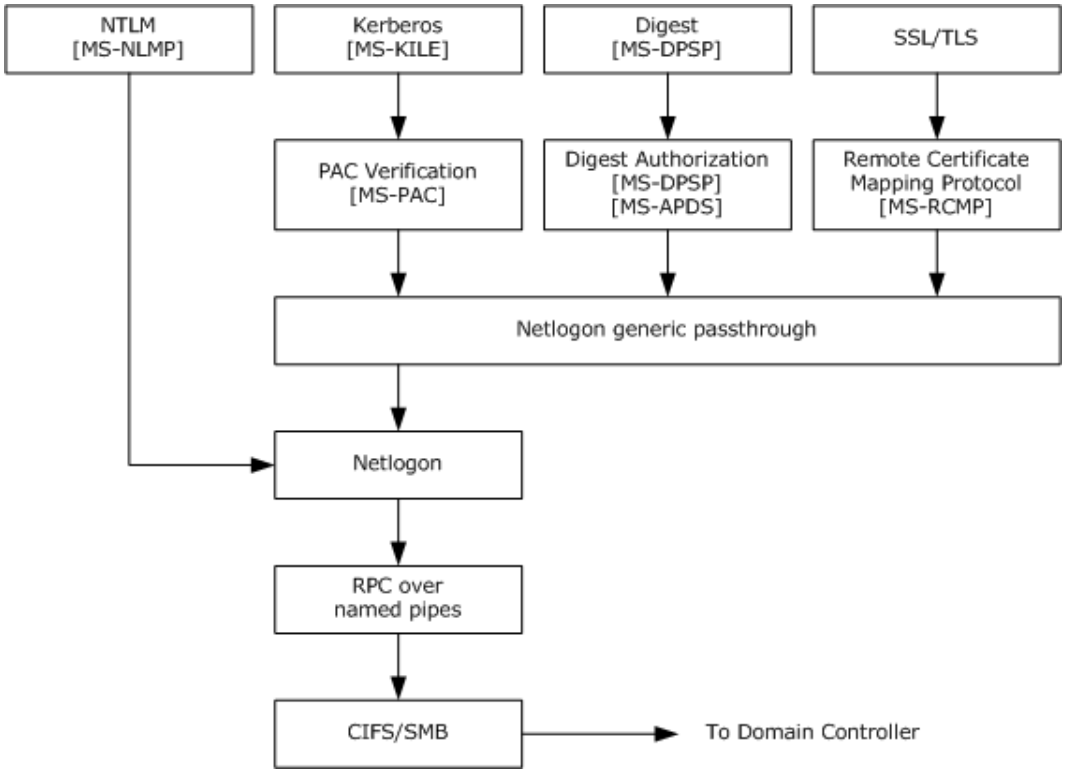

Figure 20: Domain-based authentication examples

In the diagram, the authentication protocols on the top line use the Netlogon protocol [MS-NRPC], either directly in the case of NTLM [MS-NLMP] or indirectly through the generic pass-through facility of **Netlogon**. The latter allows Certificate Mapping as described in [MS-RCMP]; Digest Authentication, as described in [MS-APDS]; and PAC verification, as defined in [MS-PAC] and [MS-APDS], all of which take place at the domain controller.

2.3.6 Credential Security Support Provider Protocol Overview

The Credential Security Support Provider (CredSSP) Protocol enables an application to securely delegate a user's **credentials** from a client to a target server. CredSSP first establishes an encrypted channel between the client and the target server using **Transport Layer Security (TLS)** (as specified in [RFC2246]). It uses **TLS** as an encrypted pipe; it does not rely on client/server authentication services available in **TLS**. CredSSP then uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) Protocol Extensions to negotiate a **GSS** mechanism that performs **mutual authentication** and **GSS** confidentiality services to securely bind to the **TLS** channel and encrypt the **credentials** for the target server. All **GSS** security tokens are sent over the encrypted **TLS** channel.

The CredSSP protocol enables an application to securely delegate a user's **credentials** from a client to a target server. For example, the Microsoft Terminal Server uses CredSSP Protocol to securely delegate the user's password or SmartCard PIN from the client to the server to remotely log in the user and establish a terminal services session.

Policy settings control whether a client delegates the user's **credentials** to assure that the user's **credentials** are not delegated to an unauthorized server such as a computer under the administrative control of an attacker. While **trust** may exist to facilitate authentication between the

client and server (for example, based on **Kerberos** (as specified in [RFC4120]) or **NTLM** (as specified in [MS-NLMP]) this does not mean that the target server is trusted with the user's **credentials**.

2.3.6.1 Logical Dependencies

The CredSSP Protocol is a composite protocol that relies on other, standards-based, **security protocols**. It first uses the **TLS**, as specified in [MS-TLSP], to establish an encrypted channel between the CredSSP client and the CredSSP server (the client is anonymous at this point; the client and the server may have no common trusted **CA** root).

All subsequent messages are sent over this channel. The CredSSP Protocol then uses - SPNEGO (as specified in [MS-SPNG]) to authenticate the user and server within the encrypted **TLS** session.

SPNEGO provides a framework for two parties engaged in authentication to select from a set of possible authentication mechanisms, in a fashion that preserves the opaque nature of the **security protocols** to the **application protocol** that uses **SPNEGO**; in this case, CredSSP Protocol.

CredSSP Protocol uses **SPNEGO** to mutually authenticate the CredSSP client and CredSSP server. It then uses the encryption key established under **SPNEGO** to "securely bind" to the **TLS** session (the process by which the server's public key used in the **TLS** handshake is authenticated) The client encrypts the server's public key using the encryption key established under **SPNEGO** and sends it to the server; the server verifies that it indeed is the same public key used in the **TLS** handshake and sends an acknowledgment back to the client also encrypted under the **SPNEGO** encryption key. This step is described in detail in section 3.1. Lastly, the client sends the user's **credentials** encrypted under the **SPNEGO** encryption key to the server. All subsequent data that may be sent between the client and server application using the CredSSP Protocol is encrypted under **TLS** as specified in [MS-CSSP]. The only new, on-the-wire formats introduced by the CredSSP Protocol are the encapsulation of the **SPNEGO** tokens sent over the **TLS** channel, the binding between the **TLS** and **SPNEGO** protocols, and the format of the user credentials.

2.3.6.2 CredSSP Relationship to Other Protocols

The CredSSP Protocol uses the TLS protocol, as specified in [MS-CSSP], to encrypt all traffic between the CredSSP client and CredSSP server. The **TLS** protocol requires a reliable transport such as the TCP protocol, as specified in [RFC793], for all messages exchanged between the client and the server.

The CredSSP Protocol uses **SPNEGO**, as specified in [MS-SPNG], for **mutual authentication** between the CredSSP client and CredSSP server **. SPNEGO** requires at least one other **Generic Security Services (GSS)** (as specified in [RFC2078]) compatible authentication protocol (in addition to **SPNEGO** itself) to be present for it to work. **SPNEGO** has no dependence on any specific **GSS**-compatible protocols; although in practice, **Kerberos** (as specified in [MS-KILE]) is generally used.

The Remote Desktop Protocol (RDP) uses the CredSSP Protocol to delegate credentials from the RDP client to the RDP server, and encrypt all data that follows using the **TLS** channel that is established as part of the CredSSP Protocol.

2.3.6.3 Directory Replication Service Remote Protocol Relationship to Other Protocols

The Directory Replication Service Remote Protocol (drsuapi) is based on RPC and therefore has the prerequisites identified in [MS-RPCE] as being common to all RPC interfaces.

2.3.6.4 Directory Services Setup Remote Protocol Relationship to Other Protocols

The Directory Services Setup Remote Protocol (drssetup) is dependent upon Microsoft Remote Procedure Call (RPC) Extensions, as specified in [MS-RPCE]), which is used to communicate between computers on a network.

2.3.6.5 Net Logon Remote Protocol Relationship to Other Protocols

The Net Logon Remote Protocol depends on RPC and the mailslot datagram delivery service, as specified in [MS-SMB], which are its transports.

Net Logon replication uses the mailslot datagram delivery mechanism; therefore, it depends on this mailslot delivery mechanism being operational before Net Logon begins operation. For mailslot operational requirements, see [MS-MAIL]. For more information about the mailslot delivery mechanism, see [MS-SMB].

2.3.6.6 Windows Client Certificate Enrollment Protocol Relationship to Other Protocols

The Windows Client Certificate Enrollment Protocol depends on the Distributed Component Object Model (DCOM) Remote Protocol, as specified in [MS-DCOM]. Data structures that are specified in the certificate templates structure specification (as specified in [MS-CRTD]), may be retrieved over the Lightweight Directory Access Protocol (LDAP), as specified in [RFC2559], and used throughout the enrollment process.

No other Microsoft Windows protocol directly depends on the Windows Client Certificate Enrollment Protocol. Indirectly, as an example, other protocols that rely on certificates for authentication (such as the Transport Layer Security Protocol, [RFC2246]) may use this protocol for certificate enrollment and issuance.

2.3.6.7 Backup Key Remote Protocol Relationship to Other Protocols

This protocol is implemented on top of RPC, as specified in [MS-RPCE].

The Backup Key Remote Protocol uses remote procedure call (RPC) as specified in [MS-RPCE], with encryption for confidentiality, and SPNEGO (as specified in [MS-SPNG]) for mutual authentication between the client and the key wrapping server.

No other protocols rely on the Backup Key Remote Protocol.

2.3.6.8 NTLM Authentication: Network News Transfer Protocol Extension Relationship to Other Protocols

The NTLM Authentication: Network News Transfer Protocol (NNTP) Extension uses the NNTP AUTHINFO GENERIC extension mechanism as specified in [RFC2980] and is an embedded protocol. Unlike standalone application protocols, such as Telnet or the HTTP, this protocol extension's packets are embedded in NNTP commands and server responses.

This extension specifies only the sequence in which an NNTP server and NNTP client must exchange NTLM messages in order to successfully authenticate the client to the server. It does not specify how the client obtains NTLM messages from the local NTLM software, or how the NNTP server should process NTLM messages.

The NNTP client and NNTP server implementations depend on the availability of an implementation of the NTLM Authentication Protocol (as specified in [MS-NLMP]) in order to obtain and process

NTLM messages; and on the availability of the base64 encoding and decoding mechanisms (as specified in [RFC1521]) to encode and decode the NTLM messages that are embedded in NNTP packets.

Because the NTLM Authentication: NNTP Extension depends on NTLM to authenticate the client to the server, both server and client must have access to an implementation of the NTLM Authentication Protocol (as specified in [MS-NLMP]) that is capable of supporting connection-oriented NTLM.

2.3.6.9 NTLM Authentication Protocol; NTLM Authentication: Post Office Protocol - Version 3 (POP3) Extension Relationship to Other Protocols

The NTLM POP3 Extension uses the POP3 AUTH extension mechanism, as specified in [RFC1734], and is an embedded protocol. Unlike standalone application protocols, such as Telnet or HTTP, [MS-POP3] packets are embedded in POP3 commands and server responses.

The POP3 protocol specifies only the sequence in which a POP3 server and POP3 client must exchange NTLM messages to successfully authenticate the client to the server. It does not specify how the client obtains NTLM messages from the local NTLM software, or how the POP3 server should process NTLM messages. The POP3 client and POP3 server implementations depend on the availability of an implementation of the NTLM Authentication Protocol (as specified in [MS-NLMP]) to obtain and process NTLM messages and on the availability of the base64 encoding and decoding mechanisms (as specified in [RFC1521]) to encode and decode the NTLM messages embedded in POP3 packets.

Because the POP3 depends on NTLM to authenticate the client to the server, both server and client must have access to an implementation of the NTLM Authentication Protocol (as specified in [MS-NLMP]) that is capable of supporting connection-oriented NTLM.

2.3.6.10 Remote Certificate Mapping Protocol Relationship to Other Protocols

Any protocol that authenticates clients based on public key certificates can make use of the Remote Certificate Mapping Protocol to obtain authorization information about the client. The Net Logon Remote Protocol serves as the transport for Remote Certificate Mapping Protocol messages.

2.3.6.11 NTLM Authentication Protocol; NTLM Authentication: SMTP Extension Relationship to Other Protocols

The NTLM Authentication: SMTP Extension uses the SMTP-AUTH extension mechanism, as specified in [RFC2554], and is an embedded protocol. Unlike standalone application protocols, such as Telnet or HTTP, NTLM Authentication: SMTP Extension packets are embedded in SMTP commands and server responses.

The SMTP specifies only the sequence in which an SMTP server and SMTP client must exchange NTLM messages to successfully authenticate the client to the server. It does not specify how the client obtains NTLM messages from the local NTLM software, or how the SMTP server should process NTLM messages. The SMTP client and SMTP server implementations depend on the availability of an implementation of the NTLM Authentication Protocol (as specified in [MS-NLMP]) to obtain and process NTLM messages and on the availability of the base64 encoding and decoding mechanisms (as specified in [RFC1521]) to encode and decode the NTLM messages embedded in SMTP packets.

2.3.6.12 Local Security Authority (Domain Policy) Remote Protocol Relationship to Other Protocols

The Local Security Authority (Domain Policy) Remote Protocol is dependent on RPC, which is used for communication between domain members and DCs. This protocol and the Security Account Manager (SAM) Remote Protocol (Client-to-Server) [MS-SAMR] access the same abstract data Server Role Information.

This protocol and the Workstation Service Remote Protocol [MS-WKST] access the same Domain Name from the abstract data Account Domain Information.

This protocol depends on Server Message Block (SMB) protocols for sending messages on the wire. It also has the prerequisites specified in [MS-RPCE] as being common to protocols that depend on RPC.

2.3.6.13 Kerberos Network Authentication Service (V5) Service for User (S4U) Extension Relationship to Other Protocols

The Service for User (S4U) extensions are based on Kerberos, as specified in [RFC4120] and [MS-KILE]. The Service-for-User-to-Self (S4U2self) extension is used to obtain a privilege attribute certificate (PAC), as specified in [MS-PAC], to determine the authorization capabilities of the user. In addition, the PAC is used in Service-for-User-to-Proxy (S4U2proxy) to validate that S4U2proxy service tickets have not been misused.

2.4 Implementation Scenarios

This section provides two scenarios that illustrate common functionality that can be implemented with the protocols included in Authentication Services. The protocols involved in these scenarios include Kerberos, as specified in [RFC4120], and the Local Security Authority (Domain Policy), together with other general networking protocols used in the course of standard TCP communication between the client and server. The functionality illustrated in the scenarios is as follows:

- Windows Vista operating system client logs on to domain hosted on Windows Server 2003 operating system
- Windows Vista client logs off of domain hosted on Windows Server 2003

It should be noted that the following scenarios depict processes that occur over TCP and thus assumes various TCP acknowledgment frames throughout the conversation that are not necessarily depicted in the trace details.

2.4.1 Scenario 1 - Vista Client Logs On To Windows Domain

2.4.1.1 Scenario Overview

In the following scenario, a Vista Enterprise Edition client is a member workstation of a Windows Server 2003 operating system R2SP2 ADS domain.

A manual logon procedure is executed by a domain user account to initiate the domain logon session.

2.4.1.2 Scenario Configuration

This scenario was implemented with a Windows Server 2003 operating system domain controller and Windows Vista operating system client machine configured as follows:

Server configuration

- Machine: WS200301.contoso.com (WS200301)
- Operating system: Windows Server 2003 R2 with latest updates
- IP Address: 10.0.10.1
- Subnet Mask: 255.255.255.0
- AD DC – contoso.com
- NetMon 3.1

Client configuration

- Machine: ClientVista.contoso.com (ClientVista)
- Operating system: Windows Vista Ultimate with latest updates
- IP Address: 10.0.10.10
- Subnet Mask: 255.255.255.0
- DNS – 10.0.10.1
- Domain member – contoso.com

Network Topology

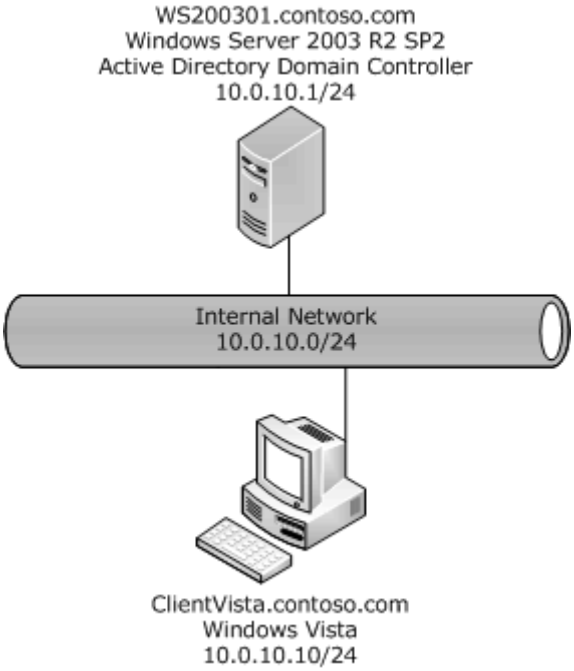

Figure 21: Network Topology

2.4.1.3 Setting Up the Trace

1. Log onto all machines as CONTOSO\Administrator
2. Turn off firewall on all machines
3. On WS200301, start NetMon 3.1
4. BEGIN TRACE
   - On ClientVista, press CTRL+ALT+DEL
   - Log on as CONTOSO\Administrator
5. STOP TRACE

2.4.1.4 Netmon Trace Digest

- From the Windows Vista operating system client machine, a standard logon process was initiated per the steps listed in section 2.4.1.3. Assuming the initial TCP/IP session remains established between the client and the domain controller (DC), all Address Resolution Protocol (ARP), Netbios (NbtNS) and domain name service (DNS) resolution, TCP flags and server message block (SMB) negotiated dialect remain established. If not, the trace will show these frames establishing an SMB session between the client and domain controller (initiated by the client) as shown in the following example.

```
ClientVista   WS200301  ARP  ARP: Request, 192.168.200.3
asks for 192.168.200.2
WS200301   ClientVista  ARP  ARP: Response, 192.168.200.2 at
00-C0-9F-23-E7-EE
```

- Kerberos clients require a way to locate the Key Distribution Center (KDC); this is done through the use of DNS SRV records and LDAP lookups. The component in Windows that is responsible for dynamic location through SRV records is called the Locator; for more information, see [MS-ADTS] section "Locator". For more information about Kerberos V5, see [RFC4120].

```
ClientVista    WS200301  LDAP  LDAP: (CLDAP)Search Request,
MessageID: 6, BaseObject: NULL, SearchScope: base Object,SearchAlias:
 neverDerefAliases
WS200301    Vista  LDAP  LDAP: (CLDAP)Search Result Entry, MessageID: 6,
Status: Success
ClientVista   WS200301  KerberosV5  KerberosV5: AS Request
Cname: administrator Realm: PROTOCOL Sname: krbtgt/PROTOCOL
```

- The following error message from the server to the client forces the client to prove or pre-authenticate. Pre-authentication takes place during the AS exchange, when the client first authenticates to the KDC. A client pre-authenticates if it supplies additional information that proves it knows the key it shares with the KDC before the TGT is issued. Pre-authentication is done by supplying one or more pre-authentication messages in the PA-data field of the AS-REQ message.

```
WS200301    ClientVista  KerberosV5  KerberosV5: KRB_ERROR  -
KDC_ERR_PREAUTH_REQUIRED (25)
```

- This is the request for the TGT. A TGT is generated using the username and password and is sent to the KDC.

```
ClientVista    WS200301  KerberosV5  KerberosV5: AS Request Cname:
administrator Realm: PROTOCOL Sname: krbtgt/PROTOCOL
```

- In response to receiving the AS-REQ for a TGT, the KDC authenticates the user by checking that the credentials used in the AS-REQ are the same as that of the user's, as specified in [RFC4120]. The KDC builds an AS-REP from the TGT and other requisite data, and sends it back to the client.

```
WS200301    ClientVista  KerberosV5  KerberosV5: AS Response
Ticket[Realm: PROTOCOL.LOCAL, Sname: krbtgt/PROTOCOL.LOCAL]
```

- Once the service ticket to the application server is obtained, the client authenticates itself to the server by sending an AP-REQ wrapped in Generic Security Services (GSS) formatting, as specified in section 3.3 and [RFC1964].

```
ClientVista    WS200301  KerberosV5  KerberosV5: TGS Request Realm:
PROTOCOL.LOCAL Sname: host/vistacnt.protocol.local
WS200301    ClientVista  KerberosV5  KerberosV5: TGS Response Cname:
Administrator
```

2.4.2 Scenario 2 - Vista Client Logs Off Windows Domain

2.4.2.1 Scenario Overview

In the following scenario, a Vista Enterprise Edition client is a member workstation of a Windows Server 2003 R2 operating system SP2 ADS domain. A manual logoff procedure was initiated by a domain user account to terminate the domain logon session.

2.4.2.2 Scenario Configuration

This scenario was implemented with a Windows Server 2003 operating system domain controller, and Windows Vista operating system client machine configured as follows:

Server configuration

- Machine: WS200301.contoso.com (WS200301)
- Operating System: Windows Server 2003 R2 operating system with latest updates
- IP Address: 10.0.10.1
- Subnet Mask: 255.255.255.0
- AD domain controller – contoso.com
- NetMon 3.1

Client configuration

- Machine: ClientVista.contoso.com (ClientVista)
- Operating system: Windows Vista Ultimate with latest updates

- IP Address: 10.0.10.10
- Subnet Mask: 255.255.255.0
- DNS – 10.0.10.1
- Domain member – contoso.com

Network Topology

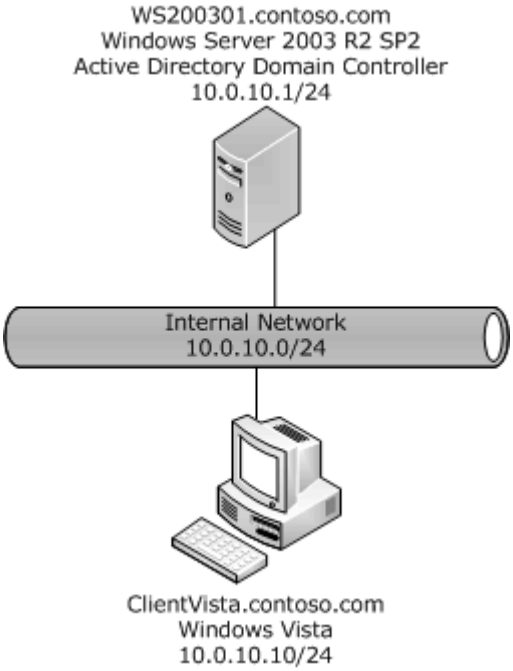

Figure 22: Network Topology

2.4.2.3 Setting Up the Trace

1. Log onto all machines as CONTOSO\Administrator
2. Turn off firewall on all machines
3. On WS200301, start NetMon 3.1
4. BEGIN TRACE
   - On ClientVista, click Start
   - Log off CONTOSO\Administrator
5. END TRACE

2.4.2.4 Netmon Trace Digest

- From the Windows Vista operating system client machine, a standard logon process was initiated per the steps listed in section 2.4.2.3. Assuming the initial TCP/IP session remains established between the client and the domain controller, all ARP, NbtNS, and DNS resolution, TCP flags and

SMB negotiated dialect remain established. If not, the trace will show these frames establishing a SMB session between the client and domain controller (initiated by the client) as shown in the following example.

```
ClientVista  WS200301  ARP  ARP: Request, 10.0.0.20 asks for
 10.0.0.1

ClientVista  10.255.255.255  NbtNs  NbtNs: Query Request for
WS200301.DOMAIN.DOM

WS200301  ClientVista  ARP  ARP: Response, 10.0.0.1 at
00-03-FF-A1-5D-52

ClientVista  WS200301  DNS  DNS: QueryId = 0x1838, QUERY (Standard
query), Query  for  dc.domain.dom of type Host Addr on class Internet

WS200301  ClientVista  DNS  DNS: QueryId = 0x1838, QUERY (Standard
query), Response - Success

ClientVista  WS200301  TCP  TCP: Flags=.S......, SrcPort=49222,
DstPort=Microsoft-DS(445), Len=0, Seq=3101792531, Ack=0,
Win=8192 (scale factor not found)

WS200301  ClientVista  TCP  TCP: Flags=.S..A..., SrcPort=Microsoft-DS
(445), DstPort=49222, Len=0, Seq=4109447982, Ack=3101792532,
Win=16384 (scale factor not found)

ClientVista  WS200301  TCP  TCP: Flags=....A..., SrcPort=49222,
DstPort=Microsoft-DS(445), Len=0, Seq=3101792532, Ack=4109447983,
 Win=256 (scale factor not found)

ClientVista  WS200301  SMB  SMB: C; Negotiate,
Dialect = PC NETWORK PROGRAM 1.0, LANMAN1.0,
Windows for Workgroups 3.1a, LM1.2X002, LANMAN2.1, NT LM 0.12,
 SMB 2.001

WS200301  ClientVista  SMB  SMB: R; Negotiate, Dialect is  (#5)

ClientVista  WS200301  TCP  TCP: Flags=....A..., SrcPort=49222,
DstPort=Microsoft-DS(445), Len=0, Seq=3101792680, Ack=4109448158,
 Win=255 (scale factor not found)

ClientVista  10.255.255.255  NbtNs  NbtNs: Query Request for
DC.DOMAIN.DOM
```

- The first evident logoff traffic is an SMB Session Setup request by the client to conduct authentication business over a SMB named pipe.

```
ClientVista  WS200301  SMB  SMB: C; Session Setup Andx

WS200301  ClientVista  SMB  SMB: R; Session Setup Andx
```

- The client then attempts to establish an SMB tree connection to the IPC$ share on the domain controller, and if successful it then follows with an attempt to open the Security Account Manager resource as called in the following example by the filename \samr.

```
ClientVista  WS200301  SMB  SMB: C; Tree Connect Andx,
Path = \\DC.DOMAIN.DOM\IPC$, Service = ?????

WS200301  ClientVista  SMB  SMB: R; Tree Connect Andx, Service = IPC

ClientVista  WS200301  SMB  SMB: C; Nt Create Andx, FileName = \samr

WS200301  ClientVista  SMB  SMB: R; Nt Create Andx, FID = 0x4002
```

- Next, the client utilizes a TRANS2_QUERY_FILE_INFORMATION SMB extension to request attribute information for the newly opened \samr file and the domain controller responds indicating the file was successfully opened and read and produces the attributes requested.

```
ClientVista  WS200301  SMB  SMB: C; Transact2, Query File Info,
Query File Standard Info, FID = 0x4002

WS200301  ClientVista  SMB  SMB: R; Transact2, Open2, FID = 0x0000
```

- The client now initiates a series of MSRPC requests to bind and write to the Security Account Manager database. The sequence updates various account attributes about the user account in the Security Account Manager file (such as last logoff, and so on).

  First, the client initiates a bind into the Security Account Manager file for the purpose of writing (note the use of the SMB_COM_WRITE extension in the Smb line of the frame breakout).

```
ClientVista  WS200301  MSRPC  MSRPC: c/o Bind:
UUID{12345778-1234-ABCD-EF00-0123456789AC} Security Account Manager
  Call=0x1  Assoc Grp=0x0  Xmit=0x10B8  Recv=0x10B8
```

- The server issues an SMB_COM_WRITE extension indicating STATUS_SUCCESS to accept the client for the purpose of writing. The client then issues a SMB_COM_READ extension to validate its written data. The writing test now complete, the DC acknowledges a successful bind via MSRPC_C/O_BIND_ACK.

```
WS200301  ClientVista  SMB  SMB: R; Write Andx, FID = 0x0000,
116 bytes

ClientVista  WS200301  SMB  SMB: C; Read Andx, FID = 0x4002,
1024 bytes at Offset 0

WS200301  ClientVista  MSRPC  MSRPC: c/o Bind Ack:  Call=0x1
Assoc Grp=0x602A  Xmit=0x10B8  Recv=0x10B8
```

- A conversation between the client and DC commences over MSRPC using C/O_REQUEST and corresponding C/O_RESPONSE extensions as the client initiates various transactions into the Security Account Manager file to modify the user account attributes (such as last logoff time).

```
ClientVista  WS200301  MSRPC  MSRPC: c/o Request: unknown
Call=0x1  Opnum=0x40  Context=0x0  Hint=0x44

WS200301  ClientVista  MSRPC  MSRPC: c/o Response: unknown   Call=0x1
  Context=0x0  Hint=0x28  Cancels=0x0
```

```
ClientVista  WS200301  MSRPC  MSRPC: c/o Request: unknown   Call=0x2
  Opnum=0x6  Context=0x0  Hint=0x1C

WS200301  ClientVista  MSRPC  MSRPC: c/o Response: unknown   Call=0x2
  Context=0x0  Hint=0x68  Cancels=0x0

ClientVista  WS200301  MSRPC  MSRPC: c/o Request: unknown   Call=0x3
  Opnum=0x5  Context=0x0  Hint=0x34

WS200301  ClientVista  MSRPC  MSRPC: c/o Response: unknown   Call=0x3
  Context=0x0  Hint=0x24  Cancels=0x0

ClientVista  WS200301  MSRPC  MSRPC: c/o Request: unknown   Call=0x4
  Opnum=0x7  Context=0x0  Hint=0x34

WS200301  ClientVista  MSRPC  MSRPC: c/o Response: unknown   Call=0x4
  Context=0x0  Hint=0x18  Cancels=0x0

ClientVista  WS200301  MSRPC  MSRPC: c/o Request: unknown   Call=0x5
  Opnum=0x7  Context=0x0  Hint=0x28

WS200301  ClientVista  MSRPC  MSRPC: c/o Response: unknown   Call=0x5
  Context=0x0  Hint=0x18  Cancels=0x0

ClientVista  WS200301  MSRPC  MSRPC: c/o Request: unknown   Call=0x6
  Opnum=0x11  Context=0x0  Hint=0x42

WS200301  ClientVista  MSRPC  MSRPC: c/o Response: unknown   Call=0x6
  Context=0x0  Hint=0x24  Cancels=0x0

ClientVista  WS200301  MSRPC  MSRPC: c/o Request: unknown   Call=0x7
  Opnum=0x22  Context=0x0  Hint=0x1C

WS200301  ClientVista  MSRPC  MSRPC: c/o Response: unknown   Call=0x7
  Context=0x0  Hint=0x18  Cancels=0x0

ClientVista  WS200301  MSRPC  MSRPC: c/o Request: unknown   Call=0x8
  Opnum=0x24  Context=0x0  Hint=0x16

WS200301  ClientVista  MSRPC  MSRPC: c/o Response: unknown   Call=0x8
  Context=0x0  Hint=0x188  Cancels=0x0

ClientVista  WS200301  MSRPC  MSRPC: c/o Request: unknown   Call=0x9
  Opnum=0x3  Context=0x0  Hint=0x18

WS200301  ClientVista  MSRPC  MSRPC: c/o Response: unknown   Call=0x9
  Context=0x0  Hint=0x98  Cancels=0x0

ClientVista  WS200301  MSRPC  MSRPC: c/o Request: unknown   Call=0xA
  Opnum=0x27  Context=0x0  Hint=0x14

WS200301  ClientVista  MSRPC  MSRPC: c/o Response: unknown   Call=0xA
  Context=0x0  Hint=0x1C  Cancels=0x0

ClientVista  WS200301  MSRPC  MSRPC: c/o Request: unknown   Call=0xB
  Opnum=0x10  Context=0x0  Hint=0x68

WS200301  ClientVista  MSRPC  MSRPC: c/o Response: unknown   Call=0xB
  Context=0x0  Hint=0x14  Cancels=0x0
```

```
ClientVista  WS200301  MSRPC  MSRPC: c/o Request: unknown   Call=0xC
  Opnum=0x1  Context=0x0  Hint=0x14

WS200301  ClientVista  MSRPC  MSRPC: c/o Response: unknown   Call=0xC
  Context=0x0  Hint=0x18  Cancels=0x0

ClientVista  WS200301  MSRPC  MSRPC: c/o Request: unknown   Call=0xD
  Opnum=0x1  Context=0x0  Hint=0x14

WS200301  ClientVista  MSRPC  MSRPC: c/o Response: unknown   Call=0xD
  Context=0x0  Hint=0x18  Cancels=0x0

ClientVista  WS200301  MSRPC  MSRPC: c/o Request: unknown   Call=0xE
  Opnum=0x1  Context=0x0  Hint=0x14

WS200301  ClientVista  MSRPC  MSRPC: c/o Response: unknown   Call=0xE
  Context=0x0  Hint=0x18  Cancels=0x0

ClientVista  WS200301  MSRPC  MSRPC: c/o Request: unknown   Call=0xF
  Opnum=0x1  Context=0x0  Hint=0x14

WS200301  ClientVista  MSRPC  MSRPC: c/o Response: unknown   Call=0xF
  Context=0x0  Hint=0x18  Cancels=0x0
```

- Upon completion of the SMB Transactions using Named Pipes, the client disconnects from the IPC$ tree and logs the user account off.

```
ClientVista  WS200301  SMB  SMB: C; Close, FID = 0x4002

WS200301  ClientVista  SMB  SMB: R; Close, FID = 0x0000

ClientVista  WS200301  SMB  SMB: C; Tree Disconnect

WS200301  ClientVista  SMB  SMB: R; Tree Disconnect

ClientVista  WS200301  SMB  SMB: C; Logoff Andx

WS200301  ClientVista  SMB  SMB: R; Logoff Andx
```

3 Certificate Services Protocols

This section provides an overview of the Certificate Services protocols. It provides a high-level conceptual overview of the core protocol, classifying it broadly according to how the credentials are passed over a Windows network.

3.1 Certificate Services Overview

These protocols can be used to provide X.509 certificate requests, renewal, key archival, certificate issuance, and lifecycle-management services.

3.1.1 Protocols List

The Certificate Services includes a single core protocol that provides the certificate services functionality and a set of protocols that are required to implement it.

The following table shows the core protocol included in Certificate Services. It does not include data structures, file structures, or algorithms that this core protocol depends on. The details for these technical dependencies are documented in [MS-WCCE], the technical specification for the core protocol.

Table: Certificate Services Core Protocol

Protocol name	Protocol description	Document short name
Windows Client Certificate Enrollment Protocol	Consists of a set of DCOM interfaces that allow clients to request various services from a certificate authority (CA). These services enable X.509 digital certificate enrollment, issuance, revocation, and property retrieval.	[MS-WCCE]

The following sections provide some basic certificate services protocol concepts.

3.2 Certificate Services Functionality

Section of this document provides examples that illustrate the interaction of some of the protocols in two sample configurations.

Familiarity with public key infrastructure (PKI) concepts such as asymmetric and symmetric cryptography, digital certificate concepts, and cryptographic key exchange are required for a complete understanding of this specification. In addition, a comprehensive understanding of the x.509 standard, as specified in [X509], is required for a complete understanding of the protocol and its usage. Certificate concepts are as specified in [X509].

A certificate authority (CA) is an entity (individual, department, company, or organization) that issues digital certificates to verify the identity of users, applications, or organizations.

Before issuing a digital certificate to someone, the certificate authority (CA) must verify the user's identity according to a strictly established policy. This verification can involve face-to-face communication, examination of a driver's license with a photograph, or another method of establishing a user's identity. When the user's identity has been verified, the certificate is issued to the user. This certificate can then be presented by the user to digitally identify himself or herself during network transactions.

CAs can be trusted third parties such as the private companies VeriSign, CyberTrust, and Nortel Networks; or any organization can establish its own CAs using the certificate services functionality in Windows Server operating system. CAs can be stand-alone authorities with their own self-signed certificates (that is, they validate their own identity as a root CA), or they can be part of a hierarchy in which each CA is certified by the trusted CA above it (up to a root CA, which must always be self-certified).

To ensure that digital certificates work as an identification scheme, both client and server programs must trust the CA. When a client program presents a certificate to a server program, the server program must be able to validate that the certificate was issued by a valid and trusted CA. Certificate authorities also maintain a certificate revocation list (CRL) for revoked certificates. Certificates issued by CAs expire after a specified period of time.

CAs are necessary for the functioning of public key infrastructure (PKI), which is essential to the widespread acceptance and success of any public key cryptography system. Windows Server 2008 operating system,Windows Server 2003 operating system and Windows 2000 Server operating system can use standard X.509 digital certificates to authenticate connections across unsecured networks such as the Internet and to provide single sign-on by using smart card authentication systems.

3.3 Certificate Services Logical Dependencies and Protocol Stack Views

This section describes the logical dependencies for the certification protocol and shows the protocol stack view.

3.3.1 Windows Client Certificate Enrollment Protocol Logical Relationships

The Windows Client Certificate Enrollment Protocol, as specified in [MS-WCCE], uses the DCOM as a transport. Thus this protocol depends on the DCOM to accomplish the task of allowing clients to manage its X.509 certificates through the entire life cycle.

This protocol is used by clients for a variety of tasks, including querying a certificate authority (CA) for its capabilities, enrolling for certificates, requesting certificates on behalf of others, renewing certificates, backing up keys, escrowing, and archiving its keys.

A number of protocols that use a public key infrastructure (PKI) depend on these processes being completed or accomplished in some manner, some protocols depend directly on this protocol or use it directly. That is, any of a variety of protocols can trigger activity that results in the client having to use this protocol but none explicitly do so. In fact, local use of encryption that is based on asymmetric keys, such as the Encrypting File System (EFS), can also require WSCCS-based enrollment and procurement of certificates.

Examples of protocols that use X.509 certificates include Transport Layer Security (TLS)/Secure Sockets Layer (SSL), Internet Protocol security (IPsec)/Internet Key Exchange (IKE), and EFS.

3.3.2 Windows Client Certificate Enrollment Protocol Stack View

The following diagram shows the transport stack used by the Windows Client Certificate Enrollment Protocol.

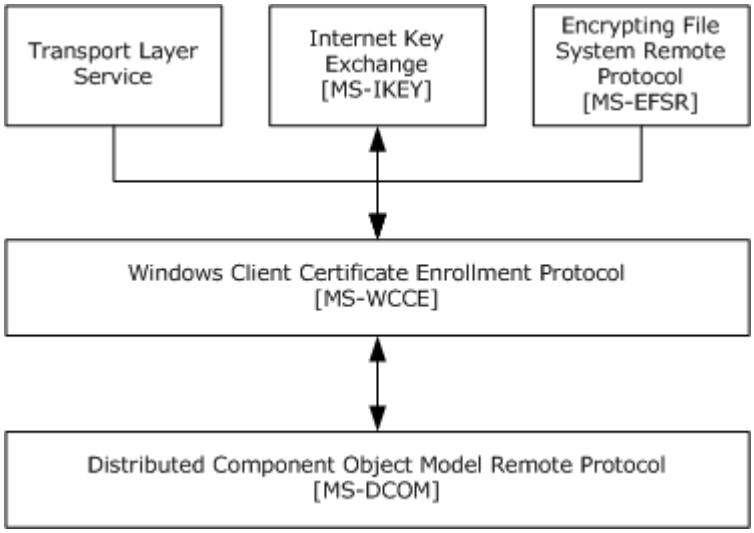

Figure 23: Preferred protocols for X.509 certificates, stack view

3.3.3 PKI Services Logical Dependencies and Stack View

The protocols, taken together, form a set of services that meet the need of applications that use PKI services. Their logical interdependencies are represented in the following figure. Note that the protocols across the top of the figure are simply applications that use PKI or X.509 certificates.

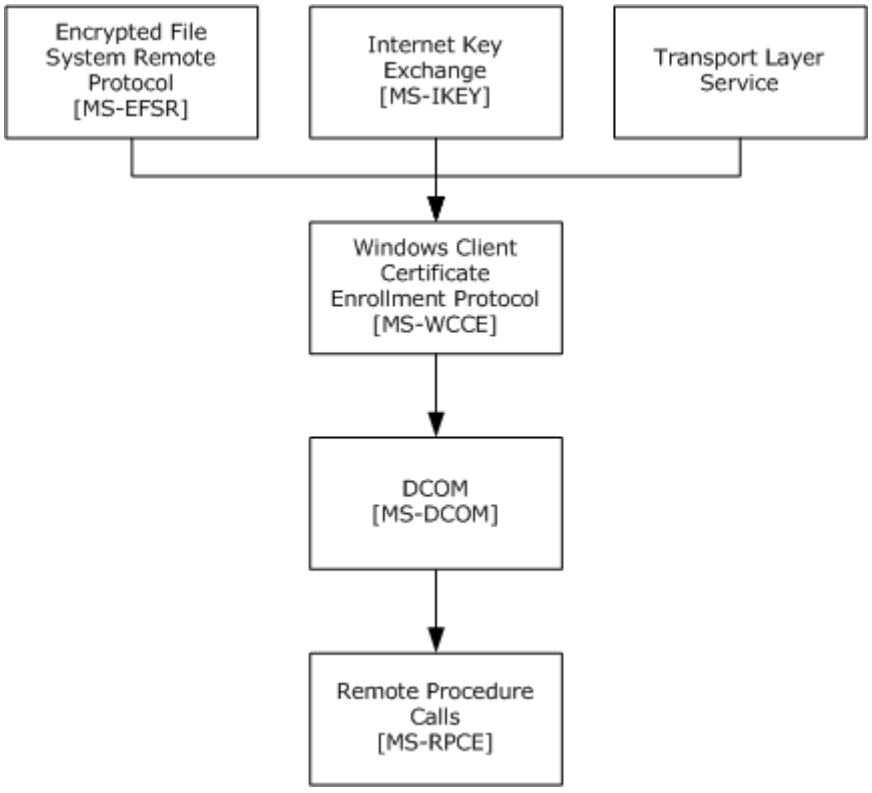

Figure 24: Dependencies on PKI

3.4 Implementation Scenarios

This section will describe two scenarios that illustrate common functionality that can be implemented with the protocols included in Certificate Services:

- Auto enrollment, the process of a client identifying itself and enrolling in the certificate service
- Certificate renewal, the process by which a client renews its certificate

3.4.1 Scenario 1 - Auto Enrollment

3.4.1.1 Scenario Overview

In this scenario, the client computer connects to the network and auto enrolls for the certificate services. The following sections show the network traffic related to the process of a client auto enrolling for certificate services and making use of the licensed technologies for this task.

3.4.1.2 Scenario Configuration

The following machine configurations are used for this scenario:

Windows Server 2003 operating system Domain Controller - WS200301

- Machine name: WS200301.contoso.com (WS200301)
- Operating system: Windows Server 2003 R2 operating system with latest patches
- IP Address: 10.0.10.1
- Subnet Mask: 255.255.255.0
- Active Directory Domain Controller - contoso.com

Windows Server 2003 Certificates Server - WS200302

- Machine name: WS200302.contoso.com (WS200302)
- Operating system: Windows Server 2003 R2 with latest patches
- IP Address: 10.0.10.2
- Subnet Mask: 255.255.255.0
- Domain member - contoso.com

Windows XP operating system Client Configuration-ClientXP

- Machine name: clientXP.contoso.com (CLIENTXP)
- Operating system: Windows XP Professional operating system SP2 with latest patches
- IP Address: 10.0.10.10
- Subnet Mask: 255.255.255.0
- Netmon 3.1

- Domain member - contoso.com

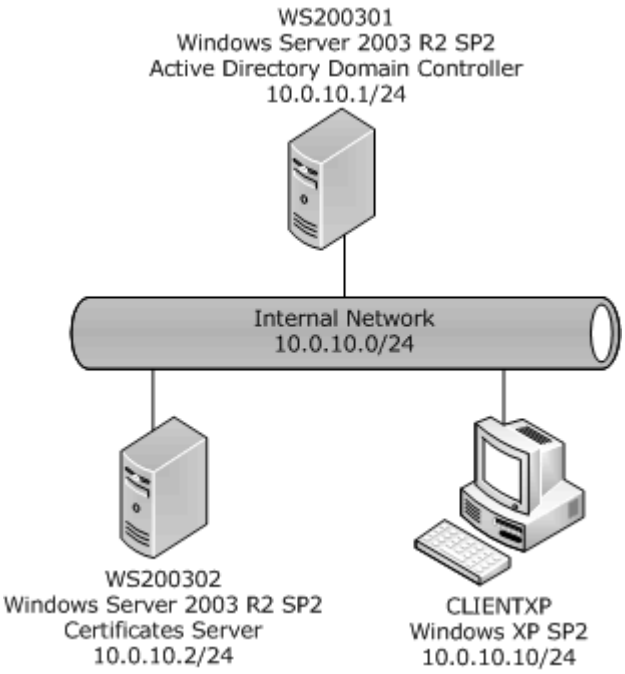

Figure 25: Network Topology

3.4.1.3 Setting Up the Trace

1. WS200302: Create a new workstation certificate.
2. ClientXP:
   - BEGIN TRACE
     - Enroll for new workstation certificate
   - END TRACE

3.4.1.4 Netmon Trace Digest

- The Windows Client Certificate Enrollment Protocol depends on the Distributed Component Object Model (DCOM) Remote Protocol. Data structures that are specified in the Certificate Templates Structure specification may be retrieved over the Lightweight Directory Access Protocol (LDAP) (as specified in [RFC2559]) and used throughout the enrollment process.
- As seen in this capture segment, the LDAP query is performed almost immediately in the auto-enrollment process:

```
{LDAP:11, TCP:10, IPv4:7}CLIENTXPWS200301LDAPLDAP:
Search Request, MessageID: 1, BaseObject: NULL, SearchScope:
base Object, SearchAlias: neverDerefAliases

{LDAP:11, TCP:10, IPv4:7}WS200301CLIENTXPLDAPLDAP:
Search Result Entry, MessageID: 1
```

```
{TCP:10, IPv4:7}CLIENTXPWS200301TCPTCP:
Flags=....A..., SrcPort=1113, DstPort=LDAP(389), Len=0, Seq=1201604453,
 Ack=3818299795, Win=65535 (scale factor 0) = 65535

{LDAP:12, TCP:10, IPv4:7}CLIENTXPWS200301LDAPLDAP:
 Bind Request, MessageID: 3, Version: 3

{LDAP:12, TCP:10, IPv4:7}WS200301CLIENTXPLDAPLDAP:
 Bind Response, MessageID: 3, Status: Success

{LDAP:13, TCP:10, IPv4:7}CLIENTXPWS200301LDAPLDAP:
 Search Request, MessageID: 4, BaseObject: NULL, SearchScope:
base Object, SearchAlias: neverDerefAliases

{LDAP:13, TCP:10, IPv4:7}WS200301CLIENTXPLDAPLDAP:
 Search Result Entry, MessageID: 4, Status: Success

{LDAP:14, TCP:10, IPv4:7}CLIENTXPWS200301LDAPLDAP:
 Search Request, MessageID: 5, BaseObject: CN=Certificate Templates,
CN=Public Key Services,CN=Services,CN=Configuration,DC=contoso,
DC=com, SearchScope: WholeSubtree, SearchAlias: neverDerefAliases

{LDAP:14, TCP:10, IPv4:7}WS200301CLIENTXPLDAPLDAP:
 Search Result Entry, MessageID: 5
```

A Kerberos Key Distribution Center (KDC) is used to authenticate the parties for authenticated DCOM messages, as shown here:

```
{MSRPC:34, TCP:33, IPv4:1}CLIENTXPWS200302  KerberosV5KerberosV5:
 AP Request Ticket[Realm: CONTOSO.COM, Sname:
RPCSS/ws200302.contoso.com]

{MSRPC:34, TCP:33, IPv4:1}WS200302  CLIENTXPKerberosV5KerberosV5:
 AP Response
{MSRPC:38, TCP:37, IPv4:1}CLIENTXPWS200302  KerberosV5KerberosV5:
 AP Request Ticket[Realm: CONTOSO.COM, Sname:
host/ws200302.contoso.com]

{MSRPC:38, TCP:37, IPv4:1}WS200302  CLIENTXPKerberosV5KerberosV5:
 AP Response

{MSRPC:38, TCP:37, IPv4:1}CLIENTXPWS200302  MSRPCMSRPC:
c/o Alter Cont:  UUID{00000143-0000-0000-C000-000000000046}
DCOM-IRemUnknown2  Call=0x1

{MSRPC:38, TCP:37, IPv4:1}WS200302  CLIENTXPMSRPCMSRPC:
c/o Alter Cont Resp:  Call=0x1  Assoc Grp=0x1B18E  Xmit=0x16D0
Recv=0x16D0

{MSRPC:38, TCP:37, IPv4:1}CLIENTXPWS200302  DCOMDCOM:
IRemUnknown2:RemQueryInterface Request, Unparsed Data

{MSRPC:38, TCP:37, IPv4:1}WS200302  CLIENTXPDCOMDCOM:
IRemUnknown2:RemQueryInterface Response
```

The high-level operations performed by this protocol can be broken down into six discrete operations:

- Get a new certificate for self in which the client directly requests a new certificate from the CA for itself. This operation uses one ICertRequestD::Request or ICertRequestD2::Request2 call from the client to the CA.
- Get new certificate on behalf of another in which the RA requests a certificate on behalf of a client person (usually) or machine (potentially). This operation uses one ICertRequestD::Request or ICertRequestD2::Request2 call from the RA to the CA.
- Renew a certificate in which the client requests a replacement certificate (presumably with a later expiration date) in return for an old certificate that is reaching its end of life. This operation uses one ICertRequestD::Request or ICertRequestD2::Request2 call from the client to the CA.
- Get CA properties in which a client or RA queries the CA for its configuration and state. This operation uses one ICertRequestD::GetCACert or ICertRequestD2::GetCAProperty call to the CA.
- Ping a CA in which an End Entity (EE) or RA queries the CA to discover availability of the CA service. This operation uses one ICertRequestD::Ping or ICertRequestD2::Ping2 call to the CA.
- Archive a key in which a client uses a public key belonging to the CA to encrypt a copy of the private key corresponding to an encryption certificate and sends that encrypted private key to the CA for archiving. This is an optional sub-protocol. This operation uses two calls from the client to the CA: ICertRequestD::GetCACert or ICertRequestD2::GetCAProperty to retrieve the CA exchange certificate, followed by ICertRequestD::Request or ICertRequestD2::Request2 to deliver the encrypted private key.

Such operations generate protocol traffic identical or similar to this, as specified in [MS-WCCE]:

```
{MSRPC:40, TCP:39, IPv4:1}WS200302  CLIENTXPWCCSWCCS:
ICertRequestD2:Request2 Response, Status =
```

Almost all other traffic generated by this task relates to LDAP or LDAP over TCP.

3.4.2 Scenario 2: Certificate Renewal

This section documents the licensed actions involved in certificate renewal.

3.4.2.1 Scenario Overview

The following sections describes the network traffic related to the process of a client renewing an existing certificate.

3.4.2.2 Scenario Configuration

The following machine configurations are used for this scenario:

Windows Server 2003 operating system Domain Controller - WS200301

- Machine name: WS200301.contoso.com (WS200301)
- Operating system: Windows Server 2003 R2 operating system with latest patches
- IP Address: 10.0.10.1

- Subnet Mask: 255.255.255.0
- Active Directory Domain Controller - contoso.com

Windows Server 2003 Certificates Server - WS200302

- Machine name: WS200302.contoso.com (WS200302)
- Operating system: Windows Server 2003 R2 with latest patches
- IP Address: 10.0.10.2
- Subnet Mask: 255.255.255.0
- Domain member - contoso.com

Windows XP operating system Client Configuration-ClientXP

- Machine name: clientXP.contoso.com (CLIENTXP)
- Operating system: Windows XP Professional operating system SP2 with latest patches
- IP Address: 10.0.10.10
- Subnet Mask: 255.255.255.0
- Netmon 3.1
- Domain member - contoso.com

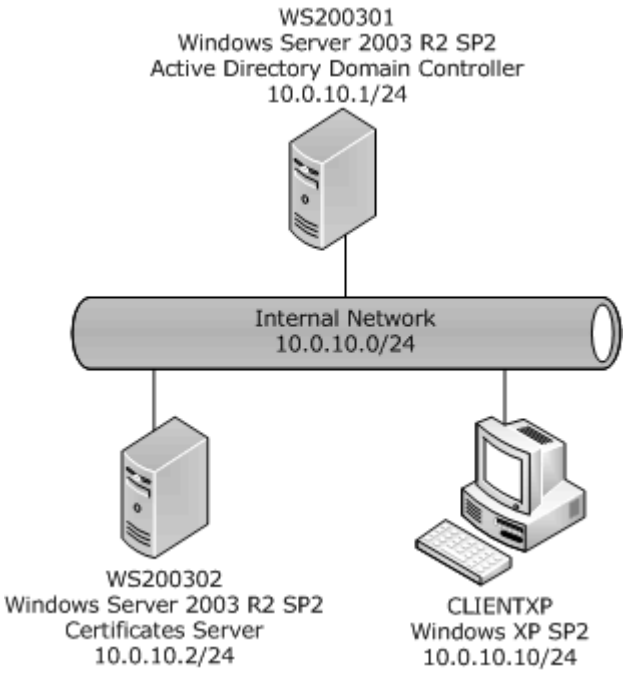

Figure 26: Network Topology

3.4.2.3 Setting Up the Trace

These steps produce the trace of a client renewing an existing certificate.

1. BEGIN TRACE
2. Renew Workstation certificate
3. END TRACE

3.4.2.4 Netmon Trace Digest

- The Certificate Renewal Scenario generates very similar network traffic to the Auto-Enrollment scenario.
- First a series of frames related to LDAP Search Requests is seen, followed by generation of a Kerebos authentication ticket (and all of its related traffic). With Kerberos approval received, the following packet traffic specific to Windows Client Certificate Enrollment Protocol is seen, as specified in [MS-WCCE]:

```
{MSRPC:17, TCP:16, IPv4:1}10.0.10.1010.0.10.2WCCSWCCS:
ICertRequestD2:Request2 Request, Authority: ContosoRootCA,
SerialNumber: NULL, Request Attributes:
cdc:ws200301.contoso.com
rmd:clientxp01.contoso.com
ccm:clientxp01.contoso.com
```

4 Collaboration Services Protocols

This section provides an overview of the Collaboration Services protocols. It provides a high-level conceptual overview of the core collaboration server protocols.

4.1 Collaboration Services Overview

These protocols can be used to provide audio and/or video conferencing, instant messaging, white boarding, application sharing, and similar services.

4.1.1 Protocols List

The Collaboration Services protocols includes a set of core protocols that provide collaboration services. The protocols are a subset of the networking protocols group described in "Introduction" section of this document.

The following table lists the core protocols included in the Collaboration Services protocols. This list does not include data structures, file structures, or algorithms that these core protocols depends on. The details for these technical dependencies are documented in the technical specifications for each core protocol.

Table: Collaboration Server core protocols

Protocol name	Description	Short name
Microsoft NetMeeting Protocol (includes extensions to the T.120 and H.323 protocol suites)	Implements various protocols in the International Telecommunications Union (ITU) T.120 suite for multimedia conferencing. T.120 is the umbrella protocol. H.323 is a communication standard produced by the ITU, is used to provide multimedia communications services over packet networks, including Voice over IP (VoIP, Internet Telephony, or IP Telephony).	[MS-MNPR]
Session Description Protocol (SDP) Extensions	SDP describes multimedia sessions for a variety of purposes associated with sessions. Microsoft has extended SDP with SDP: Microsoft Extensions to support encryption of data streams in multimedia sessions.	[MS-SDP]
Session Initiation Protocol Extensions	SIP is used to establish, modify, and terminate multimedia sessions or calls. Microsoft has extended SIP in a number of areas, including authentication, presence reporting, and subscription management.	[MS-SIP]
Telephony API Internet Locator Service Protocol	Internet Locator Service (ILS) Schema. Retrieves information about remote contacts such as people or conferences.	[MS-TAIL]
Telephony Remote Protocol	Defines the set of procedure calls that can be made between client and server. The client sends RPC buffers to the server in order to call Telephony API (TAPI) functions on the server. The server responds by sending asynchronous event messages regarding events that affect devices attached to the client.	[MS-TRP]

The following sections provide some basic collaboration concepts and individual sections that provide different conceptual views to help visualize the physical and logical relationships among the core protocols in Collaboration Services. These views are called the protocol stack view and logical dependencies view as explained in the "Introduction" section of this document.

4.2 Collaboration Services Concepts

The Collaboration Services technologies enable people to collaborate in desktop or browser-based workspaces. At the same time, these services provide a manageable infrastructure and extensible application platform for improving the efficiency of business processes.

The Collaboration Services technologies include protocols that provide audio and video conferencing, Instant Messaging, Whiteboarding, collaborative file transfer, and application-sharing services through Windows- or web-based consoles.

The "Implementation Scenarios" section 4.4, provides examples that illustrate the interaction of some of the protocols in two sample configurations.

4.2.1 S20 Protocol

The S20 Protocol includes extensions to the public T.12X and H.323 suites of protocols.

4.2.1.1 Extensions to T.120 Protocol

Multimedia Telecommunications involve the transport of information signals in a wide range of formats, efficiently, flexibly, and securely. Moreover, the communication protocol must not be confined to point-to-point operation between identical terminals, but also permit groups working between many terminals, which may be geographically isolated and diverse in their types. Such a protocol is defined in a series of ITU Recommendations collectively referred to as "the T.120 series".

The Microsoft extensions to the T.120 series were developed to add multicasting capabilities to the T.120 protocol, for the purpose of supporting NetMeeting conferences. The NetMeeting program is the Microsoft product provided with Windows that utilizes the Microsoft NetMeeting protocol. This program allows for voice, video, and text conferencing between two or more parties via TCP/UDP networks.

Implementing the extensions as specified in Microsoft NetMeeting Protocol extensions requires a thorough understanding of the T.120 series. The following diagram provides a high-level overview of the T.120 series

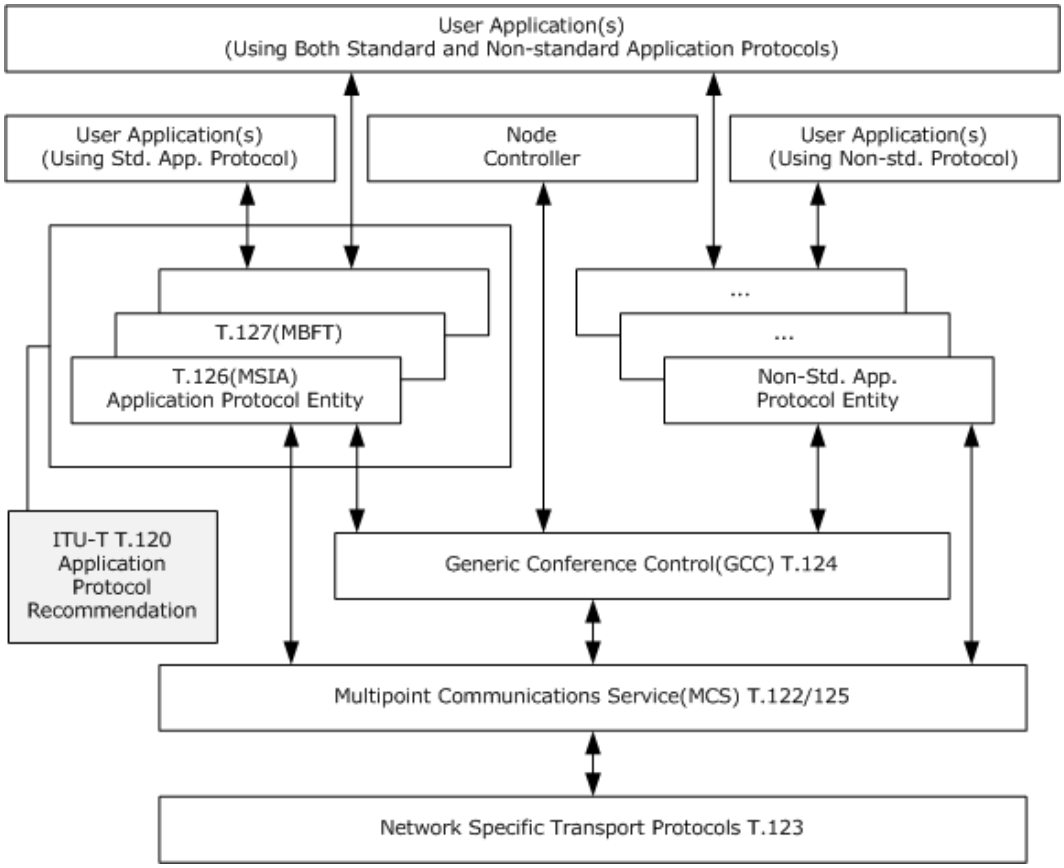

Figure 27: T.120 technical mapping

4.2.1.1.1 Netmeeting Object Manager Protocol

The Netmeeting Object Manager protocol defines a set of four control packet types (joiner, lock, wsgroup send, and operation) exchanged via the S20 layer.

4.2.1.1.2 Netmeeting Object Manager Late Joiner Protocol

The Netmeeting Object Manager implements the late joiner protocol to bring a latejoining instance up to date with the current contents of the workset group. When a Netmeeting client registers with a workset group that exists in a domain, the client is treated as a "late joiner" for the workset group.

The late joiner instance sends a message to the domain announcing its arrival, and receives one or more replies from the current domain instances. Next, the late joiner instance selects one of the replying instances as its "helper". The helper instance polls the other domain instances, assembles a current copy of the workset, and sends it to the late joiner.

4.2.1.1.3 Application Sharing

Microsoft NetMeeting implements a method of application sharing over the T.120 Multipoint Communications Services (MCS) layer, using the S20 MCS channel. The S20 MCS channel provides session management for application sharing between nodes in an application-sharing session. Share

v2.0 (part of the S20 protocol) provides the session establishment for application sharing, and MCS provides the broadcast transport for S20. For interoperability, the S20 MCS channel is designed to allow rudimentary communication with legacy application-sharing clients.

4.2.1.1.4 Whiteboard Protocol Extensions

Microsoft NetMeeting clients and servers engage in Whiteboard data sharing by exchanging ITU T.126 data.

4.2.1.1.5 Voice Communication

Microsoft NetMeeting clients and servers engage in voice communication through the Microsoft implementation of the H.323 communications standard, which is used to provide multimedia communications services over packet networks, including Voice over Internet Protocol [(VoIP), Internet Telephony, or IP Telephony)]. Microsoft NetMeeting is compliant with version 1998/02 of the H.323 standard.

4.2.2 Session Initiation Protocol (SIP) Extensions

Session Initiation Protocol Extensions, as specified in [MS-SIP] is an extension of the public standard Session Initiation Protocol (SIP), as specified in [RFC3261]. SIP is used to establish, modify and terminate multimedia sessions or calls. Microsoft has extended SIP in a number of areas, including authentication, presence reporting, and subscription management.

SIP messages can be used with transport used by SIP. This protocol defines a new format for the Presence Document.

4.2.3 Session Description Protocol (SDP): Microsoft Extensions

Microsoft extensions defines the extensions to the public SDP. The Session Description Protocol (SDP), as specified in [RFC4566] describes multimedia sessions for a variety of purposes associated with sessions. Session Description Protocol (SDP) Extensions, as specified in [MS-SDP] define additional SDP primitives to negotiate various types of sessions and also to negotiate encryption. These extensions include:

- Extensions to negotiate data collaboration sessions (application sharing & whiteboarding)
- Extensions to negotiate an instant messaging session
- Extensions to negotiate audio/video encryption
- Extensions to negotiate data collaboration encryption

SDP is used to negotiate session description between user agents. No server role relevant for SDP extensions.

4.2.4 Internet Locator Service (ILS) Schema Protocol

The Internet Locator Service (ILS) Schema Protocol, as specified in [MS-TAIL], is used for communication between a client using the Telephony Application Programming Interface (TAPI) and an ILS server.

ILS is a Dynamic Directory service that associates people and conferences with the IP addresses of their computers. ILS maintains the correct IP address for a person or conference even when their IP address changes. To do so, ILS provides a Lightweight Directory Access Protocol v3 (LDAP) interface and supports dynamic objects, as specified in [RFC2251].

4.2.4.1 ILS Variations from the LDAP V.3 Protocol

The ILS server is a dynamic directory service. It provides an LDAP v3 interface and supports dynamic objects as specified in [RFC2589]. ILS communication differs slightly from the published LDAP v.3 standards. This topic explains those variations.

ILS communication differs from LDAP v.3 in the following ways.

- ILS entries are dynamic; therefore, ILS does not support modify distinguished name (ModifyDN) requests.
- The ILS LDAP server does not support server-side sorting.
- The ILS LDAP server does not support TAPI session control operations.
- The ILS LDAP server does not support "friendly DN" as specified in [RFC1781].

4.2.4.2 NetMeeting Extensions to Standard LDAP Protocol

While connecting to an ILS server, NetMeeting does not follow the standard LDAP protocol. The following list outlines the Internet Locator Service (ILS) Schema Protocol extensions to the standard LDAP protocol, , as specified in [MS-TAIL]:

- Provide a unique structure of Distinguished Names (DNs). NetMeeting puts the most significant elements in the DN first, instead of last.
- Does not include the required "objectclass" attribute. Instead, it adds an "OBJECTCLASS" element to the end of the DN.
- Does not insert parents into the LDAP server.
- Does not understand attribute aliases.
- Requires that attributes in a search request be returned in exactly the same order they were requested.
- Specifies "base" scope in search requests, instead of "sub".
- Uses the "%" character as wildcard in search requests, instead of the "*" character specified by the standard.
- In name attributes "surname", "givenname", encodes accented European characters as 8-bit ISO 8859-1, instead of multi character UTF-8 sequences as required by LDAP ([RFC2252] and [RFC2256]).
- Uses a non-standard means of refreshing dynamic entries.
- Does not create the time-to-live attribute. Also, the client does not provide the whole DN that requires update; it only supplies the "cn" component.

The NetMeeting extensions to the standard LDAP protocol are explained in further detail in the ILS Schema Protocol specification, as specified in [MS-TAIL].

4.2.5 Telephony Remote Protocol

The Telephony Remote Protocol, as specified in [MS-TRP] enables a client to control telephony devices on the server through TAPI and to manage or administer them. The server software can generally be described as:

- TAPI Service, which is independent of device specifics and depends on device-specific software for actual device control.
- Telephony Service Provider (TSP), which is device-specific software (including the device driver software).

The TAPI service and the TSP may communicate with each other according to a well-defined interface: the Telephony Service Provider Interface (TSPI).

4.2.5.1 Telephony Remote Protocol Interfaces

The Telephony Remote Protocol consists of two interfaces: the **tapsrv** interface and the **remotesp** interface.

The **tapsrv** interface allows the client to send RPC buffers to the server, causing TAPI operations to be executed on the server. The RPC buffers in this document are named for the specific TAPI operation that will be executed.

TAPI operations can complete either synchronously or asynchronously.

- **Synchronous completion** — occurs when the requested TAPI operation is completely executed before the RPC function call returns to the client. This includes the case when the operation was not executed and an error is synchronously returned to the client.
- **Asynchronous completion** — is when the RPC function call returns to the client while the request is still being executed (for example, the RPC function calls returns while the client is dialing a number on a telephony device). A request ID is returned from the server when the asynchronous function call returns to the client. When the TAPI operation completes later, the server informs the client of completion along with success or error status, using the same request ID to identify the operation being completed.

4.3 Collaboration Services Protocols Logical Dependencies Protocol Stack View

This section provides a conceptual diagram to describe the logical dependencies among the collaboration services protocols and protocol stack views as appropriate.

4.3.1 Client and Server Integrated Collaboration Server Protocols

The following sections describe each of the protocols, and their relationships to other protocols.

4.3.2 Session Initiation Protocol Logical Dependencies and Relationship to Other Protocols

SIP messages can be transported over UDP, TCP, or Transport Layer Security (TLS). The Microsoft extensions to SIP define additional SIP primitives and Extensible Markup Language (XML) schema to support various extensions. In addition, this protocol defines some authentication extensions that make use of NT Local Area Network Manager (NTLM) and Kerberos protocols.

This protocol assumes that both the client and the server support SIP. The prerequisites for SIP extensions are the same as the prerequisites for SIP. For more information about SIP, see [RFC3261].

4.3.3 Stack View

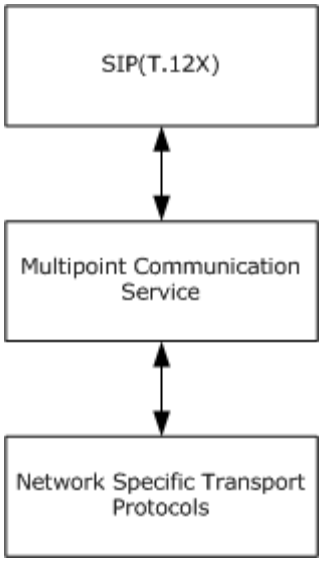

Figure 28: SIP Stack

4.3.4 Session Description Protocol: Microsoft Extensions Relationship to Other Protocols

Microsoft Session Description Protocol (SDP) Extensions depends on SDP as specified in [RFC4566] and upon Session Initiation Protocol (SIP). SDP Extensions defines additional SDP primitives needed to negotiate encryption parameters for data collaboration and audio/video sessions.

The encryption negotiation enhancements to SDP may be used when the application wants to encrypt audio/video and data collaboration media on the wire. These enhancements should be used in conjunction with Transport Layer Security (TLS) to protect the encryption key if the key is passed in the SIP/SDP signaling.

4.3.5 Stack View

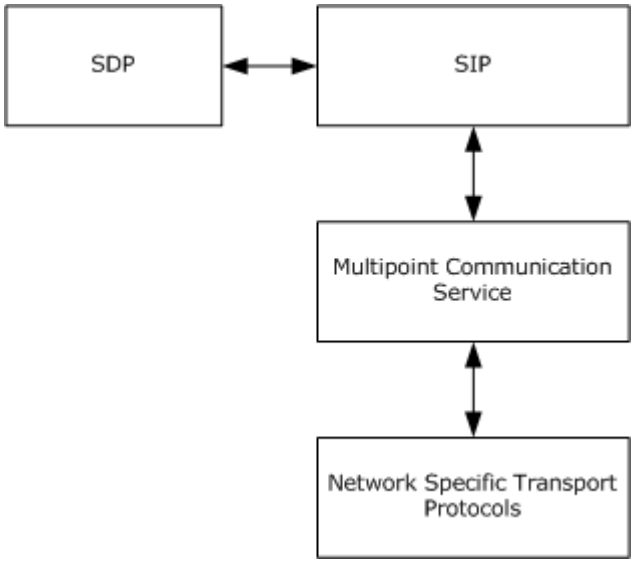

Figure 29: SDP extensions stack view

4.3.6 Telephony Remote Protocol: Relationship to Other Protocols

The Telephony Remote Protocol requires the RPC protocol for communication from client to server and for communication from server to connection-oriented client. It also depends on mailslot-mechanism support for communications from server to connection-less clients.

4.3.7 Stack View

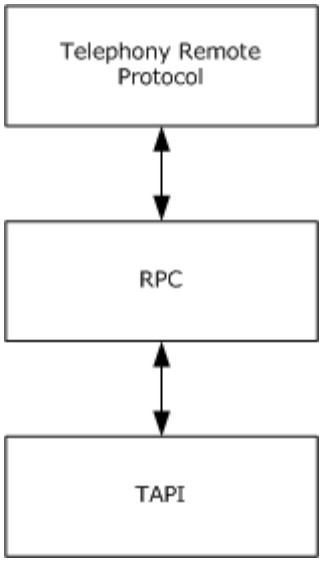

Figure 30: Telephony Remote Protocol stack view

4.4 Collaboration Services Implementation Scenarios

This section provides two scenarios that illustrate common functionality that can be implemented with the protocols included in the Collaboration Services. The protocols involved in these scenarios include the T.126 Protocol: Microsoft NetMeeting Extensions [MS-MNPR], Telephony Remote Protocol [MS-TRP], together with other general networking protocols used in the course of standard TCP communication between the client and server. The functionality illustrated in the scenarios is as follows:

- Two Windows XP operating system clients exchanging an instant message (IM)
- Windows XP client sending a text file to another Windows XP client using Windows NetMeeting

It should be noted that the following scenarios depict processes that occur over TCP and thus assumes various TCP acknowledgment frames throughout the conversation that are not necessarily depicted in the trace details.

4.4.1 Trace Scenario 1 - Windows XP Client 1 sends an IM to Windows XP Client 2

Computer Name	Operating System	Configurations
CLIENTXP01	Windows XP operating system Service Pack 2 (SP2)	NetMeeting 3.01
CLIENTXP02	Windows XP SP2	NetMeeting 3.01

4.4.1.1 Scenario Overview

In this scenario, a Windows XP operating system Client 1 (CLIENTXP01) sends an instant message to another Windows XP Client 2 (CLIENTXP02). Both clients use the built-in version of NetMeeting (version 3.01).

4.4.1.2 Scenario Configuration

This scenario is implemented with two Windows XP operating system client machines configured as follows:

ClientXP01 Configuration

- Machine name: ClientXP01
- Operating system: Windows XP Professional operating system SP2 with latest updates
- IP address: 10.0.10.12
- Subnet mask: 255.255.255.0
- Built-in Netmeeting version 3.01 installed

ClientXP02 Configuration

- Machine name: ClientXP01
- Operating system: Windows XP Professional SP2 with latest updates
- IP address: 10.0.10.13
- Subnet mask: 255.255.255.0

- Built-in Netmeeting version 3.01 installed

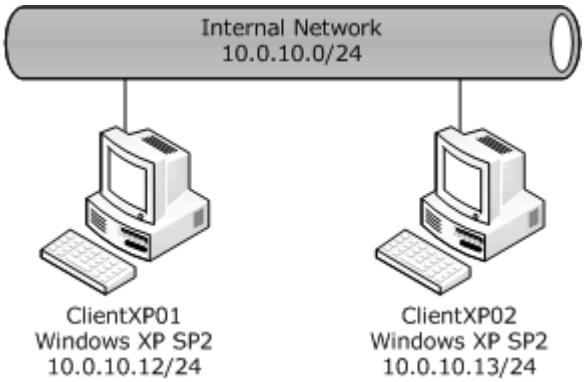

Figure 31: Network Topology

4.4.1.3 Setting up the trace

These steps produce the Windows XP operating system client IM communication trace:

1. BEGIN TRACE
2. Click "Place Call" button on ClientXP01
3. Type IP Address of ClientXP02 and then click Call
4. On ClientXP02, click "Accept"
5. On ClientXP01, click "chat"
6. Type text message, and then click Send Message
7. On ClientXP01, exit chat window
8. On ClientXP01, click "End Call"
9. END TRACE

4.4.1.4 Netmon Trace Digest

- No network traffic is generated when the Netmeeting client is opened on either machine.
- Once the user clicks the "Place Call" button on ClientXP01 with the IP address of Client02 as the target, ClientXP01 issues a request to ClientXP02 for MAC address (hardware address) using the Address Resolution Protocol (ARP):

```
CLIENTXP01CLIENTXP02ARPARP: Request,
10.0.10.12 asks for 10.0.10.13
```

- ClientXP02 responds with the MAC address of ClientXP02:

```
CLIENTXP02CLIENTXP01ARPARP: Response,
10.0.10.13 at 00-03-FF-9D-46-6B
```

- ClientXP01 issues an H.323 host call SYN to ClientXP02:

```
CLIENTXP01CLIENTXP02TCPTCP: Flags=.S....., SrcPort=1107,
DstPort=1720, Len=0
```

- ClientXP02 responds with an H.323 SYN/ACK to ClientXP01:

```
CLIENTXP02CLIENTXP01TCPTCP: Flags=.S..A..., SrcPort=1720,
 DstPort=1107, Len=0
```

- ClientXP01 issues an H.323 ACK host call to ClientXP02 to complete the handshake:

```
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1107,
 DstPort=1720, Len=0
```

- ClientXP01 issues a Q.931 SETUP command to ClientXP02:

```
CLIENTXP01CLIENTXP02Q.931TCP: Flags=...PA..., SrcPort=1107,
 DstPort=1720, Len=4
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1720,
 DstPort=1107, Len=0
```

- ClientXP02 responds with an H.225 ALERTING indicator to ClientXP02, triggering the Netmeeting client to generate an audible ringtone indicator and present the connection GUI to accept or cancel the incoming call:

```
CLIENTXP02CLIENTXP01H.225TCP: Flags=...PA..., SrcPort=1720,
 DstPort=1107, Len=4
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1107,
 DstPort=1720, Len=0
```

- When the user clicks the "Accept Call" button on ClientXP02, ClientXP02 issues an H.225 CONNECT command to ClientXP01:

```
CLIENTXP02CLIENTXP01H.225TCP: Flags=...PA..., SrcPort=1720,
 DstPort=1107, Len=4
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1107,
 DstPort=1720, Len=0
```

- ClientXP01 establishes an H.245 control channel to ClientXP02:

```
CLIENTXP01CLIENTXP02H.245TCP: Flags=.S....., SrcPort=1108,
 DstPort=1042, Len=0
CLIENTXP02CLIENTXP01TCPTCP: Flags=.S..A..., SrcPort=1042,
 DstPort=1108, Len=0
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1094,
 DstPort=1042, Len=0
```

- ClientXP01 issues an X.224 Connection Request to ClientXP02 to establish the first channel:

```
CLIENTXP01CLIENTXP02X.224TCP: Connection Request, Flags=...P....,
 SrcPort=1109, DstPort=1503, Len=25
```

- ClientXP02 responds with an X.224 Connection Confirm to ClientXP01:

```
CLIENTXP02CLIENTXP01X.224TCP: Connection Confirm, Flags=...PA...,
 SrcPort=1503, DstPort=1109, Len=21
```

- ClientXP01 and ClientXP02 exchange MCS (Multipoint Communications Service) T.125 packets to initiate a Netmeeting connection. ClientXP01 initiates this communication by sending the first MCS transport connection to ClientXP02:

```
CLIENTXP01CLIENTXP02T.125TCP: MCS Connect-Initial, SrcPort=1109,
 DstPort=1503, Len=181
```

- ClientXP02 responds with an MCS transport acceptance, acknowledged by ClientXP01:

```
CLIENTXP02CLIENTXP01T.125TCP: MCS Connect-Response, SrcPort=1503,
 DstPort=1109, Len=113
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1109,
 DstPort=1503, Len=0
```

- Netmeeting makes use of multiple channels to facilitate one-to-one and one-to-many communication, including separate channels for each service such as online state (away/free/busy/offline) or multiple independent chat sessions with different clients. ClientXP01 exchanges an X.224 connection with ClientXP02 to establish a second transport connection:

```
CLIENTXP01CLIENTXP02X.224TCP: Connection Request, Flags=...P....,
 SrcPort=1110, DstPort=1503, Len=25
CLIENTXP02CLIENTXP01X.224TCP: Connection Confirm, Flags=...PA...,
 SrcPort=1503, DstPort=1110, Len=21
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1110,
 DstPort=1503, Len=0
```

- ClientXP01 negotiates two additional X.224 connections with ClientXP02 to extend the established transport connection via two sequential ports:

```
CLIENTXP01CLIENTXP02X.224TCP: Connection Request, Flags=...P....,
 SrcPort=1111, DstPort=1503, Len=25
CLIENTXP01CLIENTXP02X.224TCP: Connection Request, Flags=...P....,
 SrcPort=1112, DstPort=1503, Len=25
CLIENTXP02CLIENTXP01X.224TCP: Connection Confirm, Flags=...PA...,
 SrcPort=1503, DstPort=1111, Len=21
CLIENTXP02CLIENTXP01X.224TCP: Connection Confirm, Flags=...PA...,
 SrcPort=1503, DstPort=1112, Len=21
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1111,
 DstPort=1503, Len=0
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1112,
 DstPort=1503, Len=0
```

- ClientXP01 generates two MCS Additional packets to extend the existing transport connection for the added channels:

```
CLIENTXP01CLIENTXP02T.125TCP: MCS Connect-Additional,
 SrcPort=1111, DstPort=1503, Len=16
CLIENTXP01CLIENTXP02T.125TCP: MCS Connect-Additional,
 SrcPort=1112, DstPort=1503, Len=16
```

- ClientXP02 acknowledges the additions by responding with a Connect-Result packet to each of the new transport connections:

```
CLIENTXP02CLIENTXP01T.125TCP: MCS Connect-Result,
 SrcPort=1503, DstPort=1111, Len=13
CLIENTXP02CLIENTXP01T.125TCP: MCS Connect-Result,
 SrcPort=1503, DstPort=1112, Len=13
CLIENTXP01CLIENTXP02TCPTCP: Flags....A..., SrcPort=1111,
 DstPort=1503, Len=0
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1112,
 DstPort=1503, Len=0
```

- With the extension of the height of the hierarchical T.125 domain due to the two added channels, ClientXP02 generates an Erect Domain request to its superior:

```
CLIENTXP02CLIENTXP01T.125TCP: MCS Erect Domain Request,
 SrcPort=1503, DstPort=1110, Len=12
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1110,
 DstPort=1503, Len=0
```

- ClientXP02 initiates an Attach User request to its superior, to register the local Netmeeting user on ClientXP02 with the top MCS provider on ClientXP01:

```
CLIENTXP02CLIENTXP01T.125TCP: MCS Attach User Request
 SrcPort=1503, DstPort=1110, Len=9
```

- ClientXP01 responds with a confirmation that includes the assigned UserID (18148):

```
CLIENTXP01CLIENTXP02T.125TCP: MCS Attach User Confirm, Result=
 rt-successful, Indicator=18148, SrcPort=1503, DstPort=1110, Len=11
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1110,
 DstPort=1503, Len=0
```

- ClientXP02 initiates channel join requests for the new UserID for channels 1 and 18148, requesting access to channels in the MCS tree at the root and UserID level:

```
CLIENTXP02CLIENTXP01T.125TCP: MCS Channel Join Request, UserID=
 18148, ChannelID=18148, SrcPort=1503, DstPort=1110, Len=12
CLIENTXP02CLIENTXP01T.125TCP: MCS Channel Join Request, UserID=
 18148, ChannelID=1, SrcPort=1503, DstPort=1110, Len=12
```

- ClientXP01 responds to the channel join requests:

```
CLIENTXP01CLIENTXP02T.125TCP: MCS Channel Join Confirm,
 ChannelID=18148, SrcPort=1110, DstPort=1503, Len=15
CLIENTXP01CLIENTXP02T.125TCP: MCS Channel Join Confirm,
 ChannelID=1, SrcPort=1110, DstPort=1503, Len=15
```

- ClientXP02 responds with an ACK and a Generic Conference Control (GCC) T.124 conference create response packet, establishing the new conference tag (1):

```
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
 DstPort=1110, Len=0
CLIENTXP02CLIENTXP01T.124TCP: GCC UserIDIndication, Tag=1,
 SrcPort=1503, DstPort=1110, Len=18
```

- ClientXP02 initiates a GCC roster update by issuing a roster update indication to the next higher node in the MCS tree:

```
CLIENTXP02CLIENTXP01T.124TCP: GCC RosterUpdateIndication,
 FullRefresh=False, NodeInformation=NodeRecordListUpdate,
 RosterInstanceNumber=1, NodesAdded=True, NodesRemoved=False,
 SrcPort=1503, DstPort=1111, Len=321
```

- ClientXP01 replies with an updated GCC roster, specifying a full refresh of the conference roster on ClientXP02:

```
CLIENTXP01CLIENTXP02T.124TCP: GCC RosterUpdateIndication,
 FullRefresh=True, NodeInformation=NodeRecordListRefresh,
 RosterInstanceNumber=2, NodesAdded=True, NodesRemoved=False,
 SrcPort=1111, DstPort=1503, Len=321
```

- ClientXP01 acknowledges the previous conference create response by sending a GCC conference unlock message to ClientXP02, completing the conference creation handshake:

```
CLIENTXP01CLIENTXP02T.124TCP: GCC ConferenceUnlockIndication,
 SrcPort=1111, DstPort=1503, Len=16
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
 DstPort=1111, Len=0
```

- ClientXP02 initiates a sequence of T.125 Attach User and Channel Join sequences for each application protocol entity taking part in the connection, including services supporting connectivity state and other notification processes. The sequence for each new user and channel is identical to that presented for the first user ID (18148) except that the UserID and ChannelID values are incremented for each successive event.
- To register each new channel with the established GCC conference, ClientXP02 issues a GCC channel registration to ClientXP01 (ChannelID 18151 shown):

```
CLIENTXP02CLIENTXP01T.124TCP: GCC RegistryRegisterChannelRequest,
 EntityID=1, Node=18146, ChannelID=18151, SrcPort=1503, DstPort=1111,
 Len=37
```

- ClientXP01 replies to the channel registration with a GCC channel registration acknowledgement to ClientXP02, returning the source node in place of the Channel ID:

```
CLIENTXP01CLIENTXP02T.124TCP: GCC RegistryResponse
 RegisterChannel, ChannelID=18146, SrcPort=1111, DstPort=1503, Len=45
```

- After all new channels have been registered in the established GCC conference, ClientXP02 sends a handle allocation request to the next higher node in the MCS hierarchy:

```
CLIENTXP02CLIENTXP01T.124TCP: GCC RegistryAllocateHandleRequest,
 EntityID=2, HandleNumber=1, SrcPort=1503, DstPort=1111, Len=41
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1111,
 DstPort=1503, Len=0
```

- ClientXP02 initiates a token request to ClientXP01, receiving the new TokenID in acknowledgement:

```
CLIENTXP02CLIENTXP01T.124TCP: GCC RegistryAssignTokenRequest,
 EntityID=1, Node=16483, SrcPort=1503, DstPort=1111, Len=35
CLIENTXP01CLIENTXP02T.124TCP: GCC RegistryResponse AssignToken,
 TokenID=16384, SrcPort=1111, DstPort=1503, Len=45
```

- To facilitate application sharing, the S.20 application protocol entity on ClientXP02 requests the new token by issuing a T.125 grab request to ClientXP01:

```
CLIENTXP02CLIENTXP01T.125TCP: Token Grab Request,
 UserID=18149, TokenID=16384, SrcPort=1503, DstPort=1111, Len=45
CLIENTXP01CLIENTXP02T.125TCP: Token Grab Confirm,
 TokenID=16384, SrcPort=1111, DstPort=16384, Len=14
```

- The S.20 application protocol entity on ClientXP02 issues an S.20 Join command to the next higher MCS node, registering the public name of the initiating client identity (here, Client2 Test) for later application sharing:

```
CLIENTXP02CLIENTXP01S.20TCP: AppShare S.20 Join,
 Name=Client2 Test, SrcPort=1503, DstPort=1111, Len=231
CLIENTXP01CLIENTXP02T.125TCP: Token Grab Confirm,
 TokenID=16384, SrcPort=1111, DstPort=16384, Len=14
```

- ClientXP02 verifies connectivity for sharing by issuing a flow test message to ClientXP01:

```
CLIENTXP02CLIENTXP01S.20TCP: AppShare FlowTestPDU,
 PDUSource=18149, SrcPort=1503, DstPort=1111, Len=22
CLIENTXP01CLIENTXP02S.20TCP: AppShare FlowResponsePDU,
 PDUSource=18144, SrcPort=1111, DstPort=1503, Len=26
```

- ClientXP02 has not yet joined the existing Netmeeting workset established by ClientXP01 when initiating the call. ClientXP02 initiates a late-join operation using the Netmeeting Object Manager protocol to issue a Hello:

```
CLIENTXP02CLIENTXP01NetmeetingTCP: AppShare Object Manager,
```

```
 MesageType=OMNET_HELLO (0x000A) Sender=18149, SrcPort=1503,
 DstPort=1111, Len=15
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1111,
 DstPort=1503, Len=0
```

- ClientXP01 replies using the Netmeeting Object Manager protocol to issue a Welcome:

```
CLIENTXP01CLIENTXP02NetmeetingTCP: AppShare Object Manager,
 MesageType=OMNET_WELCOME (0x000B) Sender=18144, SrcPort=1111,
 DstPort=1503, Len=15
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
 DstPort=1111, Len=0
```

- The Object Manager helper object on ClientXP01 sends a command to ClientXP02 to add the existing sender to its workset and initiates a catchup operation:

```
CLIENTXP01CLIENTXP02NetmeetingTCP: AppShare Object Manager,
 MesageType=OMNET_OBJECT_ADD (0x0032) Sender=18144, SrcPort=1111,
 DstPort=1503, Len=23
CLIENTXP01CLIENTXP02NetmeetingTCP: AppShare Object Manager,
 MesageType=OMNET_WORKSET_CATCHUP (0x0030) Sender=18144, SrcPort=1111,
 DstPort=1503, Len=23
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
 DstPort=1111, Len=0
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
 DstPort=1111, Len=0
```

- ClientXP01 notifies ClientXP02 of all worksets and their contents:

```
CLIENTXP01CLIENTXP02NetmeetingTCP: AppShare Object Manager,
 MesageType=OMNET_WSGROUP_SEND_MIDWAY (0x001F) Sender=18144,
 SrcPort=1111, DstPort=1503, Len=23
CLIENTXP01CLIENTXP02NetmeetingTCP: AppShare Object Manager,
 MesageType=OMNET_WSGROUP_SEND_MIDWAY (0x001F) Sender=18144,
 SrcPort=1111, DstPort=1503, Len=23
CLIENTXP01CLIENTXP02NetmeetingTCP: AppShare Object Manager,
 MesageType=OMNET_WSGROUP_SEND_MIDWAY (0x001F) Sender=18144,
 SrcPort=1111, DstPort=1503, Len=23
CLIENTXP01CLIENTXP02NetmeetingTCP: AppShare Object Manager,
 MesageType=OMNET_OBJECT_CATCHUP (0x0033) Sender=18144, SrcPort=1111,
 DstPort=1503, Len=123
CLIENTXP01CLIENTXP02NetmeetingTCP: AppShare Object Manager,
 MesageType=OMNET_OBJECT_CATCHUP (0x0033) Sender=18144, SrcPort=1111,
 DstPort=1503, Len=123
CLIENTXP01CLIENTXP02NetmeetingTCP: AppShare Object Manager,
 MesageType=OMNET_OBJECT_CATCHUP (0x0033) Sender=18144, SrcPort=1111,
 DstPort=1503, Len=123
```

- ClientXP01 signals ClientXP02 that all workset updates have been sent:

```
CLIENTXP01CLIENTXP02NetmeetingTCP: AppShare Object Manager,
 MesageType=OMNET_WSGROUP_SEND_COMPLETE (0x0020) Sender=18144,
 SrcPort=1111, DstPort=1503, Len=23
```

- ClientXP02 acknowledges all update packets:

```
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
 DstPort=1111, Len=0
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
 DstPort=1111, Len=0
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
 DstPort=1111, Len=0
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
 DstPort=1111, Len=0
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
 DstPort=1111, Len=0
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
 DstPort=1111, Len=0
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
 DstPort=1111, Len=0
```

- As the process is repeated to register all new application entities to the existing workset, ClientXP02 issues a lock request to ClientXP01 to protect against changes from other Netmeeting clients during registration. The lock is granted and then released before the next set of OMNET object add and catchup operations is conducted:

```
CLIENTXP02CLIENTXP01NetmeetingTCP: AppShare Object Manager,
 MesageType=OMNET_LOCK_REQ (0x0015) Sender=18149, SrcPort=1503,
 DstPort=1111, Len=15
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1111,
 DstPort=1503, Len=0
CLIENTXP01CLIENTXP02NetmeetingTCP: AppShare Object Manager,
 MesageType=OMNET_LOCK_GRANT (0x0015) Sender=18144, SrcPort=1111,
 DstPort=1503, Len=15
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
 DstPort=1111, Len=0
CLIENTXP02CLIENTXP01NetmeetingTCP: AppShare Object Manager,
 MesageType=OMNET_UNLOCK (0x0018) Sender=18149, SrcPort=1503,
 DstPort=1111, Len=15
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1111,
 DstPort=1503, Len=0
```

- When the user clicks the "Chat" button on ClientXP01, ClientXP01 issues a T.124 GCC roster update to ClientXP02 and transmits a GCC application invoke notification for the chat interface:

```
CLIENTXP01CLIENTXP02T.124TCP: GCC RosterUpdateIndication,
 FullRefresh=False, NodeInformation=NoderecordList,
 RosterInstanceNumber=2, NodesAdded=False, NodesRemoved=False,
 SrcPort=1111,  DstPort=1503, Len=62
CLIENTXP01CLIENTXP02T.124TCP: GCC RosterUpdateIndication,
 FullRefresh=False, NodeInformation=NoderecordList,
 RosterInstanceNumber=2, NodesAdded=False, NodesRemoved=False,
 SrcPort=1111,  DstPort=1503, Len=62
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
 DstPort=1111, Len=0
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
 DstPort=1111, Len=0
CLIENTXP01CLIENTXP02T.124TCP: GCC ApplicationInvokeIndication,
 SrcPort=1111,  DstPort=1503, Len=45
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1503,
```

```
 DstPort=1111, Len=0
```

- As ClientXP02 initiates its own chat interface, it generates an attach user request, channel join request, and GCC roster update request for the new application entity. These are acknowledged by ClientXP01 in the same manner as previously detailed. ClientXP02 also transmits a GCC application invoke notification as its chat interface is enabled:

```
CLIENTXP02CLIENTXP01T.124TCP: GCC ApplicationInvokeIndication,
 SrcPort=1503,  DstPort=1111, Len=62
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1111,
 DstPort=1503, Len=0
```

- After the user on ClientXP01 types in a text message and clicks the "Send Text", ClientXP01 issues a Netmeeting Chat exchange containing the transmitted message in clear text (security would require an additional protocol such as an IPSec tunnel):

```
CLIENTXP01CLIENTXP02NetmeetingTCP: Chat=This is a test message,
 SrcPort=1112,  DstPort=1503, Len=54
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1112,
 DstPort=1503, Len=0
```

- When the user on ClientXP01 closes the chat interface, ClientXP01 transmits a T.125 detach user indication along with a GCC roster update to ClientXP02:

```
CLIENTXP01CLIENTXP02T.125TCP: MCS Detach User Indication, Reason=
 rn-user-requested, SrcPort=1110, DstPort=1503, Len=12
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1110,
 DstPort=1503, Len=0
CLIENTXP01CLIENTXP02T.124TCP: GCC RosterUpdateIndication,
 FullRefresh=False, NodeInformation=NoderecordList: NoChange,
 RosterInstanceNumber=2, NodesAdded=False, NodesRemoved=False,
 SrcPort=1110,  DstPort=1503, Len=12
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1110,
 DstPort=1503, Len=0
```

- When the user on ClientXP01 clicks the "End Call" button, ClientXP01 transmits a self-initiated GCC eject user indication to all clients, followed by an end of data statement for each channel:

```
CLIENTXP01CLIENTXP02T.124TCP: GCC ConferenceEjectUserIndication,
 UserID NodeToEject=18148, SrcPort=1110, DstPort=1503, Len=19
CLIENTXP01CLIENTXP02T.125TCP: Disconnect Provider Ultimatum,
 Reason=rn-user-requested, SrcPort=1110, DstPort=1503, Len=9
CLIENTXP01CLIENTXP02TCPTCP: Flags=F...A..., SrcPort=1110,
 DstPort=1503, Len=0
CLIENTXP01CLIENTXP02TCPTCP: Flags=F...A..., SrcPort=1111,
 DstPort=1503, Len=0
CLIENTXP01CLIENTXP02TCPTCP: Flags=F...A..., SrcPort=1112,
 DstPort=1503, Len=0
CLIENTXP01CLIENTXP02TCPTCP: Flags=F...A..., SrcPort=1108,
 DstPort=1065, Len=0
ClientXP01 issues a Q.931 release command to ClientXP02:
CLIENTXP01CLIENTXP02Q.931TCP: Flags=...PA..., SrcPort=1107,
 DstPort=1720, Len=4
```

```
CLIENTXP02CLIENTXP01TCPTCP: Flags=....A..., SrcPort=1720,
 DstPort=1107, Len=0
```

- Once all channels have been terminated, ClientXP02 responds with an H.225 release complete signal to ClientXP01:

```
CLIENTXP02CLIENTXP01H.225TCP: Flags=F..PA..., SrcPort=1720,
 DstPort=1107, Len=4
CLIENTXP01CLIENTXP02TCPTCP: Flags=....A..., SrcPort=1107,
 DstPort=1720, Len=0
```

4.4.2 Trace Scenario 2 - Windows XP Client 1 Sends a Text file to Windows XP Client 2

4.4.2.1 Scenario Overview

In the following scenario, Client 1 (CLIENTXP01) sends a text file to Client 2 (CLIENTXP02). Both clients use the built-in version of NetMeeting (version 3.01).

4.4.2.2 Scenario Configuration

This scenario is implemented with two Windows XP operating system client machines configured as follows:

ClientXP01 Configuration

- Machine name: ClientXP01
- Operating system: Windows XP Professional operating system SP2 with latest updates
- IP address: 10.0.10.12
- Subnet mask: 255.255.255.0
- Built-in Netmeeting version 3.01 installed

ClientXP02 Configuration

- Machine name: ClientXP01
- Operating system: Windows XP Professional SP2 with latest updates
- IP address: 10.0.10.13
- Subnet mask: 255.255.255.0
- Built-in Netmeeting version 3.01 installed

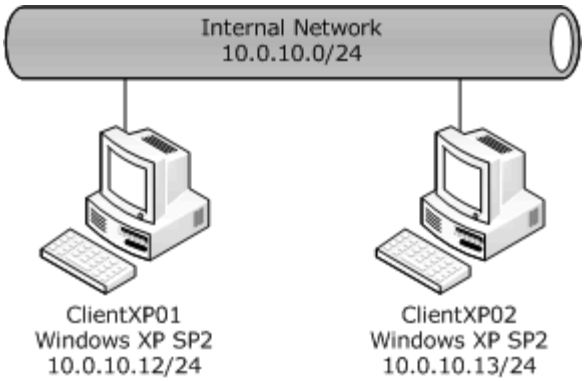

Figure 32: Network Topology

4.4.2.3 Setting up the trace

These steps produce the text-file transmission trace:

1. BEGIN TRACE
    1. Click "Place Call" button on ClientXP01
    2. Type IP Address of ClientXP02 and then click Call
    3. On ClientXP02, click "Accept"
    4. On ClientXP01, click "Transfer Files"
    5. Click "Add Files"
    6. Select test file, and then click "Add"
    7. Click "Send All"
2. END TRACE

4.4.2.4 Netmon Trace Digest

- ClientXP01 issues request to ClientXP02 for MAC address (hardware address) using the ARP protocol:

```
CLIENTXP01  CLIENTXP02  ARPARP: Request,
10.0.10.12 asks for 10.0.10.13
```

- ClientXP02 responds with the following Address Resolution Protocol (ARP) command containing the MAC address of the client:

```
CLIENTXP02  CLIENTXP01  ARPARP: Response, 10.0.10.13 at
00-03-FF-9C-46-6B
```

- Multiple TCP entries were observed before ClientXP01 and ClientXP02 exchange MCS (Multipoint Communications Service) T.125 packets to initiate and establish a NetMeeting connection.

Confirmations are sent to confirm that user is attached, the user's channel is accepted, and the broadcast channel is accepted:

```
CLIENTXP01CLIENTXP02T125T125: MCSConnect Initial

CLIENTXP02CLIENTXP01T125T125: MCSConnect Response

CLIENTXP01CLIENTXP02T125T125: X224 Connect Aditional

CLIENTXP02CLIENTXP01T125T125: Erect Domain Request,
SubHeight = 0, SubInterval = 0

CLIENTXP02CLIENTXP01T125T125: Attach User Request

CLIENTXP02CLIENTXP01T125T125: x224 Connect Result

CLIENTXP01CLIENTXP02T125T125: Attach User Confirm,
Result = rt-successful, Indicator = 33763

CLIENTXP02CLIENTXP01T125T125: Channel Join Request,
UserID = 33763,ChannelId = 33763

CLIENTXP02CLIENTXP01T125T125: Channel Join Request,
UserID = 33763,ChannelId = 1

CLIENTXP01CLIENTXP02T125T125: Channel Join Confirm,
ChannelId = 33763, Result = rt-successful
```

- Next, Generic Conference Control (GCC) – T.124, provides for setting up and managing the Multipoint conference. It provides access control and arbitration of capabilities. GCC facilities are used to coordinate independent use of the MCS channels and tokens within the same multipoint domain.

```
CLIENTXP02CLIENTXP01netmeetingnetmeeting:
GCC UserIDIndication Tag = 1
```

- Nodes can join and leave meetings at any time and GCC facilities can be used to query a node to find a desired meeting.

```
CLIENTXP02CLIENTXP01netmeetingnetmeeting: GCC
rosterUpdateIndication fullRefresh=False, NodeInformation:
nodeRecordList: nodeRecordListUpdate, Size = 3,
rosterInstanceNumber=1, nodesAdded=True, nodesRemoved=False

CLIENTXP01CLIENTXP02netmeetingnetmeeting: GCC
rosterUpdateIndication fullRefresh=True, NodeInformation:
nodeRecordList: nodeRecordListRefresh, 1, rosterInstanceNumber=2,
nodesAdded=True, nodesRemoved=False

CLIENTXP01CLIENTXP02netmeetingnetmeeting: GCC
conferenceUnlockIndication
```

- GCC makes a centralized registry facility available to applications in order to identify dynamically assigned channels and tokens.

```
CLIENTXP02CLIENTXP01netmeetingnetmeeting: GCC
registryRegisterChannelRequest EntityID = 1, channelID = 33766

CLIENTXP02CLIENTXP01netmeetingnetmeeting: GCC
registryRegisterChannelRequest EntityID = 1, channelID = 33767
```

- GCC makes a centralized registry facility available to applications in order to identify dynamically assigned channels and tokens.

```
CLIENTXP01CLIENTXP02netmeetingnetmeeting: GCC registryResponse
registerChannel, channelID = 33762

CLIENTXP01CLIENTXP02netmeetingnetmeeting: GCC registryResponse
registerChannel, channelID = 33761

CLIENTXP02CLIENTXP01netmeetingnetmeeting: GCC
registryAllocateHandleRequest EntityID = 2, HandleNumber = 1

CLIENTXP02CLIENTXP01netmeetingnetmeeting: GCC
registryAssignTokenRequest EntityID = 1
```

- Multiple applications can be running on any given node and can be dynamically launched, used, and shut down during a meeting. As part of the management, role peer GCCs exchange application information and capabilities, which can be used to maintain the local application information database.

```
CLIENTXP02CLIENTXP01netmeetingnetmeeting: AppShare S20Join,
Length=235

CLIENTXP01CLIENTXP02netmeetingnetmeeting: GCC registryResponse
assignToken, tokenID = 16384
```

- A late joiner requests to join the workset group channel by broadcasting the OMNET_HELLO message and waiting for replies. The late joiner instance sends a message to the domain announcing its arrival, and receives one or more replies from the current domain instances.

```
CLIENTXP02CLIENTXP01netmeetingnetmeeting: AppShare -
Object Manager, MessageType = OMNET_HELLO (0x000A), Sender = 33764,
Length=15

CLIENTXP01CLIENTXP02netmeetingnetmeeting: AppShare -
Object Manager, MessageType = OMNET_WELCOME (0x000B), Sender = 33759,
Length=15
```

- The OMNET_OBJECT_CATCHUP message contains the OMNET_OBJECT_ADD message format and specifies extra fields for the position-, replace-, and update- stamps. Except for deleted workset objects, the OMNET_OBJECT_CATCHUP message is followed by a data packet containing a workset object:

```
CLIENTXP01CLIENTXP02netmeetingnetmeeting: AppShare -
Object Manager, MessageType = OMNET_OBJECT_ADD (0x0032),
Sender = 33759, Length=99
```

```
CLIENTXP01CLIENTXP02netmeetingnetmeeting: AppShare -
Object Manager, MessageType = OMNET_WORKSET_CATCHUP (0x0030),
Sender = 33759, Length=27
```

- An Object Manager 'helper' instance sends an OMNET_WSGROUP_SEND_MIDWAY message to advise an Object Manager 'late joiner' instance that its list of work sets currently in use is complete:

```
CLIENTXP01CLIENTXP02netmeetingnetmeeting: AppShare -
Object Manager, MessageType = OMNET_WSGROUP_SEND_MIDWAY (0x001F),
Sender = 33759, Length=23
```

- An Object Manager 'helper' instance sends an OMNET_WSGROUP_SEND_COMPLETE message to notify an Object Manager 'late joiner' instance that it has received a complete copy of the workset group contents

```
CLIENTXP01CLIENTXP02netmeetingnetmeeting: AppShare -
Object Manager, MessageType = OMNET_WSGROUP_SEND_COMPLETE (0x0020),
 Sender = 33759, Length=23
```

- An Object Manager instance broadcasts an OMNET_OBJECT_UPDATE message on the workset group channel to update an object in a workset list with new object data sent in an attached data packet:

```
CLIENTXP02CLIENTXP01netmeetingnetmeeting: AppShare -
Object Manager, MessageType = OMNET_OBJECT_UPDATE (0x0035),
Sender = 33764, Length=43
```

5 Digital Rights Management Services Protocols

This section provides an overview of the Digital Rights Management (DRM) services protocols.

5.1 Digital Rights Management Overview

The Digital Rights Management protocols can be used to access and generate license requests relating to digital rights management.

5.1.1 Protocol List

The Digital Rights Management Services protocols include two sets of protocols. The core protocol provides the DRM services functionality. A second set of protocols is required for an implementation of the core protocol (as listed in the Normative References section of the core protocols technical specifications) to provide core networking functionality.

The following table lists the core protocol included in Digital Rights Management Services. This list does not include data structures, file structures, or algorithms that these core protocols depend on. The details for these technical dependencies are documented in [MS-DRM], the technical specification for the core protocol.

Protocol name	Description	Short name
Windows Media Digital Rights Management (WMDRM)	The Windows Media Digital Rights Management License Acquisition Protocol provides secure distribution, promotion and sale of digital media content	[MS-DRM]

5.2 Digital Rights Management Functionality

This section provides background for the use of the Windows Media Digital Rights Management (WMDRM) protocol. It does not cover the supporting licensed technologies that are required to implement the Digital Rights Management Services. Implementation of the Digital Rights Management provides and protects digital content used by the Windows client, Windows Media Player, or similar digital system for using protected digital content.

Section 5.4 in this document, provides examples that illustrates the interaction of some of the protocols in two sample configurations.

5.2.1 WINDRM Protocol Overview

The WMDRM protocol provides secure distribution, promotion, and sale of digital media content. The protocol that is used to acquire licenses for Microsoft Windows Media comes in three versions: WMDRM version 1, WMDRM version 7, and WMDRM version 11 technologies, as described below.

5.2.1.1 WMDRM Version 1

With WMDRM Version 1, the client-generated license request is sent to the rights management server as an HTTP GET request. The server responds by returning a web page. This web page contains either an ActiveX control, or a Netscape plug-in that stores the license granted by the server on the requesting client.

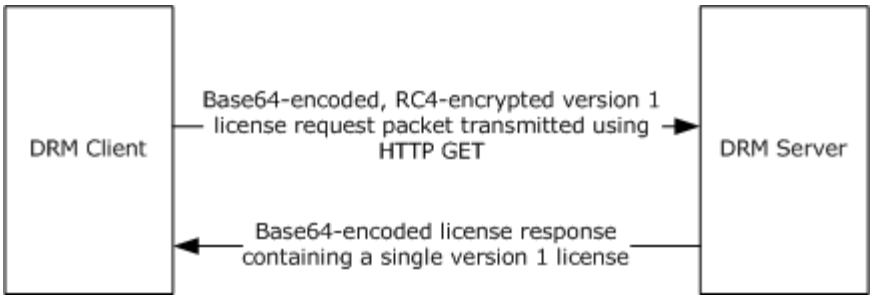

Figure 33: MDRM Version 1 license request and response.

5.2.1.2 WMDRM Version 7

With WMDRM Version 7 of the protocol, the client license request is a packet that contains the license request in extensible markup language (XML) format. This packet is sent as an HTTP POST request. The server response, also in XML form, may contain any number of licenses in either version 1 or version 7 formats.

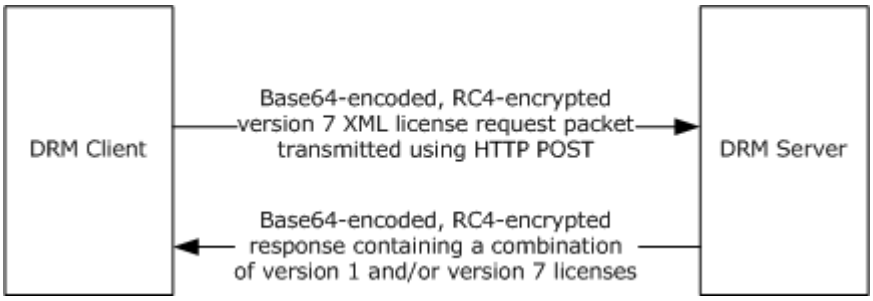

Figure 34: WMDRM Version 7 license request and response.

5.2.1.3 WMDRM Version 11

Version 11 of the protocol is functionally identical to version 7, and includes additional XML fields in the license request challenge body.

5.3 WMDRM Protocol logical dependencies and stack view

This section describes how the DRM Protocol and the protocols that are needed to implement it depend on each other.

5.3.1 Logical Relationships

DRM uses the industry standard protocols:

- HTTP
- HTTPS
- XML
- XSD

The WMDRM protocol versions use HTTP or HTTPS to communicate its packets between the server providing the license content and the client requesting the content.

5.3.2 Stack view

The following diagrams shows the transport stack used by RMS.

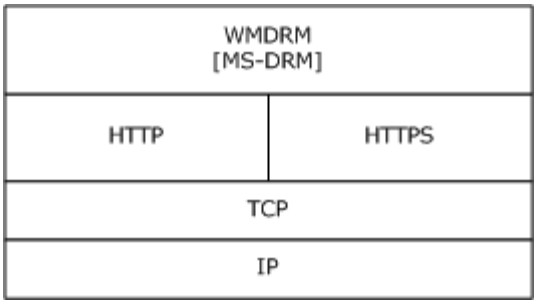

Figure 35: Protocol Stack

WMDRM defines the payload of the data packets being transmitted over HTTP/HTTPS for the purpose of using licensed content.

Implementations of WMDRM include solutions that acquire licenses for authorized content. The implementation scenarios in the following section illustrate two scenarios where WMDRM is used.

5.4 Implementation Scenarios

This section will describe two scenarios, that illustrates common functionality that can be implemented with the protocols included in Digital Rights Management Services:

1. Consuming licensed content.
2. Attempt to use licensed content on unauthorized client.

5.4.1 Scenario 1: Licensing Digitally Protected Content

5.4.1.1 Scenario Overview

In this scenario the client browses online, purchases and downloads a DRM-protected content. Licensed client then opens DRM-protected content.

The actions taken in this scenario describe the overall process by which licensed content is acquired from a commercial provider of protected digital content and used by a Windows Vista client. The scenarios show the network traffic results that pertain to DRM.

5.4.1.2 Scenario Configuration

The following machine configurations were used for this scenario:

ClientVista01

- Machine name: ClientVista01
- Operating system: Windows Vista operating system Ultimate with latest updates

- IP Address: 192.168.131.65
- Subnet Mask: 255.255.255.0

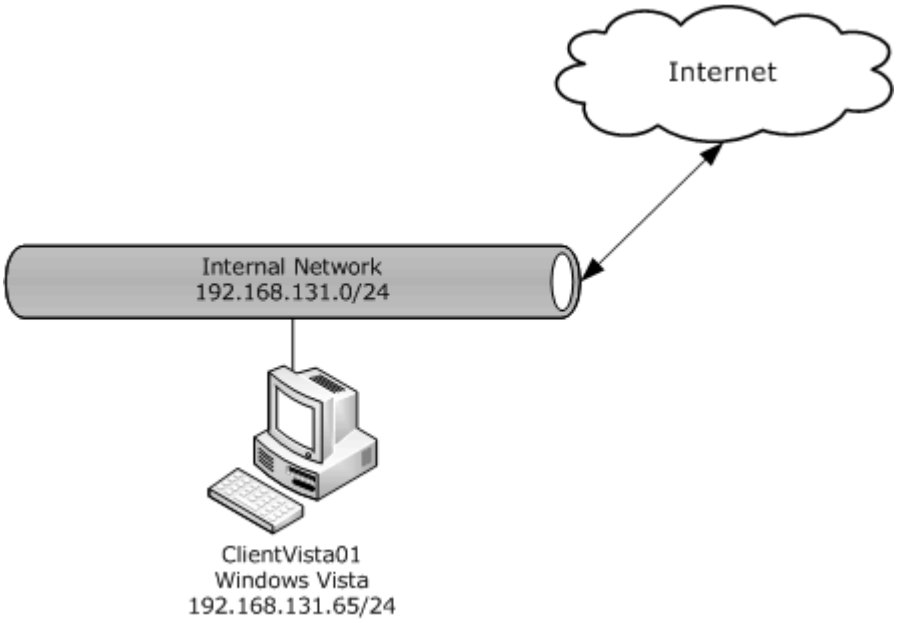

Figure 36: Network topology

5.4.1.3 Setting Up the Trace

1. Start ClientVista01.
2. Log onto URGE.
   - Start Windows Media Player 11 and configure "Express Settings" when prompted.
   - On the toolbar, click URGE and install the feature.
   - On the toolbar, click "Sign In".
   - Enter your logon credentials. If you do not have an URGE account, click "Create New Account" and follow the steps to create a new account.

     Note The free 14-day trial was used for this trace.
3. Close all windows.
4. BEGIN TRACE.
   - Start Windows Media Player.
   - Click URGE.
   - Click Sign In.
   - Enter Logon credentials.

- Download the first song in the top songs category. In this case, the song "Beautiful Girls by Sean Kingston" is used.
- When the download is complete, close Windows Media Player.
- Open the folder where the song was saved, play the song.
- The song should play successfully. Let it play for 15 seconds.

5. END TRACE.
6. Copy the downloaded song to ClientVista02 desktop for next scenario.

5.4.1.4 Netmon Trace Digest

- Because a commercial source for digital content was used, this trace uses the URGE service included with the Windows Media Player version 11. Initial communication in the trace, as shown by the following packets, is done over HTTP.

```
192.168.131.65redir.metaservices.windowsmedia.com.akadns.netHTTP
HTTP: Request, GET /redir/allservices/

redir.metaservices.windowsmedia.com.akadns.netClientVista01HTTP
HTTP: Response, HTTP/1.1, Status Code = 302
```

- Searching for the file to acquire is done via HTTP.

```
ClientVista0164.62.193.126HTTPHTTP: Request,
GET /sitewide/img/content/cta/o.t.png

ClientVista0163.236.41.146HTTPHTTP: Request,
GET /shared/dms/assets/playlist_images/m/must_have_jazz_organ_funk/
main/92x52.jpg

ClientVista0164.62.193.126HTTPHTTP: Request,
GET /sitewide/img/content/cta/o.tr.png
```

- Authentication for the digital content is performed using SSL with the CA.

```
ClientVista01e594.r.akamaiedge.netTCPTCP:
Flags=....A..., SrcPort=49211, DstPort=HTTPS(443), Len=0,
Seq=3589901703, Ack=3348211, Win=65392 (scale factor not found)

ClientVista01e594.r.akamaiedge.netSSLSSL:   Client Key Exchange.
Change Cipher Spec. Encrypted Handshake Message.

e594.r.akamaiedge.netClientVista01SSLSSL:   Change Cipher Spec.
Certificate Verify.
```

- The client requests a DRM license using an HTTP POST, and the server responds with a license.

```
ClientVista01redir.metaservices.windowsmedia.com.akadns.netHTTP
```

```
HTTP: POST

redir.metaservices.windowsmedia.com.akadns.netClientVista01HTTP
HTTP: Response, HTTP/1.1
```

- A significant amount of traffic passes between the client computer and the digital media source computer. The key portion of the traffic is the DRM communication that identifies the DRM version being used.

```
app-sav.musicnet.comClientVista01DRMDRM: DRM Version 7,
License Response - DRM encoded
```

- These few packets are the only traffic in the trace that identifies as explicit DRM traffic.

5.4.2 Scenario 2: Attempt to Use Licensed Content on Unauthorized Client

5.4.2.1 Scenario Overview

In this scenario, the client computer attempts to access digital content not licensed for use on it.

5.4.2.2 Scenario Configuration

The following machine configuration is used for this scenario:

ClientVista02

- Machine name: ClientVista02
- Operating system: Windows Vista operating system Ultimate with latest updates
- IP Address: 192.168.131.66
- Subnet Mask: 255.255.255.0

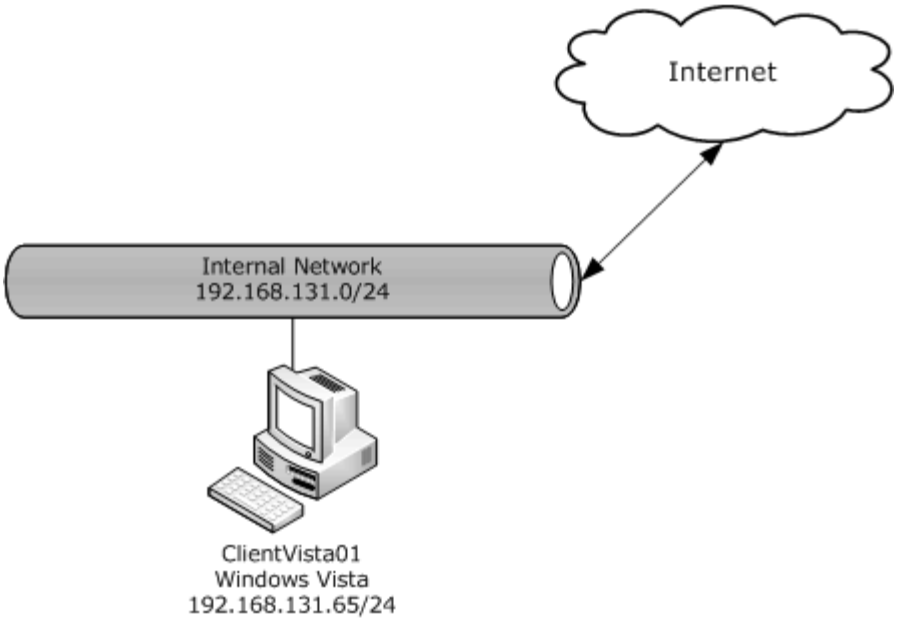

Figure 37: Network topology

5.4.2.3 Setting Up the Trace

1. Start ClientVista02.
2. Start Windows Media Player 11 and configure "Express Settings" when prompted.
3. Close all windows.
4. BEGIN TRACE.
   - on the desktop, double-click the song.
   - A Media Usage Rights Acquisition dialog box appears. Click Cancel.
5. END TRACE.

5.4.2.4 Netmon Trace Digest

With an unlicensed client, the client attempts to get DRM authentication repeatedly from the content provider, using HTTP, as shown in the following packet traces. As there is no authentication provided, no DRM information is passed back to the client.

```
ClientVista0267.131.237.23HTTPHTTP: Request, POST /getlicense/

67.131.237.23ClientVista02HTTPHTTP: Response, HTTP/1.1,
Status Code = 200

ClientVista02206.220.43.92HTTPHTTP: Request, POST /getlicense/

206.220.43.92ClientVista02HTTPHTTP: Response, HTTP/1.1,
Status Code = 301
```

```
ClientVista0267.131.237.23HTTPHTTP: Request, GET /getlicense/

67.131.237.23ClientVista02HTTPHTTP: Response, HTTP/1.1,
Status Code = 200
```

6 File Services Protocols

This section provides an overview of the File Services protocols. It provides a high-level conceptual overview of the core File Services protocols, classifying them broadly according to how the credentials are passed over a Windows network.

6.1 File Services Overview

These protocols can be used to communicate data packets from data stored on storage media on other network-attached computers.

These protocols do not include those related to underlying data transport protocol, nor do they include the protocols associated with user authentication. It is presumed that the appropriate mechanisms and protocols have already authenticated the requested file server access.

6.1.1 Protocols List

The File Services protocols include two sets of protocols: the core protocols that provide the file services functionality and a set of protocols that are required for an implementation of the core protocols (as listed in the Normative References section of the core protocols technical specifications) or provide core networking functionality. The latter set of protocols is referred to as the Networking Protocols Group as described in the "Overview" section of this document.

The following table lists the core protocols included in File Services. This list does not include data structures, file structures, or algorithms that these core protocols depend on. The details for these technical dependencies are documented in the technical specifications for each core protocol.

Protocol Name	Description	Document Short Name
Common Internet File System (CIFS) Browser Protocol	Enables interaction with the browser service that creates and maintains a view of resources that are available on a network.	[MS-BRWS]
Disk Management Remote Protocol	Enables remote management of disks on a computer running Windows Server 2003 operating system, Windows XP operating system, or Windows 2000 operating system.	[MS-DMRP]
Distributed Component Object Model (DCOM) Remote Protocol	Enables a client object to call the methods of a remote COM object over a network. DCOM also monitors connection integrity between server objects and clients.	[MS-DCOM]
Encrypting File System Remote (EFSRPC) Protocol	Enables performing maintenance and management operations on encrypted data that is stored remotely and accessed over a network.	[MS-EFSR]
Server Message Block (SMB) Protocol	Requests file and print services from server systems over a network.	[MS-SMB]
Server Message Block (SMB) Version 2 Protocol	Update of SMB 1.0 for Windows Vista operating system, Windows Server 2008 operating system, Windows 7 operating system, and Windows Server 2008 R2 operating system.	[MS-SMB2]
Server Service Remote	Provides the capability to manage file and print serving	[MS-SRVS]

Protocol Name	Description	Document Short Name
Protocol	resources, and responds to requests made by other computers for shared resources on the local computer.	
Content Indexing Services Protocol	Provides methods of querying and managing index catalogs using a named pipe.	[MS-MCIS]
Distributed File System (DFS): Namespace Management Protocol.	Maps clients logical file requests to server physical files.	[MS-DFSNM]
Removable Storage Manager (RSM) Remote Protocol	Enables remote management of removable media libraries on Windows 2000 and Windows XP.	[MS-RSMP]
Virtual Disk Service (VDS) Remote Protocol	Enables remote management of disks on a computer running Windows Server 2008.	[MS-VDS]
Web Distributed Authoring and Versioning (WebDAV) Protocol: Client Extensions	Provides file access and content management over the Internet, enabling an Internet-based file system.	[MS-WDV]

The following sections provide general functionality followed by a specific overview of each protocol and a description of the logical dependencies among the protocols that provide services in those areas.

6.2 File Services Functionality

This section provides a general overview of how the core protocols operate as a group to provide File Services functionality, followed by an individual view of each core protocol. The "Implementation Scenarios" section later in this document provides examples that illustrate the interaction of some of the protocols in two sample configurations.

The File Services includes the following key components:

- Distributed File System (DFS) Namespace: This component links together shared folders on different servers to create a hierarchical structure that behaves like a single shared folder. Users can navigate the namespace without having to know the physical server names or shared folders that host the data. DFS also provides increased availability, storage scalability, load sharing, and simplified maintenance.
  - Disk Quotas: Windows Server 2003 operating system provides disk-quota functionality that tracks quotas on a per-user, per-volume basis. After a system administrator enables the warning level and limit, these settings apply to all users who own files stored on the volume. Any user who creates a new file on that volume is automatically assigned the current warning level and limit.
  - File Replication Service (FRS): This component is a multi-master replication service used to replicate files and folders in the SYSVOL shared folder on domain controllers and in DFS and non-DFS shared folders. FRS works by detecting changes to files and folders in a replica set, and then replicating those changes to other replica members, which are connected in a replication topology. FRS is a multi-master replication service, so any member of a replica set can generate changes. In addition, FRS can resolve file and folder conflicts to make data consistent among the replica members.

- Shadow Copies for Shared Folders: This component provides point-in-time copies of files that are located on file servers. With Shadow Copies for Shared Folders, users can quickly recover deleted or changed files stored on the network without administrator assistance, which increases productivity and reduces administrative costs.
- World Wide Web Distributed Authoring and Versioning (WebDAV)—This component is the Internet Engineering Task Force (IETF) standard (see RFC 2518 and related documents) for collaborative authoring on the web. The standard defines a set of extensions to Hypertext Transfer Protocol (HTTP) that are designed to facilitate collaborative file management and editing between remote users connected by the Internet.

The following subsections describe the purpose of each individual protocol.

6.2.1 Disk Management Remote Protocol

Enables remote management of disks on a computer running Windows 2000 operating system, Windows XP operating system, or Windows Server 2003 operating system.

The interfaces of the Disk Management Remote Protocol are Distributed Component Object Model (DCOM) interfaces that use DCOM structures. For more information, see [MS-DCOM].

6.2.2 Virtual Disk Service Protocol

The Virtual Disk Service Protocol enables remote management of disks on computers running Microsoft Windows Server 2008 operating system. The interfaces of the VDS Protocol are DCOM interfaces that use DCOM structures. For more information, see [MS-DCOM].

6.2.3 Encrypting File System Remote Protocol

The Encrypting File System Remote Protocol as specified in [MS-EFSR], is a Microsoft protocol that is used for performing maintenance and management operations on encrypted data that is stored remotely and accessed over a network. It is used in Microsoft Windows to manage files that reside on remote file servers and are encrypted using the Encrypting File System (EFS).

6.2.4 Distributed Component Object Model Remote Protocol

The Distributed Component Object Model (DCOM) Remote Protocol as specified in [MS-DCOM], is a protocol for exposing application objects by way of remote procedure calls (RPCs). The protocol consists of a set of extensions layered on Microsoft Remote Procedure Call Protocol Extensions as specified in [MS-RPCE].

6.2.5 Server Message Block Version 1.0 Protocol

The Server Message Block (SMB) Version 1.0 Protocol defines extensions to the existing Common Internet File System [MS-CIFS] specification that have been implemented by Microsoft since the publication of the [MS-CIFS] specification. It also defines Windows behavior with respect to optional behavior in the base specification.

6.2.6 Server Message Block Version 2 Protocol

The Server Message Block (SMB) Version 2 Protocol was introduced with Windows Vista operating system and is a redesign of the older SMB 1.0 protocol. It adds larger type capabilities and a fixed header size to the SMB messages.

Note When this document references SMB without specifying a version, the content applies to SMB Version 1 and SMB Version 2.

6.2.7 Server Service Remote Protocol

The Server Service Remote Protocol is a RPC–based protocol that is used for remotely enabling file and printer sharing and named pipe access to the server through the Server Message Block (SMB) Protocol, as specified in [MS-SMB]. The protocol is also used for remote administration of Windows servers.

6.2.8 Browser Remote Protocol

The Browser Remote Protocol is the Microsoft implementation of the Common Internet File System (CIFS) Browser Protocol as specified in [MS-BRWS].

6.2.9 Microsoft Content Indexing Services

This protocol provides a methodology for managing and querying the Microsoft Content Indexing Services over named pipes.

6.2.10 Microsoft Distributed File System

The Microsoft Distributed File System (DFS): Referral Protocol lets file system clients resolve names on specific file servers. After names have been resolved, clients can directly access files on the identified servers using file system protocols such as SMB (as specified in [MS-SMB]), NFS (as specified in [RFC3530]), and NCP (as specified in Novell's Guide to Netware LAN Analysis, 2nd Edition).

6.2.11 Removable Storage Manager Remote Protocol

Removable Storage Manager Remote Protocol, as specified in [MS-RSMP], enables data management client applications to access removable media in automated libraries, including calling for media to be moved from storage slots to drives on Windows 2000 operating system and Windows XP operating system.

6.2.12 World Wide Web Distributed Authoring and Versioning (WebDAV) Protocol Extensions

The client extensions in the WebDAV Protocol: Client extensions extend WebDAV as specified in [RFC2518] by introducing the new headers that both enable the file types that are not currently manageable and optimize protocol interactions for client file systems. This allows client computers to transparently access files stored on web servers using the same user interface that would apply if those files were stored on a local computer or network. This transparency gives the Windows client computer end-user access to the web server-stored data resources with the need to use any additional tools or applications; the Windows operating system presents the resources to the user using the same interface conventions the user expects with any stored data file.

6.3 File Server Logical Dependencies and Protocol Stack Views

This section provides several conceptual file server areas to describe the logical dependencies among the file server protocols and protocol stack views as appropriate.

6.3.1 Logical Relationships

Basic file services include two remote file access protocols and a number of supporting or related protocols. Very broadly, the protocols are composed as depicted in the following figure.

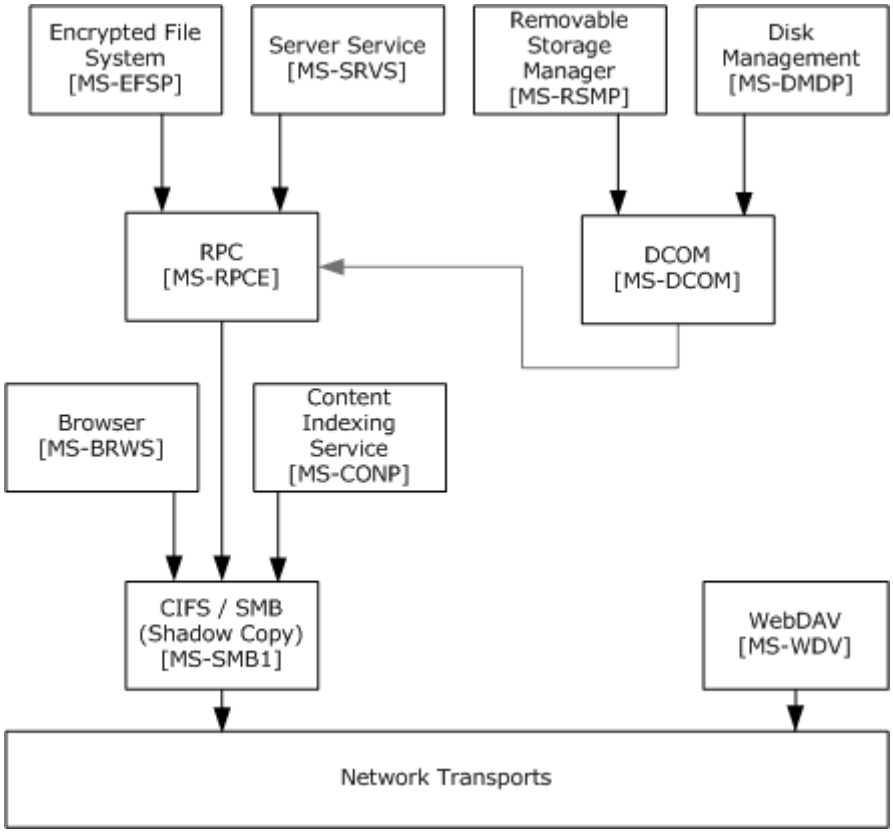

Figure 38: Basic file services protocols

In this diagram, the WebDAV Protocol: Client Extensions appear on the side because they are independent of the other protocols. The CIFS protocol, as specified in [MS-CIFS], and the SMB Protocol, as specified in [MS-SMB], are the more interesting protocols because of their relationship to other protocols.

6.3.2 Stack View

The client for an SMB or CIFS connection could be connecting to the server over a variety of transports. After the client is connected, it may make several related requests. The next diagrams show the protocol stack that is used by CIFS and SMB. The first diagram is the lower part of the stack because it shows the relationship of CIFS and SMB to the underlying network transports.

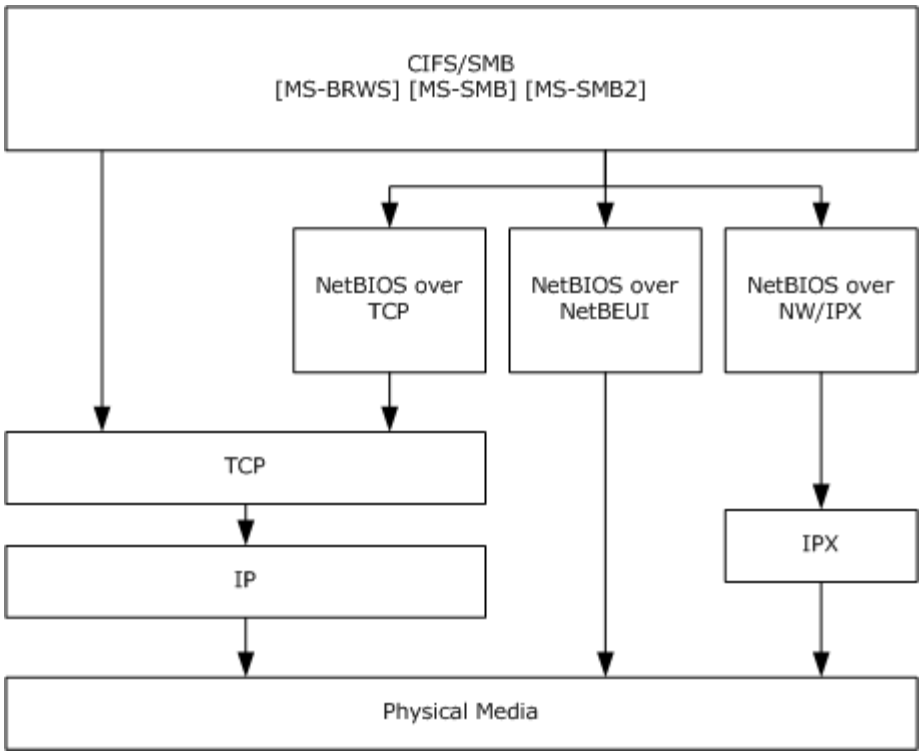

Figure 39: Basic file services protocol stack

The CIFS/SMB Protocol, can use either TCP natively, as specified in [RFC793], through port 445, or a NetBIOS transport, as specified in [RFC1088]. NetBIOS, in turn, is implemented as NetBIOS over another transport, either TCP (as specified in [RFC1001]), NetBEUI (as specified in [NETBEUI]), or Netware/IPX (as specified in [IPX]). Those underlying layers are responsible for the actual transmission over the physical medium. The encapsulation for NetBIOS over TCP is specified in [MS-SMB]; for Netware/IPX in [MS-CIFS]; and for NetBEUI in [MS-CIFS].

The next diagram shows the higher protocols, which rely on the CIFS Protocol and the SMB Protocol for their transport or rely on other protocols that are shown in the following figure.

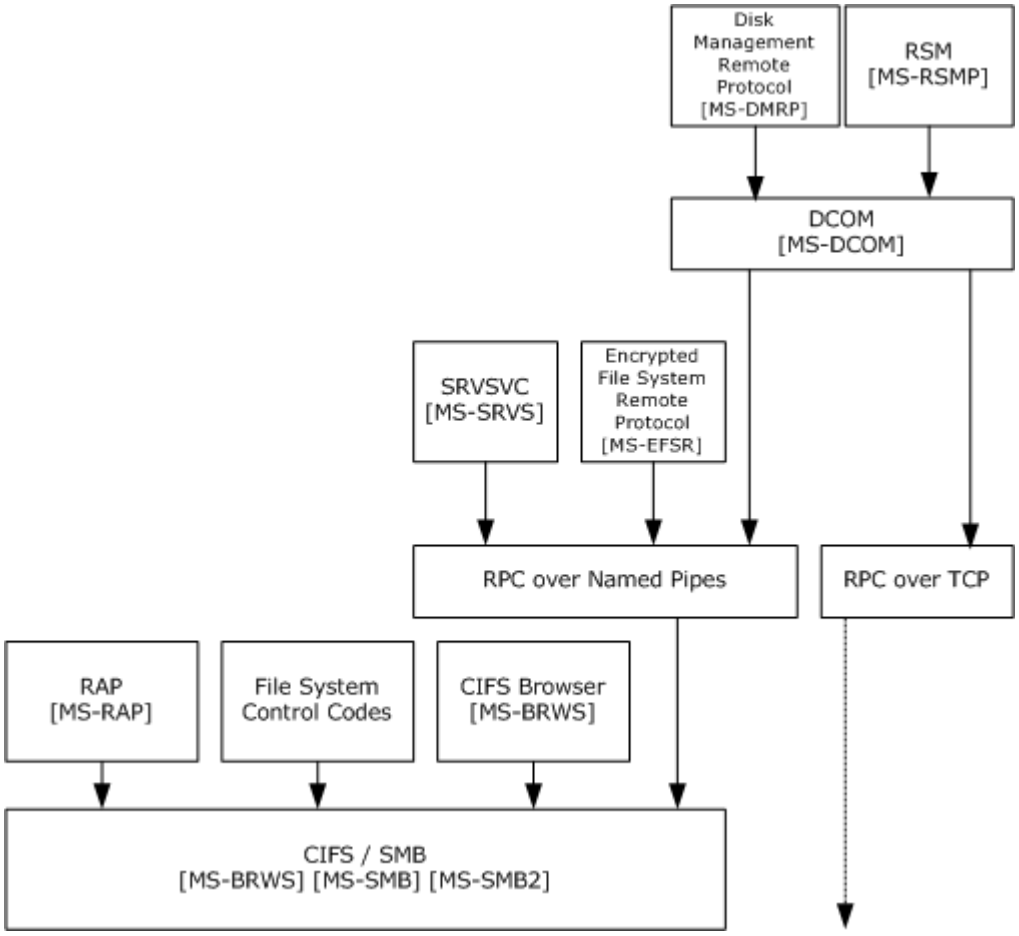

Figure 40: Protocols that depend on CIFS/SMB

Above the CIFS/SMB block are protocols that extend the functionality of the file server beyond that of simple file sharing. The File System Control Codes, as specified in [MS-FSCC], are composed of the structured commands that can be sent to a device or file system that has been accessed through the CIFS Protocol and the SMB Protocol. These are layered on top the CIFS protocol, because they cannot be used or even expressed outside a CIFS/SMB connection.

Additionally, the Server Service RPC interface, as specified in [MS-SRVS], is used to query the server for additional information, such as the named share points that can be used for connections and similar administrative information. Likewise, legacy clients (Microsoft DOS and Windows 95 operating system, Microsoft Windows 98 operating system, and Windows Millennium Edition operating system) use the Remote Administration Protocol, as specified in [MS-RAP], for similar purposes. The Remote Access Protocol, as specified in [MS-RAP], is expressed within the SMB Protocol, through the Transact command.

The Server Service RPC interface, however, is an RPC interface. The Server Service RPC interface is only accessible over an RPC named pipes binding. Named pipes are an abstraction that are offered by the CIFS Protocol and the SMB Protocol. The RPC layer performs the necessary encoding and decoding of the commands within the Server Service RPC interface.

Similarly, the Encrypting File System Remote (EFSRPC) Protocol, as specified in [MS-EFSR], also layers on top of a named pipe endpoint, and takes advantage of the RPC runtime for encoding and decoding.

6.3.3 Logical Dependencies

Unlike the transports and CIFS/SMB upper-layer protocols that were shown in the previous diagrams, other protocols used to implement a file server do not appear in neat layers. The following diagram outlines the authentication and security protocols that are used by the CIFS/SMB Protocol in order to authenticate a client. The security protocols are not layered; but instead are contained in the CIFS/SMB Protocol as specified in [MS-SMB].

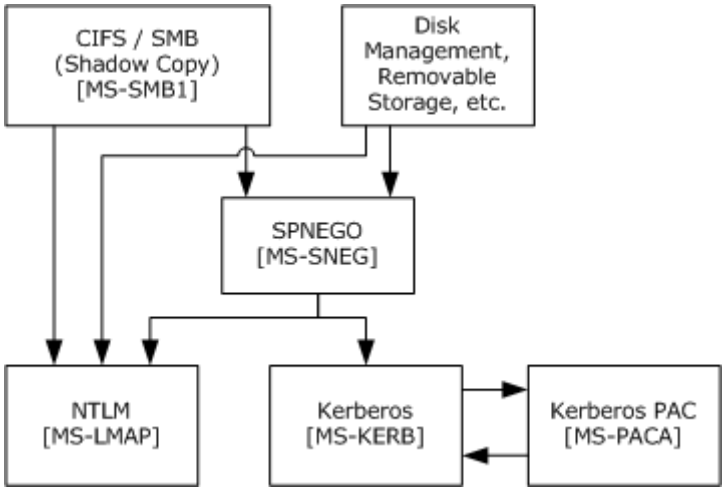

Figure 41: Basic file services protocol dependencies

As the preceding diagram shows, CIFS/SMB can use the SPNEGO Protocol Extensions (as specified in [MS-SPNG]) as its authentication scheme; or it can directly use NTLM (as specified in [MS-NLMP]). In the CIFS/SMB specifications, the use of extended security for the SPNEGO Protocol Extensions or "legacy" NTLM is fully discussed, as specified in [MS-SMB]. For purposes of this document, it is sufficient to note that NTLM is used directly by the CIFS Protocol and the SMB Protocol primarily for supporting legacy clients. Note also that in this mode, NTLM is not exchanged as whole NTLM messages, but is actually "shredded" so that only the challenge/response fields are sent; this is specified fully in the NTLM documentation, as specified in [MS-NLMP].

With extended security, the CIFS/SMB Protocol uses the SPNEGO Protocol Extensions to select an authentication protocol. In this case, the Kerberos protocol, as specified in [MS-KILE], is typically chosen. However, there are scenarios where a downgrade to NTLM is acceptable, such as when attempting to authenticate a legacy client that does not support Kerberos.

Additional information about authentication protocols is available in the Authentication Services section of this document, and in the individual authentication protocol specifications.

The two implementation scenarios in the following section are representative of the File Services technologies and protocols.

6.4 Implementation Scenarios

This section will describe two implementation scenarios that illustrate common functionality that can be implemented with the protocols included in File Services. The functionality illustrated in these scenarios is as follows:

1. Use WebDAV to transfer files from Windows Vista operating system Ultimate client to Windows Server 2003 operating system Enterprise Edition R2 x64.

2. Use the DFS: Namespace Management Protocol to create and populate a DFS namespace on the primary test server.

6.4.1 Scenario 1: WebDAV Test

6.4.1.1 Scenario Overview

This test scenario uses WebDAV to transfer files from a client (Windows Vista operating system Ultimate) to a server (Windows Server 2003 R2 Enterprise x64 Edition operating system).

Section 6.4.1.4 describes the trace seen on the network when the client accesses the files on the server using WebDAV.

6.4.1.2 Scenario Configuration

The following machine configurations were used for this scenario:

Windows Vista operating system Client Configuration

- Machine Name: VistaUltimate
- Operating System: Windows Vista Ultimate with latest updates
- IP Address: 192.168.1.10
- Subnet Mask 255.255.255.0

Windows Server 2003 operating system Configuration

- Machine Name: Win2K3Serverx64EE
- Operating System: Windows Server 2003 R2 Enterprise x64 Edition operating system
- IP Address: 192.168.1.20
- Subnet Mask: 255.255.255.0
- Configured with the following roles:
  - Application Server
  - Internet Information Server

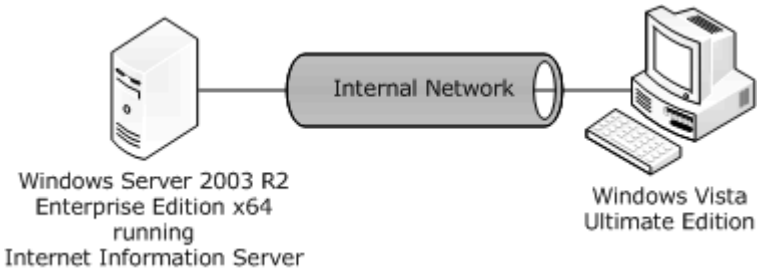

Figure 42: Network topology

6.4.1.3 Setting Up the Trace

These are the required steps for creating the WebDAV trace:

1. Configure the Windows Server operating system to support Internet Information Services (IIS).
2. Configure IIS to allow WebDav access on the web server.
3. Make a test directory available to IIS.
4. Configure a test directory for access via WebDAV.

Network Monitor was run on the server side of the connection. The captured sequence is as follows:

1. BEGIN TRACE.
2. Use Internet Explorer 7 to connect to the WebDAV server directory using the File | Open command and checking the Open as a web folder checkbox.
3. A folder that contains multiple files is dragged from the client desktop to the web folder.
4. END TRACE.

6.4.1.4 Netmon Trace Digest

The initial connection between client and server is made via TCP/IP over HTTP over Port 80. All data transfers in this sequence are also made via HTTP.

```
VistaUltimateW2K3Serverx64EETCPTCP: Flags=.S......,
SrcPort=49978, DstPort=HTTP(80), Len=0, Seq=525715980, Ack=0, Win=8192
 (scale factor 2) = 32768
```

After the server is contacted, WebDAV uses the HTTP 1.1 OPTIONS method to request information about the communication options that are available on the request/response chain.

```
VistaUltimateW2K3Serverx64EEWEBDAVWEBDAV: Request,
OPTIONS /
```

The WebDAV client then uses the WebDAV PROPFIND method, as specified in [RFC2518], to retrieve properties that are defined on the target resource. /TestDir is the target directory on the web server.

```
VistaUltimateW2K3Serverx64EEWEBDAVWEBDAV: Request,
PROPFIND /VistaUltimateW2K3Serverx64EEWEBDAVWEBDAV: Request,
PROPFIND /TestDir
```

The client then attempts to place the selected files in the web folder. The files that are selected for copying are contained in a folder called "TestData". The entire content of the client folder is copied to the server. Because there is no existing folder of that name on the server, the WebDAV MKCOL method, as specified in [RFC2518], is used to create a new resource in the web folder.

```
VistaUltimateW2K3Serverx64EEWEBDAVWEBDAV: Request,
PROPFIND /TestDir/TestData

VistaUltimateW2K3Serverx64EEWEBDAVWEBDAV: Request, MKCOL
/TestDir/TestData
```

Each file that is copied to the server uses the following process:

The WebDAV PROPFIND method is used to determine if the file exists; the HTTP 1.1 PUT method is then used to copy the file to the server.

```
VistaUltimateW2K3Serverx64EEWEBDAVWEBDAV: Request,
PROPFIND /TestDir/ TestData/TestFile1

VistaUltimateW2K3Serverx64EEWEBDAVWEBDAV: Request,
PUT /TestDir /TestData/TestFile1
```

Responsibly coded WebDAV access uses the LOCK and UNLOCK methods, as necessary.

```
VistaUltimateW2K3Serverx64EEWEBDAVWEBDAV: Request,
UNLOCK /TestDir/TestData/TestFile3.PDF
```

This process is repeated until all the selected data is copied to the WebDAV-enabled folder on the IIS site. There is significant TCP traffic on Port 80 by using HTTP to move the file data while this occurs. This process continues until all the files are transferred via WebDAV and TCP.

6.4.2 Scenario 2: DFS Test

6.4.2.1 Scenario Overview

In this test scenario, a DFS namespace is created on the primary test server. The namespace is populated with shared folders from the primary and secondary test servers.

6.4.2.2 Scenario Configuration for DFS Test

DFSServerHost Configuration

- Machine Name: DFSServerHost
- Operating System: Windows Server 2003 R2 Enterprise x64 Edition operating system with all updates
- IP Address: 192.168.1.25

- Subnet Mask: 255.255.255.0
- Configured with the following roles:
  - Application Server
  - File Server

DFSServerMember Configuration

- Machine Name: DFSServerMember
- Operating System: Windows Server 2003 R2 Enterprise Edition operating system with all updates.
- IP Address: 192.168.1.25
- Subnet Mask: 255.255.255.0
- Configured with the following roles:
  - Application Server
  - File Server

VistaClient Configuration

- Machine Name: VistaClient
- Operating System: Windows Vista operating system Ultimate with latest updates
- IP Address: 192.168.1.10
- Subnet Mask 255.255.255.0

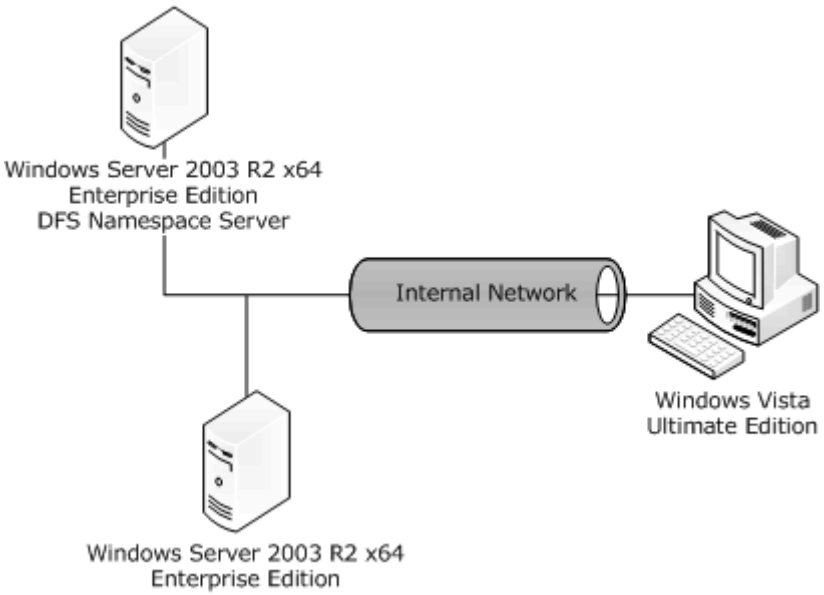

Figure 43: Network topology

6.4.2.3 Setting Up the Trace

These steps produce the DFS trace:

1. BEGIN TRACE.
2. From the Windows Vista operating system client, browse the network, resolving a share called DFStest hosted by the DFS namespace server (DFSServerHost).
3. From the client, copy a local file named DFSTestShare to a folder found in the DFStest share. This shared folder physically resides on a different server form the namespace server (DFSServerMember).
4. END TRACE.

6.4.2.4 Netmon Trace Digest

As shown in the figure, the Windows Vista operating system-based DFS client can make use of the client-to-server referral protocol to get information about the DFS share and select where in the share it chooses to communicate. Regardless of the physical location of the shared data, it will resolve to the DFS client as being in the DFS namespace resolved directory, not in the actual physical directory where the data resides on the network.

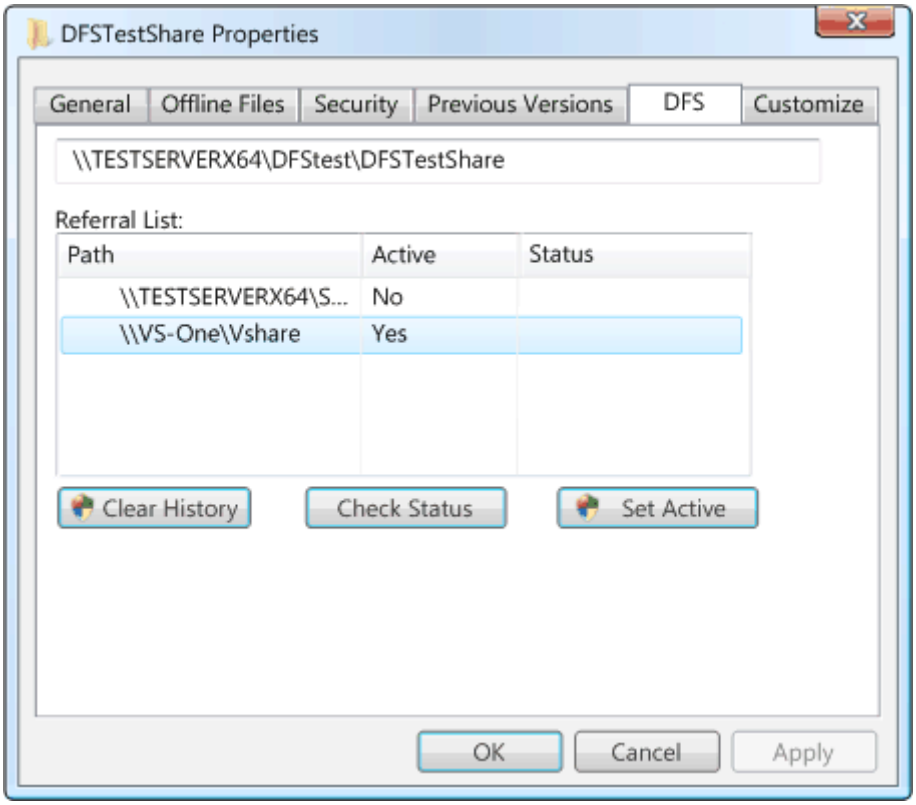

Figure 44: The DFS Share properties dialog box from Windows Vista

In the following packet capture fragment, note that the Windows Vista-based client computer resolves the share name via the DFSServerHost, which identifies the server where the namespace resides. Data transfer then takes place directly between the client and the server where the physical

share resides (DFSServerMember). The data file does not, at any time, pass through the namespace server during this process.

```
VistaClientDFSServerHostSRVSVCSRVSVC: NetrShareGetInfo
 Request,ServerName=TESTSERVERX64 NetName=DFStest Level=1

DFSServerHostVistaClientSRVSVCSRVSVC: NetrShareGetInfo Response,
Level=1, Status = ERROR_SUCCESS

VistaClientDFSServerHostSMBSMB: C; Close, FID = 0x400A
(\srvsvc@#43)

DFSServerHostVistaClientSMBSMB: R; Close, FID = 0x400A
(\srvsvc@#43)

VistaClientDFSServerHostSMBSMB: C; Transact2, Query Path Info,
 Query File Basic Info, Pattern = \TESTSERVERX64\DFStest

DFSServerHostVistaClientSMBSMB: R; Transact2, Query Path Info,
 Query File Basic Info

VistaClientDFSServerHostSMBSMB: C; Transact2, Query Path Info,
Query File Standard Info, Pattern = \TESTSERVERX64\DFStest

DFSServerHostVistaClientSMBSMB: R; Transact2, Query Path Info,
Query File Standard Info

VistaClientDFSServerHostSMBSMB: C; Transact2, Query Path Info,
Query File Basic Info, Pattern = \TESTSERVERX64\DFStest

DFSServerHostVistaClientSMBSMB: R; Transact2, Query Path Info,
Query File Basic Info

VistaClientDFSServerHostSMBSMB: C; Nt Create Andx,
FileName = \TESTSERVERX64\DFStest

DFSServerHostVistaClientSMBSMB: R; Nt Create Andx,
FID = 0x400B (\TESTSERVERX64\DFStest@#61)

VistaClientDFSServerHostSMBSMB: C; Transact2, Find First2,
ID Both Directory Info, Pattern =
\TESTSERVERX64\DFStest\DFSTestShare

DFSServerHostVistaClientSMBSMB: R; Transact2, Find First2,
 ID Both Directory Info

VistaClientDFSServerHostSMBSMB: C; Close, FID = 0x400B
(\TESTSERVERX64\DFStest@#61)

DFSServerHostVistaClientSMBSMB: R; Close, FID = 0x400B
(\TESTSERVERX64\DFStest@#61)

VistaClientDFSServerMemberSMBSMB: C; Nt Create Andx, FileName =

DFSServerMemberVistaClientSMBSMB: R; Nt Create Andx - NT Status:
 System - Error, Code = (53) STATUS_OBJECT_NAME_COLLISION

VistaClientDFSServerHostSMBSMB: C; Transact2, Query Path Info,
Query File Basic Info, Pattern = \TESTSERVERX64\DFStest
```

```
DFSServerHostVistaClientSMBSMB: R; Transact2, Query Path Info,
 Query File Basic Info

VistaClientDFSServerHostSMBSMB: C; Transact2, Query Path Info,
 Query File Standard Info, Pattern = \TESTSERVERX64\DFStest

DFSServerHostVistaClientSMBSMB: R; Transact2, Query Path Info,
 Query File Standard Info

VistaClientDFSServerHostSMBSMB: C; Transact2, Query Path Info,
 Query File Basic Info, Pattern = \TESTSERVERX64\DFStest

DFSServerHostVistaClientSMBSMB: R; Transact2, Query Path Info,
 Query File Basic Info

VistaClientDFSServerHostSMBSMB: C; Transact2, Query Path Info,
 Query File Basic Info, Pattern = \TESTSERVERX64\DFStest

DFSServerHostVistaClientSMBSMB: R; Transact2, Query Path Info,
 Query File Basic Info

VistaClientDFSServerHostSMBSMB: C; Transact2, Query Path Info,
 Query File Standard Info, Pattern = \TESTSERVERX64\DFStest

DFSServerHostVistaClientSMBSMB: R; Transact2, Query Path Info,
Query File Standard Info

VistaClientDFSServerHostSMBSMB: C; Transact2, Query Path Info,
Query File Basic Info, Pattern = \TESTSERVERX64\DFStest

DFSServerHostVistaClientSMBSMB: R; Transact2, Query Path Info,
Query File Basic Info

VistaClientDFSServerHostSMBSMB: C; Transact2, Query FS Info,
 Query FS Size Info (NT)

DFSServerHostVistaClientSMBSMB: R; Transact2, Query FS Info,
 Query FS Size Info (NT)

VistaClientDFSServerMemberSMBSMB: C; Nt Cancel

DFSServerMemberVistaClientSMBSMB: R; Nt Transact,
Unknown NT Transaction Function - NT Status: System - Error,
Code = (288) STATUS_CANCELLED

VistaClientDFSServerMemberSMBSMB: C; Close, FID = 0x400F

DFSServerMemberVistaClientSMBSMB: R; Close, FID = 0x400F

VistaClientDFSServerMemberSMBSMB: C; Transact2, Query Path Info,
 Query File Basic Info, Pattern = \TestFile1.PDF

DFSServerMemberVistaClientSMBSMB: R; Transact2, Query Path Info -
 NT Status: System - Error, Code = (52) STATUS_OBJECT_NAME_NOT_FOUND

VistaClientDFSServerMemberSMBSMB: C; Nt Create Andx,
FileName = \TestFile1.PDF
```

```
DFSServerMemberVistaClientSMBSMB: R; Nt Create Andx - NT Status:
System - Error, Code = (34) STATUS_ACCESS_DENIED

VistaClientDFSServerMemberSMBSMB: C; Transact2, Query Path Info,
 Query File Basic Info, Pattern = \TestFile1.PDF

DFSServerMemberVistaClientSMBSMB: R; Transact2, Query Path Info -
NT Status: System - Error, Code = (52) STATUS_OBJECT_NAME_NOT_FOUND

VistaClientDFSServerMemberSMBSMB: C; Nt Create Andx, FileName =
\TestFile1.PDF

DFSServerMemberVistaClientSMBSMB: R; Nt Create Andx - NT Status:
System - Error, Code = (34) STATUS_ACCESS_DENIED

VistaClientDFSServerMemberSMBSMB: C; Transact2, Query Path Info,
Query File Basic Info, Pattern = \TestFile1.PDF

DFSServerMemberVistaClientSMBSMB: R; Transact2, Query Path Info -
 NT Status: System - Error, Code = (52) STATUS_OBJECT_NAME_NOT_FOUND

VistaClientDFSServerMemberSMBSMB: C; Nt Create Andx,
FileName = \TestFile1.PDF

DFSServerMemberVistaClientSMBSMB: R; Nt Create Andx,
FID = 0x8009 (\TestFile1.PDF@#97)

VistaClientDFSServerMemberSMBSMB: C; Transact2, Query File Info,
 Query File Internal Info, FID = 0x8009 (\TestFile1.PDF@#97)

DFSServerMemberVistaClientSMBSMB: R; Transact2, Query File Info,
 Query File Internal Info, FID = 0x8009 (\TestFile1.PDF@#97)

VistaClientDFSServerMemberSMBSMB: C; Transact2, Query File Info,
 Query File Basic Info, FID = 0x8009 (\TestFile1.PDF@#97)

DFSServerMemberVistaClientSMBSMB: R; Transact2, Query File Info,
 Query File Basic Info, FID = 0x8009 (\TestFile1.PDF@#97)

VistaClientDFSServerMemberSMBSMB: C; Transact2, Query FS Info,
Query FS Volume Info (NT)

DFSServerMemberVistaClientSMBSMB: R; Transact2, Query FS Info,
 Query FS Volume Info (NT), Label =

VistaClientDFSServerMemberSMBSMB: C; Transact2, Query File Info,
 Query File Basic Info, FID = 0x8009 (\TestFile1.PDF@#97)

DFSServerMemberVistaClientSMBSMB: R; Transact2, Query File Info,
 Query File Basic Info, FID = 0x8009 (\TestFile1.PDF@#97)

VistaClientDFSServerMemberSMBSMB: C; Transact2, Query File Info,
 Query File EA Info, FID = 0x8009 (\TestFile1.PDF@#97)

DFSServerMemberVistaClientSMBSMB: R; Transact2, Query File Info,
 Query File EA Info, FID = 0x8009 (\TestFile1.PDF@#97)

VistaClientDFSServerMemberSMBSMB: C; Transact2, Query FS Info,
 Query FS Attribute Info (NT)
```

```
DFSServerMemberVistaClientSMBSMB: R; Transact2, Query FS Info,
 Query FS Attribute Info (NT), FS = NTFS

VistaClientDFSServerMemberSMBSMB: C; Transact2, Query File Info,
 Query File Basic Info, FID = 0x8009 (\TestFile1.PDF@#97)

DFSServerMemberVistaClientSMBSMB: R; Transact2, Query File Info,
 Query File Basic Info, FID = 0x8009 (\TestFile1.PDF@#97)

VistaClientDFSServerMemberSMBSMB: C; Transact2, Query File Info,
 Query File EA Info, FID = 0x8009 (\TestFile1.PDF@#97)

DFSServerMemberVistaClientSMBSMB: R; Transact2, Query File Info,
 Query File EA Info, FID = 0x8009 (\TestFile1.PDF@#97)

VistaClientDFSServerMemberSMBSMB: C; Transact2, Query FS Info,
 Query FS Size Info (NT)

DFSServerMemberVistaClientSMBSMB: R; Transact2, Query FS Info,
 Query FS Size Info (NT)

VistaClientDFSServerMemberSMBSMB: C; Transact2, Set File Info,
 Set File End of File Info, FID = 0x8009 (\TestFile1.PDF@#97)

DFSServerMemberVistaClientSMBSMB: R; Transact2, Set File Info,
 Set File End of File Info, FID = 0x8009 (\TestFile1.PDF@#97)
```

7 Health Certificate Services Protocols

This section provides an overview of the Health Certificate Services protocols. It provides a high-level conceptual view of the core Health Certificate Server protocols, classifying them broadly according to how the credentials are passed over a Windows network.

7.1 Health Certificate Services Overview

Health Certificate Services includes protocols for communicating client health status information, digital certificate requests, and digital certificate issuance or refusal data packets between a client and server.

7.1.1 Protocols List

The Health Certificate Services includes a set of core protocols that provides Health Certificate functionality.

The following table lists the core protocol included in the Health Certificate Services. This list does not include data structures, file structures, or algorithms that the core protocol depends on. The details for these technical dependencies are documented in [MS-HCEP], the technical specification for this core protocol.

Table: Health Certificate Services Protocols

Protocol name	Description	Document Short Name
Health Certificate Enrollment Protocol	Allows a network endpoint to obtain digital certificates that are conditionally issued based on the compliance of that endpoint to security policy that is defined for the network.	[MS-HCEP]

7.2 Health Certificate Services Functionality

Many network administrators who maintain a secure network require that clients accessing their networks comply with policies established for the network. For example, an administrator might require each client that accesses their network to have an active firewall. One way that administrators can ensure compliance of network endpoints is to require that clients enroll for a health certificate. Health certificates summarize the client's compliance to policy in a way that interested parties can view without performing the validation themselves.

A health certificate server is the combination of an HRA and a certification authority (CA). An HRA is a computer running Windows Server 2008 operating system and Internet Information Services (IIS). The CA can be installed on the computer running Windows Server 2008 or it can be installed on a separate computer. The health certificate server obtains health certificates for compliant computers. A health certificate can be used instead of a Statement of Health (SoH) to prove that a client is compliant with system health requirements.

Section 7.4 of this document provides examples that illustrate the interaction of some of the protocols in this task in two sample configurations.

7.2.1 Health Certificate Server Key Components

The Health Certificate Enrollment protocol is a core protocol used by the client to assert its health to the network infrastructure and to get a certificate of good health. This certificate is an X.509 public

key certificate [RFC3280], which can then be used in any authentication protocol that relies upon certificates and requires that the certificate also assert that the client is in "good health" (that is, in a state that conforms to network access policies of the enterprise).

The health certificate is created and issued by a health certificate server, which comprises a Health Registration Authority and a Certificate Authority.

The Health Certificate Enrollment Protocol has authenticated and unauthenticated modes. In the authenticated mode, the Health Certificate Enrollment Protocol supports authentication of the server, client, or both. Authentication of the server in the Health Certificate Enrollment Protocol is achieved by using the Hypertext Transfer Protocol (HTTP) over Transport Layer Security (TLS), as specified in [RFC2818]. During the establishment of the TLS channel, as specified in [RFC2446], the server is authenticated by the client. Authentication of the client in the Health Certificate Enrollment Protocol is achieved by using SPNEGO-based Kerberos and NTLM HTTP authentication, as specified in [RFC4559].

The following subsection describes the purpose of the core protocol.

7.2.2 Health Certificate Enrollment Protocol

The Health Certificate Enrollment Protocol, as specified in [MS-HCEP], is used to do the following:

- Convey information to a Health Registration Authority (HRA) about the state of the client.
- Requests that a health certificate be issued.
- Obtains a response from the HRA that may include a certificate or remediation measures.

On a Windows Server 2008 operating system Server, a typical implementation of the Health Certificate Enrollment Protocol would involve the following configuration tasks:

1. Configure certificate settings.
2. Configure the HRA.
3. Request new certificate for the HRA.
4. Configure the System Health Validator.
5. Configure Health Policies.
6. Configure NAP settings.

To further illustrate these configuration settings, a detailed implementation scenario is provided in section 7.4.

To accomplish this, the Health Certificate Enrollment Protocol uses the HTTP protocol, as specified in [RFC2616]. For the exchanges that contain confidential information, the HTTP protocol runs over TLS, as specified in [RFC2818]. HTTP is the only protocol that the Health Certificate Enrollment Protocol uses directly. HTTP uses TCP as the transport protocol underneath it.

The Health Certificate Enrollment Protocol is invoked via an API by the Quarantine Agent. This protocol is run by a component called the Health Certificate Enrollment Agent, which in turn, communicates over COM with applications that require health certificate enrollment.

On the client side, the protocol stack can be depicted as in the following diagram.

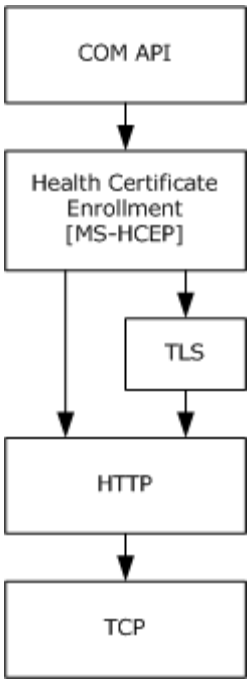

Figure 45: Health Certificate Enrollment Protocol stack

On the server side, the Health Certificate Enrollment Protocol is implemented by the Health Registration Authority (HRA), which communicates with the Internet Authentication Service (IAS) or **network policy server (NPS)** over RADIUS. RADIUS extensions are the Microsoft Vendor-Specific extension to the RADIUS messages that allow the RADIUS protocol to become health-aware. RADIUS is an industry standard protocol used by network access servers to authenticate clients that are accessing the network. The protocol itself runs between the network access server (which from the perspective of the RADIUS protocol is the client) and a back-end authentication server (the server in the RADIUS protocol).

7.2.2.1 Health Certificate Enrollment Protocol Request-Response Model

The Health Certificate Enrollment Protocol is a simple request-response protocol. The Health Certificate Enrollment Protocol allows a network endpoint to obtain digital certificates. These certificates are conditionally issued based on the compliance of that endpoint to security policy defined for the network.

The protocol is always a single Health Certificate Enrollment Protocol request followed by a single Health Certificate Enrollment Protocol response, as shown in the following diagram.

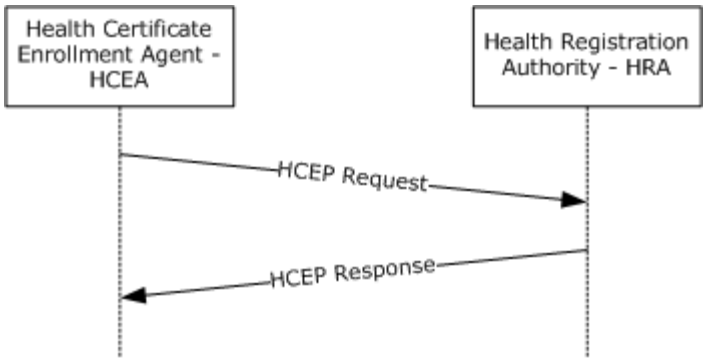

Figure 46: Health Certificate Enrollment Protocol single request and response

The health certificate enrollment agent (HCEA) sends a Health Certificate Enrollment Protocol request, and the HRA responds with a Health Certificate Enrollment Protocol response. The protocol is used by the HCEA to obtain a health certificate based on its compliance to the security policies defined for the network.

7.2.2.2 HCEP Authentication Modes

The Health Certificate Enrollment Protocol has authenticated and unauthenticated modes. In the authenticated mode, the Health Certificate Enrollment Protocol supports authentication of the server, client, or both.

Authentication of the server in the Health Certificate Enrollment Protocol is achieved by using HTTP over Transport Layer Security (TLS), as specified in [RFC2818]. During the establishment of the TLS channel, as specified in [RFC2246], the client authenticates the server.

Authentication of the client in the Health Certificate Enrollment Protocol, as specified in MS-HCEP, is achieved by using SPNEGO-based Kerberos and NTLM HTTP authentication, as specified in [RFC4559].

7.2.2.3 Security Considerations

The health state of the machine sending the Health Certificate Enrollment Protocol request can include sensitive information. Therefore implementers should ensure that the choice of transport of Health Certificate Enrollment Protocol messages is appropriate. For example, using HTTP over TLS (as specified in [RFC2818]) to authenticate the server and to provide confidentiality and integrity is better than using HTTP alone for Health Certificate Enrollment Protocol messages.

When the HCEA is authenticated using Kerberos-based HTTP authentication (as specified in [RFC4559]) it allows the HRA to validate data present in the certificate request. HRA should not impersonate the client and should only identify the client using this authentication method. Otherwise, if the HRA is compromised, it can potentially allow attackers to impersonate clients.

7.3 Health Certificate Services Logical Dependencies and Protocol Stack Views

The Health Certificate Enrollment Protocol uses HTTP (as specified in [RFC2616]) or HTTP over TLS (as specified in [RFC2818]) as the transport for its messages, as specified in section Transport. The payload of a Health Certificate Enrollment Protocol request message, sent by the HCEA, contains a PKCS #10 certificate request (as specified in [RFC2986]), which contains an SoH message.

The HRA sends a Health Certificate Enrollment Protocol response, the payload of which contains an SoHR. If the client is compliant with health policies, it will also include a PKCS #7 message (as specified in [RFC2315]) with possibly an X.509 certificate, as specified in [RFC3280].

The protocols logical view for a Windows Server 2008 operating system server deployment is shown in the figure. Note that the server-to-server communication is specific to the Microsoft implementation, and is shown here for informational purposes only. A different server topology can be implemented and not affect the communications between the HCEA and the HRA.

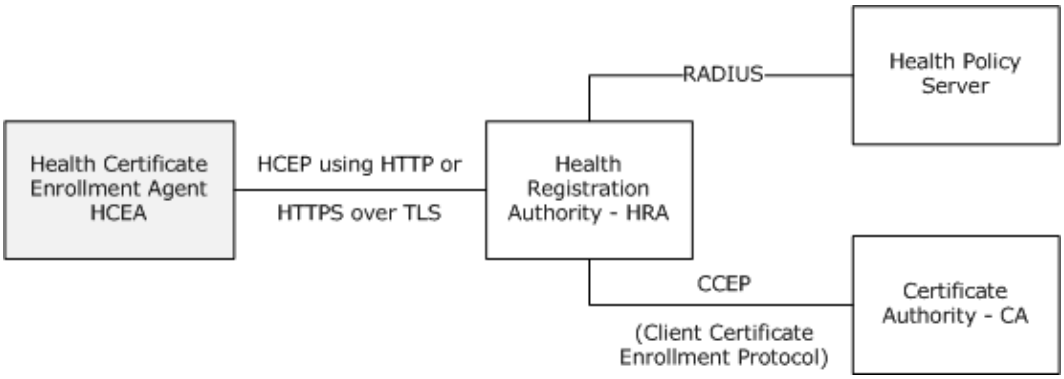

Figure 47: Health Certificate Enrollment Protocols logical overview

In the deployment illustrated in the above figure "Health Certificate Enrollment Protocols logical overview", the following events would cause the Health Certificate Enrollment Protocol to be used:

- The SHA (System Health Agent) determines that either due to the expiration of a timer or due to a change in the policy the health state needs to be re-certified.
- The health state itself has changed.
- The previously obtained health certificate is close to expiration.
- A new IP address has been obtained by the system. This is notification of a system event.

When any of these events occur, the Health Certificate Enrollment protocol is used to refresh the health certificate. The health state is communicated to the HRA, a response is received, and the certificate is deposited in the certificate store. The certificate is now available to IKE [RFC 2409], which uses the certificate to prove the identity and good health to the system that this system is attempting to connect to.

In the common case, the following events cause the HCEP protocol to be used:

- The SHA (System Health Agent) determines that either due to the expiration of a timer or due to a change in the policy the health state needs to be re-certified.
- The health state itself has changed.
- The previously obtained health certificate is close to expiration.
- A new IP address has been obtained by the system. This is notification of a system event.

When any of these events occur, a cooperating set of COM objects uses the Health Certificate Enrollment protocol to refresh the health certificate. The health state is communicated to the HRA, a response is received, and the certificate is deposited in the certificate store. The certificate is now

available to IKE [RFC2409], which uses the certificate to prove the identity and good health to the system that this system is attempting to connect to.

7.3.1 Health Certificate Enrollment Protocol Dependencies

The Health Certificate Enrollment Protocol is encapsulated within and depends on HTTP or HTTP over TLS for delivery of messages. The Health Certificate Enrollment Protocol does not impose any message retransmissions or other requirements on those transports. The choice of the transport is based entirely upon the URL present at the HCEA.

The Health Certificate Enrollment Protocol can be used to ensure that client computers have installed firewalls, critical patches, or meet other conditions before being allowed access to the network. In the following section, two such scenarios are traced and described.

7.4 Implementation Scenarios

This section provides two scenarios that illustrate common functionality that can be implemented with the protocols included in Health Certificate Services. The protocols involved in these scenarios includes the Health Certificate Enrollment Protocol, as specified in [MS-HCEP], together with other general networking protocols used in the course of standard TCP communication between the client and server. The functionality illustrated in the scenarios is as follows:

- Compliant client receives a health certificate.
- Non-compliant client is auto-remediated and receives a health certificate.

It should be noted that the following scenarios depict processes that occur over TCP and thus assumes various TCP acknowledgment frames throughout the conversation that are not necessarily depicted in the trace details.

7.4.1 Scenario 1 - Compliant Client Receives Health Certificate

7.4.1.1 Scenario Overview

In the following scenario, a compliant Windows Vista operating system client comes online and receives a health certificate.

For this scenario, the Windows Server 2008 operating system Network Access Protection (NAP) feature is configured so that network workstations must have the Windows firewall enabled. When a compliant workstation comes online, the workstation's configuration is compared with a network health policy to determine if the workstation is healthy.

7.4.1.2 Scenario Configuration

This scenario is implemented with a Windows Server 2003 operating system machine, a Windows Server 2008 operating system machine, and a Windows Vista operating system client machine configured as follows:

Sever1 Configuration

- Machine name: WS200301.contoso.com (WS200301)
- Operating system: Windows Server 2003 R2 operating system with latest updates
- IP Address: 10.0.10.1

- Subnet Mask: 255.255.255.0
- Active Directory Domain Controller – contoso.com
- Create NAP Exemption group:
    1. Open Active Directory Users and Computers.
    2. Create a new global security group called NAP Exemption.
    3. Add WS200301 and WS200801 computers to this group.
- Configure autoenrollment in Group Policy:
    1. In Active Directory Users and Computers, in the console tree, right-click contoso.com and then click Properties.
    2. Click the Group Policy tab.
    3. Click Edit.
    4. In the console tree, expand Computer Configuration | Windows Settings | Security Settings and then click Public Key Policies.
    5. In the details pane, double-click Autoenrollment Settings.
    6. Select the Renew expired certificates, and Update certificates. Check boxes and then click OK.
- Force Group Policy update:
    1. Click Start | Run.
    2. Type "gpupdate /force" and then click OK.

Server 2 Configuration

- Machine name: WS200801.contoso.con (WS200801)
- Operating system: Windows Server 2008 Beta 3
- IP Address: 10.0.10.2
- Subnet Mask: 255.255.255.0
- Domain member – contoso.com
- Installed roles:
    1. Active Directory Certificate Services
        1. Enterprise
        2. Root CA
        3. contoso-WS200801-CA
    2. Network Policy and Access Services
        1. Role services:

1. Network policy server
2. Health Registration Authority

2. Use local Certificate Server:

- Configure certificate settings:

1. In Server Manager, in the console tree, expand Roles | Active Directory Certificate Services, and then click Certificate Templates.
2. In the details pane, right-click Workstation Authentication, and then click Duplicate Template.
3. Click OK.
4. For Template display name, type "System Health Authentication".
5. Select the Publish certificate in Active Directory check box.
6. Click the Extensions tab.
7. Click Edit.
8. Click Add.
9. Scroll down and click the second application policy labeled System Health Authentication.
10. Click OK twice.
11. Click the Security tab.
12. Click Add.
13. Type "NAP Exemption" and then click OK.
14. In the Permissions field, under Allow, select the Enroll and Autoenroll check boxes.
15. Click OK.
16. In the console tree, expand contoso-WS200801-CA and then click Certificate Templates.
17. Right-click Certificate Templates and then click New | Certificate Template to Issue.
18. Click System Health Authentication and then click OK.
19. In the console tree, right-click contoso-WS200801-CA and then click Properties.
20. Click the Security tab.
21. Click Add.
22. Type "Network Service" and then click OK.
23. In the Permissions field, under Allow, select the Issue and Manage Certificates and Manage CA check boxes.
24. Click OK.

- Configure HRA:

1. Click Start | Administrative Tools | Health Registration Authority.
2. In the console tree, right-click Certificate Authority and then click Properties.
3. Click use enterprise certificate authority.
4. In the Authenticated compliant certificate template and Anonymous compliant certificate template lists, click SystemHealthAuthentication.
5. Click OK.

- Request new certificate for HRA:

1. Click Start | Run | MMC.
2. On the File menu, click Add/Remove Snap-in.
3. Click Certificates and then click Add.
4. Click Computer account and then click Next.
5. Click Finish.
6. Click OK.
7. In the console tree, expand Certificates (Local Computer) | Personal and then click Certificates.
8. Right-click Certificates and then click All Tasks | Request New Certificate.
9. Click Next.
10. Select the System Health Authentication check box and then click Enroll.
11. Click Finish.

- Configure System Health Validator:

1. In the Server Manager console tree, expand Roles | Network Policy and Access Services | NPS (Local) | Network Access Protection, and then click System Health Validators.
2. In the details pane, double-click Windows Security Health Validator.
3. Click Configure.
4. Clear all check boxes except Firewall.
5. Click OK twice.

- Configure Health Policies:

1. In the console tree, expand Polices and then click Health Policies.
2. Right-click Health Policies and then click New.
3. In the Policy name field, type "Compliant".
4. Select the Windows Security Health Validator check box.
5. Click OK.

6. Right-click Health Policies and then click New.
7. In the Policy name field, type "Non Compliant".
8. In the Client SHV checks list, click Client fails one or more SHV checks.
9. Select the Windows Security Health Validator check box.
10. Click OK.

- Remove existing network policies:
  1. In the console tree, under Policies, click Network Policies.
  2. Delete the two existing network policies.
- Configure NAP settings:
  1. In the console tree, click NPS (Local).
  2. In the details pane, click Configure NAP.
  3. In the Network connection method list, click IPsec with Health Registration Authority (HRA).
  4. Click Next four times and then click Finish.

Client Configuration

- Machine name: ClientVista.contoso.com (ClientVista)
- Operating system: Windows Vista Ultimate with latest updates
- IP Address: 10.0.10.10
- Subnet Mask: 255.255.255.0
- NetMon 3.1
- Configure MMC:
  1. Click Start and then type "MMC" and then click OK.
  2. On the File menu, click Add/Remove Snap-in.
  3. Add the following snap-ins:
     1. Certificates – Computer account – Local computer
     2. NAP Client Configuration – Local computer
     3. Services – Local computer
     4. Save this MMC
- Enable NAP services:
  1. In the client MMC, click Services.
  2. In the details pane, double-click Network Access Protection Agent.

3. In the Startup type list, click Automatic.
4. Click Start.
5. Click OK.

- Configure NAP client configuration:
  1. In the client MMC, click NAP Client Configuration.
  2. In the details pane, click Enforcement Clients.
  3. Right-click IPsec Relaying Party and then click Enable.
  4. In the console tree, click Health Registration Settings.
  5. In the details pane, click Trusted Server Groups.
  6. In the console tree, right-click Trusted Server Groups and then click New.
  7. In the Group Name field, type "Trusted HRA Servers" and then click Next.
  8. Clear the Require server verification check box. In the Add URLs… field, type "http://ws200801.contoso.com/domainhra/hcsrvext.dll".
  9. Click Add.
  10. Click Finish.
- Force Group Policy update and restart NAP agent:
  1. Click Start and then type "CMD" and then click OK.
  2. Type "gpupdate /force" and then press Enter.
  3. Type "net stop napagent && net start napagent" and then press Enter.
- Verify that certificate has been issued:
  1. In the client MMC, expand Certificates | Personal and then click Certificates.
  2. You should see a certificate issued with the intended purpose of System Health Authentication.

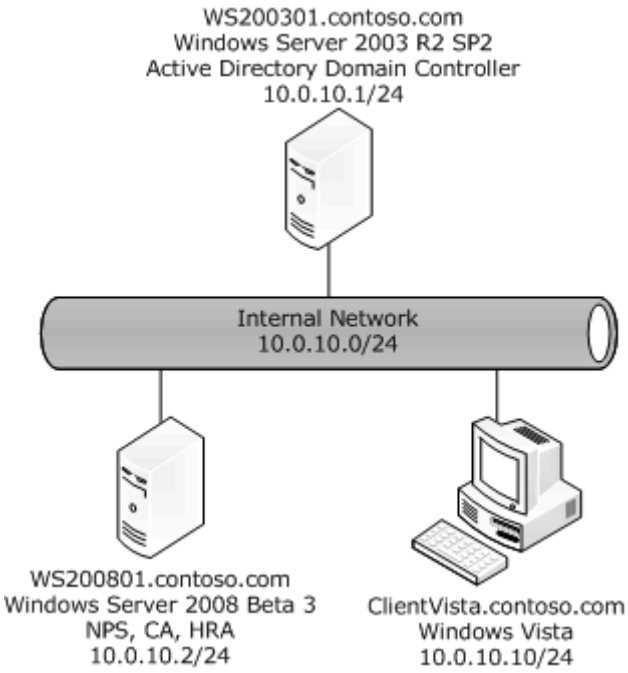

Figure 48: Trace Scenario 1 - Network topology

7.4.1.3 Setting Up the Trace

1. On WS200801, Start NetMon 3.1.
2. BEGIN TRACE.
   - On ClientVista, log on as CONTOSO\Administrator.
   - In the Certificate console, refresh the console and verify that a new health certificate has been issued. It may take a minute or two for the new certificate to show up.
3. STOP TRACE.

7.4.1.4 Netmon Trace Digest

- ClientVista issues request to WS200801 for MAC address (hardware address) using ARP protocol.

```
ClientVistaWS200801ARPARP: Request, 10.0.10.10 asks for
10.0.10.2
```

- WS200801 responds with the following ARP command containing the MAC address of the WS200801.

```
WS200801ClientVistaARPARP: Response, 10.0.10.2 at 00-03-FF-83-4E-1A
```

- ClientVista sends a health certificate request to WS200801 via HTTP request.

```
ClientVistaWS200801HTTPHTTP: Request, POST /domainhra/hcsrvext.dll
```

- The WS200801 returns an HTTP status code of 401 (Unauthorized) to ClientVista.

```
WS200801ClientVistaHTTPHTTP: Response, HTTP/1.1, Status Code = 401
```

- ClientVista resubmits the request, but encrypted with IPSec and the WS200301 issues a response.

```
ClientVistaWS200301  TCPTCP: Flags=.S......, SrcPort=49270,
DstPort=Kerberos(88), Len=0, Seq=3786090464, Ack=0, Win=8192
(scale factor 8) = 2097152

WS200301  ClientVistaTCPTCP: Flags=.S..A..., SrcPort=Kerberos(88),
 DstPort=49270, Len=0, Seq=2176118307, Ack=3786090465, Win=16384
(scale factor 0) = 16384

ClientVistaWS200301  TCPTCP: Flags=....A..., SrcPort=49270,
DstPort=Kerberos(88), Len=0, Seq=3786090465, Ack=2176118308, Win=256
 (scale factor 8) = 65536
```

- WS200801 sends a Kerberos Ticket Getting Service request to W200301, but the request is returned with a KDC_ERR_BADOPTION error.

```
WS200801WS200301KerberosV5KerberosV5: TGS Request
Realm: CONTOSO.COM Sname: ws200801$@CONTOSO.COM

WS20030110.0.10.2KerberosV5KerberosV5:
KRB_ERROR  - KDC_ERR_BADOPTION (13)
```

- WS200801 requests an RPC bind with WS200301, and WS200301 returns an acknowledgment.

```
WS200801WS200301MSRPCMSRPC: c/o Bind:
UUID{E1AF8308-5D1F-11C9-91A4-08002B14A0FA} Endpoint Mapper
Call=0x1  Assoc Grp=0x0  Xmit=0x16D0  Recv=0x16D0

WS200301WS200801MSRPCMSRPC: c/o Bind Ack:
Call=0x1  Assoc Grp=0x2DE9C  Xmit=0x16D0  Recv=0x16D0
```

- WS200801 submits a ticket getting ticket request to W200301.

```
WS200801WS200301KerberosV5KerberosV5: AS Request Cname:
 ws200801$ Realm: CONTOSO.COM Sname: krbtgt/CONTOSO.COM
```

- The request is returned from the WS200801 with a KDC_ERR_PREAUTH_REQUIRED error. This error indicates that the ClientVista did not send or did not send the appropriate type of pre-authorization.

```
WS200301WS200801KerberosV5KerberosV5: KRB_ERROR  -
KDC_ERR_PREAUTH_REQUIRED (25)
```

- WS200301 sends a new Kerberos Ticket request to WS200301.

```
WS200801WS200301KerberosV5KerberosV5: AS Request Cname:
 ws200801$ Realm: CONTOSO.COM Sname: krbtgt/CONTOSO.COM
```

- This time the WS200801 accepts the request and issues service ticket to ClientVista for the global catalog.

```
WS200301WS200801KerberosV5KerberosV5: AS Response
Ticket[Realm: CONTOSO.COM, Sname: krbtgt/CONTOSO.COM]

WS200801WS200301KerberosV5KerberosV5:
TGS Request Realm: CONTOSO.COM Sname:
GC/ws200301.contoso.com/contoso.com

WS20030110.0.10.2KerberosV5KerberosV5: TGS Response
Cname: WS200801$
```

- WS200801 issues a request to alter the context of a directory object. WS200301 responds by acknowledging the operation's success.

```
WS200801WS200301MSRPCMSRPC: c/o Alter Cont:
UUID{E3514235-4B06-11D1-AB04-00C04FC2DCD2} DRSUAPI  Call=0x1

WS200301WS200801MSRPCMSRPC: c/o Alter Cont Resp:
Call=0x1  Assoc Grp=0x21348  Xmit=0x16D0  Recv=0x16D0
```

- WS200801 sends a KRB_AS_REQ request to WS200301. WS200301 responds with a KDC_ERR_PREAUTH_REQUIRED error. This error indicates that challenge/response authentication is necessary. WS200801 resubmits the request and receives a ticket. WS200801 sends this ticket back to WS200301 and requests access to the LDAP service. WS200301 responds by issuing the requested ticket to WS200801.

```
WS200801WS200301KerberosV5KerberosV5: AS Request Cname:
 WS200801$ Realm: CONTOSO.COM Sname: krbtgt/CONTOSO.COM

WS20030110.0.10.2KerberosV5KerberosV5: KRB_ERROR  -
KDC_ERR_PREAUTH_REQUIRED (25)

WS200801WS200301KerberosV5KerberosV5: AS Request Cname:
 WS200801$ Realm: CONTOSO.COM Sname: krbtgt/CONTOSO.COM

WS200301WS200801KerberosV5KerberosV5: AS Response
Ticket[Realm: CONTOSO.COM, Sname: krbtgt/CONTOSO.COM]

WS200801WS200301KerberosV5KerberosV5: TGS Request
Realm: CONTOSO.COM Sname: ws200801$

WS200301WS200801KerberosV5KerberosV5:
```

```
TGS Response Cname: WS200801$
```

- WS200801 issues an LDAP BIND request to WS200301 for authentication purposes.

```
WS200801WS200301LDAPLDAP: Global Catalog,
Bind Request, MessageID: 55, Version: 3
```

- WS200301 responds to WS200801 indicating that the BIND request was successful.

```
WS200301WS200801LDAPLDAP: Global Catalog,
Bind Response, MessageID: 55, Status: Success
```

- An LDAP query is performed to obtain some basic information from the directory.

```
WS200801WS200301LDAPLDAP: Global Catalog,
Search Request, MessageID: 56, BaseObject: CN=WS200801,CN=Computers,
DC=contoso,DC=com, SearchScope: base Object,
SearchAlias: neverDerefAliases

WS200301WS200801LDAPLDAP: Global Catalog,
Search Result Entry, MessageID: 56, Status: Success

WS200801WS200301LDAPLDAP: Search Request,
MessageID: 57, BaseObject: NULL, SearchScope: base Object,
SearchAlias: neverDerefAliases

WS200301WS200801LDAPLDAP: Search Result Entry,
MessageID: 57
```

- WS200801 and WS200301 exchange a series of LDAP queries and responses that are encrypted with the General Security Service API (GSS-API).

```
WS200801WS200301LDAPLDAP: GSS-API Encrypted Payload,
 96 bytes ( Totally 96 bytes )

WS200301WS200801LDAPLDAP: GSS-API Encrypted Payload,
 362 bytes ( Totally 362 bytes )

WS200801WS200301LDAPLDAP: GSS-API Encrypted Payload,
 169 bytes ( Totally 169 bytes )

WS200301WS200801LDAPLDAP: GSS-API Encrypted Payload,
 286 bytes ( Totally 286 bytes )
```

- The process ends when WS200801 issues ClientVista a Statement of Health;

```
WS20080110.0.10.10SoHSoH:
```

7.4.2 Scenario 2 - Non-Compliant Client Is Auto-Remediated and Receives Health Certificate

7.4.2.1 Scenario Overview

In the following scenario, a non-compliant Windows Vista operating system client comes online, is auto-remediated, and then receives a health certificate. For this scenario, the Windows Server 2008 operating system Network Access Protection (NAP) feature is configured so that network workstations must have the Windows firewall enabled. When a compliant workstation comes online, the workstation's configuration is compared with a network health policy to determine if the workstation is healthy.

7.4.2.2 Scenario Configuration

This scenario is implemented with a Windows Server 2003 operating system machine, a Windows Server 2008 operating system machine, and a Windows Vista operating system client machine configured as follows:

Sever1 Configuration

- Machine name: WS200301.contoso.com (WS200301)
- Operating system: Windows Server 2003 R2 operating system with latest updates
- IP Address: 10.0.10.1
- Subnet Mask: 255.255.255.0
- Active Directory Domain Controller – contoso.com
- Create NAP Exemption group:
  1. Open Active Directory Users and Computers.
  2. Create a new global security group called NAP Exemption.
  3. Add WS200301 and WS200801 computers to this group.
- Configure autoenrollment in Group Policy:
  1. In Active Directory Users and Computers, in the console tree, right-click contoso.com and then click Properties.
  2. Click the Group Policy tab.
  3. Click Edit.
  4. In the console tree, expand Computer Configuration | Windows Settings | Security Settings and then click Public Key Policies.
  5. In the details pane, double-click Autoenrollment Settings.
  6. Select the Renew expired certificates, and Update certificates. Check boxes and then click OK.
- Force Group Policy update:
  1. Click Start | Run.

2. Type "gpupdate /force" and then click OK.

Server 2 Configuration

- Machine name: WS200801.contoso.con (WS200801)
- Operating system: Windows Server 2008 Beta 3
- IP Address: 10.0.10.2
- Subnet Mask: 255.255.255.0
- Domain member – contoso.com
- Installed roles:
  1. Active Directory Certificate Services:
     1. Enterprise
     2. Root CA
     3. contoso-WS200801-CA
  2. Network Policy and Access Services:
     1. Role Services:
        1. Network policy server
        2. Health Registration Authority
     2. Use local Certificate Server
- Configure certificate settings:
  1. In Server Manager, in the console tree, expand Roles | Active Directory Certificate Services, and then click Certificate Templates.
  2. In the details pane, right-click Workstation Authentication, and then click Duplicate Template.
  3. Click OK.
  4. For Template display name, type "System Health Authentication".
  5. Select the Publish certificate in Active Directory check box.
  6. Click the Extensions tab.
  7. Click Edit.
  8. Click Add.
  9. Scroll down and click the second application policy labeled System Health Authentication.
  10. Click OK twice.
  11. Click the Security tab.
  12. Click Add.

13. Type "NAP Exemption" and then click OK.
14. In the Permissions field, under Allow, select the Enroll and Autoenroll check boxes.
15. Click OK.
16. In the console tree, expand contoso-WS200801-CA and then click Certificate Templates.
17. Right-click Certificate Templates and then click New | Certificate Template to Issue.
18. Click System Health Authentication and then click OK.
19. In the console tree, right-click contoso-WS200801-CA and then click Properties.
20. Click the Security tab.
21. Click Add.
22. Type "Network Service" and then click OK.
23. In the Permissions field, under Allow, select the Issue and Manage Certificates and Manage CA check boxes.
24. Click OK.

- Configure HRA:

1. Click Start | Administrative Tools | Health Registration Authority.
2. In the console tree, right-click Certificate Authority and then click Properties.
3. Click use enterprise certificate authority.
4. In the Authenticated compliant certificate template and Anonymous compliant certificate template lists, click SystemHealthAuthentication.
5. Click OK.

- Request new certificate for HRA:

1. Click Start | Run | MMC.
2. On the File menu, click Add/Remove Snap-in.
3. Click Certificates and then click Add.
4. Click Computer account and then click Next.
5. Click Finish.
6. Click OK.
7. In the console tree, expand Certificates (Local Computer) | Personal and then click Certificates.
8. Right-click Certificates and then click All Tasks | Request New Certificate.
9. Click Next.
10. Select the System Health Authentication check box and then click Enroll.

11. Click Finish.

- Configure System Health Validator:

  1. In the Server Manager console tree, expand Roles | Network Policy and Access Services | NPS (Local) | Network Access Protection, and then click System Health Validators.
  2. In the details pane, double-click Windows Security Health Validator.
  3. Click Configure.
  4. Clear all check boxes except Firewall.
  5. Click OK twice.

- Configure health policies:

  1. In the console tree, expand Polices and then click Health Policies.
  2. Right-click Health Policies and then click New.
  3. In the Policy name field, type "Compliant".
  4. Select the Windows Security Health Validator check box.
  5. Click OK.
  6. Right-click Health Policies and then click New.
  7. In the Policy name field, type "Non Compliant".
  8. In the Client SHV checks list, click Client fails one or more SHV checks.
  9. Select the Windows Security Health Validator check box.
  10. Click OK.

- Remove existing network policies:

  1. In the console tree, under Policies, click Network Policies.
  2. Delete the two existing network policies.

- Configure NAP settings:

  1. In the console tree, click NPS (Local).
  2. In the details pane, click Configure NAP.
  3. In the Network connection method list, click IPsec with Health Registration Authority (HRA).
  4. Click Next four times and then click Finish.

Client Configuration

- Machine name: ClientVista.contoso.com (ClientVista)
- Operating system: Windows Vista Ultimate with latest updates
- IP Address: 10.0.10.10

- Subnet Mask: 255.255.255.0
- NetMon 3.1
- Configure MMC
  1. Click Start and then type "MMC" and then click OK.
  2. On the File menu, click Add/Remove Snap-in.
  3. Add the following snap-ins:
     1. Certificates – Computer account – Local computer
     2. NAP Client Configuration – Local computer
     3. Services – Local computer
     4. Save this MMC
- Enable NAP services:
  1. In the client MMC, click Services.
  2. In the details pane, double-click Network Access Protection Agent.
  3. In the Startup type list, click Automatic.
  4. Click Start.
  5. Click OK.
- Configure NAP client configuration:
  1. In the client MMC, click NAP Client Configuration.
  2. In the details pane, click Enforcement Clients.
  3. Right-click IPsec Relaying Party and then click Enable.
  4. In the console tree, click Health Registration Settings.
  5. In the details pane, click Trusted Server Groups.
  6. In the console tree, right-click Trusted Server Groups and then click New.
  7. In the Group Name field, type "Trusted HRA Servers" and then click Next.
  8. Clear the Require server verification check box. In the Add URLs… field, type "http://ws200801.contoso.com/domainhra/hcsrvext.dll".
  9. Click Add.
  10. Click Finish.
- Force Group Policy update and restart NAP agent:
  1. Click Start and then type "CMD" and then click OK.
  2. Type "gpupdate /force" and then press Enter.

3. Type "net stop napagent && net start napagent" and then press Enter.

- Verify that certificate has been issued:

  1. In the client MMC, expand Certificates | Personal and then click Certificates.
  2. You should see a certificate issued with the intended purpose of System Health Authentication.

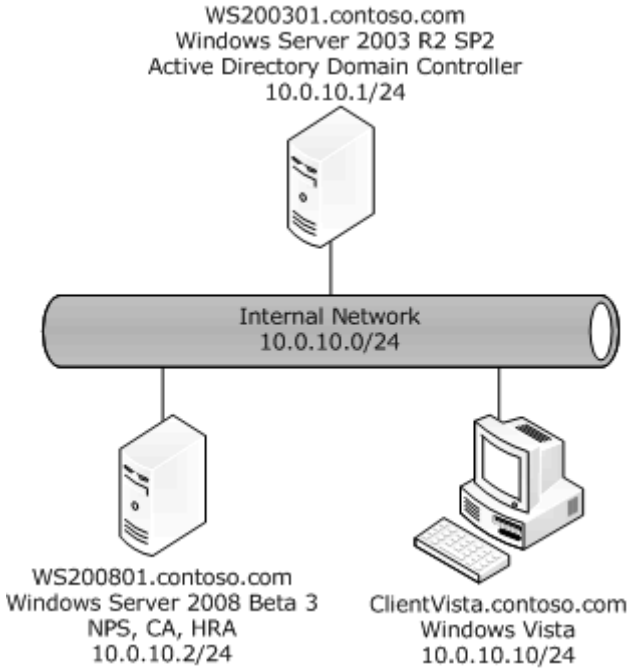

Figure 49: Trace Scenario 1 - Network topology

7.4.2.3 Setting Up the Trace

1. Start ClientVista.
2. Open Certificates on local computer and browse to the Personal Certificates store.
3. You should see a certificate issued with the intended purpose of System Health Authentication. Leave this window open.
4. Open Windows Firewall and leave it open.
5. Start NetMon 3.1.
6. BEGIN TRACE.
   - In the Certificates console, delete the existing certificate.
   - In the Windows Firewall window, turn off Windows Firewall.
     - The firewall will turn off briefly and then automatically come back on.
   - In the Certificate console, verify that a new health certificate has been issued.

7. STOP TRACE.

7.4.2.4 Netmon Trace Digest

- ClientVista issues request to server for MAC address (hardware address) using the ARP protocol.

```
10.0.10.10WS200801  ARPARP: Request, 10.0.10.10 asks for
 10.0.10.2
```

- W200801 responds with the following ARP command containing the MAC address of the server.

```
WS200801  10.0.10.10ARPARP: Response, 10.0.10.2 at
00-03-FF-83-4E-1A
```

- ClientVista sends health certificate request to W200801 via HTTP request.

```
10.0.10.10WS200801  HTTPHTTP: Request, POST /domainhra/hcsrvext.dll
```

- The W200801 returns an HTTP status code of 401 (Unauthorized) to ClientVista.

```
WS200801  10.0.10.10TCPTCP: Flags=....A..., SrcPort=HTTP(80),
DstPort=49274, Len=0, Seq=3781791760, Ack=178180008, Win=256
(scale factor not found)
WS200801  10.0.10.10HTTPHTTP: Response, HTTP/1.1, Status Code = 401
```

- ClientVista resubmits the request, but encrypted with IPSec and the server issues a response.

```
10.0.10.10WS200801  KerberosV5  KerberosV5:
  Frame:
+ Ethernet: Etype = Internet IP (IPv4)
+ Ipv4: Next Protocol = TCP, Packet ID = 1866, Total IP Length = 1500
+ Tcp: Flags=....A..., SrcPort=49274, DstPort=HTTP(80), Len=1460,
Seq=178180008 - 178181468, Ack=3781793246, Win=16425
(scale factor not found)
- Http: Request, POST /domainhra/hcsrvext.dll
    Command: POST
  + URI: /domainhra/hcsrvext.dll
    ProtocolVersion: HTTP/1.1
    Accept:  */*
    Pragma:  no-cache
    ContentType:  application/healthcertificate-request
    HCEP-Version:  1.0
    HCEP-Correlation-Id:  xkX7MI6iO0WVEV8e4avXT4ynlTq86McB
    UserAgent:  NAP IPSec Enforcement v1.0
    Host:  ws200801.contoso.com
    Connection:  Keep-Alive
    Cache-Control:  no-cache
  + Authorization:  Negotiate YIIFXAYGKwYBBQUCoIIFUDCCBUygJDAiBgkqhki
C9xIBAgIGCSqGSIb3EgECAgYKKwYBBAGCNwICCqKCBSIEggUeYIIFGgYJKoZIhvcSAQIC
AQBuggUJMIIFBaADAgEFoQMCAQ6iBwMFACAAAACjggO/YYIDuzCCA7egAwIBBaENGwtDT
05UT1NPLkNPTaInMCWgAwIBAqEeMBwbBEhUVFAbFHdzMjAwODAxLmNvb
```

```
WS200801  10.0.10.10TCPTCP: Flags=....A..., SrcPort=HTTP(80),
DstPort=49274, Len=0, Seq=3781793246, Ack=178183369, Win=256
(scale factor not found)
```

- W200801 sends a Kerberos Ticket Getting Service request to W200301, but the request is returned with a KDC_ERR_PREAUTH_REQUIRED error. This error indicates that the client did not send or did not send the appropriate type of pre-authorization.

```
WS200801  WS200301  KerberosV5KerberosV5: AS Request Cname:
ws200801$ Realm: CONTOSO.COM Sname: krbtgt/CONTOSO.COM
WS200301  WS200801  KerberosV5KerberosV5: KRB_ERROR  -
KDC_ERR_PREAUTH_REQUIRED (25)
```

- W200801 sends a new Kerberos Ticket request to W200301.

```
WS200801  WS200301  KerberosV5KerberosV5: AS Request Cname:
ws200801$ Realm: CONTOSO.COM Sname: krbtgt/CONTOSO.COM
```

- This time the server accepts the request and issues service ticket to ClientVista for the global catalog.

```
WS200301  WS200801  KerberosV5KerberosV5: AS Response Ticket[Realm:
 CONTOSO.COM, Sname: krbtgt/CONTOSO.COM]
WS200301  WS200801  KerberosV5KerberosV5: TGS Response Cname:
WS200801$
```

- W200801 issues a request to alter the context of a directory object. W200301 responds by acknowledging the operation's success.

```
WS200801  WS200301  MSRPCMSRPC: c/o Alter Cont:  UUID{E3514235-4B06
-11D1-AB04-00C04FC2DCD2} DRSUAPI  Call=0x1
WS200301  WS200801  MSRPCMSRPC: c/o Alter Cont Resp:  Call=0x1
Assoc Grp=0x1AE27  Xmit=0x16D0  Recv=0x16D0
```

- The W200801 performs an RPC bind with the Server.

```
WS200801  WS200301  MSRPCMSRPC: c/o Bind:  UUID{E1AF8308-5D1F-11C9
-91A4-08002B14A0FA} Endpoint Mapper  Call=0x1  Assoc Grp=0x0
Xmit=0x16D0  Recv=0x16D0
WS200301  WS200801  MSRPCMSRPC: c/o Bind Ack:  Call=0x1
Assoc Grp=0x22092  Xmit=0x16D0  Recv=0x16D0
WS200801  WS200301  MSRPCMSRPC: c/o Request: unknown   Call=0x1
  Opnum=0x3  Context=0x0  Hint=0x84
WS200301  WS200801  MSRPCMSRPC: c/o Response: unknown   Call=0x1
  Context=0x0  Hint=0x80  Cancels=0x0
```

- W200801 sends a KRB_AS_REQ request to W200301. W200301 responds with a KDC_ERR_PREAUTH_REQUIRED error. This error indicates that challenge/response authentication is necessary. W200801 resubmits the request and receives a ticket. W200801 sends this ticket back to W200301 and requests access to the LDAP service. W200301 responds by issuing the requested ticket to W200801.

```
WS200801  WS200301  KerberosV5KerberosV5: AS Request Cname:
ws200801$ Realm: CONTOSO.COM Sname: krbtgt/CONTOSO.COM
WS200301  WS200801  KerberosV5KerberosV5: KRB_ERROR  -
KDC_ERR_PREAUTH_REQUIRED (25)
WS200801  WS200301  KerberosV5KerberosV5: AS Request Cname:
ws200801$ Realm: CONTOSO.COM Sname: krbtgt/CONTOSO.COM
WS200301  WS200801  KerberosV5KerberosV5: AS Response Ticket[Realm:
 CONTOSO.COM, Sname: krbtgt/CONTOSO.COM]
```

- W200801 requests access to a service or a resource by sending a TGS request to the Ticket Granting Service (TGS) on the KDC. The Ticket Granting Service returns an individual ticket for the requested service that the client can submit to the server that holds the service or resource the clients wants.

```
WS200801  WS200301  KerberosV5KerberosV5: TGS Request Realm:
CONTOSO.COM Sname: GC/ws200301.contoso.com/contoso.com
WS200301  WS200801  KerberosV5KerberosV5: TGS Response Cname:
WS200801$
```

- W200801 performs a binding operation to the LDAP directory.

```
WS200801  WS200301  LDAPLDAP: Bind Request, MessageID: 15,
Version: 3
WS200301  WS200801  LDAPLDAP: Bind Response, MessageID: 15,
 Status: Success
```

- W200801 and W200301 exchange a series of LDAP queries and responses that are encrypted with the General Security Service API (GSS-API).

```
WS200801  WS200301  LDAPLDAP: GSS-API Encrypted Payload, 74 bytes
 ( Totally 74 bytes )
WS200301  WS200801  LDAPLDAP: GSS-API Encrypted Payload, 99 bytes
( Totally 99 bytes )
WS200801  WS200301  LDAPLDAP: GSS-API Encrypted Payload, 119 bytes
( Totally 119 bytes )
WS200301  WS200801  LDAPLDAP: GSS-API Encrypted Payload, 125 bytes
( Totally 125 bytes )
```

- The process ends when W200801 issues ClientVista a Statement of Health.

```
WS200801  10.0.10.10SoHSoH:
```

8 Media Streaming Services Protocols

This section provides an overview of the Media Streaming Services protocols. It provides a high-level conceptual overview of the core media streaming protocols, classifying them broadly according to how the streaming media data is transferred over the network.

8.1 Media Streaming Services Overview

The Media Streaming Services include protocols for communicating data packets that originate from downloadable and streaming audio, visual, and other multimedia data files.

8.1.1 Protocols List

The Media Streaming Services include two sets of protocols. The first set of protocols is referred to as the core protocols, which provide the media streaming services functionality. The second set of protocols is referred to as the networking protocols group, as described in the Overview section of this document. The latter set of protocols is required for an implementation of the core protocols (as listed in the Normative References section of the core protocols technical specifications) or provides core networking functionality.

The following table lists the core protocols included in Media Streaming Services. This list does not include data structures, file structures, or algorithms that these core protocols depend on. The details for these technical dependencies are documented in the technical specifications for each core protocol.

Protocol Name	Protocol Description	Document Short Name
Media Stream Broadcast (MSB) Protocol	Allows the multicast distribution of ASF packets over a network for which Internet Protocol (IP) multicasting is enabled. MSB allows clients to tune in to a broadcast on a network, much like television and radio users can tune to a particular television or radio station.	[MS-MSB]
Media Stream Broadcast Distribution (MSBD) Protocol	Used to transfer a live stream of audio-visual content from a server to a single client or multiple clients. Media Stream Broadcast Distribution (MSBD) Protocol may be used to transmit the digitized audio and video of a live event to another computer that is running appropriate streaming media server software such as Windows Media Services, or to distribute the stream to multiple clients.	[MS-MSBD]
Microsoft Media Server (MMS) Protocol	Used by Windows Media Services to stream data between the Windows Media Player (WMP) and Windows Media Server over Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). The MMS documentation contains a description of the MMS Protocol, its sequencing, and its packets.	[MS-MMSP]
Real-Time Streaming Protocol (RTSP) Windows Media Extensions	Defines extensions to Real-Time Streaming Protocol (RTSP), Real-Time Transport Protocol (RTP), Session Description Protocol (SDP), and Real-Time Transport Control Protocol (RTCP) to enable the delivery of multimedia data that is encapsulated in Advanced Systems Format (ASF) packets.	[MS-RTSP]
Windows Media HTTP Streaming Protocol	Used to transfer real-time multimedia data (for example, audio and video). The protocol depends on Hypertext Transfer Protocol (HTTP) for the transfer of all protocol messages, including the transfer of multimedia data.	[MS-WMSP]

Protocol Name	Protocol Description	Document Short Name
Windows Media HTTP Push Distribution Protocol	Used to transfer real-time multimedia data (for example, audio and video) from a client to a server. Push distribution is ideal for broadcasting company meetings or live presentations. In such scenarios, the client is likely to be an encoder software application, perhaps implemented using the Windows Media Encoder software development kit (SDK). For more information, see the Windows Media Encoder SDK.	[MS-WMHTTP]

The following sections provide some basic media streaming protocol concepts, and include individual sections that provide different conceptual views to help the reader visualize the physical and logical relationships among the core protocols in the Media Streaming Services. These views are called the protocol stack view and the logical dependencies view, as explained in the Overview section of this document.

8.2 Media Streaming Services Protocols Concepts

This section describes basic concepts for media streaming services protocols, and provides some background for the Windows media streaming model.

The main purpose of media streaming services is to deliver real-time or downloadable audio-visual content via the transfer of streams from a server to a single client or multiple clients.

Section 8.4, later in this document, provides examples that illustrate the interaction of some of the protocols in two sample configurations.

8.2.1 Windows Media Server Streaming Content Delivery

A streaming media system based on Windows Media technologies typically consists of a computer running Windows Media Encoder, a server running Windows Media Server, and a number of client computers running Windows Media Player (WMP). The encoder converts both live and prerecorded audio and video content to Windows Media format. The Windows Media Server distributes the content over a network or the Internet. The player then receives the content.

One legacy method of media content delivery requires a client machine to download an entire file before the content can be played, which consumes both time and disk space. In addition, because the entire file must be downloaded to a computer before it can play, downloading cannot be used with live content. Moreover, downloading does not make efficient use of available bandwidth. When a client begins to download a digital media file, all available network bandwidth is used to transfer the data as quickly as possible. As a result, other network functions may slow down or be disrupted.

Streaming uses bandwidth more efficiently than downloading, because streaming sends data over the network only at the speed that is necessary for the client to render it properly. This method helps prevent the network from becoming overloaded and helps maintain system reliability. There is typically a delay between the time the player receives the stream and the point at which the content begins to play, because the player must first buffer the data in case there are delays or gaps in the stream. Because data streaming and rendering occur at the same time, streaming also enables the delivery of live content.

In the typical user scenario, the user clicks a link on a web page to request content. The web server then redirects the request to the Windows Media Server and starts WMP on the user's computer. At this point, the web server no longer plays a role in the streaming media process, because the WMP establishes a direct connection with the Windows Media Server which begins to stream the content directly to the user.

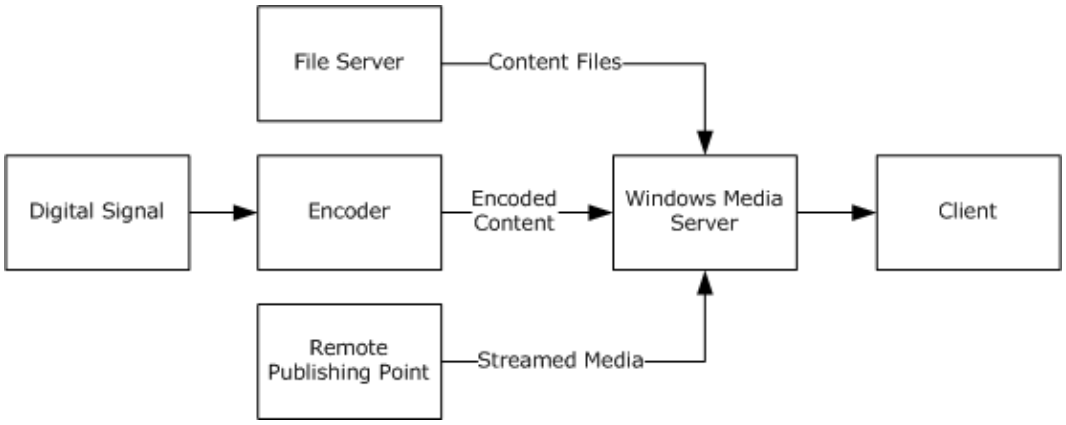

Figure 50: Media Streaming Server content delivery

The Windows Media Server can receive content from several different sources. Prerecorded content can be stored locally on the server, or retrieved from a networked file server. Live events can be captured using a digital recording device, and processed through the encoder before they are sent to the Windows Media Server for broadcast. Windows Media Server can also rebroadcast content streamed from a publishing point on a remote Windows Media Server.

The following sections provide information about each of the core protocols, and examples of how they are used.

8.2.2 Real-Time Streaming Protocol: Windows Media Extensions

The Real-Time Streaming Protocol (RTSP) Windows Media Extensions specification defines extensions to RTSP, Real Time Transport Protocol (RTP), Session Description Protocol (SDP), and Real-Time Transport Control Protocol (RTCP) to enable the delivery of multimedia data that is encapsulated in Advanced Systems Format (ASF) packets.

The ASF is the extensible file storage format developed by or for Microsoft for authoring, editing, archiving, distributing, streaming, playing, referencing, or otherwise manipulating content. A wide variety of coder-decoders (codecs) can be stored in an ASF file and played back with the WMP (provided the appropriate codecs are installed), streamed with Windows Media Services, or optionally packaged with Windows Media Rights Manager (WMRM). Microsoft Windows Media Rights Manager is an add-on component for Windows Media Player that handles packaging, license acquisition, and playing a media file that is protected by Microsoft Digital Rights Management. While the client portion of WMRM is included with WMP, the functionality to encode media with Microsoft DRM is provided with the Windows Media Rights Manager Software Developer's Kit.

The basic Windows Media Rights Manager process is as follows:

1. Packaging

   Windows Media Rights Manager packages the digital media file. The packaged media file has been encrypted and locked with a "key". This key is stored in an encrypted license, which is distributed separately. Other information is added to the media file, such as the URL where the license can be acquired. This packaged digital media file is saved in Windows Media Audio format (with a .wma file name extension) or Windows Media Video format (with a .wmv file name extension).

2. Distribution

The packaged file can be placed on a website for download, placed on a media server for streaming, distributed on a CD, or sent by email to consumers. Windows Media Rights Manager permits consumers to send copy-protected digital media files to their friends as well.

3. Establishing a License Server

   The content provider chooses a license clearing house that stores the specific rights or rules of the license and implements the Windows Media Rights Manager license services. The role of the clearing house is to authenticate the consumer's request for a license. Digital media files and licenses are distributed and stored separately, making it easier to manage the entire system.

4. License Acquisition

   To play a packaged digital media file, the consumer must first acquire a license key to unlock the file. The process of acquiring a license begins automatically when the consumer attempts to acquire the protected content, acquires a predelivered license, or plays the file for the first time. Windows Media Rights Manager either sends the consumer to a registration page where information is requested or payment is required, or "silently" retrieves a license from a clearing house.

5. Playing the Media File

   To play the digital media file, the consumer needs a media player that supports Windows Media Rights Manager. The consumer can then play the digital media file according to the rules or rights that are included in the license. Licenses can have different rights, such as start times and dates, duration, and counted operations. For instance, default rights may allow the consumer to play the digital media file on a specific computer and copy the file to a portable device. Licenses, however, are not transferable. If a consumer sends a packaged digital media file to a friend, this friend must acquire his or her own license to play the file. This PC-by-PC licensing scheme ensures that the packaged digital media file can only be played by the computer that has been granted the license key for that file.

The following table summarizes the standards-based protocols extended by the RTSP Windows Media Extensions.

Protocol Name	Protocol Description	Document Short Name
Real-Time Streaming Protocol (RTSP v1.0)	Transfers real-time multimedia data (for example, audio and video) between a server and a client.	[RFC2326]
Real-Time Transport Control Protocol (RTCP)	Provides feedback on the quality of service (QoS) RTP is providing. RTCP gathers statistics on a media connection and information such as bytes sent, packets sent, lost packets, jitter, feedback, and round-trip delay. An application may use this information to increase the quality of service, perhaps by limiting flow, or by using a low-compression codec instead of a high-compression codec.	[RFC3350]
Real-Time Transport Protocol (RTP)	Provides end-to-end network transport functions suitable for applications that transmit real-time data, such as audio, video, or simulation data, over multicast or unicast network services.	[RFC3350]
Session Description Protocol (SDP)	Describes multimedia sessions for the purposes of session announcement, session invitation, and other forms of multimedia session initiation.	[RFC4566]

8.2.2.1 RTSP Overview

The RTSP Windows Media Extensions are used to stream multimedia from WMS to Microsoft WMP or to another instance of WMS. To understand the extensions, it is important to understand the underlying protocol upon which the extensions are based.

RTSP, as specified in [RFC2326], is used to transfer real-time multimedia data (for example, audio and video) between a server and a client. RTSP is a streaming protocol; this means it attempts to facilitate scenarios in which the multimedia data is being transferred and rendered (that is, video displayed and audio played) simultaneously. RTSP establishes and controls either a single or several time-synchronized streams of continuous media. This protocol does not typically deliver the continuous streams itself, although interleaving of the continuous media stream with the control stream is possible. In other words, RTSP acts as a "network remote control" for multimedia servers.

RTSP typically uses a Transmission Control Protocol (TCP) connection for control of the streaming media session, although User Datagram Protocol (UDP) also can be used for this purpose.

The entity that sends the RTSP request that initiates the session is referred to as the client, and the entity that responds to that request is referred to as the server. Typically, the multimedia data flows from the server to the client. RTSP also allows multimedia data to flow in the opposite direction. However, the extensions defined in the RTSP Windows Media Extensions specification were not designed for such scenarios.

Clients can send RTSP requests to the server, to request information about content before a session has been established. The information returned by the server is formatted using the SDP syntax, as specified in [RFC4566].

Clients use RTSP requests to control the session and to request the server to perform actions such as starting or stopping the flow of multimedia data. For each request, a corresponding RTSP response is sent in the opposite direction. Servers can also send RTSP requests to clients; for example, to inform them that session state has changed.

While RTSP provides a transport mechanism for streaming, the actual multimedia content is relayed across the network using the Real-time Transport Protocol (RTP) as specified in [RFC3550]. For each RTP stream, the server and client can also exchange Real-time Transport Control Protocol (RTCP) packets, as specified in [RFC3556].

8.2.2.2 RTSP States and Request Methods

RTSP controls a stream that may be sent via a separate protocol, independent of the control channel. For example, RTSP control may occur on a TCP connection while the data flows via UDP. Thus, data delivery continues, even if no RTSP requests are received by the media server. Also, during its lifetime, a single media stream may be controlled by RTSP requests issued sequentially on different TCP connections.

Therefore, the server needs to maintain "session state" to be able to correlate RTSP requests with a stream. Many methods in RTSP do not contribute to state. However, the following play a central role in defining the allocation and usage of stream resources on the server: SETUP, PLAY, RECORD, PAUSE, and TEARDOWN.

- SETUP: Causes the server to allocate resources for a stream and start a RTSP session.
- PLAY and RECORD: Starts data transmission on a stream allocated via SETUP.
- PAUSE: Temporarily halts a stream without freeing server resources.

- TEARDOWN: Frees resources associated with the stream. The RTSP session ceases to exist on the server.

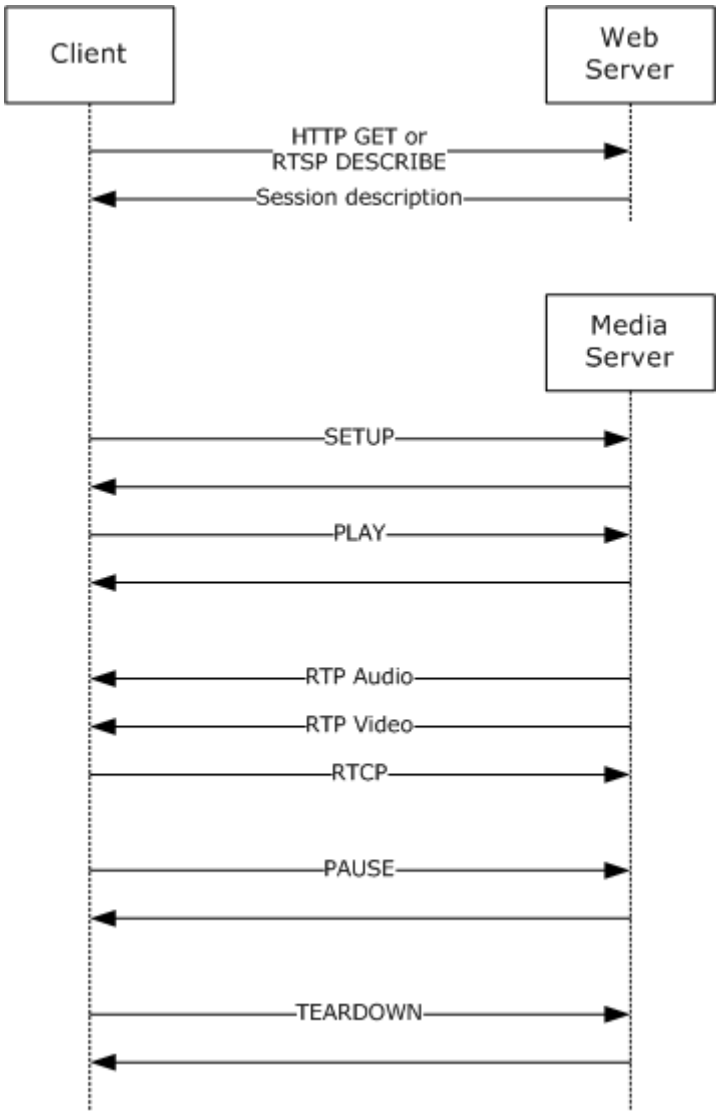

Figure 51: RTSP session example

8.2.2.3 RTSP Request Types

RTSP defines logical requests that are sent from the client to the server, or from the server to the client.

The requests from the client and the corresponding responses from the server are exchanged using RTSP request methods. Each request type is mapped using one of the following RTSP methods:

- ANNOUNCE: The purpose of the ANNOUNCE request is to send information to the client on a new entry in a server-side playlist that the server will start to stream. The ANNOUNCE request describes the playlist entry using SDP syntax; this request provides (among other things) the

ASF file header as well as a URL for each stream, and information on the RTP payload format for each stream.

- DESCRIBE: The purpose of the DESCRIBE request is to request information on one particular piece of multimedia content, which is identified by a URL. The client sends this request before it asks the server to start streaming the content. The server's response describes the content using SDP syntax, providing (among other things) the ASF file header as well as a URL for each stream, and information on the RTP payload format for each stream.
- GET_PARAMETER: The purpose of the GET_PARAMETER request is to retrieve the value of a parameter for a presentation or stream.
- PAUSE: The purpose of the PAUSE request is to request that the server stop streaming RTP packets for all of the currently selected streams.
- PLAY: The main purpose of the PLAY request is to ask the server to start streaming RTP packets for the currently selected streams. If the server has switched to a new entry in a server-side playlist, and the server is using Predictive Stream selection to select streams on the client's behalf, the PLAY request is also used as a way for the client to confirm the stream selection made by the server, and to confirm that it has started to play the RTP packets for the new playlist entry.
- SET_PARAMETER: The purpose of the SET_PARAMETER request is to set the value of a parameter for a presentation or stream. For example, the ENDOFSTREAM request is mapped to the SET_PARAMETER method. The purpose of the ENDOFSTREAM request is to inform the client that the server has transmitted the last RTP packet for all of the selected streams in the content.
- SETUP: The purpose of the SETUP request is to cause the server to allocate resources for a stream and start an RTSP session.
- TEARDOWN: The purpose of the TEARDOWN request is to free resources associated with the stream. With this request, the RTSP session ceases to exist on the server.

8.2.2.4 RTSP Client-Side State

The state machine for RTSP clients is as specified in [RFC2326], and as depicted in the following diagram.

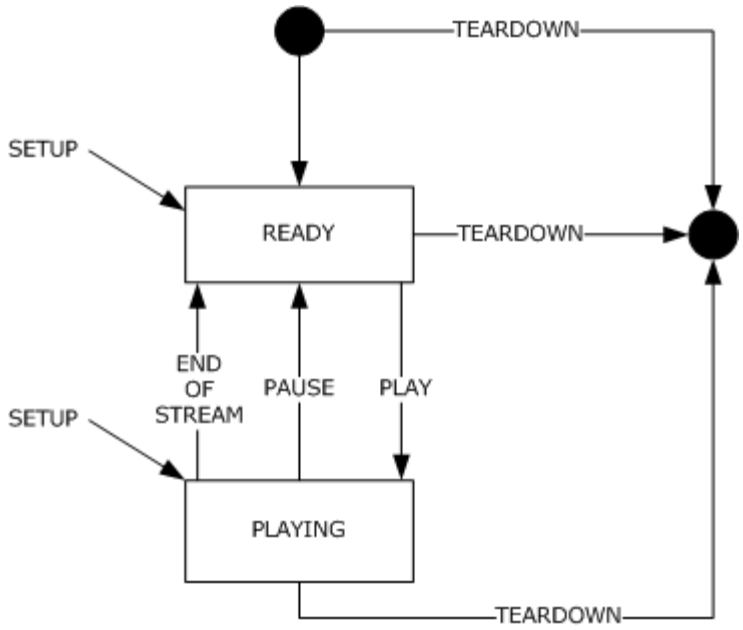

Figure 52: RTSP state diagram (client perspective)

RTSP defines an additional state transition: An ENDOFSTREAM request can cause the client to transition from PLAYING to READY state.

8.2.2.5 RTSP Server-Side State

The state machine for RTSP servers is as specified in [RFC2326], and as depicted in the following diagram.

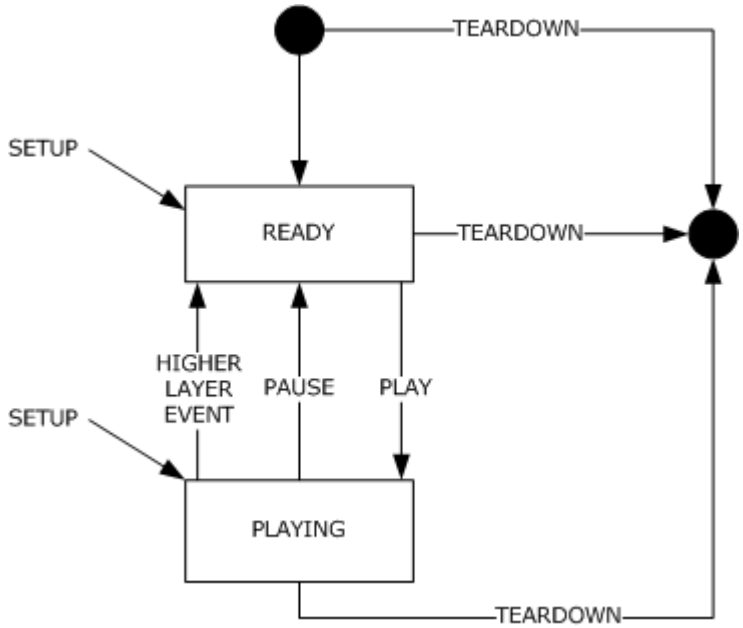

Figure 53: RTSP state diagram (server perspective)

RTSP defines an additional state transition: A higher-layer event can cause the server to transition from PLAYING to READY state.

8.2.3 Microsoft Media Server Protocol

The Microsoft Media Server (MMS) Protocol is used to transfer real-time multimedia data (for example, audio and video). Because it is a streaming protocol, the MMS Protocol attempts to facilitate scenarios in which the multimedia data is being transferred and rendered (such as video displayed and audio played) simultaneously.

The MMS Protocol is suitable for streaming delivery of real-time multimedia data. The term streaming means that data is transmitted at some fixed rate, or at some rate related to the rate at which the data is consumed (for example, displayed) by the receiver.

It is appropriate to use this protocol when a client is streaming from a server that does not support the RTSP Windows Media Extensions or Windows Media HTTP Streaming Protocol. Otherwise, use of the MMS Protocol is generally not appropriate.

The MMS Protocol does not support seeking in a server-side playlist. That functionality is available in the RTSP Windows Media Extensions and in the Windows Media HTTP Streaming Protocol.

If MMS is used in a scenario in which the multimedia data is always transferred over TCP, the Windows Media HTTP Streaming Protocol might be more appropriate to use instead because it does not include the UDP functionality.

The MMS Protocol uses a TCP connection for control of the streaming media session. The entity that initiates the TCP connection is referred to as the client, and the entity that responds to the TCP connection is referred to as the server. The multimedia data flows from the server to the client.

The client can send MMS Protocol request messages to the server over the TCP connection, to request the server to perform actions such as starting and stopping the flow of multimedia data. The multimedia data is transferred either over the same TCP connection or as a flow of UDP packets.

While the server is transmitting multimedia data to the client, the client can send MMS Protocol messages to the server, requesting that it change the stream being transmitted. For example, the client can request that the server replace the currently transmitted video stream with a lower bit-rate version of the same video stream.

If UDP is used to transmit the multimedia data to the client, the client can send an MMS Protocol message to the server requesting that it resend a UDP packet. This approach is useful if the client does not receive a UDP packet the server transmitted. Unlike other MMS Protocol messages sent by the client, the request to resend a UDP packet is sent using UDP.

Streaming media content may be made accessible only to authorized clients. The authentication sequences that the MMS Protocol supports are based on a basic and new technology local area network (NT LAN) Manager (NTLM) sequence. Authentication is not covered in detail as part of this task overview. See the Authentication Services overview for more information on authentication protocols.

8.2.3.1 MMS Protocol General Sequence

The following diagram shows the sequence of TCP packets that pass over the wire when the client attempts to stream a media file from the server using the MMS Protocol. The sequence assumes

that the media stream does not require authentication, and that the client does not request packet-pair bandwidth estimation.

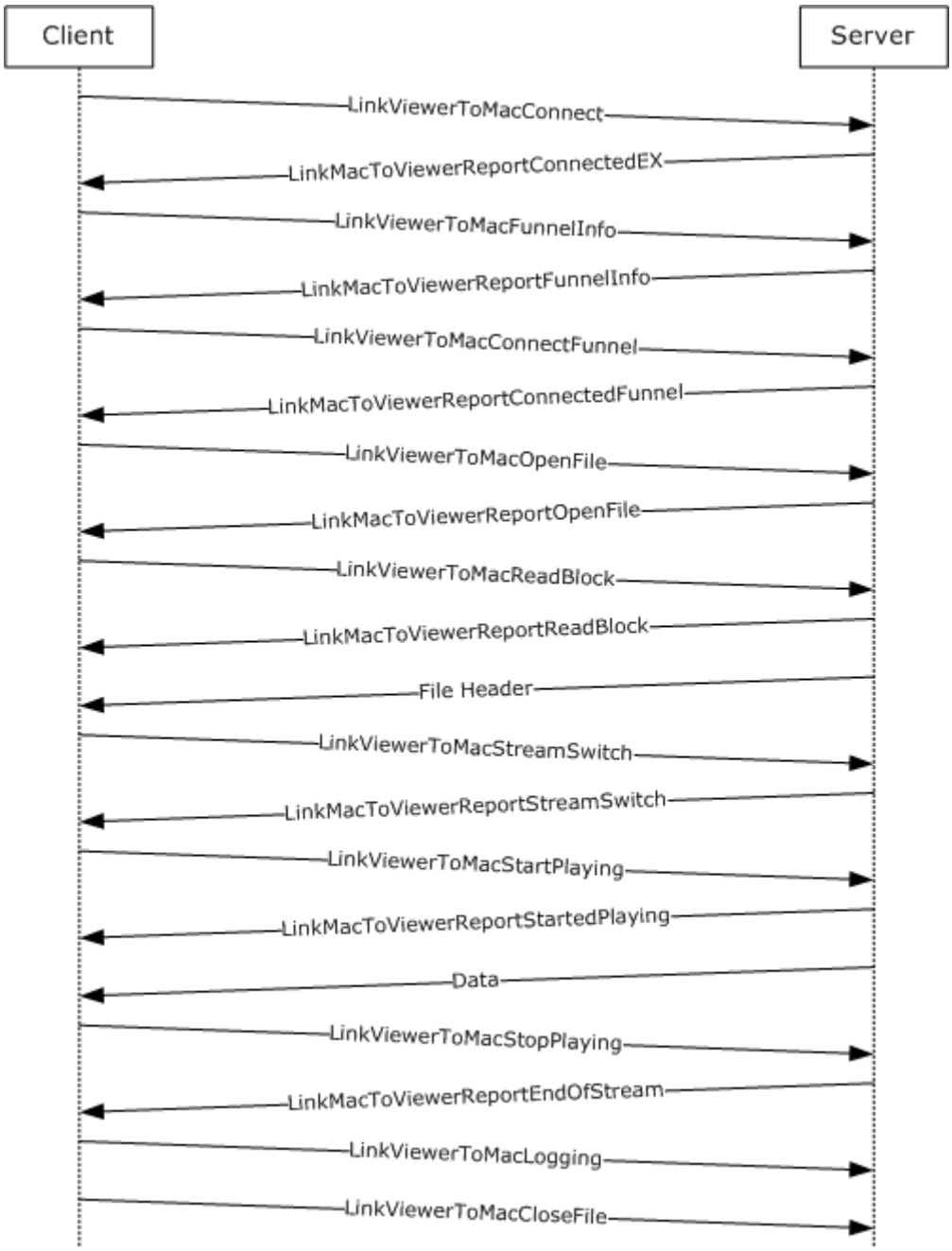

Figure 54: MMS sequence of TCP packets

The steps in this sequence are as follows:

1. The client sends a LINKVIEWERTOMACCONNECT message to request a connection from the server.

2. The server responds with a LINKMACTOVIEWERREPORTCONNECTEDEX message.
3. The client requests information from the server with a LINKVIEWERTOMACFUNNELINFO message.
4. The server responds with a LINKMACTOVIEWERREPORTFUNNELINFO message.
5. The client sends a LINKVIEWERTOMACCONNECTFUNNEL message to choose a specific port and request that the server use that port to send streaming media content.
6. The server responds, sending a LINKMACTOVIEWERREPORTCONNECTEDFUNNEL message to notify the client that it will use the requested port.
7. The client sends a LINKVIEWERTOMACOPENFILE message to request by name the specific media file it needs to have streamed to it from the server.
8. The server responds that it was able to open the requested file with a LINKMACTOVIEWERREPORTOPENFILE message.
9. The client sends a LINKVIEWERTOMACREADBLOCK message to request the file header from the server.
10. The server responds with a LINKMACTOVIEWERREPORTREADBLOCK message; it also sends the file header information as a Data packet.
11. The client sends a LINKVIEWERTOMACSTREAMSWITCH message to request the streams it needs from the server.
12. The server responds with a LINKMACTOVIEWERREPORTSTREAMSWITCH message.
13. The client sends a LINKVIEWERTOMACSTARTPLAYING message to request that the server begin streaming the media file.
14. The server responds with a LINKMACTOVIEWERREPORTSTARTEDPLAYING message; it also sends a stream of Data packets.
15. During file playback, the client may send a LINKVIEWERTOMACSTOPPLAYING message.
16. The server sends a LINKMACTOVIEWERREPORTENDOFSTREAM message to the client either in response to the client's request, or because the end of the stream has been reached.
17. The client sends a LINKVIEWERTOMACLOGGING message that contains the log information.
18. The client sends a LINKVIEWERTOMACCLOSEFILE message. The client closes the network socket connection.

8.2.3.2 Playback Adjustment Sequence

Windows Media Services uses the MMS Protocol to stream data between the WMP and Windows Media Server by means of TCP and UDP.

During the playback process, the user is able to drag the seek bar to a new position and fast forward and rewind the content. The following sequence demonstrates one of these sequencing maneuvers: sequencing during playback.

If the user drags the seek bar to a new position, the sequence shown in the following diagram occurs:

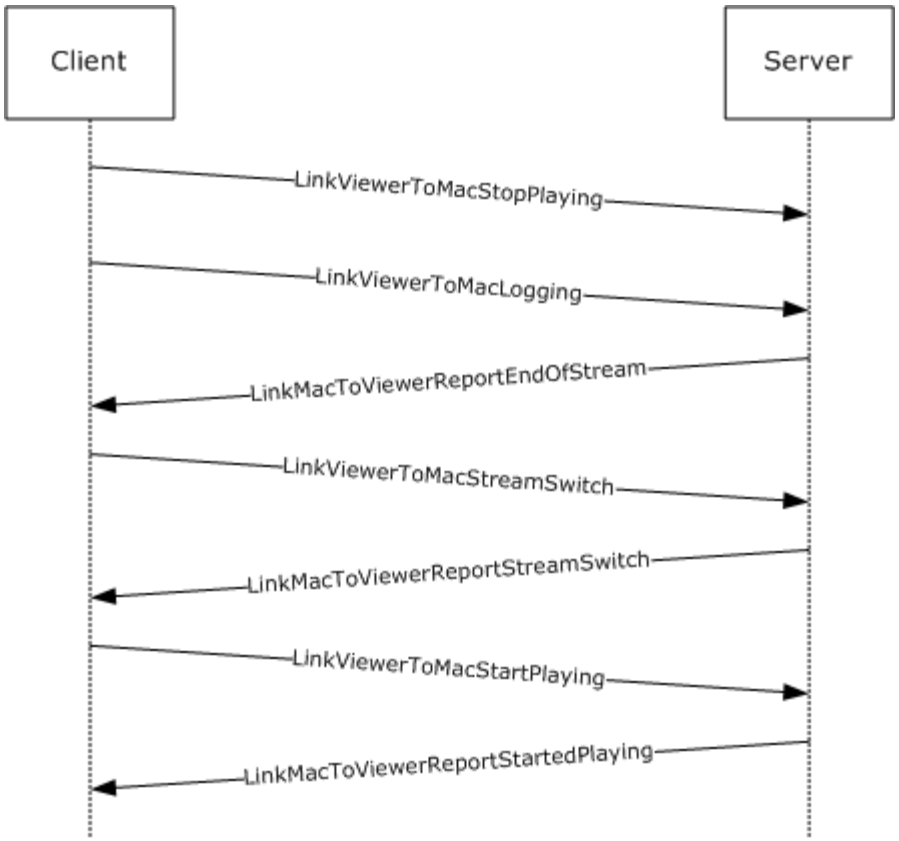

Figure 55: Sequencing during playback - new position of seek bar

The steps in this sequence are as follows:

1. The client sends the LinkViewerToMacStopPlaying message followed by a LinkViewerToMacLogging message.
2. The server confirms with a LinkMacToViewerReportEndOfStream message.
3. The client sends a LinkViewerToMacStreamSwitch message.
4. The server responds with a LinkMacToViewerReportStreamSwitch message.
5. The client sends a LinkViewerToMacStartPlaying message that contains the new playback position.
6. The server responds with a LinkMacToViewerReportStartedPlaying message, while also streaming data to the client.

8.2.4 Media Stream Broadcast Protocol

The Media Stream Broadcast (MSB) Protocol ([MS-MSB]) allows the multicast distribution of ASF packets over a network for which IP multicasting is enabled. The MSB Protocol allows clients to tune in to a broadcast on a network, much like television and radio users can tune to a particular television or radio station.

To access a network broadcast, clients listen for Media Stream Broadcast packets on a particular IP address and UDP port. The specific IP multicast address and UDP port are delivered to clients by a Windows Media Station (.nsc) file. The .nsc file is delivered to the clients by some other means, such as hosting the file at a URL for retrieval by means of HTTP, or sending the file as an email attachment.

8.2.4.1 Media Stream Broadcast Protocol General Sequence

The following diagram shows a typical communication sequence between a client and a server.

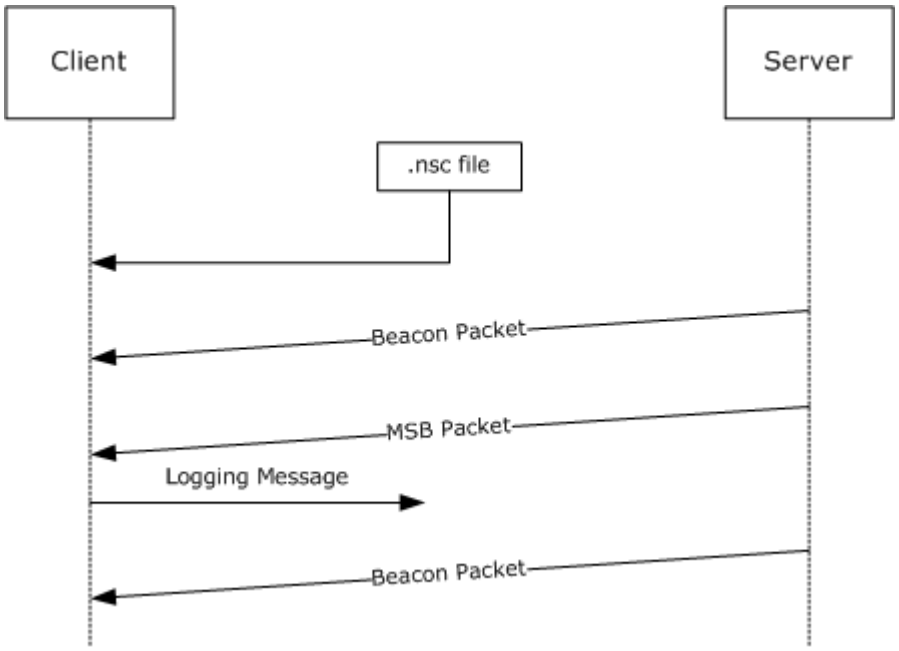

Figure 56: Typical media stream broadcast communication sequence between a client and a server

The steps in this sequence are as follows:

1. The client retrieves an .nsc file by some means, such as retrieving the file from a URL or through an email attachment.
2. The server (optionally) transmits Beacon packets prior to transmission of Media Stream Broadcast packets.
3. The server sends one or more Media Stream Broadcast packets.
4. If the Log URL property is specified in the .nsc file, the client submits a logging message at the end of the stream.
5. The server transmits Beacon packets if it intends to transmit future Media Stream Broadcast packets, and if it wants the client to remain ready to receive the Media Stream Broadcast packets.

8.2.4.2 Server-Side Playlist Streaming

The following diagram shows a typical communication sequence between a client and a server when content is being streamed from a server-side playlist. It is important to note that when the MSB Protocol is used, the client does not have the capability to seek or skip to a new entry in a server-side playlist.

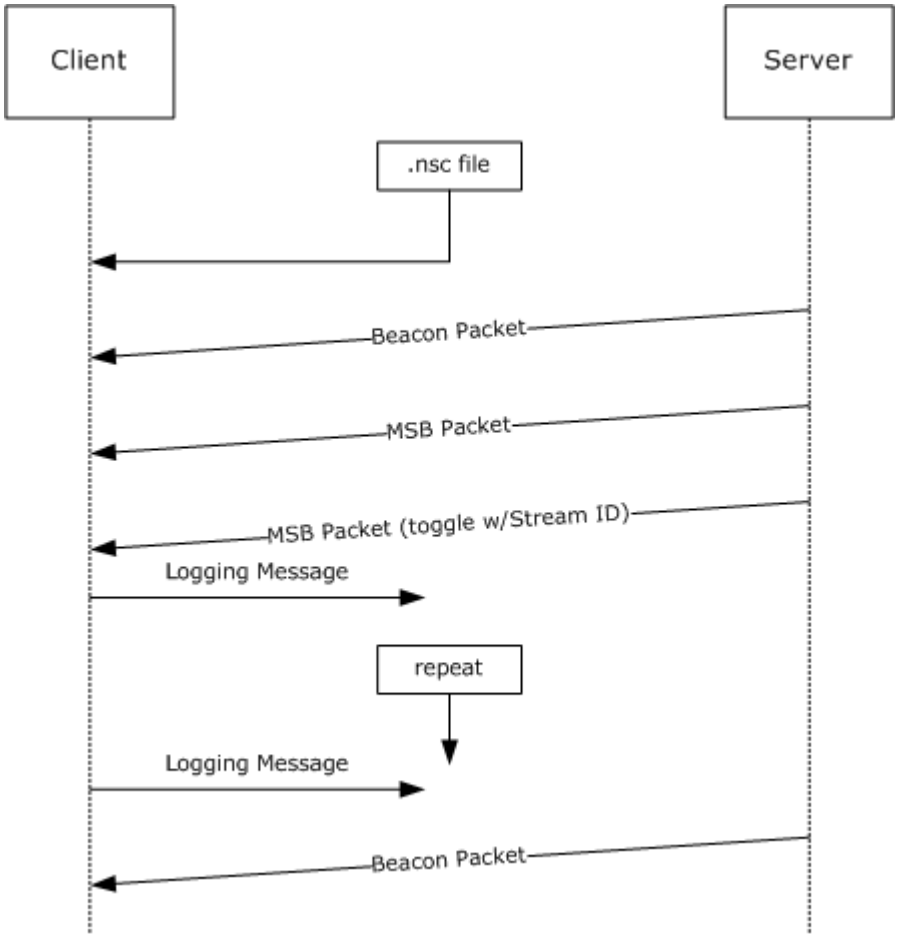

Figure 57: Server-side playlist sequence during client/server communication

The steps in this sequence are as follows:

1. The client retrieves an .nsc file by some means, such as retrieving the file from a URL or through an email attachment.
2. The server (optionally) transmits Beacon packets prior to transmission of Media Stream Broadcast packets.
3. The server sends one or more Media Stream Broadcast packets.
4. When the entry changes in the server-side playlist, the most significant bit of the wStreamID is toggled, and Media Stream Broadcast packets for the new playlist entry are sent. If the server is set to loop or repeat when the end of stream is reached, the server toggles the wStreamID when it restarts the stream.

5. If the Log URL property is specified in the .nsc file, the client submits a logging message after it receives a Media Stream Broadcast packet with a new wStreamID.

6. Steps 4 and 5 are repeated until the end of the server-side playlist is reached.

7. If the Log URL property is specified in the .nsc file, the client submits a logging message at the end of the stream.

8. The server transmits Beacon packets if it intends to transmit future Media Stream Broadcast packets, and if it wants the client to remain ready to receive the Media Stream Broadcast packets.

The client needs to know the IP multicast address and UDP port to which the Media Stream Broadcast packets will be transmitted. Additionally, the client needs to have a way to associate the ASF packets that are contained in the Media Stream Broadcast packets with an ASF file header.

The .nsc file contains the above information; therefore, the usual way to satisfy these preconditions is by delivering an .nsc file to the client. The MSB Protocol cannot negotiate protocol versioning or capabilities.

8.2.5 Media Stream Broadcast Distribution Protocol

The Media Stream Broadcast Distribution (MSBD) Protocol is used to transfer a stream of audio-visual content from a server to a single client or to multiple clients. For example, the MSBD Protocol might be used for transmitting the digitized sound and images of a lecture from a computer that is encoding the lecture in real time, to another computer that is running the appropriate streaming media server software, such as Windows Media Services 4.1.

The MSBD Protocol also can be used for transmitting a stream from one Windows Media Services server installation to another. Certain versions of WMP also support the MSBD Protocol, and can use it to monitor a live stream; they do this by connecting to a server running Windows Media Services, or directly to a real-time encoder.

8.2.5.1 Media Stream Broadcast Distribution Client-Server Communication Sequence

The following diagram shows a typical communication sequence between the client and the server:

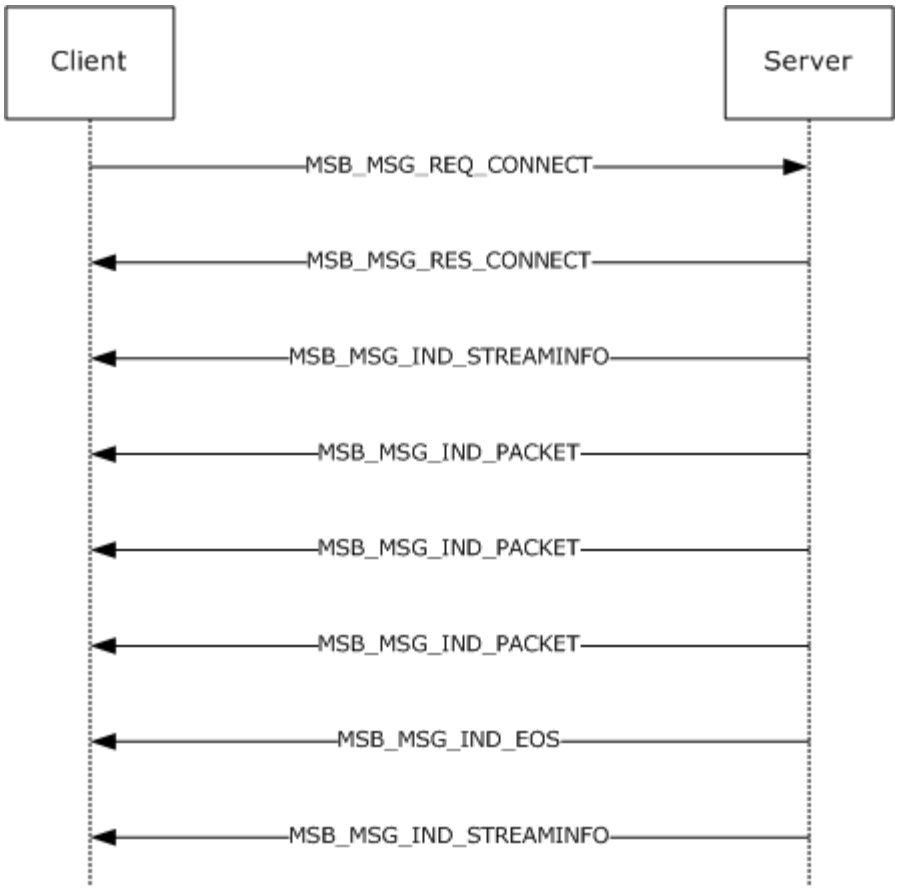

Figure 58: Client/server communication sequence

The steps in this sequence are as follows:

1. The client opens a TCP connection to the server and sends an MSB_MSG_REQ_CONNECT packet.
2. The server responds with an MSB_MSG_RES_CONNECT packet, followed by an MSB_MSG_IND_STREAMINFO packet that describes the media stream.
3. The server sends one or more MSB_MSG_IND_PACKET packets, each of which contains one ASF packet. The packets are sent either over the TCP connection, or using UDP to an IP multicast group.
4. When the stream ends, the server sends an MSB_MSG_IND_EOS packet that is followed by an MSB_MSG_IND_STREAMINFO packet with an empty payload.

During the previous message exchange, the server periodically pings the client using an MSB_MSG_REQ_PING packet.

The MSBD Protocol can be used to distribute ASF packets over an IP-based network. The MSBD Protocol is designed for distribution of content between servers or from an encoder to a server. The MSBD Protocol is not intended for use by an end-user application such as WMP, except for troubleshooting. The MSBD Protocol is supported only in WMP 6.4.

8.2.6 Windows Media HTTP Streaming Protocol

The Windows Media HTTP Streaming Protocol is used to transfer real-time multimedia data (that is, audio and video). This protocol is a streaming protocol, which means that it attempts to facilitate scenarios in which the multimedia data is being transferred and rendered (that is, video displayed and audio played) simultaneously.

This protocol depends on HTTP for the transfer of all protocol messages, including the transfer of the multimedia data. In this specification, the entity that initiates the HTTP connection is referred to as the client, and the entity that responds to the HTTP connection is referred to as the server. The multimedia data flows from the server to the client.

The client can send feedback to the server in the form of HTTP-based request messages. Based on the feedback, the server may, for example, replace the currently transmitted video stream with a lower bit-rate version of the same video stream.

Unlike HTTP, which is a stateless protocol, the Windows Media HTTP Streaming Protocol does maintain state, referred to as session state. The session state allows a server to relate an HTTP request that contains feedback information to an HTTP request used for the transfer of the multimedia data.

8.2.6.1 Nonpipelined Mode of Operation

In its most common mode of operation, feedback from the client is sent to the server using HTTP requests on TCP connections that are separate from the TCP connection that the server uses to transfer the multimedia data to the client. This transfer is referred to as the normal, or nonpipelined mode of operation.

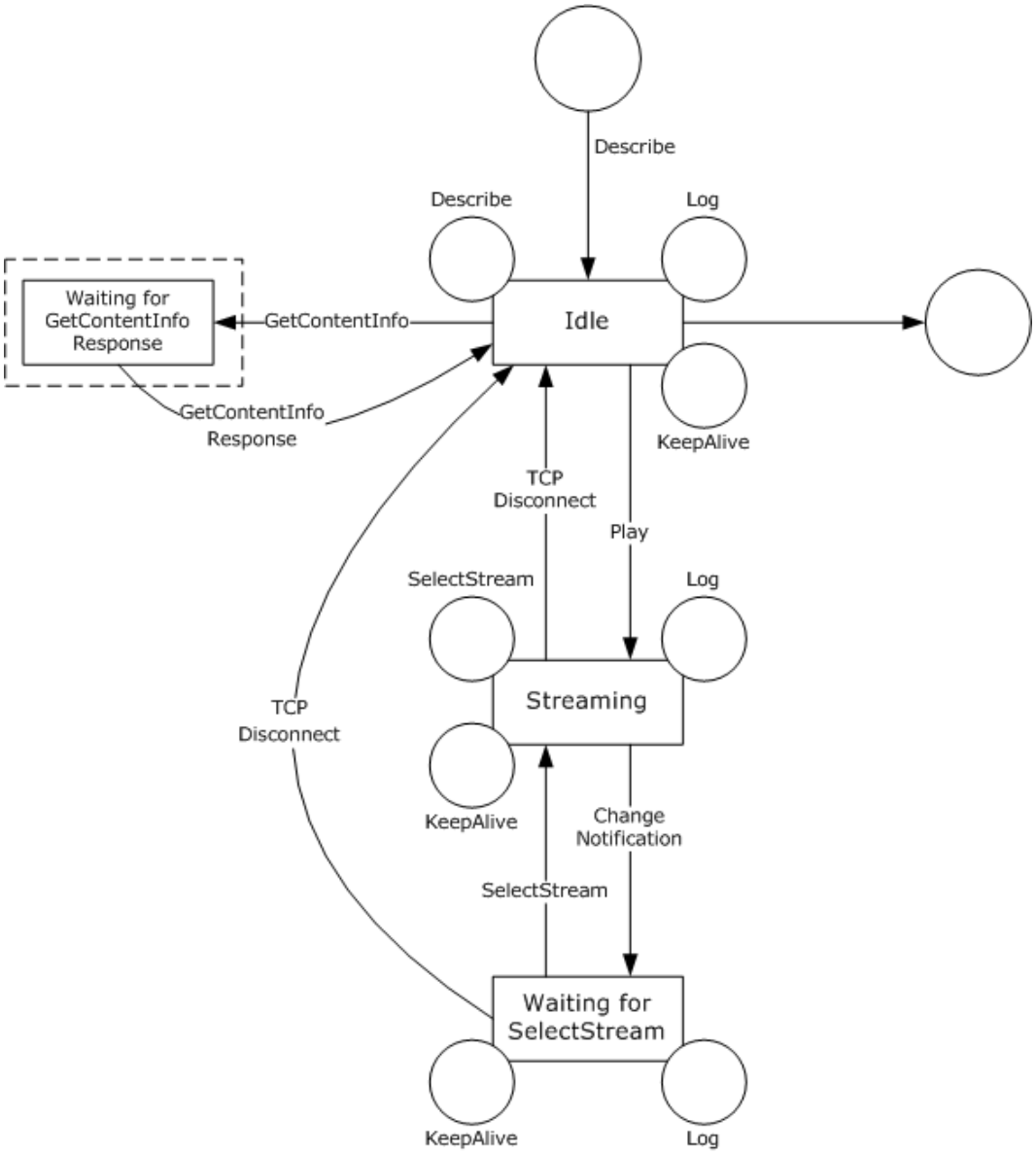

Figure 59: Server states in nonpipelined mode

The steps in this sequence are described as follows:

1. All requests illustrated in the diagram, with the exception of Change Notification, are sent by the client to the server in the context of an HTTP request message using either the Get or Post methods. All server responses are in the context of an HTTP response message.

2. Requests that loop back on a given state indicate that the requests are sent by the client to the server on a separate TCP connection. These requests do not interrupt other requests or sessions, and can be sent multiple times without changing the server state.

3. The presence of a caching proxy server introduces an additional state; the dotted box in the above diagram indicates this state. This state is not applicable during direct server and client interaction.

4. A TCP connection is initiated with a server when a client issues its first DESCRIBE request that contains a uniform resource identifier (URI) that corresponds to the virtual publishing point for content on the server (the content can be stored or live). This first DESCRIBE request initiates the server IDLE state.

5. In response to the initial DESCRIBE request, the server creates a client ID token value that the client uses throughout that individual streaming session. However, the server can send a new client ID that the client must subsequently use.

6. TCP Disconnect occurs when the connection used for the PLAY request and server response is closed. The server may close the TCP connection after it sends the response, or because of an error or time-out that occurred on the server. Clients may also close the TCP connection in stop and resume scenarios. A TCP Disconnect causes the server to change from its current state to the IDLE state.

7. Change Notification, which contains the $C (Stream Change Notification) packet, is sent by the server on the same TCP connection as the PLAY request. A client usually sends DESCRIBE, LOG, SELECTSTREAM, and KEEPALIVE requests on separate connections. A TCP Disconnect on one of these separate connections does not cause a state transition. All transitions from IDLE state to final state are server-specific implementations.

8. By default, when the nonpipelined mode of the protocol is used, TCP connections are not reused. Therefore, the server closes the connection used for the DESCRIBE request when the server sends its response. However, the client can include a Connection: Keep-Alive header with the request to keep this connection open. If the server can comply, it includes a Connection: Keep-Alive header in response, to notify the client. In this case, the client can send subsequent requests, such as the PLAY request, on the same connection. The presence of the Connection: Keep-Alive header does not trigger a state transition or otherwise change the state of the protocol. The Connection: Keep-Alive header is described in Hypertext Transfer Protocol 1.1, [RFC2616].

8.2.6.2 Pipelined Mode of Operation

In another mode of operation, feedback from the client can be sent to the server using HTTP requests on the same TCP connection that the server uses to transfer multimedia data to the client. This transfer is referred to as the pipelined mode of operation.

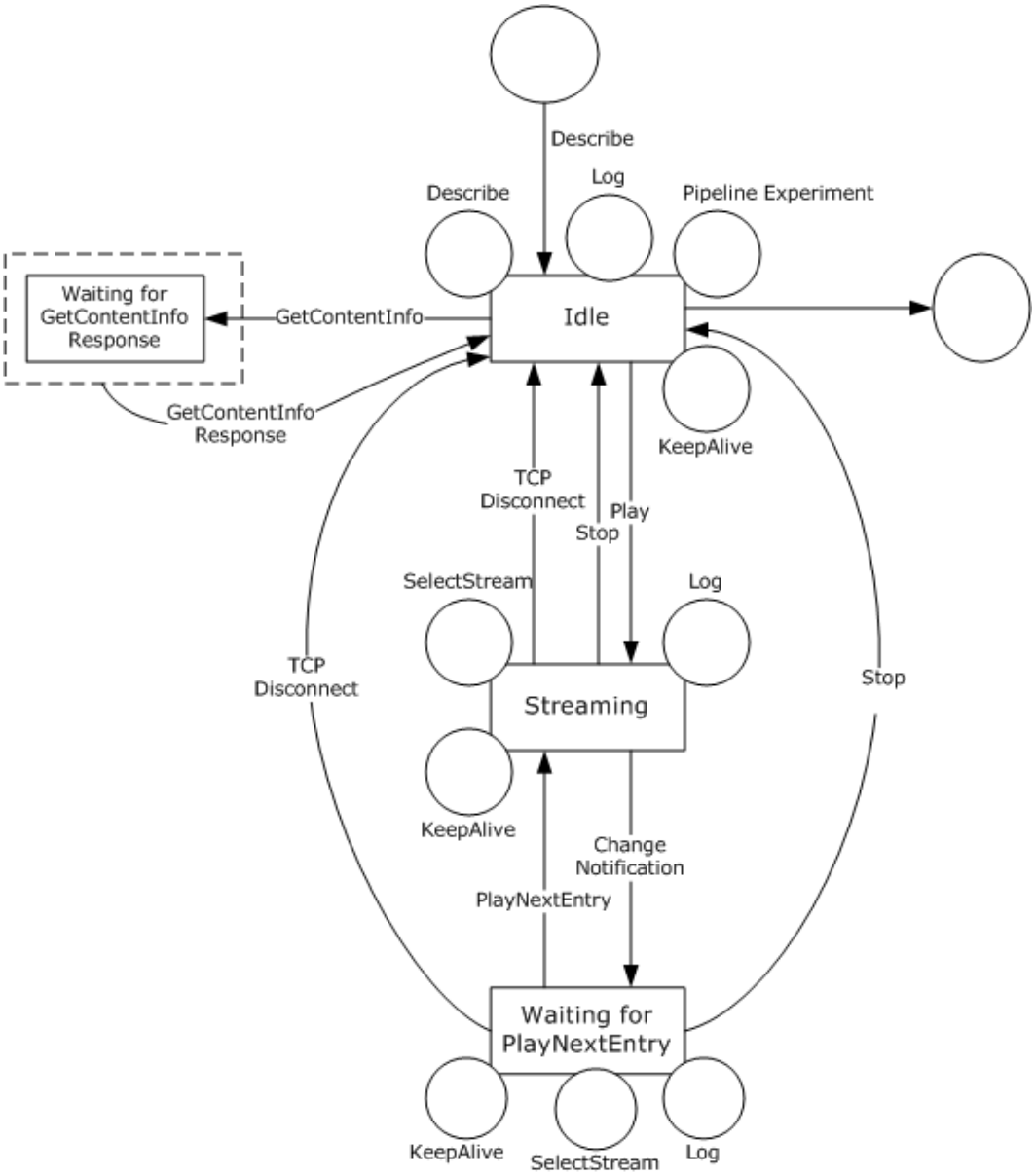

Figure 60: Pipelined mode of operation

The steps in this sequence are described as follows:

1. All requests illustrated in the diagram are sent by the client to the server in the context of an HTTP request message using either the Get, Post, or Options methods. All server responses are in the context of an HTTP response message.

2. Requests that loop back on a state indicate that the requests are sent by the client to the server on the same TCP connection. The response to the client is not returned until the server terminates the previous request.

3. The presence of a caching proxy server introduces an additional state, indicated by the dotted box in the above diagram. This state is not applicable during direct server and client interaction.

4. A TCP connection is initiated with a server when a client issues its first DESCRIBE request that contains a URI that corresponds to the virtual publishing point for content on the server (the content can be stored or live). This first DESCRIBE request initiates the server IDLE state.

5. In response to the initial DESCRIBE request, the server creates a client ID token value that the client uses throughout that individual streaming session. However, the server can send a new client ID that the client must subsequently use.

6. By default, when the pipelined mode of the protocol is used, the TCP connection created with the initial DESCRIBE request is kept open. If the server closes the connection, it includes a Connection: Close header with the response to notify the client. For more information about Connection: Close header, see HTTP 1.1, as specified in [RFC2616]. A TCP Disconnect causes the server to change from its current state to the IDLE state; however, TCP connections are not required to be disconnected for the protocol to be in the IDLE state. All transitions from IDLE state to final state are server-specific implementations.

7. In an auto-reconnect scenario, the server maintains the session state. The client must maintain the point or offset from which to resume playback. This offset is sent in the stream-offset token on the Pragma header with the next PLAY request. The server does not expect a new DESCRIBE request, because it has already sent the $H (Header) packet for the media stream. Also, the server does not need to conduct the packet-pair experiment again to determine the bit rate of the new connection. The sequence is identical to a server resuming from a PAUSE state.

8.2.6.3 Playlist Streaming

The following sequence occurs between a client and server during playlist streaming using predictive stream selection. The sequencing applies to both the pipelined mode and the nonpipelined mode of the protocol.

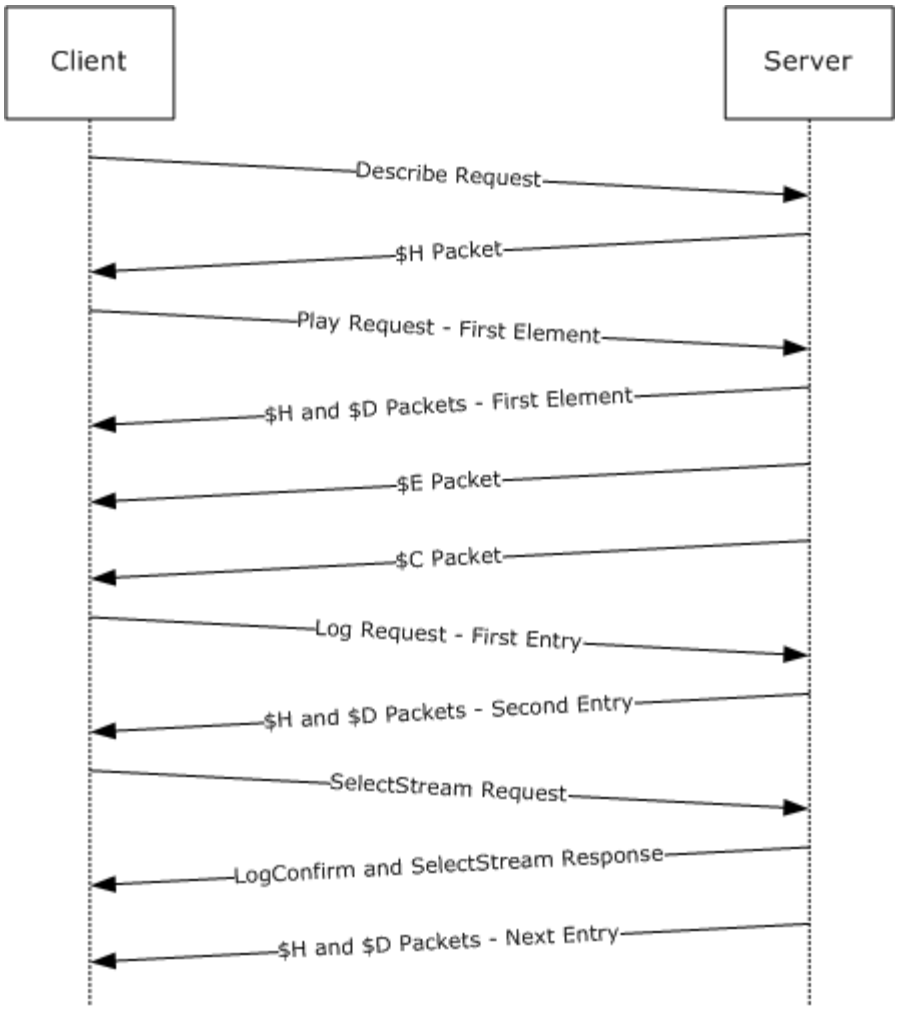

Figure 61: Sequencing during normal playlist streaming

The steps in this sequence are described as follows:

1. The client sends a DESCRIBE request to retrieve the ASF header.
2. The server responds with a $H (Header) packet.
3. The client sends a PLAY request for the file, selecting one or more streams.
4. The server responds with a $H (Header) packet and $D (Data) packets of the first playlist element.
5. When the end of the stream is reached, the server sends a $E (EndOfStream) packet to the client.
6. After all data packets of the first playlist element have been sent to the client, the server sends a $C (Stream Change Notification) packet.
7. The client sends a LOG request for the first element of the playlist.

- When the nonpipelined mode of the protocol is used, the client uses a separate TCP connection to send the LOG request to the server; therefore, the server can respond to the LOG request while still sending stream data through the previously opened connection.
- When the pipelined mode of the protocol is used, the client sends requests to the server on the same TCP connection. The response to the client is not returned until the server terminates the previous request.

8. The server sends $H (Header) and $D (Data) packets of the next element in the playlist.
9. The client sends a SELECTSTREAM request to notify the server that it is rendering the second element of the playlist. The client might select other streams than those the server has predictably selected.
   - When the nonpipelined mode of the protocol is used, the client uses a separate TCP connection for the SELECTSTREAM request. The server responds immediately to this request while still sending stream data through the previously opened connection.
   - When the pipelined mode of the protocol is used, the client uses the same TCP connection; therefore, the server does not respond to the SELECTSTREAM request until the server terminates the previous request.
10. The server responds to the LOG and SELECTSTREAM requests.
11. The server continues to loop playlist element data packets (step 5) until the end of the playlist is reached.

The Windows Media HTTP Streaming Protocol does not provide a mechanism for a client to discover the URL to the server. Thus, before this protocol can be used, a prerequisite is that the client must obtain a URL to the server.

8.2.7 Windows Media HTTP Push Distribution Protocol

The Windows Media HTTP Push Distribution Protocol is used to transfer real-time multimedia data (for example, audio and video) from a client to a server. Push distribution is ideal for broadcasting company meetings or live presentations. In such scenarios, the client is likely to be an encoder software application, perhaps implemented using the Windows Media Encoder Software Development Kit (SDK). For more information, see [MS-WMESDK].

The Windows Media HTTP Push Distribution Protocol depends on HTTP for the transfer of all protocol messages, including the transfer of the multimedia data. In this specification, the entity that initiates the HTTP connection is referred to as the client, and the entity that responds to the HTTP connection is referred to as the server. With the Windows Media HTTP Push Distribution Protocol, multimedia data flows from the client to the server—the opposite of other streaming protocols such as the Windows Media HTTP Streaming Protocol, as specified in [MS-WMSP].

This protocol also may be appropriate if a firewall prevents the server from initiating a TCP connection to the client, or if the client administrator needs to maintain control of the broadcast. The Windows Media HTTP Push Distribution Protocol supports HTTP access authentication, as specified in [RFC2616].

8.2.7.1 Push Distribution General Sequence

The following sequence occurs between a client and a server during a push distribution.

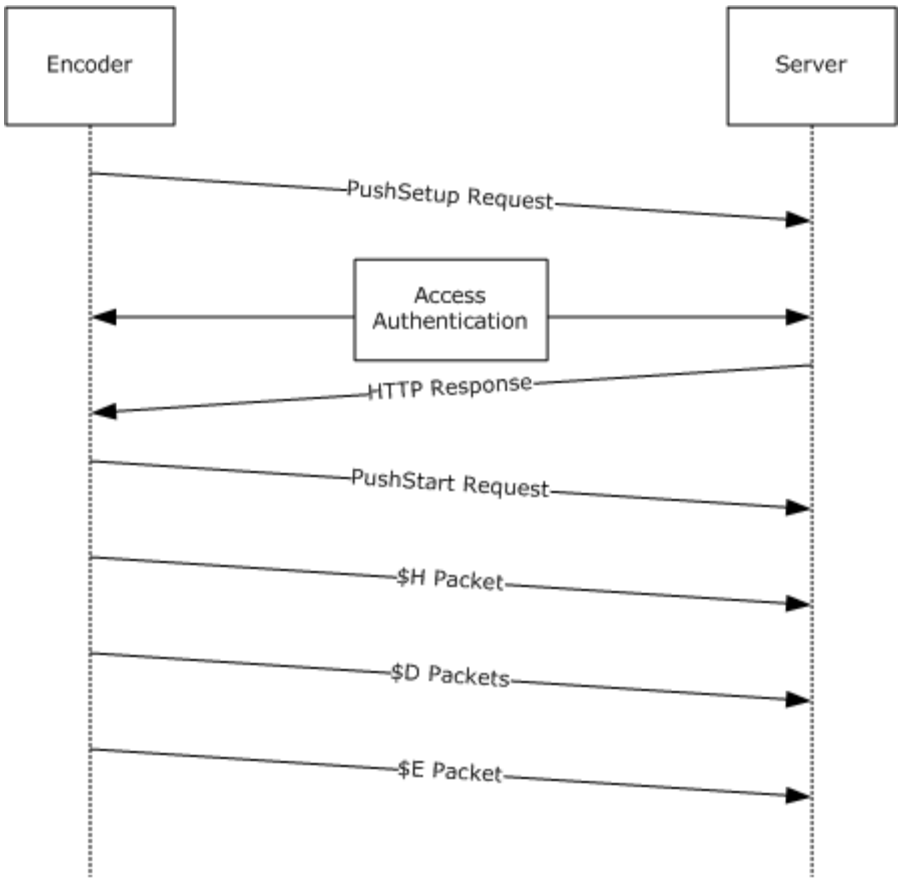

Figure 62: General push distribution sequence

The steps in this sequence are as follows:

1. The client sends a PushSetup request.
2. If the server requires the client to be authenticated, the server and client exchange access authentication HTTP headers as specified in [RFC2616]. (Note that the selected authentication scheme defines the HTTP exchanges required for authentication.)
3. If authentication is not required, or if authentication has succeeded, the server responds with a 204 No Content HTTP response.
4. The client sends a PushStart request, followed by a $H (header) packet and $D (data) packets.
5. After all $D (Data) packets have been sent to the server, the client sends an $E (end-of-stream notification) packet to indicate that the data transfer has been completed.

If there are no other tracks to be sent, the client sends the $E packet with the **Reason** field that is not 0x00000001, and the server closes the TCP connection to the client without sending a PushStart response and deletes the session state. Otherwise, the server must restart the Idle-Timeout timer and wait for either a higher-level triggered event or the reception of a $C, $E, or $F packet from the client.

If there is a playlist entry remaining to be sent, the client will transmit an $E packet with the **Reason** field set to 0x0000001. When the client switches to the new playlist entry, the client will transmit a $C packet.

8.3 Media Streaming Protocols Logical Dependencies and Protocol Stack Views

This section describes the logical dependencies among the media streaming protocols and protocol stack views, as appropriate.

8.3.1 Integrated Media Streaming Protocols

The high-level view of the integrated media-streaming protocols is depicted in the following figure.

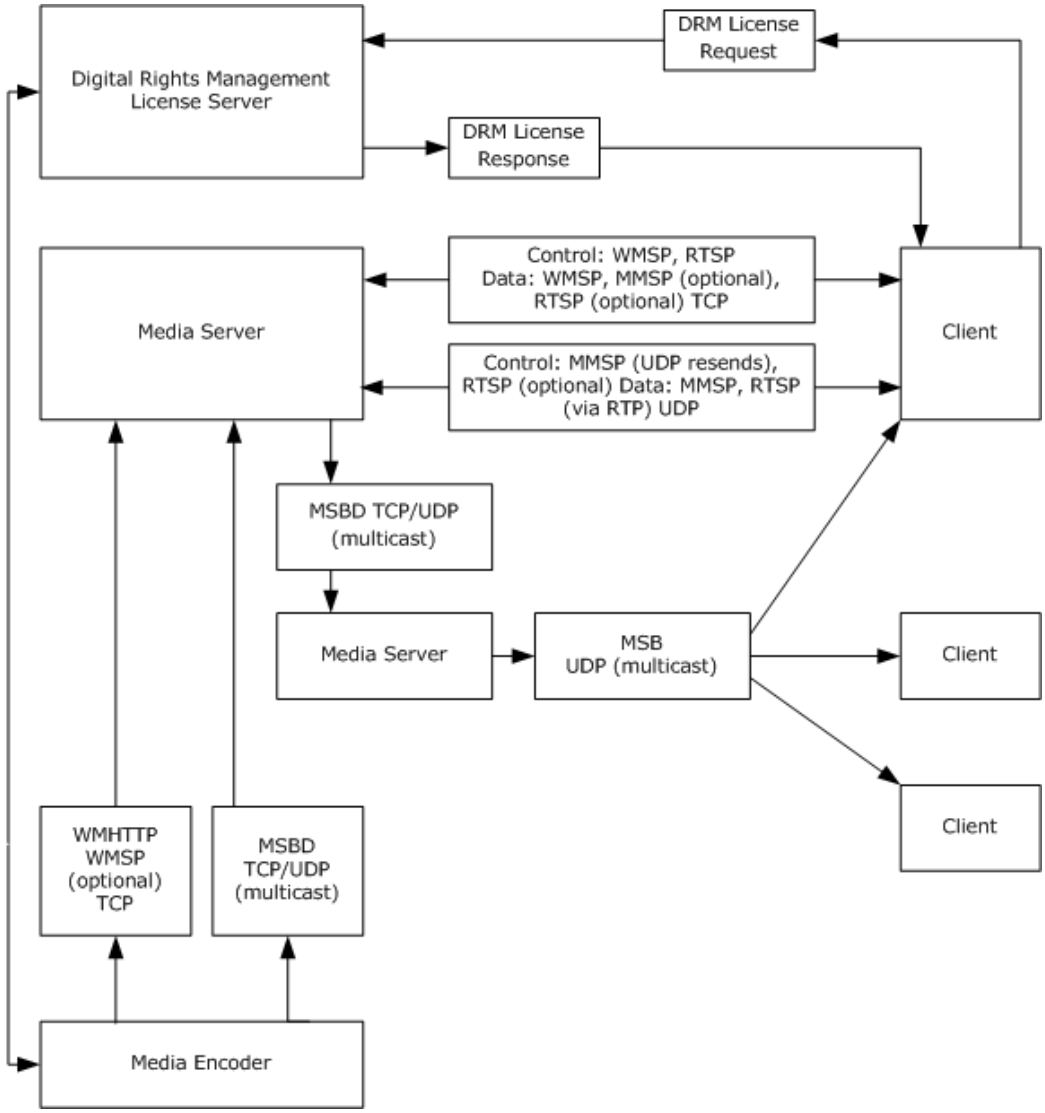

Figure 63: Integrated media streaming protocols logical view

Note The Windows Media Station (.nsc) file contains a base64 encoded ASF header with the relevant DRM objects.

Implementation scenarios for these protocols will be described in detail later in this document. Following are the logical dependencies of each core protocol.

8.3.1.1 RTSP: Windows Media Extensions Logical Dependencies and Relationship to Other Protocols

The RTSP Windows Media Extensions rely on TCP to control the streaming media session. Although UDP is also allowed, it is rarely used for this purpose.

RTSP uses the Session Description Protocol (SDP) syntax to describe the properties of content.

The RTSP Windows Media Extensions use RTP for the delivery of multimedia data, and use RTCP for RTP feedback and statistics. RTP and RTCP packets are transmitted over either UDP or TCP. It is possible to transmit some RTP streams over UDP and other RTP streams over TCP.

The RTSP Windows Media Extensions depend on ASF. ASF is used both in the SDP syntax and in the payload of the RTP packets.

The RTSP Windows Media Extensions are similar in functionality to the MMS Protocol (for more information, see [MS-MMSP]). However, the RTSP Windows Media Extensions provide additional functionality that is not available in MMS.

The RTSP Windows Media Extensions are similar in functionality to the Windows Media HTTP Streaming Protocol, as specified in [MS-WMSP]. However, in that protocol, the delivery of ASF packets is limited to TCP only.

8.3.1.2 MMS Protocol Logical Dependencies and Relationship to Other Protocols

The MMS Protocol relies on TCP for the connection that controls the streaming media session. Both the client and the server send MMS Protocol messages over the TCP connection. The multimedia data that is being transferred by the server is sent over either TCP or UDP. The client also relies on UDP to send requests that ask the server to resend lost UDP packets.

The MMS Protocol is similar in functionality to the RTSP Windows Media Extensions (for more information, see [MS-RTSP]). However, the RTSP Windows Media Extensions support functionality that is not available in MMS.

8.3.1.3 Media Stream Broadcast Protocol Relationship to Other Protocols

Media Stream Broadcast (MSB) Protocol packets are encapsulated in UDP. The UDP packets can be transmitted over either IP version 4 (IPv4) or IP version 6 (IPv6). The MSB Protocol packets are used to transport ASF packets. In addition, the MSB Protocol uses the forward error correction (FEC) algorithm, as specified in [RFC3452], for error detection.

8.3.1.4 Media Stream Broadcast Distribution Protocol Logical Dependencies and Relationship to Other Protocols

MSBD Protocol packets are encapsulated in TCP. However, one MSBD Protocol packet type, MSB_MSG_IND_PACKET, can also be encapsulated in the UDP. The UDP encapsulation mode is used only to transmit packets to an IPv4 multicast group.

The UDP encapsulation mode of this protocol might not be suitable for content that uses large ASF data packets. Large ASF data packets might cause the UDP packets to be fragmented into multiple

IP datagrams, and fragmentation of IP datagrams might be undesirable. In such cases, it is recommended to use the TCP encapsulation mode instead.

8.3.1.5 Windows Media HTTP Streaming Protocol Logical Dependencies and Relationship to Other Protocols

The Windows Media HTTP Streaming Protocol depends on HTTP 1.0, as specified in [RFC1945]. The pipelined mode of the protocol can be used only if the client, the server, and any intermediate HTTP proxy servers support the pipelining feature of HTTP 1.1, as specified in [RFC2616].

This protocol can be used instead of the MMS Protocol as specified in [MS-MMSP]. This protocol can also be used instead of the RTSP Windows Media Extensions, as specified in [MS-RTSP]. However, it should be noted that although these two other protocols allow the multimedia data to be transmitted over either UDP or TCP, the Windows Media HTTP Streaming Protocol allows multimedia data to be transmitted only over TCP (because HTTP always uses TCP). The Windows Media HTTP Streaming Protocol is a good choice, where the multimedia data needs to pass through a proxy or a firewall as it uses HTTP, which is typically configured to pass through the proxy or firewall.

8.3.1.6 Windows Media HTTP Push Distribution Protocol Logical Dependencies and Relationship to Other Protocols

The Windows Media HTTP Push Distribution Protocol depends on the Hypertext Transfer Protocol (HTTP/1.1), as specified in [RFC2616].

This protocol also utilizes headers, packet types, and other components from the Windows Media HTTP Streaming Protocol, as specified in [MS-WMSP].

8.3.2 Stack Views

The relationships between these protocols are best represented in terms of logical dependencies. Therefore, stack views are not provided.

8.4 Implementation Scenarios

This section provides two scenarios that illustrate common functionality that can be implemented with the protocols included in the Media Streaming Services. The protocols involved in these scenarios include the Real-Time Streaming Protocol (RTSP): Windows Media Extensions, as specified in [MS-RTSP], and the Session Description Protocol (SDP) Extensions, as specified in [MS-SDP], together with other general networking protocols used in the course of standard TCP communication between the client and server. The functionality illustrated in the scenarios is as follows:

- A Windows Vista operating system client downloads and plays a video file from a Windows Server 2008 operating system using Windows Media Player.
- A Windows Vista client downloads and plays a video file from a Windows Server 2003 operating system using Windows Media Player.

The scenarios illustrate streaming video and audio being served from Windows servers configured as streaming media servers.

It should be noted that the following scenarios depict processes that occur over TCP, and thus assume various TCP acknowledgment frames throughout the conversation that are not necessarily depicted in the trace details.

8.4.1 Scenario 1 - Vista Client Downloads Streaming Media from Windows Server 2008

8.4.1.1 Scenario Overview

In the following scenario, a Windows Vista operating system client downloads and plays a video file from Windows Server 2008 operating system using Windows Media Player.

8.4.1.2 Scenario Configuration

This scenario was implemented with a Windows Server 2008 operating system server, and a Windows Vista operating system client machine configured as follows:

Server Configuration

- Machine: WS200801
- Operating system: Windows Server 2008 Beta 3
- IP Address: 10.0.10.4
- Subnet Mask: 255.255.255.0
- Streaming Media Services role configured
- Created an on-demand publishing point that contained a playlist of two media files (a Windows Media Audio [.wma] file and a Windows Media Video [.wmv] file).

Client Configuration

- Machine: ClientVista
- Operating system: Windows Vista with latest updates
- IP Address: 10.0.10.11
- Subnet Mask: 255.255.255.0

Network Topology

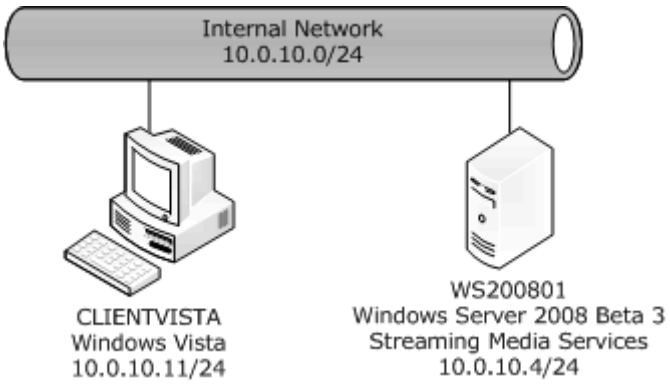

Figure 64: Network topology

8.4.1.3 Setting Up the Trace

1. Start WS200801.
2. Log on as local Administrator.
3. Start ClientVista.
4. On ClientVista, start NetMon 3.1.
5. BEGIN TRACE.
    1. Open Windows Media Player.
    2. On the File menu, click Open URL.
    3. Type mms://WS200801/, and then click OK.
    4. Click Next.
6. STOP TRACE.
7. Close WMP.

8.4.1.4 Netmon Trace Digest

- The client issues a request to the server for a MAC address (hardware address) using the Address Resolution Protocol (ARP), and the server responds with its address:

```
CLIENTVISTA WS200801 ARP ARP: Request, 10.0.10.11 asks for 10.0.10.4
WS200801 CLIENTVISTA ARP ARP: Response, 10.0.10.4 at 00-03-FF-A3-46-6B
```

- The process is then repeated when the server requests the MAC address of the client:

```
WS200801 CLIENTVISTA ARP ARP: Request, 10.0.10.4 asks for 10.0.10.11
CLIENTVISTA WS200801 ARP ARP: Response, 10.0.10.11 at
00-03-FF-AC-46-6B
```

- The client issues an RTSP request to the server. This request is sent before the client asks the server to start streaming the content.

```
{TCP:70, IPv6:66}FE80:0:0:0:7DFF:F7C3:7D5A:C9C9
FE80:0:0:0:A8D3:6D66:D1FF:C3A6 RTSP RTSP:  REQUEST, DESCRIBE
rtsp://WS200801/PublishingPoint1
```

- SDP is used to describe streaming media initialization parameters:

```
FE80:0:0:0:A8D3:6D66:D1FF:C3A6FE80:0:0:0:7DFF:F7C3:7D5A:C9C9SDPSDP:
Session Name=24 Hours of Exchange Server 2007 (Part 01 of 24):
Integration of Exchange Server 2007 and Active Directory, Version=0
```

- TCP is used to control the RTSP requests and responses:

```
FE80:0:0:0:7DFF:F7C3:7D5A:C9C9FE80:0:0:0:A8D3:6D66:D1FF:C3A6 TCP
TCP: Flags=....A..., SrcPort=49161, DstPort=RTSP(554), Len=0,
Seq=3911700222, Ack=409635840, Win=258 (scale factor 8) = 66048

FE80:0:0:0:7DFF:F7C3:7D5A:C9C9FE80:0:0:0:A8D3:6D66:D1FF:C3A6TCP
TCP: Flags=....A..., SrcPort=49161, DstPort=RTSP(554), Len=0,
Seq=3911700222, Ack=409641600, Win=258 (scale factor 8) = 66048
```

- An RTSP SETUP request follows. This request specifies how each media stream is to be transported.

```
FE80:0:0:0:7DFF:F7C3:7D5A:C9C9FE80:0:0:0:A8D3:6D66:D1FF:C3A6 RTSP RTSP:
REQUEST, SETUP rtsp://WS200801/PublishingPoint1/rtx
```

- The server responds with the protocol version:

```
FE80:0:0:0:A8D3:6D66:D1FF:C3A6FE80:0:0:0:7DFF:F7C3:7D5A:C9C9 RTSP RTSP:
RESPONSE, RTSP/1.0, Status Code = 200 - OK
```

- The SET_PARAMETER method requests to set the value of a parameter for a presentation or stream:

```
FE80:0:0:0:7DFF:F7C3:7D5A:C9C9FE80:0:0:0:A8D3:6D66:D1FF:C3A6 RTSP RTSP:
REQUEST, SET_PARAMETER rtsp://WS200801/PublishingPoint1
```

- The client issues a PLAY request to ask the server to start streaming RTP packets for the currently selected streams:

```
FE80:0:0:0:7DFF:F7C3:7D5A:C9C9FE80:0:0:0:A8D3:6D66:D1FF:C3A6 RTSP RTSP:
REQUEST, PLAY rtsp://WS200801/PublishingPoint1
```

- Using RTP, the streaming payload is then sent to the client:

```
FE80:0:0:0:A8D3:6D66:D1FF:C3A6FE80:0:0:0:7DFF:F7C3:7D5A:C9C9 RTP RTP:
PayloadType = Dynamic, SSRC = 1304733200, Seq = 9879, TimeStamp = 0, Mark

FE80:0:0:0:A8D3:6D66:D1FF:C3A6FE80:0:0:0:7DFF:F7C3:7D5A:C9C9 RTP RTP:
PayloadType = Dynamic, SSRC = 1304733200, Seq = 9880, TimeStamp = 0, Mark
```

8.4.2 Scenario 2 - Vista Client Downloads Streaming Media from Windows Server 2003

8.4.2.1 Scenario Overview

In the following scenario, a Windows Vista operating system client downloads and plays a video file from Windows Server 2003 operating system using Windows Media Player.

8.4.2.2 Scenario Configuration

This scenario was implemented with a Windows Server 2003 operating system machine, and a Windows Vista operating system client machine configured as follows:

Server Configuration

- Machine: WS200301
- Operating system: Windows Server 2003 R2 operating system SP2 with latest updates
- IP Address: 10.0.10.2
- Subnet Mask: 255.255.255.0
- Streaming Media Server role configured
- Created an on-demand publishing point that contained a playlist of two media files (a .wma file and a .wmv file).

Client Configuration

- Machine: ClientVista
- Operating system: Windows Vista with latest updates
- IP Address: CLIENTVISTA
- Subnet Mask: 255.255.255.0

Network Topology

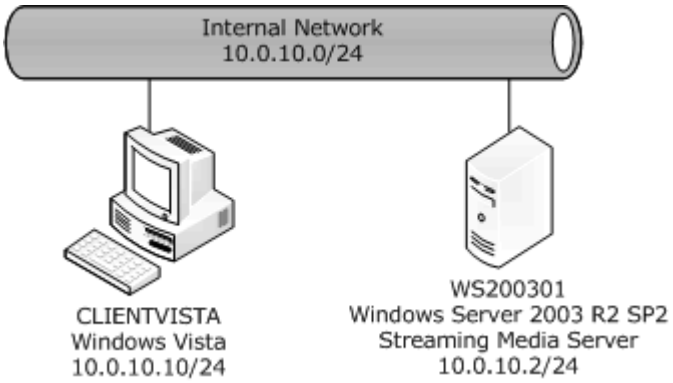

Figure 65: Network topology

8.4.2.3 Setting up the Trace

1. Start WS200301.
2. Log on as local Administrator.
3. Start ClientVista.
4. On ClientVista, start NetMon 3.1.

5. BEGIN TRACE.
    1. Open WMP.
    2. On the File menu, click Open URL.
    3. Type mms://WS200301/, and then click OK.
    4. Click Next.
6. STOP TRACE.
7. Close WMP.

8.4.2.4 Netmon Trace Digest

- The client issues a request to the server for a MAC address (hardware address) using the Address Resolution Protocol (ARP), and the server responds with its address:

```
CLIENTVISTA WS200301 ARP ARP: Request, CLIENTVISTA asks for 10.0.10.2

WS200301 CLIENTVISTA ARP ARP: Response, 10.0.10.2 at 00-03-FF-AE-46-6B
```

- The client issues an RTSP request to the server. This request is sent before the client asks the server to start streaming the content:

```
{TCP:38, IPv4:1} CLIENTVISTA WS200301 RTSP RTSP:  REQUEST, DESCRIBE
  rtsp://WS200301/PublishingPoint1
```

- SDP is used to describe streaming media initialization parameters:

```
WS200301 CLIENTVISTA SDP SDP: Session Name=24 Hours of Exchange Server
 2007 (Part 01 of 24): Integration of Exchange Server 2007 and Active
Directory, Version=0
```

- As with the previous scenario, TCP is used for control of the RTSP requests and responses:

```
WS200301 CLIENTVISTA TCP TCP: Flags=....A..., SrcPort=RTSP(554),
  DstPort=49161, Len=0, Seq=2236623144, Ack=3287479131, Win=65097
  (scale factor 0) = 65097

CLIENTVISTA WS200301 TCP TCP: Flags=....A..., SrcPort=49161,
  DstPort=RTSP(554), Len=0, Seq=3287479131, Ack=2236626064, Win=256
  (scale factor 8) = 65536

CLIENTVISTA WS200301 TCP TCP: Flags=....A..., SrcPort=49161,
  DstPort=RTSP(554), Len=0, Seq=3287479131, Ack=2236630444, Win=256
  (scale factor 8) = 65536
```

- An RTSP SETUP request follows. This request specifies how each media stream is to be transported.

```
CLIENTVISTA WS200301 RTSP RTSP: REQUEST, SETUP
  rtsp://WS200301/PublishingPoint1/rtx
```

- The server responds with the protocol version:

```
WS200301 CLIENTVISTA RTSP RTSP: RESPONSE, RTSP/1.0,
Status Code = 200 - OK
```

- The SET_PARAMETER method requests to set the value of a parameter for a presentation or stream:

```
CLIENTVISTA WS200301 RTSP RTSP: REQUEST, SET_PARAMETER
  rtsp://WS200301/PublishingPoint1
```

- The client issues a PLAY request to ask the server to start streaming RTP packets for the currently selected streams:

```
CLIENTVISTA WS200301 RTSP RTSP: REQUEST, PLAY
  rtsp://WS200301/PublishingPoint1
```

- Using RTP, the streaming payload is then sent to the client:

```
WS200301 CLIENTVISTA RTP RTP: PayloadType = Dynamic,
  SSRC = 3091800863, Seq = 34626, TimeStamp = 0, Mark

WS200301 CLIENTVISTA RTP RTP: PayloadType = Dynamic,
  SSRC = 3091800863, Seq = 34629, TimeStamp = 1114, Mark
```

9 Multiplayer Games Services Protocols

This section provides an overview of the Multiplayer Games Services (MGS) protocols. It provides a high-level conceptual overview of the core MGS protocols, classifying them broadly according to how the multiplayer game data is transferred over the network.

9.1 Multiplayer Games Services Overview

The Multiplayer Games Services protocols provide DirectPlay 4 and DirectPlay 8 implementation functionality as follows:

- Game lobby functionality - The game lobby is the screen that users see while connecting to a game server, making configurations to their game properties, and waiting for other players to join.
- Game state information - Game status messages are typically provided to inform users of game server events, such as when another player joins the game.
- Side channels between game users - Most multiplayer games provide some communication capabilities between players, usually text chat, for team coordination and taunting of opponents.
- Remote administration functionality to support covered Multiplayer Games services - The remote administration capabilities allow users to change the configuration of the game server.

9.1.1 Protocol List

Multiplayer Games Services (MGS) consists of eight protocol specifications. The two core protocols that represent the base MGS functionality are the DirectPlay Protocol, which is presented from the perspective of the DXDiag application, and DirectPlay Voice Protocol. The DirectPlay Protocol is also broken down into six protocols that support these core protocols. The six supporting protocols supply additional functionality required for an implementation of the core protocols, or provide core networking functionality. The supporting protocols that are required for an implementation of an MGS core protocol are identified in the Normative References section of the MGS core protocol technical specification.

The DirectPlay Protocol is broken down into a series of specifications which describe both DirectPlay 4 and DirectPlay 8. DirectPlay 4 and DirectPlay 8 are further distinguished in their technical specifications regarding the manner in which the protocol provides for reliable and unreliable communications. The technical specifications for these protocols ([MC-DPL4R] and [MC-DPL8R]) document how the protocols provide game data transmission for both reliable and unreliable transmission. The core and service providers technical specifications for these protocols ([MC-DPL4CS] and [MC-DPL8CS]) describe how the protocols provide the functionality necessary for multiplayer game communication, including the ability to create and manage game sessions.

The following tables list the protocols included in Multiplayer Games Services. The tables do not identify the data structures, file structures, or algorithms upon which these protocols depend. The details for these technical dependencies are documented in the technical specifications for each protocol, which can be located by clicking on the document short name.

Table: MGS Core Protocols

Protocol Name	Protocol Description	Document Short Name
DirectPlay DXDiag Usage Protocol	Specifies implementation details regarding the DirectPlay Protocol that are presented from the perspective of how the protocol is used by the DXDiag application. DXDiag.exe is a Windows diagnostic utility that is used to test Microsoft DirectX functionality, including DirectPlay traffic.	[MS-DPDX]
DirectPlay Voice Protocol	Describes voice communications for applications that use the DirectPlay Protocol to communicate.	[MC-DPLVP]

Table: MGS Supporting Protocols

Protocol Name	Protocol Description	Document Short Name
DirectPlay 4 Protocol: Core and Service Providers	Specifies the core protocol services of the DirectPlay 4 Protocol which is used to facilitate communication between computer games. The DirectPlay 4 Protocol enables the implementation of functions to enumerate hosted game sessions and players, to add and remove game players, and to interchange data between game instances.	[MC-DPL4CS]
DirectPlay 4 Protocol: Reliable	Pertains to the DirectPlay 4 Protocol and describes functionality related to the reliable delivery of DirectPlay 4 messages.	[MC-DPL4R]
DirectPlay 8 Protocol: Core and Service Providers	Specifies the core protocol services of the DirectPlay 8 Protocol which provides functionality necessary for multiplayer game communication.	[MC-DPL8CS]
DirectPlay 8 Protocol: Reliable	Pertains to the DirectPlay 8 Protocol and describes functionality related to the delivery of mixed messages, both reliable and unreliable, over existing datagram protocols such as UDP.	[MC-DPL8R]
DirectPlay 8 Protocol: Host and Port Enumeration	Pertains to the DirectPlay 8 Protocol and describes the technology available for enumerating DirectPlay 8 hosts and ports.	[MC-DPLHP]
DirectPlay 8 Protocol: NAT Locator	Pertains to the DirectPlay 8 Protocol and describes technology available for the support of network environments that involve network address translation (NAT).	[MC-DPLNAT]

The following sections provide some general DirectPlay Protocol concepts, and specific views to show the physical and logical relationships for the core protocols in MGS.

9.2 Multiplayer Games Services Functionality

This section describes general MGS functionality protocols concepts, and provides background for the Multiplayer Games model of Windows.

A multiplayer application has these characteristics:

- Two or more individual users, each with a game client on their computer.
- Network links that enable the users' computers to communicate with each other, perhaps through a centralized server, but typically through peer-to-peer sessions. One peer computer becomes the host for the multiplayer game and the other peers become clients.

DirectPlay 8 provides a layer that primarily isolates a gaming application from the underlying network. Most gaming applications can use the DirectPlay protocols instead of TCP/IP. The DirectPlay 4 and DirectPlay 8 protocol documents describe the messages and responses sent back and forth over the network.

DirectPlay 8 provides features that simplify the process of implementing aspects of a multiplayer application, including:

- Creates and manages peer-to-peer and client/server sessions.
- Manages users and groups within a session.
- Manages messaging between the members of a session over different network links and varying network conditions.
- Enables applications to interact with lobbies.
- Enables users to communicate with each other by voice.

Section 9.4 of this document provides an example that illustrates the core protocols.

9.2.1 DirectPlay 8 DXDiag Usage Protocol

The DirectPlay 8 DXDiag Usage protocol implementation details regarding the DirectPlay Protocol are presented from the perspective of how the protocol is used by the DXDiag application. The complete DirectPlay8 protocol for handling network messaging for networked gaming is described in [MC-DPL8R], [MC-DPL8CS], [MC-DPLHP], and [MC-DPLNAT]. DXDiag.exe (DXDiag) is a diagnostic utility included with Windows. DXDiag can be used to test Microsoft DirectX functionality, which includes DirectPlay 8.

9.2.1.1 DirectPlay 4 and 8 Messaging

Two types of messages are supported:

- Reliable messages have guaranteed delivery to the target application.
- Sequential messages are received by the target application in the same order in which they are sent.

Games use messaging for various purposes, each with different demands. To support this range of messaging, the DirectPlay 8 protocol designates a message as one of four categories:

- Reliable and sequential
- Unreliable and sequential
- Reliable and non-sequential
- Unreliable and non-sequential

9.2.1.2 How DXDiag Uses DirectPlay 8

The DXDiag.exe diagnostic utility tests Microsoft DirectX functionality, including DirectPlay 8 traffic. DXDiag uses the two types of DirectPlay 8 packets: session and transport. These packets are transported by the User Datagram Protocol (UDP), as specified in [RFC768].

In the DXDiag chat session, client identifiers are contained in a name table. When a new client joins a chat session, the client receives a name table that lists all the clients currently in the session. When a client departs the session, the identifier for that client is removed from the name table. The name table is kept current through the use of a version number.

DirectPlay 8 is a sliding window protocol. It requires the receiver to acknowledge received UDP packets before more packets are transmitted. An acknowledgment can be conveyed in one of two ways: either bundled in back traffic sent from the receiver, or when no back traffic is flowing, sent from the receiver as a dataless Selective Acknowledgment (SACK) packet.

When the acknowledgment is bundled in back traffic, fields in the header indicate the sequence number of the next expected packet. This acknowledges that all packets with sequence numbers less than the specified number have been received correctly. If an acknowledgment is not received in a specified amount of time (which is derived from a round-trip time calculated using keep-alive packets), the original packet is resent with the same sequence number that was previously assigned. If the original sender specifies POLL (ACK now) in the packet header, the receiver must immediately acknowledge the packet when it arrives.

The DirectPlay 8 protocol was designed for multiplayer network gaming, but not for other uses of client/server or peer-to-peer messaging. The protocol and its data structures were developed and tuned for gaming uses.

The DirectPlay 8 protocol does not provide file transfer or security. It is expected that security is provided by other technologies, such as IPsec. It is also not intended as a generic replacement for TCP/IP.

9.2.1.3 DirectPlay 8 Packets

The DirectPlay 8 protocol includes two types of DirectPlay 8 packets: session and transport. DirectPlay 8 uses session packets to locate sessions and to test network paths. DirectPlay 8 uses transport packets also, and these include: command, user data, and acknowledgment packet types.

9.2.1.4 Message Processing Events and Sequencing Rules

DXDiag allows a client and server to create a chat session. Sequencing depends upon the number and role of gaming participants, as shown in the following examples.

9.2.1.4.1 Example 1: Client Joins a DirectPlay 8 Session with No Other Clients

1. The client sends a session packet in search of a chat session.
2. The server responds to the client session packet.
3. The client requests a connection using a user data packet.
4. The server responds with a user data packet. The client issues a user data packet to acknowledge the connection.
5. The server and client exchange request packets. The client sends user information.
6. The client sends an acknowledgment to the server.
7. The server sends an instruction to form a connection.
8. The client responds. The server sends a version synchronization packet.
9. The client acknowledges the packet.

9.2.1.4.2 Example 2: Client Joins a DirectPlay 8 Session with Multiple Other Clients

1. The client sends a session packet in search of a chat session.
2. The server responds to the client session packet.
3. The client requests a connection using a user data packet.
4. The server responds with a user data packet.
5. The client issues a user data packet to acknowledge the connection.
6. The server and client exchange request packets.
7. The client sends user information.
8. The server, after receiving the user info packet, alerts existing clients to the existence of the new client.
9. The new client tests the network path to the existing clients.
10. The new client sends a packet to acknowledge the session information previously received from the server.
11. The existing clients send the server a name table packet.
12. The server commands all clients to resynchronize their name tables.
13. The server sends a packet to each existing client to instruct it to connect to the new client.
14. Each client initiates the connection with the new client using a connect packet.
15. The new client responds with an accept packet.
16. The existing client acknowledges the connection with an accept packet.
17. The new client and the existing client exchange keep alive packets, and the existing client transmits its user identifier to the new client.
18. Finally, all clients acknowledge the session server and each other using acknowledgment packets.

9.2.1.4.3 Example 3: Client Disconnects from a Chat Session

When a client disconnects, the server makes an announcement to the remaining clients, and then the departing client disconnects.

1. The client issues an end-of-stream packet.
2. The server responds with four acknowledgment packets and an end-of-stream packet (in any order).
3. The client returns four acknowledgment packets before disconnecting.
4. If other clients are present in the session, the server sends each one a destroy-player packet to remove the departing client from its name table. (This can happen before the server receives the final acknowledgment packet from the departing client.)

5. Each remaining client sends the server an integrity-check packet and an acknowledgment packet to acknowledge the removal of the client.

9.2.1.4.4 Example 4: Server Disconnects from a Chat Session

A server can leave without destroying the chat session. The DirectPlay 8 DXDiag Usage protocol allows hosting to migrate to another member currently in the session. By default, the DXDiag utility transfers the hosting to the client that has been in the chat longest.

1. The hosting migration begins with the server sending an end-of-stream packet to all clients in the session.
2. The server responds with four acknowledgment packets and an end-of-stream packet (in any order).
3. The server sends four acknowledgment packets to each client, and disconnects.
4. The client that has been in the session the longest becomes the new server, and contacts each client with a migration user data packet.
5. Each client sends a name table packet to the new server.
6. The new server ends the migration sequence with migration and destroys player packets.
7. The destroy-player packet identifies the previous host to remove from the client session list.

9.2.2 DirectPlay Voice Protocol

The DirectPlay Voice Protocol enables DirectPlay clients within a DirectPlay session to communicate voice. The exchange is coordinated by a voice server and works independent of the version of DirectPlay in use. The protocol depends on the underlying DirectPlay session to handle connectivity and transport between the voice clients and the voice server.

The DirectPlay Voice Protocol provides a virtual session that exists within the DirectPlay session. It requires an existing DirectPlay session to operate but maintains its own session coordinated by the voice server. This means that not every DirectPlay client is required to participate. The voice server maintains the list of voice clients in the session and coordinates the distribution of this information.

The voice server and voice clients use the Connection Subprotocol to establish initial connectivity and then the voice server tells the voice clients which voice session subprotocol to use based on the voice session type determined at initialization of the voice server.

The DirectPlay Voice Protocol also provides an extension called the Host Migration Extension. This extension allows sessions running under the Peer Voice Session Subprotocol to elect a new voice server if the existing voice server becomes unavailable.

The upper layer records audio from the user and uses a CODEC to create an encoded voice stream. The encoded voice stream is broken into voice bursts and then further into speech messages. The exact methodology of distributing speech messages varies depending on the type of voice session subprotocol.

Only a single CODEC is used for any given session. When a voice server is started, a CODEC is chosen to be used by that voice server. Any voice clients that wish to connect to the voice server MUST support the CODEC chosen by the voice server.

9.3 DirectPlay 8 Protocol Physical Dependencies and Stack View

This section provides conceptual authentication areas to describe the logical dependencies among the DirectPlay 8 protocol and protocol stack views, as appropriate.

9.3.1 Logical Relationships

The DirectPlay 8 and DirectPLay 8 Voice protocols have a dependency on the User Datagram Protocol (UDP) and TCP/IP for the transport layer. Use of UDP and TCP/IP depends on a variety of factors, including the networking context or type of message. For more detailed information see the [MC-DPLVP], [MC-DPL8R], [MC-DPL8CS], [MC-DPLHP], and [MC-DPLNAT] specifications. The DXDiag is another application which uses DirectPlay. No other protocols depend on DirectPlay 8.

9.3.2 Stack View

The DirectPlay 8 stack view is shown here.

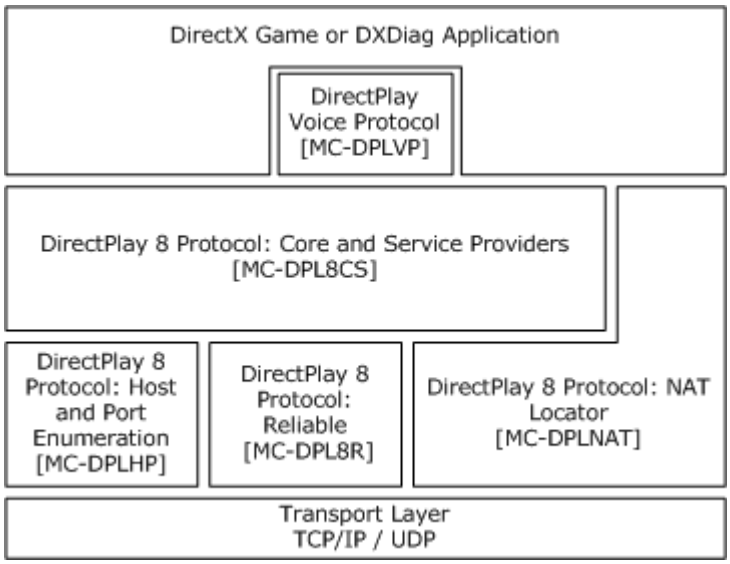

Figure 66: DirectPlay 8 protocol stack

9.4 Implementation Scenario

This section provides a scenario that illustrates common functionality that can be implemented with the protocol included in Multiplayer Games Services. The protocols involved in this scenario include the DirectPlay 8 DXDiag Usage protocol, as specified in [MS-DPDX], [MC-DPL8R], [MC-DPL8CS], [MC-DPLHP], and [MC-DPLNAT], together with other general networking protocols used in the course of standard TCP communication between the client and server. The functionality illustrated in the scenario is as follows:

- Peer-to-peer gaming session

It should be noted that the following scenario depicts processes that occur over TCP, and thus assumes various TCP acknowledgment frames throughout the conversation that are not necessarily depicted in the trace details.

9.4.1 Scenario 1: Peer-to-Peer Gaming Scenario

9.4.1.1 Scenario Overview

The scenario illustrated below is a gaming session using two Windows XP Professional operating system stand-alone computers that use peer-to-peer networking. To provide network activity to document this task, the game "Dungeon Siege" was played by one user (the game client) against a Windows XP operating system game server. The protocol involved in this scenario is the DirectPlay 8 protocol as documented in [MC-DPL8CS], [MC-DPL8R], and [MS-DPDX].

9.4.1.2 Trace Configuration

This scenario is implemented with two Windows XP operating system machines, configured as:

Windows XP Game Server:

- Machine name: Windows XP Game Server (GameServer)
- Operating system: Windows XP Professional operating system - original version, unpatched
- IP Address: 10.0.0.1
- Subnet Mask: 255.0.0.0
- Game - Dungeon Siege

Windows XP Game Client:

- Machine name: Windows XP Game Client (GameClient)
- Operating system: Windows XP Professional – original version, unpatched
- Windows Installer 3.1 installed
- .NET Framework 2.0 installed
- IP Address: 10.0.0.2
- Subnet Mask: 255.0.0.0
- Game – Dungeon Siege

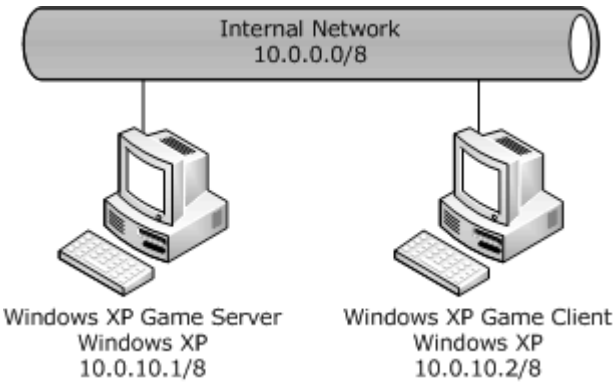

Figure 67: Network topology

9.4.1.3 Setting Up the Trace

These are the steps required to produce the peer-to-peer gaming trace:

Windows XP operating system Game Server:

- Launch Dungeon Siege and host a new multiplayer game.

Windows XP Game Client:

1. Start NetMon 3.1.
2. Launch Dungeon Siege.
3. BEGIN TRACE.
4. Join multiplayer game hosted on Windows XP Game Server.
5. Broadcast a test chat message.
6. END TRACE.

9.4.1.4 Netmon Trace Digest

- Communication between the game server and game client is performed exclusively by using the DirectPlay 8 protocol. Initial communication between the client and server is negotiated using DirectPlay session packets. The session is initiated when the game client issues a session enumeration query to the server, which has the single initiating client currently registered in the session name table:

```
GameClient GameServer DirectPlay 8  DirectPlay: 0x1 - Seq:
 Session Enumeration Query
 GameServer GameClient DirectPlay 8 DirectPlay: 0x1 - Ser:
 Session Enumeration Response, Name=D, Players=1
```

- The client then issues a connection request to the game server, handshaking acknowledgements with the server to establish the connection to the existing session:

```
GameClient GameServer DirectPlay 8 DirectPlay: 0x1 - Tcc:
 Connection Request
Gameserver  GameClient DirectPlay 8 DirectPlay: 0x1 - Tcca:
 Accept Connection Request
GameClient GameServer DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_USERDATA_ACK_SESSION_INFO
```

- In order to calculate the round-trip time (RTT) used to determine when a packet should be resent if not yet acknowledged, the game client sends a keepalive packet that is returned by the server:

```
GameClient GameServer DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_USERDATA_KEEPALIVE (SessID: 0x8E5CCB6A)
GameServer GameClient DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_USERDATA_KEEPALIVE (SessID: 0x8E5CCB6A)
```

- The client initiates a player registration in the name table by sending player connection information to the game server:

```
GameClient GameServer DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_USERDATA_PLAYER_CONNECT_INFO (0xc1)
```

- The server relays session information to the game client:

```
GameServer GameClient DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_USERDATA_SEND_SESSION_INFO (0xc2)
```

- The client acknowledges receipt of session information to the game server:

```
GameClient GameServer DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_USERDATA_ACK_SESSION_INFO (0xc3)
```

- The game server instructs the client to establish a connection to the session:

```
GameServer GameClient DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_USERDATA_INSTRUCT_CONNECT (0xc6)
```

- The game client announces the version number of the current client-side name table:

```
GameClient GameServer DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_USERDATA_NAMETABLE_VERSION (0xc9)
```

- The server issues an instruction to the game client to resynchronize the name table version with that of the server:

```
GameServer GameClient DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_USERDATA_RESYNC_VERSION (0xca)
```

- The game client concludes the session packet exchange by acknowledging the resynchronization instruction:

```
GameClient GameServer DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_COMMAND_SACK
```

- The negotiated DirectPlay 8 connection provides two-way transport for state and messaging communication between the game server and game client using a sliding window format. Each packet is designated reliable and sequential, requiring each UDP packet to be acknowledged by the receiver before more packets are transmitted:

```
GameClient GameServer DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_USERDATA_HEADER (Command 0x3F, Len=20)
GameServer GameClient DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_COMMAND_SACK
GameServer GameClient DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_USERDATA_HEADER (Command 0x3F, Len=32)
Gameclient Gameserver DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_COMMAND_SACK
```

- Although the DirectPlay 8 protocol does not provide security parameters or robust security like that available through IPSec, the transport functionality bundles state and communication information into sequential packets. Alone, chat data sent from the game client to the server employs the TRANS_USERDATA_SEND_MESSAGE packet, which is designated sequential but not reliable (an immediate acknowledgement is not required). Dungeon Siege bundles its chat and state information into the same stream, sequestering chat text from casual examination and ensuring all packets are exchanged as sequential and reliable:

```
GameClient GameServer DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_USERDATA_HEADER (Command 0x3F, Len=63)
GameServer GameClient DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_COMMAND_SACK
```

- At the conclusion of the connection, DirectPlay returns to session packet exchange initiated by an end of stream packet sent from the disconnecting client to the game server :

```
GameClient GameServer DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_USERDATA_END_OF_STREAM
```

- The game servers replies to the end of stream announcement with four sequential identical acknowledgements:

```
GameServer GameClient DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_COMMAND_SACK
GameServer GameClient DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_COMMAND_SACK
GameServer GameClient DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_COMMAND_SACK
GameServer GameClient DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_COMMAND_SACK
```

- The game server transmits and end of stream announcement to the disconnecting client:

```
GameServer GameClient DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_USERDATA_END_OF_STREAM
```

- The game client then replies to the end of stream announcement with four sequential identical acknowledgements:

```
GameClient GameServer DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_COMMAND_SACK
GameClient GameServer DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_COMMAND_SACK
GameClient GameServer DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_COMMAND_SACK
GameClient GameServer DirectPlay 8 DirectPlay: 0x1 -
 PacketType: TRANS_COMMAND_SACK
```

- Because the example includes only the game server and one client, the server does not need to transmit a TRANS_USERDATA_DESTROY_PLAYER packet to any additional clients to remove the departing player from their name table copies.

10 Print/Fax Services Protocols

This section provides an overview of the Print/Fax Services protocols.

10.1 Print/Fax Protocols Overview

The Print/Fax Services include protocols that manage the interaction between file/print and fax servers and clients. These protocols include authentication to the print/fax server, notification of print/fax status, and management of remote print jobs.

These protocols do not include any printer protocols or printer-specific commands or content.

10.1.1 Print/Fax Protocols List

The Print/Fax Services include two sets of protocols: the core protocols that provide the print/fax services functionality, and a set of protocols that are required for an implementation of the core protocols (as listed in the Normative References section of the core protocols technical specifications) or that provide core networking functionality. The latter set of protocols is referred to as the Networking Protocols Group, as described in the Overview section of this document.

Protocol name	Description	Document short name
Fax Server and Client Remote Protocol	Manages and sends faxes; manages the fax server and its queues; and provides for notifications. Fax server services are provided through the fax interface, and fax client services are provided through the faxclient interface.	[MS-FAX]
Print System Asynchronous Notification Protocol	An asynchronous RPC-based protocol; used by clients to receive print status notifications from a print server and send any server-requested responses to those notifications back to the server. A set of notifications and responses is defined together, and is referred to as a "notification type". The Print System Asynchronous Notification Protocol defines one such notification type, called Async UI, but allows for the independent creation of other notification types. The Async UI notification type allows for a server-resident printer driver to request the display on a client of an informative alert; allows for a client to send user input that is requested by such an alert back to the server; and allows for the server-resident driver to request the execution on the client of code that is contained in a client-resident printer driver.	[MS-PAN]
Print System Asynchronous Remote Protocol	An asynchronous RPC-based protocol; used by a print client to send print jobs to a print server and to direct their processing; and to add or remove print queues or perform other print system management functions. It provides asynchronous forms of the synchronous operations that are supported by the Print System Remote Protocol, and extends them with additional asynchronous operations.	[MS-PAR]
Print System Remote Protocol	Synchronous RPC-based protocol; used by a print client to send print jobs to a print server and direct their processing; and to add or remove print queues or perform other print system management functions.	[MS-RPRN]
Web Point-and-Print Protocol	Allows world-wide distribution of pointer driver software using standard HTTP packets. This protocol uses HTTP redirect commands	[MS-WPRN]

Protocol name	Description	Document short name
	with a custom query string on the URL to download a print driver.	

The following sections provide some basic print/fax protocols concepts, and different conceptual views to help the user visualize the physical and logical relationships among the core protocols in Print/Fax Services. These views are called the protocol stack view and the logical dependencies view, as explained in section 10.3 of this document.

10.2 Print/Fax Services Functionality

The section describes basic concepts for print/fax protocols, and provides some background for the Print/Fax protocol model.

Section 10.4 of this document provides examples that illustrate the interaction of some of the protocols in two sample configurations.

Network printing is a multi-step process that starts with the authentication of the client's right to use the selected resource, and ends with notification back to the client that the print job has completed. Additional functionality, such as printer driver installation on demand, makes use of the basic level of functionality described here.

10.2.1 Fax Server and Client Remote Protocol

This specification defines a Microsoft protocol referred to as the FAX Server and Client Remote Protocol. The FAX Server and Client Remote Protocol is an RPC-based, client/server protocol used to send faxes and manage the fax server and its queues.

10.2.2 Print System Remote Protocol

The Print System Remote Protocol (as specified in [MS-RPRN]) defines the communication of print job processing and print system management between a print client and any print server. It is built on the RPC protocol

10.2.3 Print System Asynchronous Remote Protocol

The Print System Asynchronous Remote Protocol (as specified in [MS-PAR]) defines the communication of print job processing and print system management information between a print client and any print server. The Print System Asynchronous Remote Protocol is built on the RPC protocol and can be used as an enhanced replacement for the Print System Remote Protocol, as specified in [MS-RPRN].

The Print System Asynchronous Remote Protocol is designed for use by asynchronous clients. The implementations of these clients permit them to continue execution without waiting for an RPC method call to return.

10.2.4 Print System Asynchronous Notification Protocol

The Print System Asynchronous Notification Protocol (as specified in [MS-PAN]) is an asynchronous protocol used by clients to receive print status notifications from a print server, and to send any server-requested responses to those notifications back to the server. It is based on the RPC protocol.

A set of notifications and responses is defined together, and is referred to as a notification type. The RPC interfaces and methods defined by the Print System Asynchronous Notification Protocol provide a transport mechanism for arbitrary notification types.

The Print System Asynchronous Notification Protocol defines one notification type, called AsyncUI. The AsyncUI notification type allows for a print-server-resident notification source to request the display on a client of an informative alert, for a client to send user input requested by such an alert back to the server, and for the server-resident notification source to request the execution on the client of code that is resident on the client.

10.2.5 Web Point-And-Print Protocol

The Web Point-and-Print Protocol (as specified in [MS-WPRN]) provides a mechanism for a client to load necessary printer driver software. It also provides a mechanism for automatically downloading the printer driver directly from a print device, network server, or website.

10.2.6 Fax Server and Client Remote Protocol

The Fax Server and Client Remote Protocol (as specified in [MS-FAX]) can be used to submit and manage faxes. It can be further used to change configuration on the fax server—for example, setting the Inbound Routing Rules/Outbound Groups. It can be used to change settings, such as whether the fax service should archive the faxes it sends or receives, the number of days it should keep an archive, or the number of rings before a call should be answered. Practically everything that manages the behavior of the fax server can be controlled using the Fax Server and Client Remote Protocol. This can be used either locally, where both the client and server are on the same machine, or remotely, where the client and server are on different machines.

10.3 Print/Fax Server Protocols Logical Dependencies and Protocol Stack Views

This section describes the logical dependencies among the print/fax protocols and protocol stack views as appropriate.

10.3.1 Print RPC

Print RPC is composed of three protocols that; work together to provide mechanisms for gaining access to and managing print system functionality in a distributed environment:

- Print System Remote Protocol (as specified in [MS-RPRN]): A protocol built on the RPC protocol and supports synchronous access to and management of print system functionality.
- Print System Asynchronous Remote Protocol (as specified in [MS-PAR]): A protocol built on the RPC protocol and is closely modeled after Print System Remote Protocol (as specified in [MS-RPRN]), the key enhancement being that Print System Asynchronous Remote Protocol provides asynchronous access to and management of print system functionality.
- Print System Asynchronous Notification Protocol (as specified in [MS-PAN]): A protocol built on the RPC protocol and allows server-resident print system components, such as a printer driver, to asynchronously communicate with a client-resident print system component. One particular application of these interfaces is to enable a client-resident print system component to interact with the user on behalf of a server-resident printer driver.

10.3.1.1 Print RPC Stack View

The following diagram illustrates the protocol stack for Print RPC.

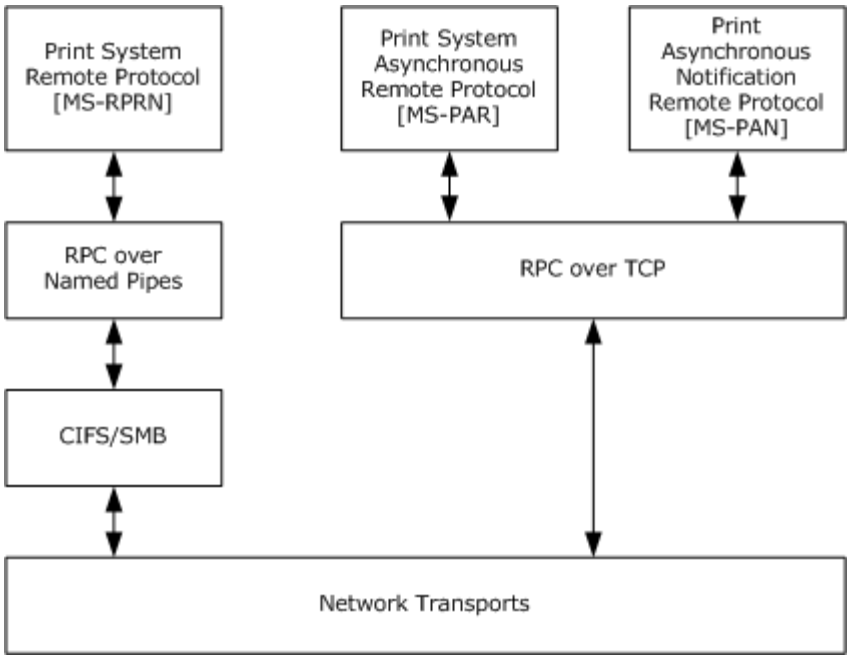

Figure 68: Print RPC protocol stack

10.3.1.2 Print RPC Logical Dependencies

The protocols in Print RPC do not depend on any protocols other than their underlying transports. Both of their underlying transports, in turn, have dependencies resulting from the authentication schemes that they support. The standard authentication support is specified in section 3.2.

10.3.2 Internet Print

Internet Print includes the Web Point-and-Print Protocol (as specified in [MS-WPRN]), which provides a mechanism for clients to obtain printer driver software from servers by using the HTTP and HTTPS protocols.

10.3.2.1 Web Point-and-Print Stack View

The following diagram illustrates the layering of the protocol in this section with other protocols in its stack.

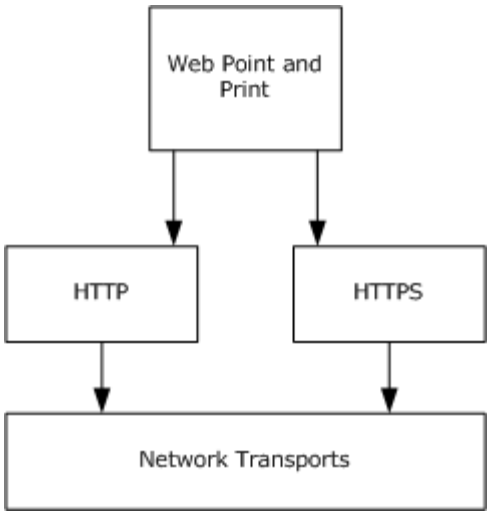

Figure 69: Internet Printing protocol stack

As shown, the Web Point-and-Print Protocol (as specified in [MS-WPRN]) is layered directly over the HTTP and HTTPS protocols by using their standard TCP port numbers.

The protocol stack for this section does not contain any other protocols.

10.3.2.2 Internet Print Logical Dependencies

Internet Print has no logical dependencies other than its underlying transports.

10.3.3 Fax Service

The Fax service provides fax capabilities, which allows users to send and receive faxes from desktop applications using either a local fax device or a shared network fax device.

10.3.3.1 Fax Service Stack View

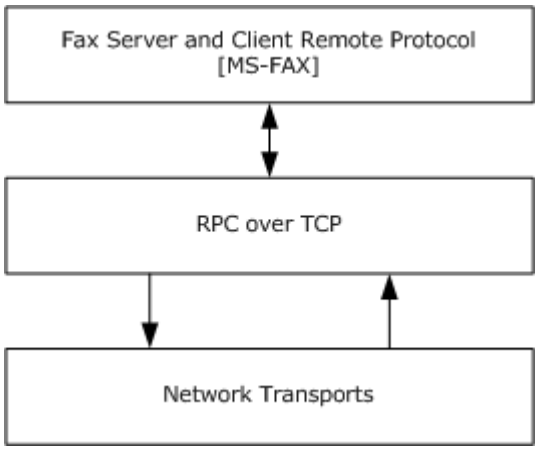

Figure 70: Fax client with RPC over HTTP stack

10.4 Implementation Scenarios

This section describes two scenarios that illustrate common functionality that can be implemented with the protocols included in Print/Fax Services.

- Network printing
- Remotely managing a network printer

10.4.1 Scenario 1 - Network Printing

10.4.1.1 Scenario Overview

The actions described here represent network traffic generated between Microsoft protocols when a file is printed on a server-attached printer from a network-attached Windows client.

A file is printed from a Windows Vista operating system client to a printer attached to a Windows Server 2003 operating system server.

10.4.1.2 Scenario Configuration

W2K3ServerR2 Configuration

- Machine Name: W2K3ServerR2
- Operating System: Windows Server 2003 R2 operating system with all current updates
- IP Address: 192.168.1.10
- Subnet Mask: 255.255.255.0
- Configured with the following roles.
  - File/Print Server

VistaClient Configuration

- Machine Name: VistaClient
- Operating System: Windows Vista operating system Ultimate with all current updates
- IP Address: 192.168.1.5
- Subnet Mask: 255.255.255.0

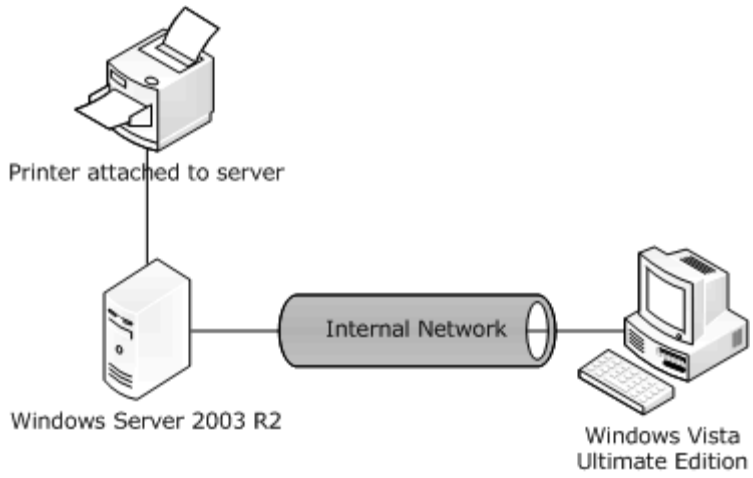

Figure 71: Network topology

10.4.1.3 Setting up the Trace

1. Set a network printer as default printer for a Windows Vista operating system client.
2. Print file

10.4.1.4 Netmon Trace Digest

In this task, a file is printed from a Windows Vista operating system client to a print server (printer directly attached to a Windows Server 2003 R2 operating system machine). By default, notification from the client to server of the status of print jobs is disabled. This means that traffic is not generated from the Print System Asynchronous Remote Protocol (as specified in [MS-PAR]) and the Print System Asynchronous Notification Protocol (as specified in [MS-PAN]). If traffic is present from the notification protocols, it will be sent asynchronously, with the primary data stream sent via MS-RPRN. As the default status is used in this scenario, all related traffic is therefore generated via the Print System Remote Protocol (as specified in [MS-RPRN]).

- As shown below, the initial traffic from the client to the server is the creation of the spool file via Server Message Block (SMB).

```
VistaClientW2K3ServerR2SMBSMB: C; Nt Create Andx, FileName =
\spoolss
```

- Using RPC calls, the client requests the selected printer.

```
W2K3ServerR2VistaClientMSRPCMSRPC: c/o Bind Ack:  Call=0x1  Assoc
 Grp=0x6EF8  Xmit=0x10B8  Recv=0x10B8
VistaClientW2K3ServerR2WinspoolWinspool: RpcOpenPrinterEx Request,
 Printer = \\192.161.1.4\HP LaserJet 6P
W2K3ServerR2VistaClientWinspoolWinspool: RpcOpenPrinterEx Response,
 Handle: {01000000-00000000-0000-0000-0000-000000000000}, Status =
ERROR_SUCCESS
VistaClientW2K3ServerR2WinspoolWinspool: RpcClosePrinter Request
W2K3ServerR2VistaClientWinspoolWinspool: RpcClosePrinter Response,
 Status = ERROR_SUCCESS
```

- RpcRemoteFindFirstPrinterChangeNotificationEx creates a remote change notification object that monitors changes to printer objects and sends change notifications to the client.

```
VistaClientW2K3ServerR2WinspoolWinspool:
RpcRemoteFindFirstPrinterChangeNotificationEx Request, LocalMachine:
\\VistaClient, Flags: 0x00000000, Printer:
{01000000-00000000-0000-0000-0000-000000000000}
W2K3ServerR2VistaClientWinspoolWinspool:
RpcRemoteFindFirstPrinterChangeNotificationEx Response, Status =
ERROR_ACCESS_DENIED
VistaClientW2K3ServerR2WinspoolWinspool:
RpcRouterRefreshPrinterChangeNotification Request, Printer:
{01000000-00000000-0000-0000-0000-000000000000}
W2K3ServerR2VistaClientWinspoolWinspool:
RpcRouterRefreshPrinterChangeNotification Response, Status =
ERROR_INVALID_HANDLE
```

- Because the notification services are disabled by default, the notification fails.
- Data is transferred to the spooler file, as necessary, until the entire print job has been passed to the server.
- RpcWritePrinter sends the data to the print server. Status-ERROR_SUCCESS indicates successful command execution in all of these remote procedure calls.

```
VistaClientW2K3ServerR2WinspoolWinspool: RpcWritePrinter Request
W2K3ServerR2VistaClientWinspoolWinspool: RpcWritePrinter Response,
 Status = ERROR_SUCCESS
```

- RpcEndPagePrinter notifies the print server that the application is at the end of a page in the print job.

```
VistaClientW2K3ServerR2WinspoolWinspool: RpcEndPagePrinter Request
W2K3ServerR2VistaClientWinspoolWinspool: RpcEndPagePrinter
Response, Status = ERROR_SUCCESS
```

- RpcEndDocPrinter notifies the print server that the application has finished its print job.

```
VistaClientW2K3ServerR2WinspoolWinspool: RpcEndDocPrinter Request
W2K3ServerR2VistaClientWinspoolWinspool: RpcEndDocPrinter Response,
 Status = ERROR_SUCCESS
```

- RpcClosePrinter closes the handle being used by the print job.

```
VistaClientW2K3ServerR2WinspoolWinspool: RpcClosePrinter Request
W2K3ServerR2VistaClientWinspoolWinspool: RpcClosePrinter Response,
 Status = ERROR_SUCCESS
```

- The print session ends with an SMB message that closes the print spooler file.

```
W2K3ServerR2VistaClientSMBSMB: R; Close, FID = 0x400F
(\spoolss@#258)
```

10.4.2 Scenario 2 - Remote Management of Printers

10.4.2.1 Scenario Overview

This scenario demonstrates the network traffic that happens when a Windows client attempts to remotely manage a printer attached to a Windows Server.

In this scenario, a Windows Vista operating system client attempts to access a printer attached to a Windows Server 2003 R2 operating system SP2 server with the intent of modifying the properties of the printer.

10.4.2.2 Scenario Configuration

W2K3ServerR2 Configuration

- Machine Name: W2K3ServerR2
- Operating System: Windows Server 2003 R2 operating system with all current updates
- IP Address: 192.168.1.10
- Subnet Mask: 255.255.255.0
- Configured with the following roles.
  - File/Print Server

VistaClient Configuration

- Machine Name: VistaClient
- Operating System: Windows Vista operating system Ultimate with all current updates
- IP Address: 192.168.1.5
- Subnet Mask: 255.255.255.0

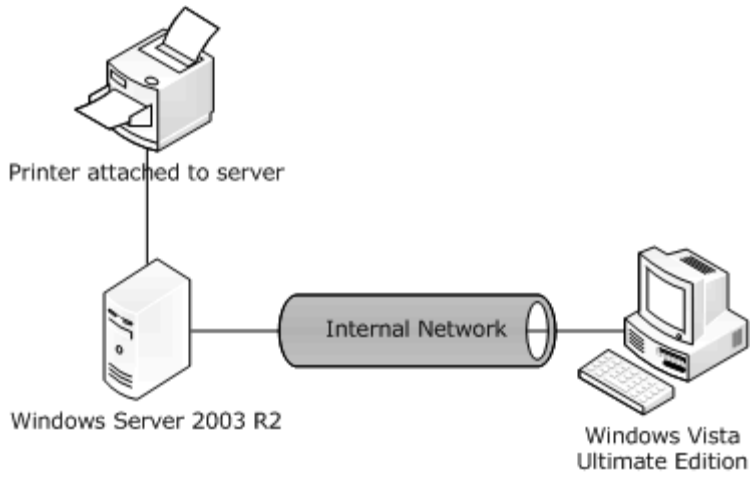

Figure 72: Network topology

10.4.2.3 Setting up the Trace

The following steps produce the remote printer management trace.

1. On the server, enable the group policy object Allow Spooler to Permit Client Connections.
2. Enable the Printer Browsing Group Policy Object.
3. Start trace.
4. Connect to remote printer via Printer Properties context menu.
5. Change properties.
6. End trace.

10.4.2.4 Netmon Trace Digest

- In this task, a client views the properties of a remote printer.

  The initial sequences in the Netmon capture file deals with authenticating the client computer and user account for access to the server. See section 2.2 for more information about authentication. Resolution of the name of the available printer is performed via Distributed File System (DFS).

```
VistaClientW2K3ServerR2DFSDFS: Get DFS Referral Request,
FileName: \TEST-SERVER-1\EPSONSty.2, MaxReferralLevel: 4
```

- Details about the Distributed File System (DFS): Referral Protocol can be found in [MS-DFSC].
- A series of SMB requests are processed in order to establish a tree connection to the server share. For example:

```
W2K3ServerR2VistaClientSMBSMB: R; Transact2, Get Dfs Referral
- NT Status: System - Error, Code = (14) STATUS_NO_SUCH_DEVICE
W2K3ServerR2VistaClientSMBSMB: R; Open Print File,
```

```
FID = 0x4001
VistaClientW2K3ServerR2SMBSMB: C; Tree Connect Andx,
Path = \\TEST-SERVER-1\EPSONSTY.2, Service = ?????
W2K3ServerR2VistaClientSMBSMB: R; Tree Connect Andx,
Service = LPT1:
9366.968750{TCP:12, IPv4:1}VistaClientW2K3ServerR2TCP
TCP: Flags=....A..., SrcPort=49211, DstPort=NETBIOS Session
Service(139),
 Len=0, Seq=2771857192, Ack=905870110, Win=66 (scale factor 8)
= 16896
VistaClientW2K3ServerR2SMBSMB: C; Close Print File,
FID = 0x4001
W2K3ServerR2VistaClientSMBSMB: R; Close Print File,
FID = 0x4001
VistaClientW2K3ServerR2SMBSMB: C; Open Print File,
FileName = ADMINISTRATOR
W2K3ServerR2VistaClientSMBSMB: R; Open Print File,
FID = 0x4002
```

- Complete details on the use of these commands can be found in the [MS-SMB] protocol document.
- The NetShareEnum command is used by the Remote Administration Protocol (as specified in [MS-RAP]) to enumerate the shared services to the client.

```
VistaClientW2K3ServerR2SRVSVCSRVSVC: NetrShareEnum Request,
ServerName=\\TEST-SERVER-1 Level=1 PreferredMax=0xFFFFFFFF
W2K3ServerR2VistaClientSRVSVCSRVSVC: NetrShareEnum Response,
 Level=1 Entries=10, Status = ERROR_SUCCESS
```

- Complete details about using this command can be found in [MS-RAP].
- The Print System Remote Protocol (as specified in [MS-RPRN]) can now make RPC calls via Winspool to manipulate printer data.

```
VistaClientW2K3ServerR2WinspoolWinspool: RpcOpenPrinterEx
Request, Printer = \\TEST-SERVER-1
W2K3ServerR2VistaClientWinspoolWinspool: RpcOpenPrinterEx
Response, Handle: {01000000-00000000-0000-0000-0000-000000000000},
Status = ERROR_SUCCESS
VistaClientW2K3ServerR2WinspoolWinspool: RpcGetPrinterData
Request
W2K3ServerR2VistaClientWinspoolWinspool: RpcGetPrinterData
Response, Status = ERROR_SUCCESS
VistaClientW2K3ServerR2WinspoolWinspool: RpcGetPrinter Request
W2K3ServerR2VistaClientWinspoolWinspool: RpcGetPrinter
Response, Status = ERROR_SUCCESS
VistaClientW2K3ServerR2WinspoolWinspool: RpcClosePrinter
Request
W2K3ServerR2VistaClientWinspoolWinspool: RpcClosePrinter
Response, Status = ERROR_SUCCESS
```

- At this point in the sequence, the user opens the selected Printer Properties dialog box and changes the share name of the printer object. The Print System Remote Protocol (as specified in [MS-RPRN]) handles the notification of the change via RpcRemoteFindFirstPrinterChangeNotificationEx.

```
W2K3ServerR2VistaClientWinspoolWinspool:
RpcRemoteFindFirstPrinterChangeNotificationEx Response,
Status = ERROR_ACCESS_DENIED
VistaClientW2K3ServerR2WinspoolWinspool:
RpcRouterRefreshPrinterChangeNotification Request,
Printer: {01000000-00000000-0000-0000-0000-000000000000}
W2K3ServerR2VistaClientWinspoolWinspool:
RpcRouterRefreshPrinterChangeNotification Response,
Status = ERROR_INVALID_HANDLE
VistaClientW2K3ServerR2WinspoolWinspool:
RpcOpenPrinterEx Request, Printer = \\TEST-SERVER-1
W2K3ServerR2VistaClientWinspoolWinspool:
RpcOpenPrinterEx Response, Handle:
{01000000-00000000-0000-0000-0000-000000000000},
Status = ERROR_SUCCESS
VistaClientW2K3ServerR2WinspoolWinspool:
RpcRemoteFindFirstPrinterChangeNotificationEx Request,
LocalMachine: \\Vishnu, Flags: 0x000000FF, Printer:
{01000000-00000000-0000-0000-0000-000000000000}
```

11 Rights Management Services Protocols

This section provides an overview of the Rights Management Services protocols. It provides a high-level conceptual overview of the core protocol and its relationship to other protocols, showing its operation in typical rights-management tasks.

11.1 Rights Management Services Overview

Rights Management Services Server (RMS Server) includes only one protocol as specified in section 11.1.1.

The Rights Management Services (RMS): Client-to-Server Protocol [MS-RMPR] provides support for information protection through content encryption and fine-grained policy definition and enforcement. In doing so, the RMS Protocol enables end users to create and consume protected information.

Only the RMS actions that transmit data on the network are described in this document.

11.1.1 Protocols List

Rights Management Services include a single core protocol that provides the certificate services functionality. Rights Management Services include a second set of protocols that are required to implement it. Those are referred to as the networking protocols group.

The following table lists the core protocol included in Rights Management Services. This list does not include data structures, file structures, or algorithms that this core protocol depends on. The details for these technical dependencies are documented in the technical specifications for the core protocol.

Table 1: RMS Server core protocol

Protocol name	Protocol description	Document short name
Rights Management Services (RMS): Client-to-Server Protocol	The Rights Management Services (RMS): Client-to-Server Protocol is used to obtain and issue certificates and licenses used for creating and working with protected content.	[MS-RMPR]

11.2 Rights Management Services Functionality

Implementation of the Rights Management Services allows for the creation, access, and consumption of protected content. When this functionality has been implemented, the creator of a document can limit and control the use of the protected content. This can be accomplished from within the application that creates the protected content.

Examples that illustrate the interaction of some of the protocols are provided in two sample configurations in section 11.4.

Access to protected content is under the central control of the RMS server. This server-based control allows the implementation to use a single authentication mechanism for the purpose of granting access rights to protected content.

The server is required to undergo a one-time bootstrapping process to begin functioning in the RMS system. In RMS 1.0, RMS 1.0 SP1, and RMS 1.0 SP2, this operation involves contacting the cloud

service. In RMS 2.0, this operation is done entirely offline. The creator and consumer contact the server for a one-time bootstrapping process to function in the RMS system.

The consumer, upon receiving the document from the creator and opening it, supplies the server with the PL and the RMS account certificate (RAC) that was acquired during bootstrapping. If the consumer is allowed access according to the access policy in the PL, the server issues the consumer a use license (UL) that specifies the access policy for the consumer and binds the content decryption key to the consumer's RAC. The RAC key is encrypted by the key of a trusted software module called the security processor. When the consumer attempts to access the document, the security processor decides whether the requesting application on the consumer machine is capable of enforcing the access policy. If so, it supplies plain text of the document to the application along with the policy that the application is to enforce.

Consumption of the protected content means that a client received the proper licenses issued, via the RMS server, to allow access to the protected content, under the terms and restrictions of the license that the server granted and that the document creator requested.

Regardless of functionality, the RMS system components comprise four active entities: the creator, the consumer, the server, and the cloud service.

- **Creator**: The creator is the entity that builds the document to be protected. After the document has been created, the creator can then choose an access policy for the document or make use of a pre-existing rights policy template. After encrypting the document, the creator can then make use of a publishing license to bind the selected access policy to the document.
- **Consumer**: After consumers have received a protected document, they will attempt to open it. The consumer system contacts the server, and the server makes use of the publishing license and RMS account certificate (RAC) already acquired. After the server decides that access is permitted, it issues a use license to the client that specifies the access policy for the client and binds the encryption key to the consumer's RAC.
- **Server**: The server controls the access to the protected content. It does this by issuing signed certificates and keys. The server is the sole authority for the evaluation of client identity and issues authorizations based on client-provided identity credentials.
- **Cloud**: The Cloud service contains the Enrollment service, which allows clients to get the credentials necessary for obtaining the rights to digitally protected content. The Cloud service also contains the Activation service necessary for RMS clients earlier than version 2.0.

A client can play the role of a creator, a consumer, or both, depending on the implementation. The client is responsible for requesting certificates, licenses, and policies from the server. It is also responsible for enforcing authorization policies as they apply to protected information and encrypting or decrypting content as appropriate. The RMSv2 client in Windows Server 2008 operating system and Windows Vista operating system with Service Pack 1 (SP1) also has the capability to fetch rights policy templates from an RMSv2 server.

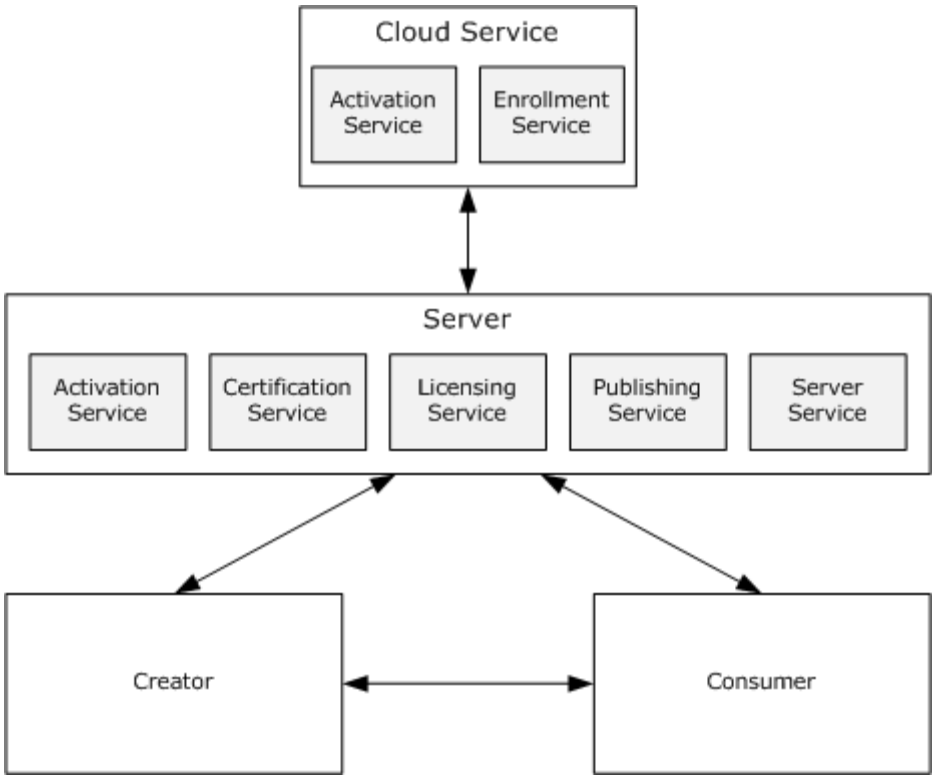

Figure 73: High-level overview of the RMS system

The following are some of the ways that implementers can integrate this functionality into their services.

This baseline RMS scenario involves three actors: the publisher/author, the consumer/recipient, and the RMS server. The publisher and consumer are both roles of the RMS client. For this scenario, we assume that the publisher, consumer, and server are all bootstrapped (that is, started automatically.)

1. A publisher creates content in an RMS-enabled application. The publisher protects the content and applies a usage policy that specifies recipients by their email addresses. Recipients can be individual users or distribution groups in a directory. The publisher then specifies what access rights are to be granted to each recipient: such as read, edit, and print. The publisher also specifies any conditions on those rights: for example, expiry after a period of time. This protection step calls in to the RMS client to encrypt the file and create the policy. The policy lives in a Publishing License (PL). The PL describes the access policy and contains the content key; all of this data is encrypted with the RMS server's public key.

2. The publisher sends the content to the recipient using any mechanism, such as USB flash drive, email, or file share. RMS does not do the actual distribution of the content.

3. The recipient receives the file and needs to access it. When the recipient tries to open the file in an RMS-enabled application, the application realizes that the file is protected and that the user does not yet have an authorization token [Use License (UL)] for the content.

4. The application calls into the RMS client to acquire a UL. The client makes an AcquireLicense request to the RMS server. (This is the first and only client-server interaction in this scenario.)

The AcquireLicense request submits the PL, along with the user's RAC to the server. The RAC is the user's encryption certificate. The client may discover the RMS server location through Active Directory, existing client configuration data, or from the DISTRIBUTIONPOINT element in an existing license.

5. The RMS server checks the signatures of the PL and the RAC to determine whether they can be trusted.

6. The RMS server decrypts the policy data and content key from the PL and checks to see whether any of the principals named in the PL match the principal named in the RAC. This check may involve expanding groups by contacting a directory.

7. If a match is found, the RMS server determines the rights that are to be granted to the user, along with any conditions on those rights. It also encrypts the content key with the user's RAC public key. This authorization data and encrypted content key are packaged in a UL and sent to the client in the response to the request.

8. With the UL, the recipient now has the data needed to consume the content. The application passes the RAC and UL into the RMS client so that the RMS client can decrypt the content. The RMS client evaluates what rights are to be granted along with any conditions on those rights, and then uses the security processor to decrypt the RAC private key. The private key is then used to decrypt the content key from the UL, which is then used to decrypt the content.

11.3 Rights Management Protocol Logical Dependencies and Protocol Stack Views

This section describes how the Rights Management Services (RMS) Protocol and the protocols that are needed to implement it depend on each other. Protocol stack views are also described as appropriate.

11.3.1 Logical Relationships

The RMS Protocol uses the Simple Object Access Protocol (SOAP) messaging protocol for formatting requests and responses. It transmits these messages using the Hypertext Transfer Protocol (HTTP) and/or HTTP Secure (HTTPS) protocols. SOAP is considered the wire format used for messaging. HTTP and HTTPS are the underlying transport protocols. The SOAP protocol is specified in [SOAP1.1]. The content files are downloaded using HTTP 1.1, as specified in [RFC2616].

11.3.2 Stack View

The following diagram shows the transport stack used by RMS.

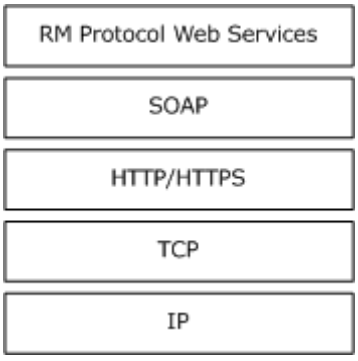

Figure 74: RMS transport stack

RMS defines the payload of the SOAP message. SOAP defines the format of the message and then HTTP/HTTPS over TCP/IP defines the transport of the message. To accomplish RMS tasks, both client and server RMS packages MUST be installed. These packages need to contain the RMS application programming interfaces (APIs) and the appropriate code for creating and responding to RMS requests.

11.4 Implementation Scenarios

This section describes a scenario that illustrate common functionality that can be implemented with the protocols included in Rights Management Services. This scenario can be performed with either Windows Server 2008 operating system or Windows Server 2003 R2 operating system and produce the same results.

- Creating and consuming protected content with Windows Server 2003 R2.
- Creating and consuming protected content with Windows Server 2008.

11.4.1 Scenario1 - Protected Content with Windows Server 2003

11.4.1.1 Scenario Overview

The scenario for protected content includes the following actions.

- File creation
- File consumption

Section 11.2 describes the overall process by which content is protected, accessed, and consumed. These scenarios show the network traffic results that pertain to RMS in each specific step taken for each of the four scenarios described in section 11.4.1.

11.4.1.2 Scenario Configuration

The following machine configurations were used for this scenario.

WS200801 Configuration

- Machine Name: WS200801.contoso.com (WS200801)
- Operating System: Windows Server 2008 operating system Beta 3
- Internal IP Address: 10.0.10.4
- Subnet Mask: 255.255.255.0
- Configured with the following roles.
  - Application Server
  - Domain Certificate Authority
  - DNS Server
  - Domain Controller
    - Domain Name: Contoso.com

- Users User1 and User2 created

CONTOSORMS2003 Configuration

- Machine Name: CONTOSORMS2003.contoso.com (CONTOSORMS2003)
- Operating System: Windows Server 2003 R2 operating system with the latest updates
- IP Address: 10.0.10.15
- Subnet Mask: 255.255.255.0
- Configured with the following roles.
  - Application Server
  - Rights Management Server

RMSCLIENT03-1 Configuration

- Machine Name: RMSCLIENT03-1.contoso.com (RMSCLIENT03-1)
- Operating System: Windows Vista operating system with latest updates
- IP Address: 10.0.10.17
- Subnet Mask: 255.255.255.0
- Microsoft Office 2007 Enterprise installed
- Member of Domain Contoso.com

RMSCLIENT03-2 Configuration

- Machine Name: RMSCLIENT03-2.contoso.com (RMSCLIENT03-2)
- Operating System: Windows Vista with latest updates
- IP Address: 10.0.10.14
- Subnet Mask: 255.255.255.0
- Microsoft Office 2007 Enterprise Installed
- Member of Domain Contoso.com

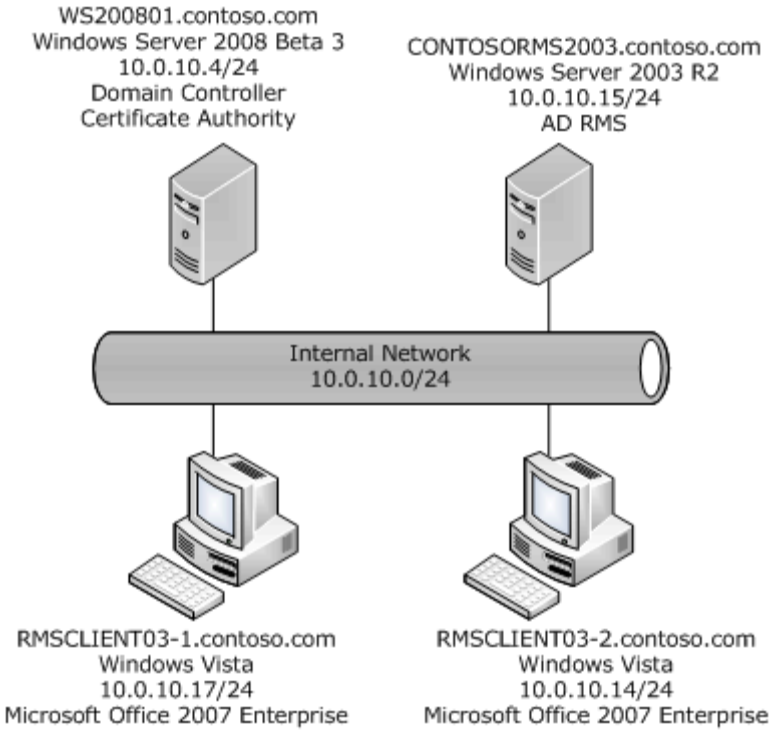

Figure 75: Network topology

11.4.1.3 Setting up the Trace - Document Creation

The following steps are used to produce the client document creation trace.

1. Boot all virtual machines.
2. Log on to RMSCLIENT03-1 as Contoso\User1.
3. Start trace.
4. Open Microsoft Office.
5. In the Microsoft Word document, type "Protected Document".
6. Click the **Review** tab.
7. Click **Protect Document**.
8. Click **Restrict Access**. The Permission dialog box appears.
9. Select **Restrict Permissions to this document**.
10. In the **Read** field, type "user2@contoso.com".
11. Click **OK**.
12. Save the document to a network share.
13. End trace.

11.4.2 Scenario 2: Protected Content Trace Scenario for Windows Server 2003

11.4.2.1 Scenario Overview

The scenario for protected content includes the following actions.

- Client document creation
- Client document consumption
- File creation
- File consumption

Section 11.2 describes the overall process for content protection, access, and consumption. The scenarios show the network traffic results that pertain to RMS in each specific step taken for each scenario described in section 11.4.2.

11.4.2.2 Scenario Configuration for Protected Content Windows Server 2008

The following machine configurations were used for this scenario.

WS200801 Configuration

- Machine Name: WS200801.contso.com (WS200801)
- Operating System: Windows Server 2008 operating system Beta 3
- Internal IP Address: 10.0.10.4
- Subnet Mask: 255.255.255.0
- Configured with the following roles.
  - Application Server
  - Domain Certificate Authority
  - DNS Server
  - Domain Controller
    - Domain Name: Contoso.com

  Users User1 and User2 created

WS2008RMS Configuration

- Machine Name: WS2008RMS.contoso.com (WS2008RMS)
- Operating System: Windows Server 2008 Beta 3
- Internal IP Address: 10.0.10.5
- Subnet Mask: 255.255.255.0
- Configured with the following roles.
  - Application Server

- Rights Management Server

RMSCLIENT1 Configuration

- Machine Name: RMSCLIENT1.contoso.com (RMSCLIENT1)
- Operating System: Windows Vista operating system with latest updates
- IP Address: 10.0.10.6
- Subnet Mask: 255.255.255.0
- Microsoft Office 2007 Enterprise installed
- Member of Domain Contoso.com

RMSCLIENT2 Configuration

- Machine Name: RMSCLIENT2.contoso.com (RMSCLIENT2)
- Operating System: Windows Vista with latest updates
- IP Address: 10.0.10.7
- Subnet Mask: 255.255.255.0
- Microsoft Office 2007 Enterprise installed
- Member of Domain Contoso.com

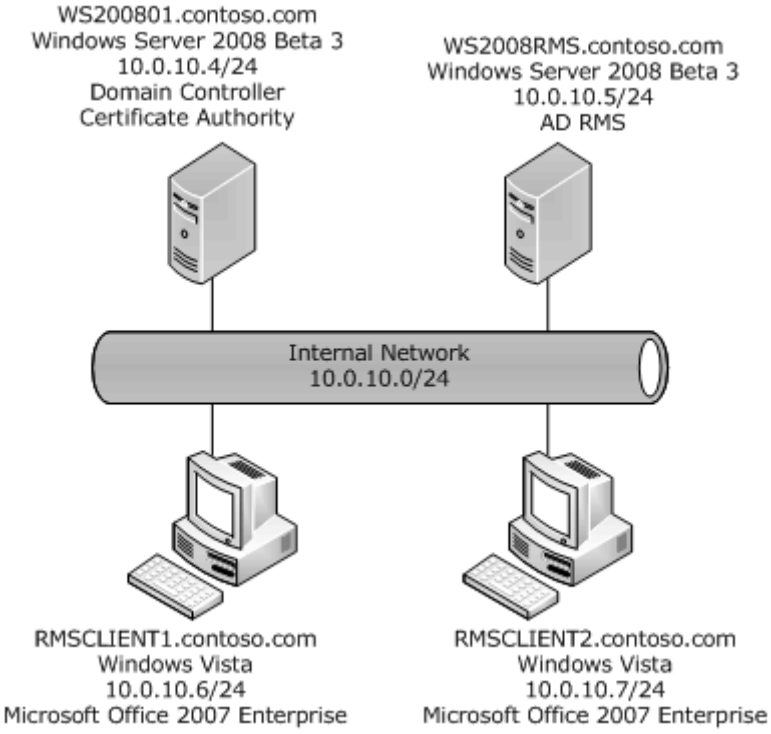

Figure 76: Network topology

11.4.2.3 Setting up the Trace - Document Creation

The following steps produce the client document creation trace.

1. Boot all virtual machines.
2. Log on to RMSCLIENT1 as Contoso\User1.
3. Start trace.
4. Open Microsoft Word.
5. In the Microsoft Word document, type "Protected Document".
6. Click the **Review** tab.
7. Click **Protect Document**.
8. Click **Restrict Access**. The Permission dialog box appears.
9. Select **Restrict Permissions to this document**.
10. In the **Read** field, type "user2@contoso.com".
11. Click **OK**.
12. Save the document to a network share.
13. End trace.

11.4.2.4 Netmon Trace Digest - Document Creation

Traffic related to client document creation with Windows Server 2008 operating system is identical to that found with the Windows Server 2003 R2 operating system configuration.

11.4.2.5 Setting up the Trace - Document Consumption

The following steps produce the client document consumption trace.

1. Log on to RMSCLIENT2 as Contoso\User2.
2. Start trace.
3. Browse to the network share.
4. Open the document.
5. End trace.

11.4.2.6 Netmon Trace Digest - Document Consumption

- With the need to consume content, the client must acquire a license. It does this by sending a request over HTTP to the server, using SOAP.

```
RMSCLIENT2 WS2008RMSHTTPHTTP: Request,
POST /_wmcs/licensing/License.asmx
```

- Frame details reveal the AcquireLicense SOAP action, as expected.

```
SOAPAction:  "http://microsoft.com/DRM/LicensingService/
AcquireLicense"
```

- RMS data is returned to the client.

```
RMSCLIENT2WS200801RMSRMS: RMS Payload
```

- With frame details that include such information as the following SOAP traffic.

```
<AcquireLicense xmlns="http://microsoft.com/DRM/LicensingService">
```

11.4.2.7 Setting up the Trace - File Creation

The following steps produce the file creation trace.

1. Start trace.
2. On RMSCLIENT1, open Microsoft Word.
3. In the Microsoft Word document, type "Protected Document".
4. Click the **Review** tab.
5. Click **Protect Document**.
6. Click **Restrict Access**.
7. The Permission dialog box appears. Select **Restrict Permissions to this document**.
8. In the **Read** field, type "user2@contoso.com".
9. Click **OK**.
10. Save the document to a network share.
11. End trace.

11.4.2.8 Netmon Trace Digest - File Creation

- With the file creation scenario, initial RMS request is made via HTTP to the server. The RMS requests fail until NTLM authentication is complete, at which point RMS data is passed.

```
RMSCLIENT1 WS2008RMSHTTPHTTP: Request,
POST /_wmcs/certification/ServiceLocator.asmx
RMSCLIENT1 WS2008RMSRMSRMS: RMS Payload
WS2008RMSRMSCLIENT1RMSRMS: 401 Unauthorized
RMSCLIENT1 WS2008RMSRMSRMS: RMS Payload
RMSCLIENT1 WS2008RMSRMSRMS: RMS Payload
```

- Responses to the request for the location of Certification Services, so that a certificate can be generated, are answered with the response from the Certification Service residing on the IIS Server.

```
WS2008RMSRMSCLIENT1RMSRMS:
Response from  Microsoft-IIS/7.0 powered by  ASP.NET version
 2.0.50727
RMSCLIENT1 WS2008RMSRMSRMS: RMS Payload
RMSCLIENT1 WS2008RMSRMSRMS: RMS Payload
ws2008rms.contoso.comRMSCLIENT1  RMSRMS:
Response from Microsoft-IIS/7.0 powered by  ASP.NET version
 2.0.50727
```

11.4.2.9 Setting up the Trace - File Consumption

The following steps produce the file consumption trace.

1. Start trace.
2. On RMSCLIENT2, browse to the network share.
3. Open the document.
4. End trace.

11.4.2.10 Netmon Trace Digest - File Consumption

- First, the standard authentication traffic appears.

```
RMSCLIENT2 WS2008RMSHTTPHTTP: Request, POST
/_wmcs/certification/ServiceLocator.asmx
RMSCLIENT2 WS2008RMSRMSRMS: RMS Payload
WS2008RMSRMSCLIENT2RMSRMS: 401 Unauthorized
RMSCLIENT2 WS2008RMSNTLMSSPNTLMSSP: NTLM NEGOTIATE MESSAGE,
Workstation Name: RMSCLIENT22, Workstation Domain: CONTOSO
WS2008RMSRMSCLIENT2NTLMSSPNTLMSSP: NTLM CHALLENGE MESSAGE
RMSCLIENT2 WS2008RMSNTLMSSPNTLMSSP: NTLM AUTHENTICATE MESSAGE,
Domain: CONTOSO, User: User2, Workstation: RMSCLIENT22
RMSCLIENT2 WS2008RMSRMSRMS: RMS Payload
```

- Then, the response regarding the location of the RMS services appears.

```
WS2008RMSRMSCLIENT2RMSRMS: Response from  Microsoft-IIS/7.0 powered
by  ASP.NET version  2.0.50727
```

- Authentication is performed, and then the RMS traffic resumes.

```
RMSCLIENT2 WS2008RMSNTLMSSPNTLMSSP: NTLM NEGOTIATE MESSAGE,
Workstation Name: RMSCLIENT22, Workstation Domain: CONTOSO
WS2008RMSRMSCLIENT2NTLMSSPNTLMSSP: NTLM CHALLENGE MESSAGE
RMSCLIENT2 WS2008RMSNTLMSSPNTLMSSP: NTLM AUTHENTICATE MESSAGE,
```

```
Domain: CONTOSO, User: User2, Workstation: RMSCLIENT22
RMSCLIENT2 WS2008RMSRMSRMS: RMS Payload
RMSCLIENT2 WS2008RMSRMSRMS: RMS Payload
RMSCLIENT2 WS2008RMSRMSRMS: RMS Payload
```

- The Publishing Service responds.

```
WS2008RMSRMSCLIENT2RMSRMS: Response from Microsoft-IIS/7.0
powered by ASP.NET version  2.0.50727
RMSCLIENT2 WS2008RMSRMSRMS: RMS Payload
```

- It finishes with the RMS data from the Licensing Service.

```
WS2008RMSRMSCLIENT2RMSRMS: Response from  Microsoft-IIS/7.0
powered by  ASP.NET version  2.0.50727
```

12 Firewall Services Protocols

This section provides an overview of the Firewall Services protocols.

12.1 File Services Overview

The Windows Firewall provides host-based firewall protection for computers running Windows XP operating system, Windows Server 2003 operating system, and Windows Vista operating system. Windows Firewall is designed to be a supplemental security solution. Windows Firewall should be part of a comprehensive security architecture that implements a variety of security technologies, such as border routers, perimeter firewalls, intrusion detection systems, virtual private networking (VPN), IEEE 802.1X authentication for wireless and wired connections, and Internet Protocol security (IPsec).

Windows Firewall inspects and filters all IP version 4 (IPv4) and IP version 6 (IPv6) network traffic. It is a stateful firewall, which means it tracks the state of each network connection and determines whether incoming packets should be allowed or blocked. Windows Firewall blocks incoming traffic unless it is in response to a request by the host (in which case, it is solicited traffic) or has been specifically permitted by being added to the Windows Firewall exceptions list. Aside from a few Internet Control Message Protocol (ICMP) messages, Windows Firewall allows all outgoing traffic.

12.1.1 Protocols List

Firewall Services Protocols examine and reject or forward data packets based on predefined permissions. The following table lists all of the protocols that are included in Firewall Services.

Protocol Name	Description	Document/Short Name
Internet Protocol v6 (IPv6)	Windows Firewall supports IPv6 Tunneled traffic that is encapsulated in an IPv4 header. Once the IPv4 header is removed, the IPv6 packets are inspected like all traffic.	[RFC2460] [RFC2462] [RFC2463] [RFC2464] [RFC2711]
Authentication Header/Encapsulating Security Payload (AH/ESP)	IPsec uses two transforms, the Authentication Header (AH) and the Encapsulating Security Payload (ESP) header and trailer, to encapsulate and secure IP packets or payloads. These transforms wrap the IP payload with header or header and trailer information and provide security properties, such as data origin authentication, data integrity, data confidentiality, and replay protection.	AH [RFC4302] ESP [RFC4303]
Generic Routing Encapsulation (GRE)	The Generic Routing Encapsulation (GRE) protocol is used to encapsulate the data channel of Point-to-Point Tunneling Protocol (PPTP) traffic. Any PPTP traffic that uses a GRE header to encapsulate Point-to-Point Protocol (PPP) frames will pass directly through Windows Firewall. Non-PPTP traffic that uses GRE is filtered by Windows Firewall.	[RFC2784]
Internet Group Management Protocol	IP multicast is used primarily for audio and video teleconferencing and gaming. Hosts and routers send IGMP messages so that IP multicast-enabled routers	[RFC3376]

Protocol Name	Description	Document/Short Name
(IGMP)	can determine the set of multicast host groups that have members on a subnet. All IGMP traffic passes directly through Windows Firewall.	
Pragmatic General Multicast (PGM)	The Pragmatic General Multicast (PGM) protocol is a scalable multicast protocol that enables receivers to detect loss, request retransmission of lost data, or notify a program of unrecoverable loss. PGM is often referred to as reliable multicast; it is a receiver-reliable protocol, which means the receiver is responsible for ensuring all data is received, absolving the sender of reception responsibility. Any traffic that uses the PGM protocol passes directly through Windows Firewall.	[RFC3208]

12.2 Firewall Services Functionality

Windows Firewall relies on several networking components, including the Windows Firewall/Internet Connection Sharing service (also known as the SharedAccess service), the network address translation (NAT) driver (Ipnat.sys), the IPv4-based TCP/IP driver (Tcpip.sys) and the IPv6-based TCP/IP driver (Tcpipv6.sys), and the Windows Sockets (Winsock) driver (Winsock.dll). Windows Firewall also relies on several administrative tools that allow you to configure Windows Firewall settings. The following figure shows the relationship among these components.

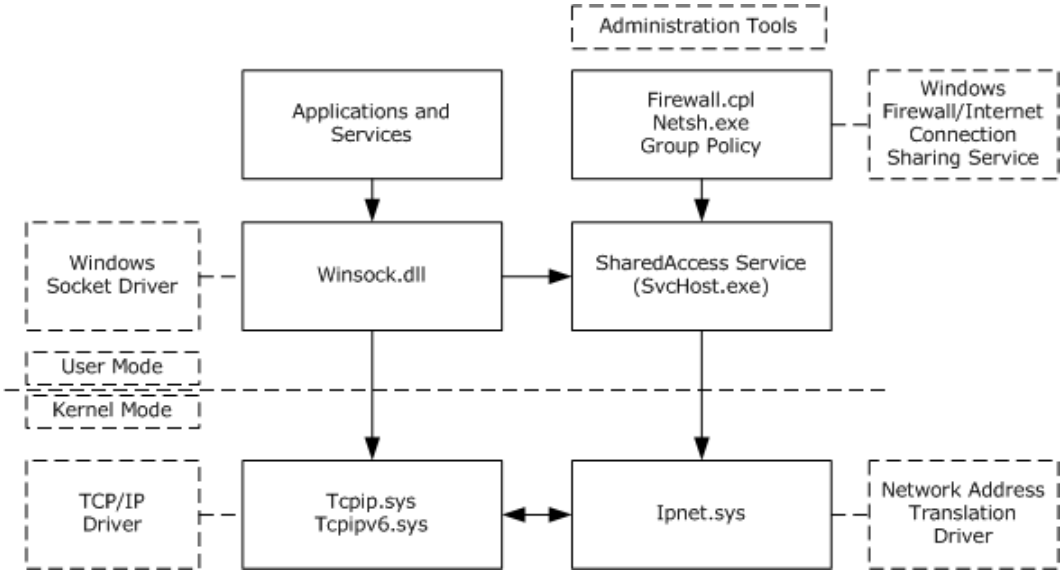

Figure 77: Firewall services functionality overview

- Administrative tools - There are several administrative tools that interact with the Windows Firewall/Internet Connection Sharing service and allow you to configure Windows Firewall settings. These tools include Windows Firewall in Control Panel (Firewall.cpl), the netsh firewall command, and Group Policy with the Group Policy Object Editor.
- The NAT driver (Ipnat.sys) provides a data store for Windows Firewall. Windows Firewall uses the data store, known as the NAT Mapping Table, to store connection state information for dynamically created exceptions. The connection state information typically consists of a 5-tuple

entry, which includes the protocol, the source and destination port numbers, and the source and destination IP addresses. Without a data store, Windows Firewall would not be a stateful firewall. Note that although Windows Firewall uses the NAT driver, it does not provide any network address translation.

- The TCP/IP driver (Tcpip.sys) controls the flow of information between a network adapter and a program or system service. As incoming traffic flows through the TCP/IP driver, the traffic is inspected by the NAT driver. The NAT driver processes the traffic based on the entries in the Windows Firewall exceptions list. If the traffic matches an exception, the NAT driver determines that the traffic is allowed; the packets continue through the TCP/IP driver. If the traffic does not match an exception, the NAT driver determines that the traffic is unsolicited; the packets are dropped and do not continue through the TCP/IP stack. Neither the NAT driver nor the TCP/IP driver sends a notification to the sender when packets are dropped (this is sometimes referred to as a silent discard).
- The Winsock driver is responsible for assigning and binding ports to a program or system service. Programs and system services use this driver when they need to listen for incoming traffic.
- The Windows Firewall/Internet Connection Sharing service runs in Svchost.exe. The Windows Firewall/Internet Connection Sharing service is responsible for processing the Windows Firewall exceptions that you configure for network traffic. The Windows Firewall/Internet Connection Sharing service then relays the exceptions to the NAT driver, which inspects the traffic that is flowing through the TCP/IP driver.

12.3 Protocol Dependencies and Stack Views

This section provides an overview of the basic concepts of the protocols used by Windows Firewall.

12.3.1 Internet Protocol v6

Internet Protocol v6 (IPv6) is designed to be the successor to Internet Protocol v4 (IPv4). IPv6 is an Internet standard, and is documented in [RFC2460]. IPv6 includes the following improvements over IPv4:

- Expanded addressing capability. IPv6 provides 128 bits of address information, as opposed to the 32 bits of address information provided by IPv4.
- Support for extensions and enhancements. IPv6 provides more flexibility for future options, relaxed limits on the length of options, and more efficient forwarding.
- Simplified header format. IPv6 has dropped, or made optional, some of the header fields used by IPv4. This has the effect of lowering the processing cost of common-case packet handling and limits bandwidth cost of the IPv6 header.
- Flow labeling capability. IPv6 includes the ability to label flows of data to allow for special handling of the data such as marking a non-default quality of service or marking the data flow as a real-time service.

12.3.1.1 Stack View

The IPv6 protocol provides network transport services for higher level protocols such as the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).

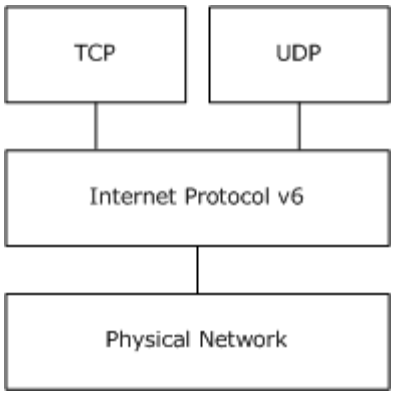

Figure 78: IPv6 Stack View

12.3.2 Authentication Header/Encapsulating Security Payload

IPsec uses two transforms, the Authentication Header (AH) and the Encapsulating Security Payload (ESP) header and trailer, to encapsulate and secure IP packets or payloads. These transforms wrap the IP payload with header or header and trailer information and provide security properties, such as data origin authentication, data integrity, data confidentiality, and replay protection.

Traffic protected with AH or ESP is processed by the IPsec components of Windows before it is processed by Windows Firewall. The IPsec components validate the security information in the AH or ESP header and trailer, decrypt the payload if needed, and remove the AH or ESP header and trailer. The resulting IP packet is handed back to the TCP/IP driver, which is then handed to Windows Firewall for processing. However, you can change this behavior so that IPsec-secured traffic bypasses Windows Firewall by configuring the Windows Firewall: Allow authenticated bypass Group Policy setting. When you enable this setting, you specify groups of IPsec-secured computers whose traffic bypasses all Windows Firewall processing and overrides any existing policies or settings that would otherwise block the traffic.

12.3.2.1 Logical View

Windows Firewall examines traffic protected by either AH or ESP, or both together. The following diagram presents a logical view of AH and ESP.

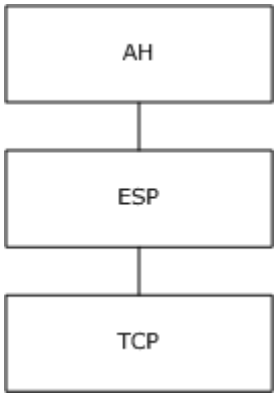

Figure 79: Authentication Header and Encapsulating Security Payload Logical View

12.3.3 Generic Routing Encapsulation

The Generic Routing Encapsulation (GRE) Protocol is used to encapsulate the data channel of Point-to-Point Tunneling Protocol (PPTP) traffic. Any PPTP traffic that uses a GRE header to encapsulate Point-to-Point Protocol (PPP) frames will pass directly through Windows Firewall. Non-PPTP traffic that uses GRE is filtered by Windows Firewall.

12.3.3.1 Logical View

GRE encapsulates data as a payload to be delivered by another protocol, such as IP. The following diagram represents this relationship.

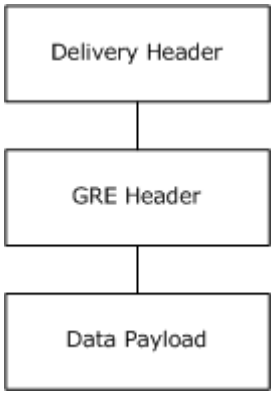

Figure 80: Generic Routing Encapsulation Logical View

12.3.4 Internet Group Management Protocol

Internet Group Management Protocol (IGMP) is an IP multicast maintenance protocol. IP multicast provides the one-to-many delivery of traffic without the overhead and performance impact that is associated with broadcast delivery and multiple unicast delivery. IP multicast is used primarily for audio and video teleconferencing and gaming. Hosts and routers send IGMP messages so that IP multicast-enabled routers can determine the set of multicast host groups that have members on a subnet. All IGMP traffic passes directly through Windows Firewall.

12.3.4.1 Logical View

The following diagram presents a logical view of the relationship between IGMP and IP.

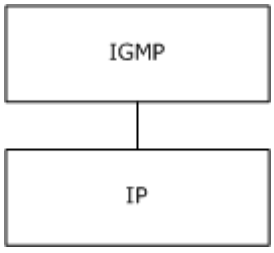

Figure 81: Internet Group Management Protocol Logical View

12.3.5 Pragmatic General Multicast (PGM) Protocol

The Pragmatic General Multicast (PGM) protocol is a scalable multicast protocol that enables receivers to detect loss, request retransmission of lost data, or notify a program of unrecoverable loss. PGM is often referred to as reliable multicast; it is a receiver-reliable protocol, which means the receiver is responsible for ensuring all data is received, absolving the sender of reception responsibility. Any traffic that uses the PGM protocol passes directly through Windows Firewall.

12.3.5.1 Logical View

The following diagram present the logical relationship between PGM and IP.

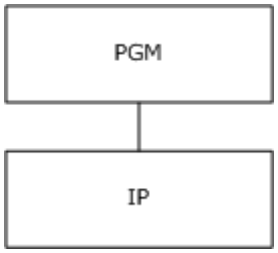

Figure 82: Pragmatic General Multicast Protocol Logical View

13 System Management Services Protocols

This section provides an overview of the System Management Server (SMS) Services protocols. It documents how the protocols are used and gives examples of the protocols in use.

13.1 System Management Services Overview

The System Management Services include a number of protocols that have system management functionality in common. The protocols themselves do not interrelate.

These protocols can be used to provide remote systems management functionality.

13.1.1 Protocols List

The System Management Services include two sets of protocols. The core protocols provide the SMS services functionality, and another set of protocols is required for an implementation of the core protocols (as listed in the Normative References section of the core protocols' technical specifications) or to provide core networking functionality.

The following table lists the core protocols included in the System Management Services. This list does not include data structures, file structures, or algorithms that these core protocols depend on. The details for these technical dependencies are documented in the technical specifications for each core protocol.

Protocol name	Protocol description	Document short name
Eventlog Remoting Protocol Version 1.0	Exposes Remote Procedure Call (RPC) methods for reading events in live and backup event logs on remote computers.	[MS-EVEN]
Eventlog Remoting Protocol Version 6.0	Exposes RPC methods for reading events in live and backup event logs on remote computers.	[MS-EVEN6]
Performance Logs and Alerts Protocol	A set of DCOM interfaces for logging diagnosis data on remote computers.	[MS-PLA]
Task Scheduler Remoting Protocol	Used to register and configure a task and to inquire about the status of running tasks on a remote computer.	[MS-TSCH]
Group Policy	Specifies the details of the communication between clients and domain controllers.	[MS-GPOL]
Windows Management Instrumentation Remote Protocol	A client-server–based framework that provides an open and automated means of systems management.	[MS-WMI]
Web Services Management Protocol Extensions	General-purpose, SOAP-based management protocol that defines procedures for carrying out remote management operations.	[MS-WSMAN]
Web Services Management Protocol Extensions Version 2.0	Microsoft extensions to the WS-Management Protocol (as specified in [DMTF-DSP0226]), the WS-Management Common Information Model (CIM) Binding Specification (as specified in [DMTF-DSP0227]), and the WS-CIM Mapping Specification (as specified in [DMTF-DSP0230]) for accessing CIM objects as a Web service in Windows Vista operating system.	[MS-WSMV]

13.2 System Management Services Functionality

System Management Services include protocols that communicate over HTTP, HTTPS, and TCP. This includes the use of such standard protocols as [DMTF-DSP0226], [DMTF-DSP0227], and [DMTF-DSP0230].

This section describes general functionality of SMS protocols and offers specific background and conceptual information about each protocol.

None of these protocols are related to each other. Each protocol listed perform independently of all other protocols included here.

Section 13.4 provides examples that illustrate the interaction of some of the protocols in two sample configurations.

13.2.1 Eventlog Remote Protocol Specification

Microsoft uses the Eventlog Remote Protocol (as specified in [MS-EVEN]) to read events from live and backup event logs generated by Windows computers. The protocol has the ability to get information about event logs, such as, how many records are contained and the age of those records. It also has the ability to clear or backup the log.

The information contained in the logs is a sequential set of records (also known as events) that can be used by administrators responsible for maintaining and monitoring the target computer. Multiple event logs can be found on each computer.

The records found in the logs are static. After they are written, they are never updated, though in some cases, where event logs have been size restricted and configured to overwrite, older entries can be overwritten by newer information as necessary. Each event written to the log is given an EventID.

Each application that writes data to the log is known as an event source. Multiple event sources can write to a single log file, or multiple log files.

Two kinds of event log exist: live and backup. A live event log is one to which current data can be written. A backup event log is a snapshot in time of a specific event log.

13.2.2 Eventlog Remote Protocol Version 6

Before the release of Windows Vista operating system, there was no versioning done with the changes to the Eventlog Remote Protocol. With the release of the Windows Vista operating system, versioning has been applied to the changes in the Eventlog Remote Protocol, with the Windows Vista version being referred to as Version 6 and getting its own protocol document. The purpose and capabilities of the protocol remain unchanged. For details of the differences between EventLog Remote Protocol versions, see the specific protocol documents [MS-EVEN] and [MS-EVEN6].

13.2.3 Performance Logs and Alerts Protocol Specification

Software components can be designed to assist in serviceability, manageability, supportability, and diagnostics. For instance, performance counters are a simple way of exposing state information that can be sampled or polled. Event-based instrumentation typically generates a state change notification. Alerts are a simple way of turning a sampled counter into an event notification, based on a threshold value.

System administrators often want to collect diagnosis data on a remote system in a periodic or ongoing basis to better support and diagnose problems on the systems. The collected data can also be processed (by tools) for more detailed problem analysis.

The Performance Logs and Alerts Protocol (as specified in [MS-PLA]) provides a set of Distributed Component Object Model (DCOM) interfaces to control data collection on a remote system. The control includes creating, starting, stopping, scheduling, configuration of data collector objects, and creation of alerts.

The capabilities of the Performance Logs and Alerts Protocol are summarized as follows:

- **Performance Counter Logging**: The Performance Logs and Alerts Protocol allows users to log performance counters data of resources on a remote system. The resource could be hardware (for example, CPU, memory) or software (for example, application processes). The logged performance counter data is often useful for the analysis of performance trends and bottlenecks. The Performance Logs and Alerts Protocol also supports logging performance counter data in a SQL database format. This option defines the name of an existing SQL database and log set within the database where the performance counter data will be read or written. This file format is useful when collecting and analyzing performance counter data at an enterprise level rather than on a per-computer basis.
- **Event Trace Logging**: The Performance Logs and Alerts Protocol allows users to log event tracing data of resources on a remote system. The event provider is software that can create event notifications, and generate events when certain activities, such as a disk I/O operation or a page fault, occur. The application that uses the Performance Logs and Alerts Protocol can enable or disable event providers and selectively log the events of interest into a file.
- **API Trace Logging**: The Performance Logs and Alerts Protocol allows users to log the API call activity of an executable on a remote system. Observing API call activity is useful for the diagnosis of various executable issues (for example, detecting unnecessary API calls.)
- **Configuration Data Logging**: The Performance Logs and Alerts Protocol allows users to log the computer configuration information on a remote system. Readjustment of an incorrect setting is one of the common diagnosis root causes.
- **Alerts**: The Performance Logs and Alerts Protocol allows users to create alerts based on performance counter values on a remote system. An alert can trigger running a program, logging the alert as an event, or starting another data collection.
- **Data Collector Set**: The Performance Logs and Alerts Protocol allows users to group multiple logging entities data collectors and apply operations to them at once. The operations include start, stop, schedule, and configure.
- **Data Management**: The Performance Logs and Alerts Protocol allows users to set a data retention policy against logged data and define post-actions of the collection. The post-actions, such as delete largest log file and compress log file, can be defined with the Performance Logs and Alerts Protocol interfaces.

13.2.4 Task Scheduler Remoting Protocol

The Task Scheduler Remoting Protocol consists of three separate RPC interfaces.

- Net Schedule (ATSvc)
- Task Scheduler Agent (SASec)
- Windows Vista operating system Task Remote Protocol (TaskSchedProt)

These interfaces are used to register and configure tasks on a remote server or to query the status of running tasks on those remote servers.

13.2.5 Group Policy Core Protocol

The Group Policy Core Protocol is a client/server protocol that allows clients to discover and retrieve policy settings that administrators of a domain create. Policy settings are administrative directives that administrators make regarding the behavior of the clients. For example, an administrator might want to configure every computer in a certain group of computers to open a specific port in their firewall. That administrator can use Group Policy to state that directive, and it will eventually be communicated to the clients through the Group Policy Protocol.

13.2.5.1 Group Policy Protocols Processing Sequences

The Group Policy Protocol allows clients to discover and retrieve policy settings that administrators of a domain create. Policy settings are administrative directives that administrators make regarding the behavior of the clients. These behaviors (or policy settings) fall into two classes: user policy settings and computer policy settings. User policy settings specify behaviors for interactively logged-in users and can potentially affect different users who are logged in to the same computer. There are also settings that should affect that user no matter what computer the user logs in to. Such settings include the desktop background image for a user or the user's default location for saving documents.

This section describes the various events that trigger Group Policy Processing sequences. This gives a very high level picture of when different events happen when a machine starts up, a user logs on to the machine, a user logs off from the machine and when machine shuts down. At the end of the sequence diagram, references are provided to the different documents that describes these message sequences.

The sequence diagram also shows when the Group Policy User Logon and User Logoff scripts as well as the Active Directory User Object Logon Scripts are run. The Active Directory based User Logon scripts are different from the Group Policy User Logon and Logoff Scripts and are run as part of the user profiles being loaded due to user logon events.

The Active Directory User Object logon scripts need to be stored in the netlogon share of the Active Directory server. The scripts that need to be run as part of the user's logon also need to be defined using the attribute scriptpath as defined in section 2.231 in the Active Directory Schema Attributes N-Z document.

Group Policy User Logon and Logoff scripts provide additional configuration opportunities to an administrator than Active Directory User Object Logon scripts. In addition to also allowing an administrator to specify a script that needs to be run at user logoff, they can also be stored at any location of the administrators choosing and not just the netlogon share.

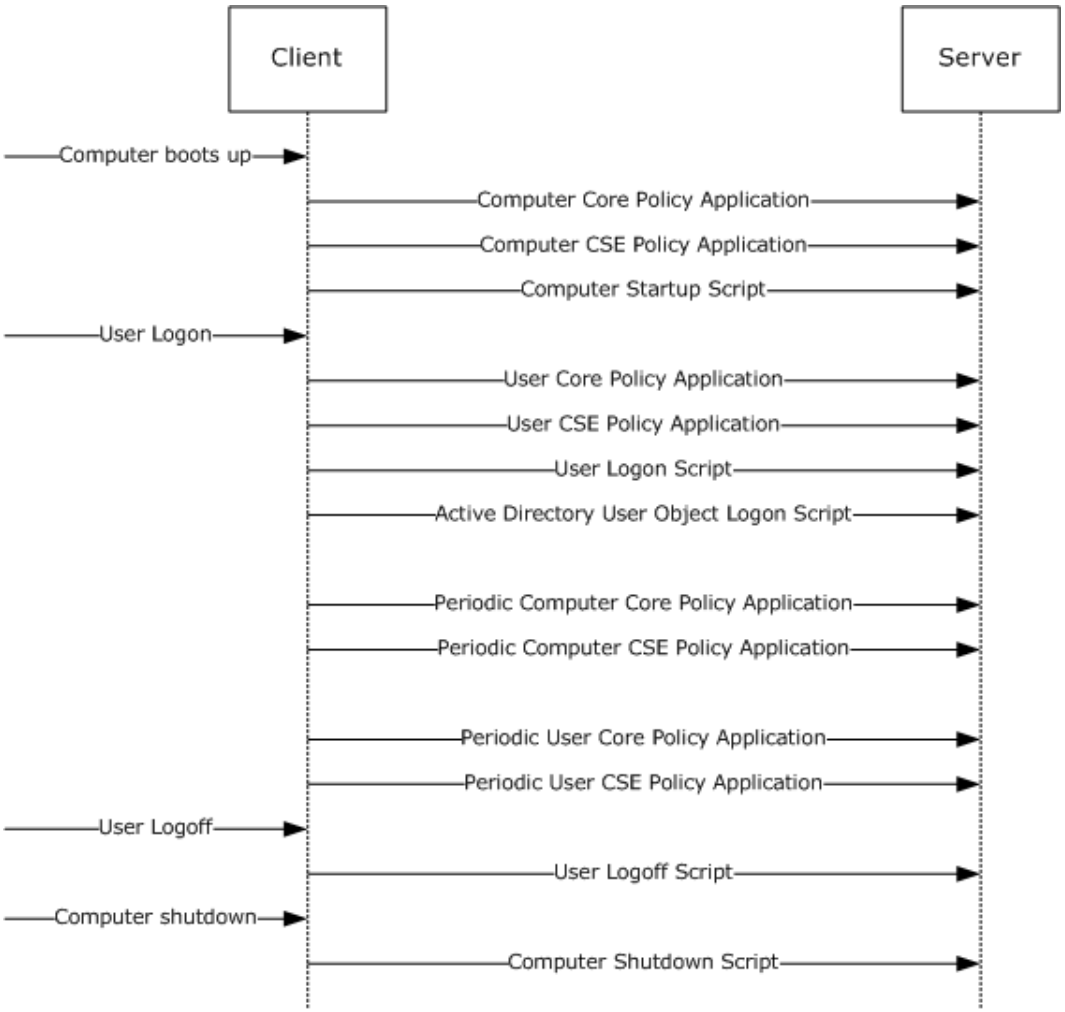

Figure 83: Sequence diagram

Protocol message	Document name	Section
Computer Core Policy Application	[MS-GPOL]: Group Policy: Core Protocol Specification	[MS-GPOL] section 1.3.3
Computer CSE Policy Application	[MS-GPOL]: Group Policy: Core Protocol Specification	[MS-GPOL] section 1.3.3
Computer Startup Scripts	[MS-GPSCR]: Group Policy Scripts Extension: Protocol Specification	[MS-GPSCR] section 3.2.5
User Core Policy Application	[MS-GPOL]: Group Policy: Core Protocol Specification	[MS-GPOL] section 1.3.3
User CSE Policy Application	[MS-GPOL]: Group Policy: Core Protocol Specification	[MS-GPOL] section 1.3.3
User Logon Scripts	[MS-GPSCR]: Group Policy Scripts Extension: Protocol Specification	[MS-GPSCR] section 1.3

Protocol message	Document name	Section
Periodic Computer Core Policy Application	[MS-GPOL]: Group Policy: Core Protocol Specification	[MS-GPOL] section 1.3.3
Periodic Computer CSE Policy Application 1	[MS-GPOL]: Group Policy: Core Protocol Specification	[MS-GPOL] section 1.3.3
Periodic User Policy Core Application	[MS-GPOL]: Group Policy: Core Protocol Specification	[MS-GPOL] section 1.3.3
Periodic User CSE Policy Application	[MS-GPOL]: Group Policy: Core Protocol Specification	[MS-GPOL] section 1.3.3
User Logoff Scripts	[MS-GPSCR]: Group Policy Scripts Extension: Protocol Specification	[MS-GPSCR] section 1.3
Computer Shutdown Scripts	[MS-GPSCR]: Group Policy Scripts Extension: Protocol Specification	[MS-GPSCR] section 1.3
Attribute scriptPath	[MS-ADA3]: Active Directory Schema Attributes N-Z	[MS-ADA3] section 2.232

Please refer to the following documents for the specific message exchanged by the different Client Side Extensions.

[MS-GPDPC]: Group Policy Protocol: Deployed Printer Connections Extension

[MS-GPEF]: Group Policy: Encrypting File System Extension

[MS-GPFR]: Group Policy: Folder Redirection Protocol Extension

[MS-GPIE]: Group Policy: Internet Explorer Maintenance Extension

[MS-GPIPSEC]: Group Policy: IP Security (IPSec) Protocol Extension

[MS-GPREG]: Group Policy: Registry Extension Encoding

[MS-GPSB]: Group Policy: Security Protocol Extension

[MS-GPSCR]: Group Policy: Scripts Extension Encoding

[MS-GPSI]: Group Policy: Software Installation Protocol Extension

[MS-GPWL]: Group Policy: Wireless/Wired Protocol Extension

13.2.6 Windows Management Instrumentation Remote Protocol

The Windows Management Instrumentation Remote Protocol is the Microsoft implementation of CIM as specified in [DMTF-DSP0004]. The Windows Management Instrumentation Remote Protocol uses CIM as the conceptual model for representing enterprise management information that can be managed by a Windows administrator. For more information, see section 19.6.1.

13.2.7 Web Services Management Protocol Extensions

The Web Services Management Protocol Extensions for Windows Server 2003 operating system are a set of modifications to the WS-Management Protocol (as specified in [DMTF-DSP0226]), the WS-

Management CIM Binding Specification (as specified in [DMTF-DSP0227]), and the WS-CIM Mapping Specification (as specified in [DMTF-DSP0230]) for compatibility with Windows Server 2003.

It should be noted that conformance to these extensions does not make an implementation compatible with the WS-Management Protocol specifications as currently published by the DMTF. Instead, conformance makes an implementation compatible with Microsoft's implementation of WS-Management in Windows Server 2003, which was based on the pre-release drafts of the WS-Management Protocol specifications available at the time Windows Server 2003 was implemented and released.

The WS-Management Protocol can provide remote access to CIM objects, as specified by [DMTF-DSP0004]. WS-Management servers expose a set of entities, which can be managed as objects with attributes and methods. WS-Management clients perform management tasks by issuing object operations against objects exposed by WS-Management servers.

13.2.8 Web Services Management Protocol Extensions Version 2

Web Services Management Protocol Extensions for Windows Vista (as specified in [MS-WSMV]) are a set of additions and modifications to the Web Services for Management (WS-Management) Protocol (as specified in [DMTF-DSP0226]), the WS-Management CIM Binding specification (as specified in [DMTF-DSP0227]), and the WS-CIM Mapping specification (as specified in [DMTF-DSP0230]) for compatibility with Windows Vista operating system and Windows Server 2008 operating system. Web Services Management Protocol Extensions for Windows Vista clients perform management tasks by issuing object operations against objects exposed by the Web Services Management Protocol Extensions for Windows Vista service.

13.3 SMS Protocol Logical Dependencies and Protocol Stack Views

The protocols included here are not related to each other. They implement independently of each other. Unless otherwise specified their only dependencies are those of the individual protocols transport mechanisms (RPC or DCOM).

13.3.1 Eventlog Remote Protocol Specification Logical Dependencies

The Eventlog Remote Protocol (as specified in [MS-EVEN]) uses RPC to read data from active event logs and backup event logs.

13.3.1.1 Stack View

The following diagram illustrates the stack view of the Eventlog Remote Protocol, as specified in [MS-EVEN].

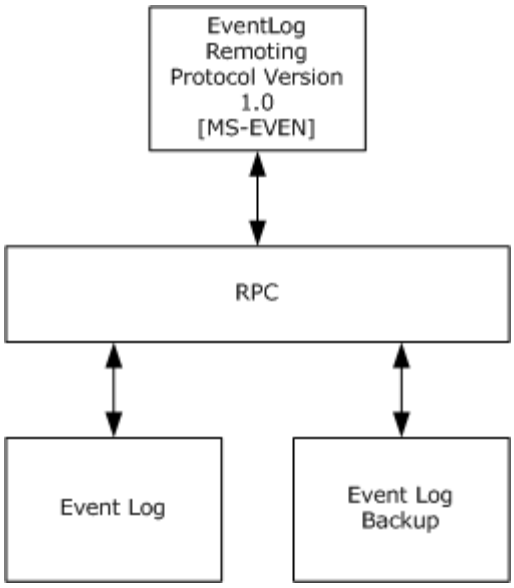

Figure 84: Eventlog Remote Protocol stack view

13.3.2 Eventlog Remote Protocol Specification Version 6 Logical Dependencies

The Eventlog Remote Protocol Version 6.0 (as specified in [MS-EVEN6]) uses RPC to read data from active event logs and backup event logs.

13.3.2.1 Stack View

The following diagram illustrates the stack view of the Eventlog Remote Protocol Version 6.0, as specified in [MS-EVEN6].

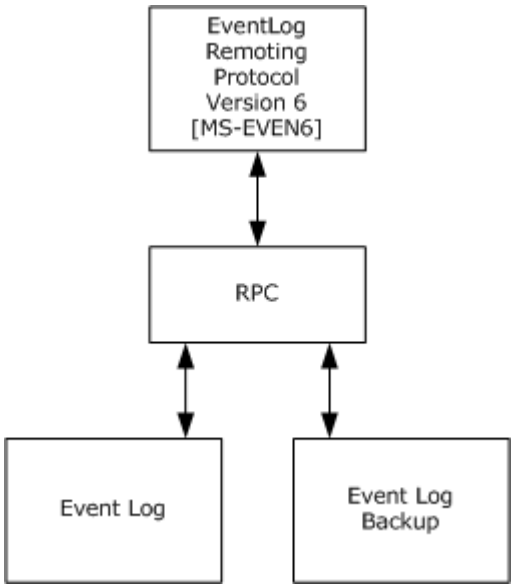

Figure 85: Eventlog Remote Protocol Version 6 stack view

13.3.3 Performance Logs and Alerts Protocol Specification Logical Dependencies

The Performance Logs and Alerts Protocol (as specified in [MS-PLA]) is implemented using DCOM over RPC.

13.3.3.1 Stack View

The following diagram illustrates the stack view of the Performance Logs and Alerts Protocol, as specified in [MS-PLA].

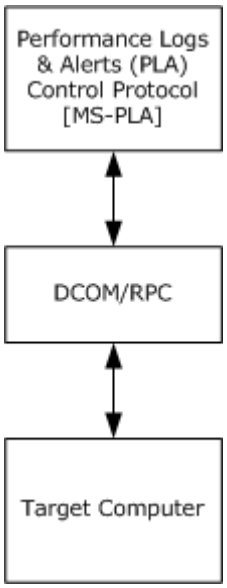

Figure 86: Performance Logs and Alerts stack view

13.3.4 Task Scheduler Remoting Protocol Logical Dependencies

The ATSvc, SASec, and TaskSchedProt interfaces in the Task Scheduler Remoting Protocol use the RPC protocol [C706] for transport. The SASec interface requires that clients also use the Windows Remote Registry Protocol Specification (see [MS-RRP]), which uses RPC, and a remote file system protocol.

No higher-layer protocols make use of the AT Service Remote Protocol.

13.3.4.1 Task Scheduler Remoting Protocol Stack View

The stack for the Task Scheduler Remoting Protocol is best described in terms of logical relationships.

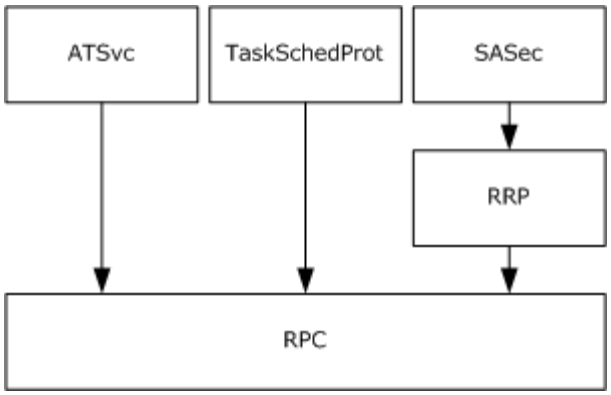

Figure 87: Task Scheduler Remoting Protocol Stack View

13.3.5 Group Policy Logical Dependencies

The Group Policy Protocol is not able to communicate policy settings to other computers directly. It communicates only by extending supported protocols and allowing those protocols to query Group Policy Objects to determine how policy should be applied.

The Group Policy Protocol depends on the following protocols to allow it to exchange information between a Client and a Group Policy server.

- Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions (as specified in [MS-SPNG]) for authentication.
- Kerberos Network Authentication Service (V5) Extensions (as specified in [MS-KILE]) for authentication.
- NT LAN Manager (NTLM) Authentication Protocol (as specified in [MS-NLMP]) for authentication.
- Domain Controller (DC) Locator, as specified in [MS-NRPC], for discovering the Group Policy server.
- DNS for discovering the Group Policy server.
- Server Message Block (SMB) Protocol (as specified in [MS-SMB] and [MS-SMB2]) for transmitting Group policy settings and instructions between the Client and the Group Policy server.
- Distributed File System (DFS): Referral Protocol (as specified in [MS-DFSC]) to provide location-independent access to the Group Policy server for Clients during policy application and policy administration.
- Lightweight Directory Access Protocol (v3), as specified in [RFC2251], for transmitting Group Policy settings and instructions between the Client and the Group Policy server.

13.3.5.1 Stack View

The following diagram shows the Group Policy stack view.

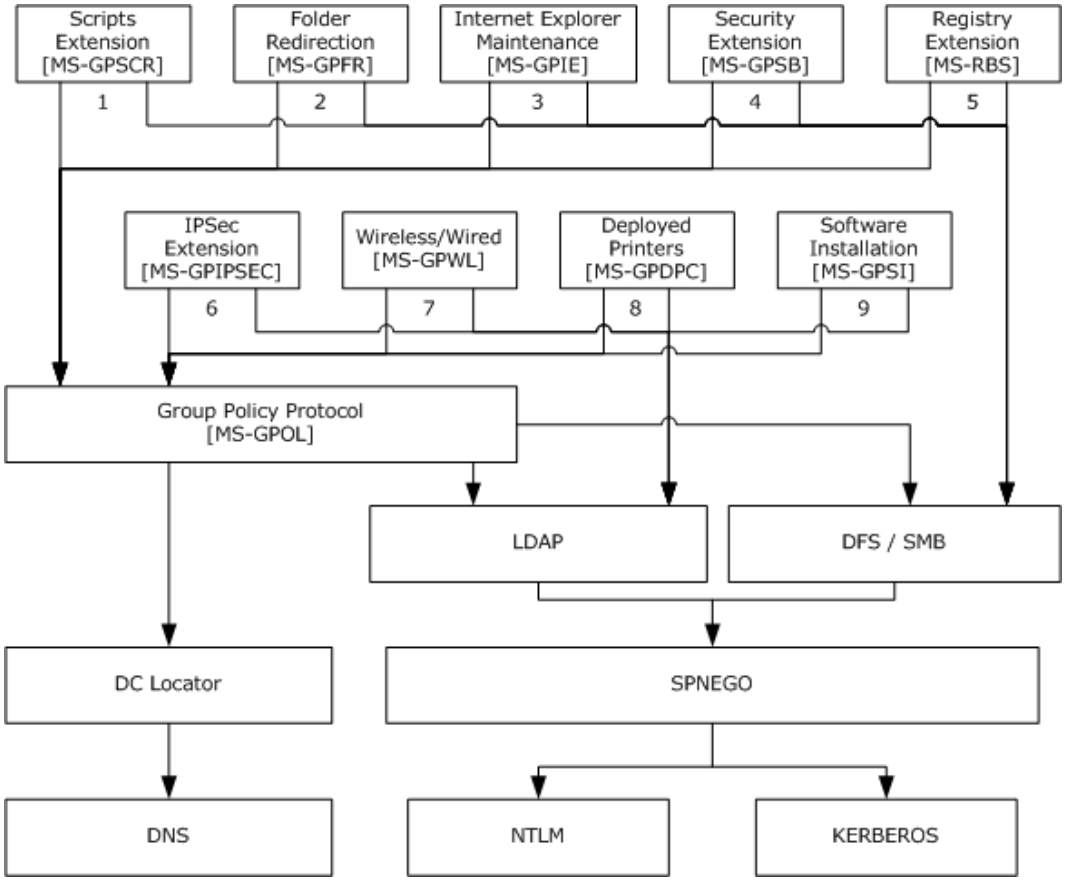

Figure 88: Stack view

13.3.6 WS-Management Protocol Logical Dependencies

The WS-Management Protocol uses SOAP over HTTP or HTTPS for communication. The WS-Management Protocol(as specified in [DMTF-DSP0226]) must be used as the transport to provide access to CIM data using binding techniques specified by [DMTF-DSP0227] and mapping techniques specified by [DMTF-DSP0230]. The Web Services Management Protocol Extensions for Windows Server 2003 operating system Protocol specifies the differences between the protocols as defined in early drafts and supported in Windows Server 2003 and the protocols as specified in [DMTF-DSP0226], [DMTF-DSP0227], and [DMTF-DSP0230].

The Windows Management Instrumentation Remote Protocol (as specified in [MS-WMI]) is an alternate network protocol for accessing CIM data on servers. There is no other logical relationship between the WS-Management Protocol and the WMI Remote Protocol. The WMI Remote Protocol is described in section 19.6 of this document.

13.3.6.1 Stack View

The following illustrates the stack view of the WS-Management Protocol.

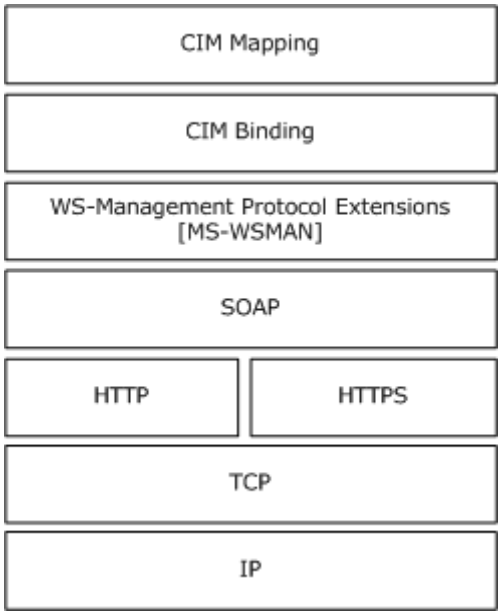

Figure 89: Web Services Management stack view

13.3.7 WS-Management Protocol Version 2 Logical Dependencies

The WS-Management Protocol Version 2 uses SOAP over HTTP or HTTPS for communication. The WS-Management Protocol, as specified in [DMTF-DSP0226], must be used as the transport to provide access to CIM data using binding techniques specified by [DMTF-DSP0227] and mapping techniques specified by [DMTF-DSP0230]. The WS-Management Protocol Version 2 Extensions specifies the differences between the protocols as defined in early drafts and supported in Windows Server 2008 and the protocols as specified in [DMTF-DSP0226], [DMTF-DSP0227], and [DMTF-DSP0230].

The Windows Management Instrumentation Remote Protocol (as specified in [MS-WMI]) is an alternate network protocol for accessing CIM data on servers, in which case. There is no other logical relationship between the WS-Management Protocol and WMI Remote Protocol. The WMI Remote Protocol is described in section 19.6 of this document.

13.3.7.1 Stack View

The following illustrates the stack view of the WS-Management Protocol Version 2.

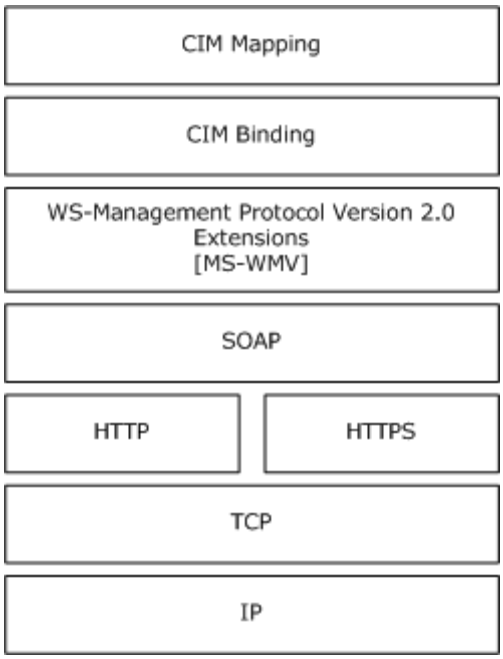

Figure 90: Web Services Management Version 2 stack view

Management protocols can be used for a variety of scenarios. Two such scenarios have been traced and analyzed in the following section.

13.4 Implementation Scenarios

This section describes two scenarios that illustrate common functionality that can be implemented with the protocols included in Systems Management Services.

- Reading Server event log from Windows Vista operating system client
- Configuring and monitoring performance counters on remote server.

13.4.1 Scenario 1 - Reading Server Event Log

13.4.1.1 Scenario Overview

This section describes the network traffic related to reading a remote server event log.

13.4.1.2 Scenario Configuration

The following machine configurations were used for this scenario.

WS200301

- Machine Name: WS200301.contoso.com (WS200301)
- Operating System: Windows Server 2003 R2 operating system with latest updates
- IP Address: 10.0.10.1
- Subnet Mask: 255.255.255.0

- Active Directory Domain Controller – contoso.com

WS200302

- Machine Name: WS200302.contoso.com (WS2003021)
- Operating System: Windows Server 2003 R2 with latest updates
- IP Address: 10.0.10.2
- Subnet Mask: 255.255.255.0
- DNS – 10.0.10.1
- Domain Member – contoso.com

ClientVista

- Machine Name: ClientVista.contoso.com (ClientVista)
- Operating System: Windows Vista operating system Ultimate with latest updates
- IP Address: 10.0.10.10
- Subnet Mask: 255.255.255.0
- DNS – 10.0.10.1
- NetMon 3.1
- Domain Member – contoso.com

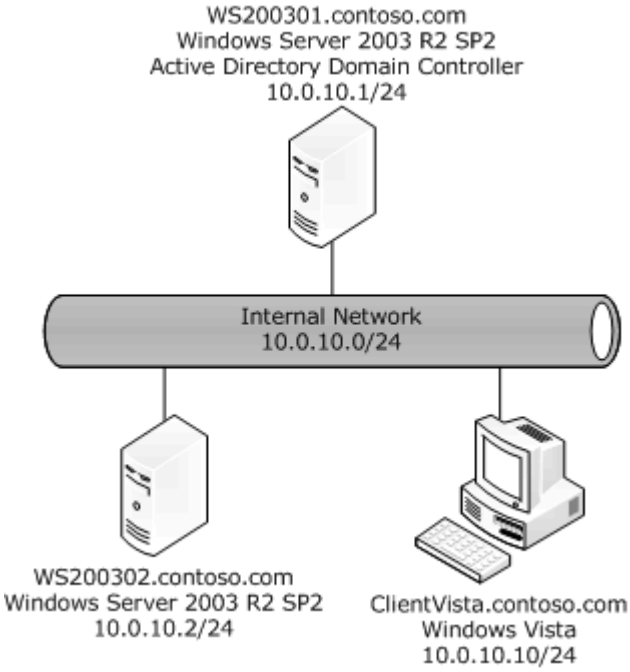

Figure 91: Network topology

13.4.1.3 Setting up the Trace

1. Start trace.
2. In Event Viewer, on the **Action** menu, click **Connect to Another Computer**.
3. Type "WS200302", and then click **OK**.
4. In the console tree, expand **Classic Event Viewer | Global Logs**, and then click **Application**.
5. In the details pane, double-click an event. (In this case it was an information event from source LoadPerf, event ID 1000.)
6. Click **Cancel**.
7. Close Event Viewer.
8. End trace.

13.4.1.4 Netmon Trace Digest

While there is significant traffic relating to authenticating the request and moving the requested data from server to client, the key information related to this task is tied to the Eventlog Remote Protocol, as shown by this communication between the Windows Vista operating system client and the WS200302 server.

```
CLIENTVISTA WS200302 EVENTLOG EVENTLOG: ElfrOpenELW Request, Module: Application
WS200302 CLIENTVISTA EVENTLOG EVENTLOG: ElfrOpenELW Response,
  Handle: {00000000-9E0C511F-CF1C-9E44-AD29-4B7B80E99241},
  Status = 0x00000000 - STATUS_SUCCESS
CLIENTVISTA WS200302 EVENTLOG EVENTLOG: ElfrNumberOfRecords Request,
  Handle: {00000000-9E0C511F-CF1C-9E44-AD29-4B7B80E99241}
WS200302 CLIENTVISTA EVENTLOG EVENTLOG: ElfrNumberOfRecords Response,
  Num Records: 103, Status = 0x00000000 - STATUS_SUCCESS
CLIENTVISTA WS200302 EVENTLOG EVENTLOG: ElfrOldestRecord Request,
  Handle: {00000000-9E0C511F-CF1C-9E44-AD29-4B7B80E99241}
WS200302 CLIENTVISTA EVENTLOG EVENTLOG: ElfrOldestRecord Response,
  Oldest Record: 1, Status = 0x00000000 - STATUS_SUCCESS
CLIENTVISTA WS200302 EVENTLOG EVENTLOG: ElfrReadELW Request,
  Record Offset: 0x67, NumBytes: 65536, Flags:  Seek Back,
  Handle: {00000000-9E0C511F-CF1C-9E44-AD29-4B7B80E99241}
WS200302 CLIENTVISTA EVENTLOG EVENTLOG: ElfrReadELW Response,
  Bytes Read: 0, Min Bytes Needed: 0, Status = 0x00000000 - STATUS_SUCCESS,
  Invalid SID: Revision (Array[94,9])
```

13.4.2 Scenario 2 - Configuring and Monitoring Remote Performance Counters

13.4.2.1 Scenario Overview

This section describes the network traffic related to remote performance monitoring tasks. In this scenario, Perfmon is remotely launched, configured, and monitored.

13.4.2.2 Scenario Configuration

The following machine configurations were used for this scenario.

WS200301

- Machine Name: WS200301.contoso.com (WS200301)
- Operating System: Windows Server 2003 R2 operating system with latest updates
- IP Address: 10.0.10.1
- Subnet Mask: 255.255.255.0
- Active Directory Domain Controller – contoso.com

WS200302

- Machine Name: WS200302.contoso.com (WS200302)
- Operating System: Windows Server 2003 R2 with latest updates
- IP Address: 10.0.10.2
- Subnet Mask: 255.255.255.0
- DNS – 10.0.10.1
- Domain Member – contoso.com

ClientVista

- Machine Name: ClientVista.contoso.com (ClientVista)
- Operating System: Windows Vista operating system Ultimate with latest updates
- IP Address: 10.0.10.10
- Subnet Mask: 255.255.255.0
- DNS – 10.0.10.1
- NetMon 3.1
- Domain Member – contoso.com

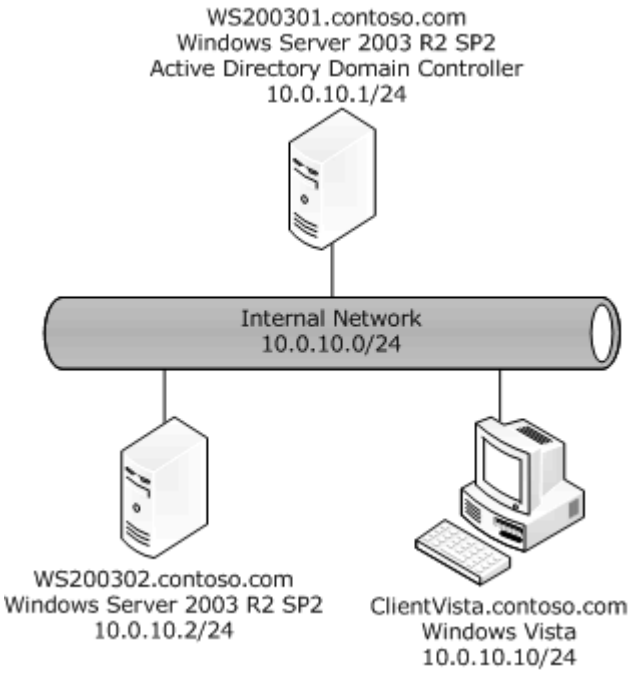

Figure 92: Network topology

13.4.2.3 Setting up the Trace

1. Open Reliability and Performance Monitor.
2. Start trace.
3. In Reliability and Performance Monitor, on the **Action** menu, click **Connect to another computer**.
4. Type "WS200302", and then click **OK**.
5. In the console tree, expand **Monitoring Tools**, and then click **Performance Monitor**.
6. On the toolbar, click **Add**.
7. In the **Select counters from this computer** list, type "\\WS200302".
8. In the counter list, expand **IPv4**, click **Datagrams Received/sec**, and then click **Add**.
9. In the counter list, expand **PhysicalDisk**, click **% Idle Time**, and then click **Add**.
10. In the counter list, expand **System**, click **System Up Time**, and then click **Add**.
11. Click **OK**.
12. Let performance monitor run for 30 seconds.
13. Close Reliability and Performance Monitor.
14. End trace.

13.4.2.4 Netmon Trace Digest

All traffic for the Performance Logs and Alerts Protocol (as specified in [MS-PLA]) was passed over MSRPC as documented in the sample packet traces shown below. After the connection is made to the remote performance log, the continuation data continues to be transmitted between client and server over RPC.

```
CLIENTVISTA WS200302 MSRPC MSRPC: c/o Alter Cont:
  UUID{000001A0-0000-0000-C000-000000000046} DCOM-IRemoteSCMActivator  Call=0x2D
WS200302 CLIENTVISTA MSRPC MSRPC: c/o Alter Cont Resp:
  Call=0x2D  Assoc Grp=0x2E60E  Xmit=0x16D0  Recv=0x16D0
CLIENTVISTA WS200302 MSRPC MSRPC: c/o Alter Cont:
  UUID{338CD001-2244-31F1-AAAA-900038001003} WINREG  Call=0x1
CLIENTVISTA WS200302 MSRPC MSRPC: c/o Bind:
  UUID{E1AF8308-5D1F-11C9-91A4-08002B14A0FA} Endpoint Mapper
  Call=0x1  Assoc Grp=0x0  Xmit=0x16D0  Recv=0x16D0
WS200302 CLIENTVISTA MSRPC MSRPC: c/o Bind Ack:
  Call=0x1  Assoc Grp=0x2E60F  Xmit=0x16D0  Recv=0x16D0
WS200302 CLIENTVISTA MSRPC MSRPC: c/o Alter Cont Resp:
  Call=0x1  Assoc Grp=0x34D32  Xmit=0x10B8  Recv=0x10B8
CLIENTVISTA WS200302 MSRPC MSRPC: c/o Alter Cont:
  UUID{338CD001-2244-31F1-AAAA-900038001003} WINREG Call=0x1
WS200302 CLIENTVISTA MSRPC MSRPC: c/o Continued Response:
  WINREG {338CD001-2244-31F1-AAAA-900038001003} Call=0x10
  Context=0x0  Hint=0x9DA4  Cancels=0x0
WS200302 CLIENTVISTA MSRPC MSRPC: c/o Continued Response:
  WINREG {338CD001-2244-31F1-AAAA-900038001003} Call=0x10
  Context=0x0  21639.056160 {MSRPC:43, SMB:42, NbtSS:37, TCP:35, IPv4:1}
  Hint=0x8D44 Cancels=0x0
```

14 Terminal Services Protocols

This section provides an overview of the Terminal Services protocols. It provides a high-level conceptual overview of the core Terminal Services protocols.

14.1 Terminal Services Overview

Terminal Services protocols provide services for communicating remote graphical desktop interaction and display data packets, and sound, file redirection, and print redirection data packets from applications accessed by authorized Windows clients to a Windows server configured as a Terminal Server.

14.1.1 Protocols List

Terminal Services include two sets of protocols: the core protocols that provide the Terminal Services functionality, and a set of protocols that are required for an implementation of the core protocols (as listed in the Normative References section of the core protocols' technical specifications), or that provide core networking functionality.

The following table lists the core protocols included in Terminal Services. This list does not include data structures, file structures, or algorithms that these core protocols depend on. The details for these technical dependencies are documented in the technical specifications for each core protocol.

Protocol name	Protocol description	Document short name
Remote Assistance Protocol	Used after a remote assistance connection is established to facilitate different capabilities used during the connection. This protocol supports five capabilities: session initialization, file transfer, chat, share control, and Voice over Internet Protocol (VoIP) control.	[MS-RA]
Remote Assistance Initiation Protocol	Provides a set of Distributed Component Object Model (DCOM) interfaces that enable an Expert to retrieve Remote Assistance (RA) connection-specific data from a remote Novice computer to initiate an RA connection.	[MS-RAI]
Remote Desktop Protocol: Basic Connectivity and Graphics Remoting	Facilitates user interaction with a remote computer system by transferring graphics display information from the remote computer to the user and transporting input from the user to the remote computer, where the input may be injected locally. This protocol also provides an extensible transport mechanism that allows specialized communication to take place between components on the user computer and components running on the remote computer.	[MS-RDPBCGR]
Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension	Implements a generic connection-oriented communication channel on top of the virtual channel protocol. A dynamic virtual channel is established over an existing static virtual channel.	[MS-RDPEDYC]
Remote Desktop Protocol: GDI Acceleration Extensions	Encodes the drawing operations that produce an image instead of encoding the actual image, thereby reducing bandwidth associated with graphics remoting.	[MS-RDPEGDI]

Protocol name	Protocol description	Document short name
Remote Desktop Protocol: Clipboard Virtual Channel Extension	Provides basic programmatic access to the clipboard provided by an operating system (OS); usually ensures that any application has the capability to place data onto the clipboard, extract data from the clipboard, enumerate the data formats available on the clipboard, and register to receive notifications when the system clipboard is updated.	[MS-RDPECLIP]
Remote Desktop Protocol: File System Virtual Channel Extension	Gives access between the server and the client file system drivers by redirecting all input/output requests and responses between the two.	[MS-RDPEFS]
Remote Desktop Protocol: Print Virtual Channel Extension	Specifies the communication used to enable the redirection of printers between a terminal client and a Terminal Server. By redirecting printers from the terminal client to the Terminal Server, applications running on a server machine can access the remote devices as if they were local printers.	[MS RDPEPC]
Remote Desktop Protocol: Licensing Extension	Allows authorized Remote Desktop Clients or users to connect to a Terminal Server. This extension involves communication between the Remote Desktop Client, the Terminal Server and a License Server. The Terminal Server can be configured to be in per device or per user license mode. Client Access Licenses (CALs) are installed on a License Server, and when a Terminal Server requests a license on a client's behalf, the License Server issues a license out of its available pool of licenses.	[MS-RDPELE]
Remote Desktop Protocol: Plug and Play Devices Virtual Channel Extension	Specifies the communication used to enable the redirection of Plug and Play devices between a Terminal Server client and Terminal Server server.	[MS-RDPEPNP]
Remote Desktop Protocol: Remote Programs Virtual Channel Extension	Is a Remote Desktop Protocol (RDP) feature (as specified in [MS-RDPBCGR]) that presents a remote application (running remotely on a Remote Applications Integrated Locally (RAIL) server) as a local user application (running on the RAIL client machine). Also known as RAIL.	[MS-RDPERP]
Remote Desktop Protocol: Smart Card Virtual Channel Extension	Enables client smart card devices to be available, within the context of a single RDP session, to server-side applications.	[MS-RDPESC]
Remote Desktop Protocol: XPS Print Virtual Channel Extension	Specifies communication between a virtual printer driver installed on a Terminal Server and a printer driver installed on the Terminal Server client. The primary purpose of this protocol is to acquire printing capabilities and to display a printer-specific user interface on the Terminal Server client.	[MS-RDPEXPS]
Terminal Services Gateway Server Protocol	Provides remote connections from Terminal Services Gateway (TSG) clients originating on the Internet to target servers behind a firewall. Based on the Remote Procedure Call Over HTTP Protocol, as specified in [MS-RPCH].	[MS-TSGU]
Terminal Services Terminal Server Runtime Interface	Is a remote procedure call (RPC)-based protocol used for remotely querying and configuring various aspects of a Terminal Server. For example, this protocol can be used to query the number of active	[MS-TSTS]

Protocol name	Protocol description	Document short name
(RPC)	sessions running on a Terminal Server.	
Remote Desktop Protocol: Audio Output Virtual Channel Extension	Transfers audio data from the server to the client. For example, when the server plays an audio file, this protocol is used by the server to transfer the audio data to the client. The client may then play the audio.	[MS-RDPEA]
Remote Desktop Protocol: Multiparty Virtual Channel Extension	Enables the remote display of desktop and application content. To effectively implement an application-sharing or collaborative solution, additional information must be conveyed to keep the participants apprised of who else is involved, in addition to which applications or windows are being shared.	[MS-RDPEMC]
Remote Desktop Protocol: Session Selection Extension	Describes the messages exchanged between an RDP client and a terminal server to facilitate the precise targeting of an application sharing context.	[MS-RDPEPS]
Remote Desktop Protocol: Serial Port Virtual Channel Extension	Specifies the communication used to enable the redirection of ports between a terminal client and a terminal server. By redirecting ports from the terminal client to the terminal server, applications running on a server machine can access the remote devices attached to those ports.	[MS-RDPESP]

14.2 Terminal Services Protocols Concepts

Terminal Services allows clients (remote computer systems or devices) to remotely execute Windows-based applications on the server, or to remotely access the Windows desktop itself. When users run an application on Terminal Services, the application execution takes place on the server, transmitting keyboard, mouse, and display information over the network between the server and the client.

Connections to the server may be made from a wide range of devices, over diverse network connections. Terminal Services allows a server to host multiple, simultaneous client sessions. Each client session is independent of any other client session and is managed transparently by the server.

A Terminal Services client can exist in various forms.

- Thin-client hardware devices that run an embedded Windows-based operating system can run the Terminal Services client software to connect to a Terminal Server.
- Windows, Mac OS, or UNIX computers can run Terminal Services client software to connect to a Terminal Server to display Windows-based applications. This combination of Terminal Services clients provides access to Windows-based applications from both Windows and non-Windows OSs.
- Remote Assistance allows an expert computer user to render assistance to a novice computer user. The novice computer user can request the expert logon through Remote Assistance to correct a problem remotely while the novice user watches the session.

The following sections describe the various core protocols used in Terminal Services implementations.

14.2.1 Remote Assistance Initiation Protocol

The Remote Assistance Initiation Protocol (as specified in [MS-RAI]) is a set of Distributed Component Object Model (DCOM) interfaces for initiating a Remote Assistance (RA) connection to a computer. The Remote Assistance Initiation Protocol allows an authorized Expert to start RA on a remote Novice computer to retrieve data that is required to make an RA connection from the Expert's computer to the Novice's computer.

For purposes of this protocol, an Expert is defined as the side of a RA connection that is able to view the remote screen of the other computer in order to provide help. A Novice is defined as the side of a RA connection that shares its screen with the other computer in order to receive help. The Expert is the DCOM client and the Novice is the DCOM server.

The following are two prerequisites for using this protocol.

1. The Expert must have the IP address or fully qualified domain name (FQDN) of the Novice's computer in order to use this protocol.

2. Before the Expert's DCOM call is executed on the Novice computer, DCOM performs a check to verify that the Expert is on the list of authorized RA helpers on the Novice computer.

14.2.2 Remote Assistance Protocol

The Remote Assistance Protocol (as specified in [MS-RA]) is used to support communications and control between two computers to allow an Expert to view or "desktop shadow" the remote screen of a Novice and thereby provide RA to the Novice. The functions supported by the Remote Assistance Protocol are as follows:

- File transfer
- Chat (text message exchange)
- Session control
- Voice-over-IP (VoIP) control

To accomplish these functions, the Remote Assistance Protocol uses four virtual channels as its underlying transport.

- Session initialization virtual channel: used to do initial setup and configuration of the RA connection and persists through the duration of the RA connection.
- File transfer virtual channel: created on-demand to transfer file data.
- Chat virtual channel: created when the RA connection is first established and persists through the duration of the RA connection.
- Channel for share control, initializing VoIP and file transfer.

14.2.3 Remote Desktop Protocol Extensions

Remote Desktop Protocol (RDP) is the core protocol used to provide terminal services functionality. Microsoft has also defined numerous extensions to RDP, which are used in various contexts such as:

- Graphics remoting
- Printing and print redirection

- Smart card implementation
- Plug and play (PNP) functions
- Clipboard functions

RDP is based upon, and is an extension of, the International Telecommunication Union (ITU) T.120 family of protocol standards. The T.120 standard comprises a suite of communication and application protocols developed and approved by the international computer and telecommunications industries. These protocols are used to create compatible products and services for real-time, multipoint data connections, and conferencing. With T.120-based programs, multiple users can participate in conferencing sessions over different types of networks and connections.

Microsoft proposed the T.128 standard as an addition to the T.120 standard. T.128 is accepted by the ITU, Telecommunication Standardization Sector (ITU-T). T.128 specifies the program sharing protocol, defining how participants in a T.120 conference can share local programs. Specifically, T.128 enables multiple conference participants to view and collaborate on shared programs, and it is the foundation for RDP.

A multichannel-capable protocol allows for separate virtual channels for carrying presentation data, serial device communication, licensing information, highly encrypted data (keyboard and mouse activity), and so on. Because RDP is an extension of the core T.120 protocol suite, several other capabilities are retained as part of RDP, such as the architectural features necessary to support multipoint (multiparty) sessions. Multipoint data delivery allows data from an application to be delivered in "real-time" to multiple parties without requiring the same data to be sent to each session individually. However, application sharing (collaboration) is not in the scope of this task and is fully documented in the Collaboration Server Services section of this Protocol document.

RDP versions 5.1, 5.2, 6.0, 6.1, and 7.0 support virtual channels. Virtual channels are custom data formats handled independently of the RDP protocol. After an RDP connection is made, and basic settings are exchanged between server and client, one or more virtual channels are established over a static channel (see the Message Flows—Standard Connection Sequence section of this Terminal Server document). Thereafter, until the disconnect sequence, all activity between client and server (for example, file operations, printing, graphics, and application remoting) takes place on these virtual channels.

14.2.4 Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Overview

Remote Desktop Protocol: Basic Connectivity and Graphics Remoting (as specified in [MS-RDPBCGR]) is the core protocol that provides Terminal Services functionality in Windows Vista operating system and Windows Server 2008 operating system (RDP 6.0).

This protocol facilitates user interaction with a remote computer system by transferring graphics display information from the remote computer to the user and transporting input from the user to the remote computer, where it can be injected locally. This protocol also provides an extensible transport mechanism, which allows specialized communication to take place between components on the user computer and components running on the remote computer.

The activity involved in sending and receiving data through the RDP stack is essentially the same as in the seven-layer Open System Interconnection (OSI) model standards for common local-area network (LAN) networking. Data from an application or service to be transmitted is passed down through the protocol stack, sectioned, directed to a channel (through Multipoint Communication Services), encrypted, wrapped, framed, packaged onto the network protocol, and finally addressed and sent over the wire to the client. The returned data works the same way, only in reverse, with

the packet being stripped of its address, then unwrapped, decrypted, and so on until the data is presented to the application for use.

14.2.5 Remote Desktop Protocol: Dynamic Virtual Channel Extension

The Remote Desktop Protocol: Dynamic Virtual Channel Extension (as specified in [MS-RDPEDYC]) implements a generic connection-oriented communication channel on top of the virtual channel protocol. The Remote Desktop Protocol: Dynamic Virtual Channel Extension is applicable to the creation of applications such as PNP device redirection and Media Infrastructure Layer Composition Engine commands.

A virtual channel application has two parts: a client-side component and a server-side component. The client-side component is a DLL that must be loaded into memory on the client computer when the Terminal Services client program runs. The server-side component is an executable program running on the Terminal Server.

The Remote Desktop Protocol: Dynamic Virtual Channel Extension operates only after the static virtual channel transport is fully established. If the static virtual channel transport is terminated, no other communication over the Remote Desktop Protocol: Dynamic Virtual Channel Extension occurs.

Dynamic virtual channels are created and maintained by dynamic virtual channel (DVC) managers. A DVC manager is running on both the Terminal Services server and the Terminal Services client. The DVC server manager is responsible for initializing the dynamic virtual channel environment and for creating individual dynamic virtual channels. The DVC client manager is responsible for creating and maintaining connections to the client-side DVC manager's applications.

14.2.6 Remote Desktop Protocol: GDI Acceleration Extension

The Remote Desktop Protocol: Graphics Device Interface (GDI) Acceleration Extension (as specified in [MS-RDPEGDI]) extends the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting (as specified in [MS-RDPBCGR]) by adding advanced drawing-order capabilities and compression techniques. The aim of Remote Desktop Protocol: GDI Acceleration Extension is to reduce the bandwidth associated with graphics remoting by encoding the drawing operations that produce an image instead of encoding the actual image.

For example, instead of sending the bitmap image of a filled rectangle from server to client, an order to render a rectangle at coordinate (X, Y) with a given width, height, and fill color is sent to the client. The client then executes the drawing order to produce the intended graphics result.

In addition to defining how to encode common drawing operations, the Remote Desktop Protocol: GDI Acceleration Extension also facilitates the use of caches to store drawing primitives such as bitmaps, color tables, and characters. The effective use of caching techniques helps to reduce wire traffic by ensuring that items used in multiple drawing operations are sent only once from server to client (retransmission of these items for use in conjunction with future drawing operations is not required once the item has been cached on the client).

14.2.7 Remote Desktop Protocol: Remote Programs Virtual Channel Extension

Remote Desktop Protocol: Remote Programs Virtual Channel, also known as RemoteApps or Remote Applications Integrated Locally (RAIL), is an RDP 6.0 feature that presents a remote application running remotely on a RAIL server as a local user application running on the RAIL client machine. RAIL extends the core RDP protocol to deliver this seamless Windows experience. Support for RAIL is optional in RDP, and it is negotiated as part of the capability negotiation process.

Microsoft Remote Applications are programs that are accessed remotely through Terminal Services and appear as if they are running on the end user's local computer. Instead of being presented to

the user in the desktop of the remote Terminal Server, the RemoteApp program is integrated with the client's desktop, running in its own resizable window with its own entry in the taskbar. The user can run RemoteApp programs side-by-side with local programs. If a user is running more than one RemoteApp program on the same Terminal Server, the RemoteApp programs share the same Terminal Services session.

The RAIL client, running on the user's local machine, creates one local window or notification icon for every window or notification icon running on the RAIL server. These local windows/icons, called RAIL windows/icons, exactly mimic the appearance of their corresponding remote windows/icons, which are created by RAs running on the RAIL server. All local user input to the RAIL windows/icons is captured by the RAIL client and redirected to the server. All display updates to the remote windows/icons on the RAIL server are captured by the server and redirected to the client.

RAIL relies on the core RDP protocol for basic connection establishment, connection security, local input redirection to server, and drawing-order updates from server to client (as specified in the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification [MS-RDPBCGR]). In addition, RAIL adds the following extensions to the RDP protocol:

- Extensions to send drawing orders from the server to the client that describe individual windows and notification icons. This enables the RAIL client to mimic the geometry of the drawing orders in RAIL windows/icons.
- Virtual channel messages from client to server that contain client information, system-parameters information, and RAIL-specific commands, such as remote program launch.
- Virtual channel updates from server to client containing responses to client messages, server system parameters information, or information regarding other RAIL-specific features such as local move/resize.

Certain classes of user input are not directly received by the RAIL window/icon as keyboard or mouse input. Examples include right-clicks on the window's taskbar icon; key combinations to minimize, maximize, or restore all windows; and all user interactions with notification icons. These interactions are posted to the RAIL window/icon as nonkeyboard or nonmouse messages; hence, they cannot be sent over the core RDP channel. The client sends these interactions to the server as RAIL Virtual Channel messages.

14.2.8 Remote Desktop Protocol: File System Virtual Channel Extension

The Remote Desktop Protocol: File System Virtual Channel Extension (as specified in [MS-RDPEFS]) provides access between the server and the client file system drivers, by redirecting all input/output requests and responses between the two. Those requests could be specific to the OS on both sides, so for this protocol that OS-specific payload is treated as opaque data.

In a typical terminal server scenario, much of the nonvolatile information used by the terminal server user (such as hard drives, flash drives and floppy disks) is located on the client. To provide access for applications running on the server to those client file systems, the server exposes a file system driver that is visible to the applications as a hard drive.

The Remote Desktop Protocol: File System Virtual Channel operates only after the static virtual channel transport is fully established. If the static virtual channel transport is terminated, no other communication over the Remote Desktop Protocol: File System Virtual Channel Extension occurs.

The Remote Desktop Protocol: File System Virtual Channel runs within the context of an RDP virtual channel established between a client and server. This protocol extension is applicable when applications running on the terminal server need to access the file system physically located on a client machine.

The Remote Desktop Protocol: File System Virtual Channel provides the virtual channel transport to redirect devices such as printers (as specified in [MS-RDPEPC]), serial/parallel ports (as specified in [MS-RDPESP]), and smart card devices (as specified in [MS-RDPESC]). Before these devices can be redirected to the Remote Desktop session, the static virtual channel provided by the Remote Desktop Protocol: File System Virtual Channel must be established.

14.2.9 Remote Desktop Protocol: Print Virtual Channel Extension

The Remote Desktop Protocol: Print Virtual Channel Extension (as specified in [MS-RDPEPC]) specifies the communication used to enable the redirection of printers between a terminal client and a terminal server. By redirecting printers from the terminal client to the terminal server, applications running on a server machine can access the remote devices as if they were local printers. This protocol handles redirection of printing tasks, while the Remote Desktop Protocol: XPS Print Virtual Channel Extension (as specified in [MS-RDPEXPS]) handles printer-driver related tasks, such as displaying the status of the redirected printer.

Essentially a subprotocol within Remote Desktop Protocol: File System Virtual Channel Extension, the Remote Desktop Protocol: Print Virtual Channel Extension follows the initialization of the Remote Desktop Protocol: File System Virtual Channel Extension (as specified in [MS-RDPEFS]) to enable printer redirection. It also shares many messages and common data types already specified in [MS-RDPEFS].

14.2.10 Remote Desktop Protocol: XPS Print Virtual Channel Extension

The Remote Desktop Protocol: XPS Print Virtual Channel Extension (as specified in [MS-RDPEXPS]) specifies communication between a virtual printer driver installed on a Terminal Server and a printer driver installed on the Terminal Server client. The primary purpose of this protocol is to acquire printing capabilities and to display a printer-specific user interface on the Terminal Server client. All other print tasks are handled by the Remote Desktop Protocol: Print Virtual Channel Extension (as specified in [MS-RDPEPC]).

This protocol extension consists of the following interfaces:

- Printer Ticket
- Printer Driver

The Printer Ticket interface is a group of messages used to specify how a particular document will be rendered in Extensible Markup Language Paper Specification (XPS) format. This interface also supplies a means for translating between older types of document properties.

The Printer Driver interface is a group of messages used to specify printer capabilities that are negotiated between the client and the server. The interface also assists in displaying a printer-specific user interface on the Terminal Server client.

In a typical Terminal Server scenario, the physical printer device is located on the Terminal Server client machine, while applications are running on the Terminal Server. Terminal Server clients often need to print documents using the printer that is physically connected to the client machine. The Remote Desktop Protocol: XPS Print Virtual Channel Extension allows these jobs to be redirected to the client printer. The platform-independent XPS document format specifies the content- and document-specific properties of the print jobs.

To maintain backward compatibility, and to extend the lifetime of the two interfaces, the Remote Desktop Protocol: XPS Print Virtual Channel Extension includes helper messages for interface manipulation; these messages are applicable to both interfaces.

14.2.11 Remote Desktop Protocol: Smart Card Virtual Channel Extension

Remote Desktop Protocol: Smart Card Virtual Channel extension (as specified in [MS-RDPESC]) is an asynchronous client/server protocol, which is designed to remotely execute requests on a client's Smart Cards for Windows.

The term smart card describes a class of credit-card-sized devices with varying capabilities: stored-value cards, contact-less cards, and integrated circuit cards (ICCs). The ICCs are of most interest to the computer industry, because these cards can perform more sophisticated operations, including signing and key exchange.

These cards all differ in functionality from one another and from the more familiar magnetic stripe cards used as standard credit, debit, and automated teller machine (ATM) cards. Smart Card Redirection is an asynchronous client/server protocol that is designed to remotely execute requests on a client's Smart Cards for Windows.

Each request executed using the Remote Desktop Protocol: Smart Card Virtual Channel extension is composed of two packets: a call packet and a return packet. The Protocol Server (Terminal Services server) sends a Call Packet after an initial announcement by the Protocol Client (Terminal Services client), and will receive a return packet after the request has been completed or an error has occurred.

RDP Device Redirection uses a static virtual channel as its transport. Smart Card Redirection redirects the Terminal Services client–side Smart Cards for Windows. When Smart Card Redirection is in effect, the Terminal Services server application smart card subsystem calls (for example, EstablishContext) are automatically remapped to the Terminal Services client–side Smart Cards for Windows, which will then receive the corresponding request. Smart Card Redirection devices are required only to understand one type of device input/output (I/O) request.

14.2.12 Remote Desktop Protocol: Plug and Play Devices Virtual Channel Extension

The Remote Desktop Protocol: Plug and Play Devices Virtual Channel extension (as specified in [MS-RDPEPNP]) is used to redirect PNP devices from a Terminal Server client to the Terminal Server. This action allows the server access to devices physically connected to the client as if the device were local to the server. For example, a user can attach a portable music device to the Terminal Services client and then synchronize music by using a media player application running on the Terminal Server server.

The Remote Desktop Protocol: Plug and Play Devices Virtual Channel extension consists of two subprotocols:

1. Plug and Play (PNP) Device Info. The PNP Device Info Subprotocol specifies the communication between the Terminal Server client and the Terminal Server component that handles the creation and removal of remote devices on the server side. This subprotocol is used to create remote device instances on the server machine that correspond to the physical devices on the client machine.

2. Plug and Play (PNP) Device I/O. For handling I/O requests, the PNP Device I/O subprotocol specifies the communication between the terminal client and the remote devices on the Terminal Server. This subprotocol is used to redirect the I/O calls from applications on the Terminal Server side to a device driver on the terminal client side. This subprotocol uses a dynamic virtual channel named FileRedirectorChannel for communication between client and server.

14.2.13 Remote Desktop Protocol: Clipboard Virtual Channel Extension

The Remote Desktop Protocol: Clipboard Virtual Channel extension (as specified in [MS-RDPECLIP]) allows users to seamlessly transfer data via the system clipboard between applications that are running on different computers. To accomplish this objective, the Remote Desktop Protocol: Clipboard Virtual Channel extension specifies how to keep two distinct system clipboards in sync, so that at any given time, the data available to an application on one computer's local clipboard is identical to the data available to another application on a remote computer's local clipboard.

All data copied to a system clipboard must conform to a format specification known as a Clipboard Format. Each Clipboard Format is identified by a unique numeric format ID. This format ID is used to tag the data on the clipboard so that any application that is enumerating the contents of the clipboard can determine the format of the data without having to extract and analyze it. The type of data that can be transferred by using Remote Desktop Protocol: Clipboard Virtual Channel extension is divided into four categories:

- Generic data
- Palette data
- Metafile data
- File stream data

Generic data is not manipulated or re-encoded by Remote Desktop Protocol: Clipboard Virtual Channel extension. Generic data is treated as opaque and passed from one virtual channel endpoint to another without any modification.

Palette data contains a predefined set of mappings from a given index to a red, green, and blue (RGB) triplet. Each triplet represents a color in the additive RGB color space. Palette data to be transferred between virtual channel endpoints is specially encoded and decoded for transport on the wire by Remote Desktop Protocol: Clipboard Virtual Channel extension.

A Windows metafile is a collection of structures that can store an image in an application-independent format. The stored image can be recreated by processing the metafile structures. Also called a vector image, a metafile contains a sequence of drawing commands and settings. The commands and settings recorded in a metafile object can be rendered on a display, output by a printer or plotter, stored in memory, or saved to a file or stream. Metafile data to be transferred between virtual channel endpoints is specially encoded and decoded for transport on the wire by the Remote Desktop Protocol: Clipboard Virtual Channel extension.

A file stream encapsulates the contents of a file that resides on some form of long-term storage. The Remote Desktop Protocol: Clipboard Virtual Channel extension provides the ability to transfer selected chunks of a file between virtual channel endpoints, as opposed to having to transfer the entire file image. A file stream can also be part of a larger collection of streams, where each stream can be referenced independently (as, for example, when transferring a group of files).

14.2.14 Remote Desktop Protocol: Licensing Extension

The Remote Desktop Protocol: Licensing Extension (as specified in [MS-RDPELE]) allows authorized Remote Desktop Clients or users to connect to a Terminal Server. This protocol involves communication between the Remote Desktop Client, the Terminal Server, and a License Server. The Terminal Server can be configured to be in per device or per user license mode. Client Access Licenses (CALs) are installed on a License Server, and when a Terminal Server requests a license on a client's behalf, the License Server issues a license out of its available pool of licenses.

The Remote Desktop Protocol: Licensing Extension involves the following components:

- Remote Desktop Client—Connects to a Terminal Server and provides the user interface through which a user interacts with a remote session.
- Terminal Server—Hosts remote desktop sessions.
- License Server—Issues licenses to users or devices using remote desktop sessions.
- Clearing House—Activates License Servers and provides Client Access Licenses.
- Active Directory—Stores licenses issued to users.
- License Manager: Administers License Servers.

14.2.15 Remote Desktop Protocol: Audio Output Virtual Channel Extension

Remote Desktop Protocol: Audio Output Virtual Channel Extension (as specified in [MS-RDPEA]) information may be exchanged between the client and server via two different transport methods:

- Static Virtual Channels, as specified in [MS-RDPBCGR]
- User Datagram Protocol (UDP)

Static Virtual Channels may be used to transmit all information between client and server and must be used for some sequences. For certain sequences, however, UDP may be used as well.

The Remote Desktop Protocol: Audio Output Virtual Channel Extension is divided into three distinct sequences:

- Initialization Sequence - The connection is established and capabilities and settings are exchanged.
- Data Transfer Sequences - The audio data is transferred to the client.
- Audio Setting Transfer Sequences - Changes to audio settings are transferred.

14.2.16 Remote Desktop Protocol: Multiparty Virtual Channel Extension

The Remote Desktop Protocol: Basic Connectivity and Graphics Remoting [MS-RDPBCGR] enables the remote display of desktop and application content. To effectively implement an application-sharing or collaborative solution, additional information must be conveyed to keep the participants apprised of who else is involved, in addition to which applications or windows are being shared. The Remote Desktop Protocol: Multiparty Virtual Channel Extension (as specified in [MS-RDPEMC]) defines a set of messages that are used to communicate the information between the participants and to signal the occurrence of significant events.

The Remote Desktop Protocol: Multiparty Virtual Channel Extension provides the following message types:

- Application and Window Filtering - A host may choose to share all application windows on a desktop or, instead, limit the sharing to a subset. The process of limiting the sharing to a subset is known as filtering. Application filtering is used when the host wants to share the current windows for a specific application in addition to any others subsequently created while the application is being shared. Although the term "application" is operating system specific, it generally denotes all windows created by a certain process as well as all windows related to the original windows by window hierarchy. Window filtering is purely explicit. A window is selected for

sharing, and any subsequent windows created by an application must be manually added to the sharing list. The precise mode of operation depends on a combination of user preference and the features of a sharing manager.

- Participant Management - Participant Management facilities allow the sharing manager to send notifications to all participants when an individual participant connects or disconnects from the sharing session or when a participant's control level changes.
- Graphics Stream Control - The host may choose to momentarily suspend or resume desktop sharing. This capability is useful when an event, such as the input of a plain-text password, would reveal sensitive information to all participants. Participants should be notified when sharing is suspended, so they know why they are no longer receiving information.

14.2.17 Remote Desktop Protocol: Session Selection Extension

The Remote Desktop Protocol: Session Selection Extension (as specified in [MS-RDPEPS]) specifies the messages exchanged between an RDP client and a terminal server to facilitate the precise targeting of an application sharing context. This extension allows for multiple processes to accept connections on the same TCP/IP port.

This protocol extension expands upon the original connectivity options specified in the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification to address a wide range of new scenarios where the Remote Desktop Protocol (RDP) is used to send the user experience of an application.

14.2.18 Remote Desktop Protocol: Serial Port Virtual Channel Extension

The Remote Desktop Protocol: Serial Port Virtual Channel Extension (as specified in [MS-RDPESP]) specifies the communication used to enable the redirection of ports between a terminal client and a terminal server. By redirecting ports from the terminal client to the terminal server, applications running on a server machine can access the remote devices attached to those ports.

This extension enables the redirection of serial and parallel port devices attached to the terminal client. With the redirection, such devices can then be accessed by the applications running on the server.

This extension can be considered as a subprotocol within the Remote Desktop Protocol: File System Virtual Channel Extension as specified in [MS-RDPEFS]. It follows the initialization of the Remote Desktop Protocol: File System Virtual Channel Extension to enable port redirection.

14.2.19 Terminal Services Gateway Server Protocol

The Terminal Services Gateway Server Protocol (as specified in [MS-TSGU]) is based on the Remote Procedure Call Over HTTP Protocol, as specified in [MS-RPCH].

The Terminal Services Gateway Server Protocol enables authorized remote users to connect to terminal servers and remote desktops (remote computers) on the corporate network from any Internet-connected device that is running Remote Desktop Connection (RDC) 6.0.

The Terminal Services Gateway Server Protocol uses RDP tunneled over Hypertext Transfer Protocol Secure (HTTPS) to help form a secure, encrypted connection between remote users on the Internet and the remote computers on which their productivity applications run, even if the remote users are located behind a network address translation (NAT) Traversal-based router.

The Terminal Services Gateway Server Protocol eliminates the need to configure legacy Virtual Private Network (VPN) connections, thus enabling remote users to connect to the corporate network

through the Internet. At the same time, this protocol provides a comprehensive security configuration model that enables the remote users to control access to specific resources on the network.

14.2.20 Terminal Services Terminal Server Runtime Interface (RPC)

The Terminal Services Terminal Server Runtime Interface Protocol Specification (as specified in [MS-TSTS]) is an RPC-based protocol used for remotely querying and configuring various aspects of a Terminal Server. For example, this protocol can be used to query the number of active sessions running on a Terminal Server.

The protocol consists of two major subcomponents:

1. Local Session Manager (LSM): a central component in the Windows operating system that creates, destroys, and manipulates sessions.

2. TermService: a specific implementation of connection manager, this is an extension for the Local Session Manager (LSM). The main purpose of TermService is to service incoming remote connection requests. The main protocols TermService currently supports are Microsoft's RDP (as specified in [MSDN-RDP]) and Citrix's ICA (as specified in [ICA]), but could be extended for others as well.

The protocol can be further divided into the following functional categories:

- LSM Session: These calls deal with collecting information, and controlling and configuring sessions that are running on the Terminal Server.
- LSM Notification: These RPC calls are asynchronous in nature and can be used to receive event notifications from LSM.
- LSM Enumeration: These calls are used to enumerate information related to sessions running on a Terminal Server.
- TermService: These calls can be used to query and configure various aspects of the TermServices running on the Terminal Server.
- TermService Listener: The TermService Listener calls are specific to the Listener session that is running on the Terminal Server and listening for incoming connection requests.
- Legacy: All the calls used by TS clients in the version of Windows prior to Windows Vista operating system can be grouped under this category.
- Terminal Server Licensing: These calls can be used to query information about the licensing mode used by the Terminal Server, and to configure it remotely.

The Terminal Services Terminal Server Runtime Interface is a simple request-response RPC-based protocol. For every method that the server receives, it executes the method and returns a completion. The client simply returns the completion status to the caller.

The Terminal Services Terminal Server Runtime Interface Protocol is only appropriate for querying and configuring a Terminal Server on a Windows operating system. Before this protocol is invoked, it is assumed that a Terminal Server Runtime Interface (RPC) Protocol client has obtained the name of a Terminal Server that supports the Terminal Server Runtime Interface (RPC) Protocol.

14.3 Terminal Services Protocols Logical Dependencies and Protocol Stack Views

This section describes the logical dependencies among the Terminal Services protocols and provides protocol stack views, as appropriate.

14.3.1 Integrated Terminal Services Protocols

The high-level view of the integrated Terminal Services core protocols is depicted in the following diagram.

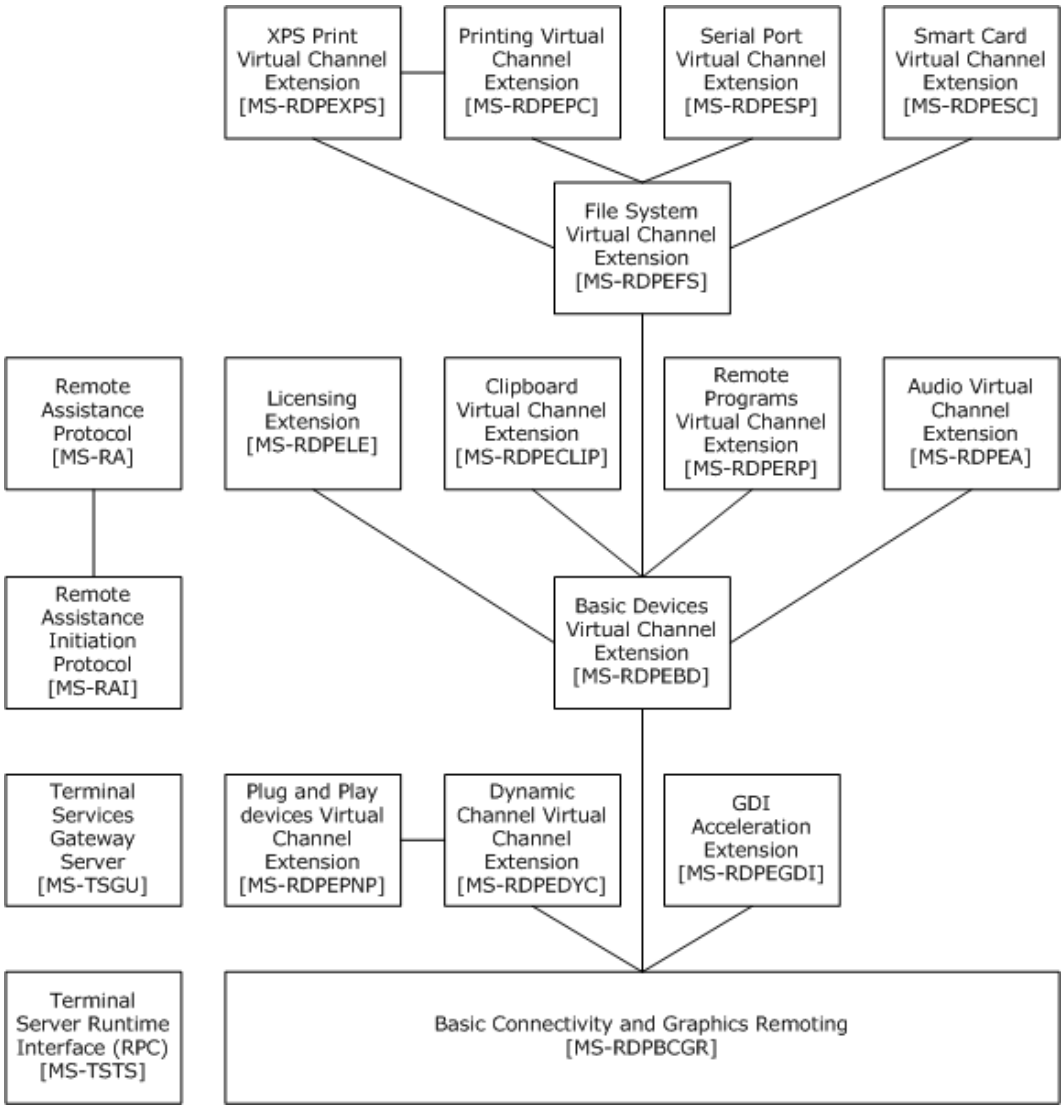

Figure 93: Terminal Services core protocols, logical view

This diagram is intended to show Terminal Services core protocols, except the Terminal Services Gateway Server Protocol, in a single logical view. In practice, these core protocols would not all be used at the same time; rather, they would be invoked as needed during an RDP session for functions such as printing, file transfer, graphics remoting, and clipboard transfers, to name a few.

The following sections describe the logical dependencies of each core protocol. A protocol may depend upon core protocols or more noncore protocols. Noncore protocols are referenced but not documented as a part of Terminal Services.

14.3.1.1 Remote Assistance Initiation Protocol Relationship to Other Protocols

The Remote Assistance Initiation Protocol (as specified in [MS-RAI]) relies on the OLE Automation Protocol (as specified in [MS-OAUT]), the Distributed Component Object Model (DCOM) Remote Protocol (as specified in [MS-DCOM]), and on Microsoft RPC, as specified in the Remote Procedure Call Protocol Extensions and [MS-RPCE].

The Remote Assistance Initiation Protocol (as specified in [MS-RAI]) is dependent upon RDP: Multiparty Virtual Channel Extension (as specified in [MS-RDPEMC]) for basic connectivity and graphics remoting, and for support for the virtual-channel extensions.

The Remote Assistance Protocol (as specified in [MS-RA]) is dependent on the Remote Assistance Initiation Protocol (as specified in [MS-RAI]).

14.3.1.2 Stack View

The relationships between this and other protocols are best represented in terms of logical dependencies. Therefore, stack views are not provided.

14.3.1.3 Remote Assistance Protocol Relationship to Other Protocols

The Remote Assistance Protocol assumes that an RA connection between two computers has been established by using the Remote Assistance Initiation Protocol (as specified in [MS-RAI]) and the Remote Desktop Protocol: Multiparty Virtual Channel Extension (as specified in [MS-RDPEMC]).

The Remote Assistance Protocol assumes that an underlying protocol, specifically the Remote Desktop Protocol: Multiparty Virtual Channel Extension (as specified in [MS-RDPEMC]), will be available to transport the protocol messages.

No other protocol is dependent on the Remote Assistance Protocol.

14.3.1.4 Stack View

The relationships between this and other protocols are best represented in terms of logical dependencies. Therefore, stack views are not provided.

14.3.1.5 Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Relationship to Other Protocols

The Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification is based on the ITU T.120 series of protocols. The T.120 standard is composed of a suite of communication and application-layer protocols that enable developers to create compatible products and services for real-time, multipoint data connections and conferencing.

14.3.1.6 Stack View

The following diagram shows the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting protocol stack.

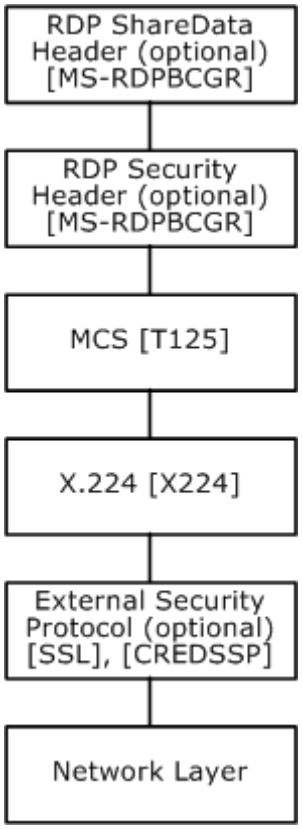

Figure 94: Remote Desktop Protocol: Basic Connectivity and Graphics Remoting protocol stack

14.3.1.7 Remote Desktop Protocol: Dynamic Channel Virtual Channel Relationship to Other Protocols

The Remote Desktop Protocol: Dynamic Channel Virtual Channel is embedded in a static virtual channel transport, as specified in [MS-RDPBCGR]. This protocol operates only after the static virtual channel transport (as specified in [MS-RDPBCGR]) is fully established. If the static virtual channel transport is terminated, no other communication over the Remote Desktop Protocol: Dynamic Channel Virtual Channel occurs.

14.3.1.8 Stack View

The following diagram shows the Remote Desktop Protocol: Dynamic Channel Virtual Channel protocol stack.

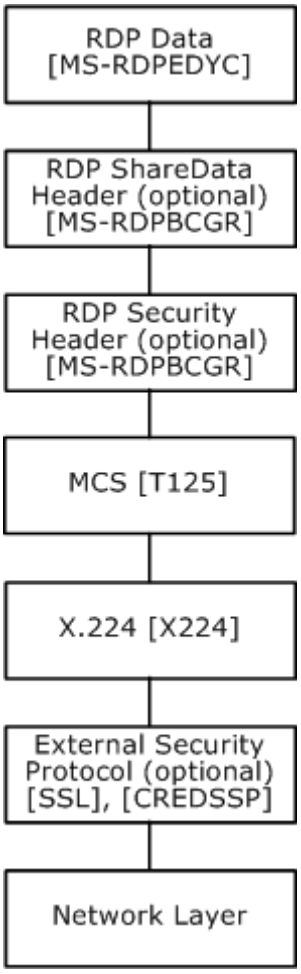

Figure 95: Remote Desktop Protocol: Dynamic Channel Virtual Channel protocol stack

14.3.1.9 Remote Desktop Protocol: GDI Acceleration Extension Logical Dependencies

The Remote Desktop Protocol: Graphics Devices Interfaces (GDI) Acceleration Extension packets are an extension to Remote Desktop Protocol: Basic Connectivity and Graphics Remoting (as specified in [MS-RDPBCGR]), and they rely on the same transport. The packets are encapsulated in TCP.

The Remote Desktop Protocol: GDI Acceleration Extensions GDI cache relies on data types specified in the Enhanced Metafile (EMF) Format: Plus Extensions Specification (see [MS-EMFPLUS]).

14.3.1.10 Stack View

The following diagram shows the Remote Desktop Protocol: Graphics Devices Interfaces (GDI) Acceleration Extension protocol stack.

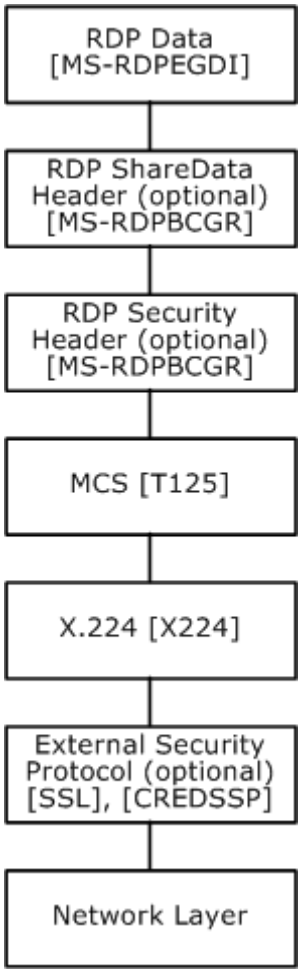

Figure 96: Remote Desktop Protocol: GDI Acceleration Extension protocol stack

14.3.1.11 Remote Desktop Protocol: Remote Programs Virtual Channel Logical Dependencies

The Remote Desktop Protocol: Remote Programs Virtual Channel is assumed to operate in a fully operational RDP connection. A fully operational RDP connection is a connection that has passed the Connection Finalization phase, as specified in [MS-RDPBCGR].

14.3.1.12 Stack View

The following diagram shows the Remote Desktop Protocol: Remote Programs Virtual Channel Extension.

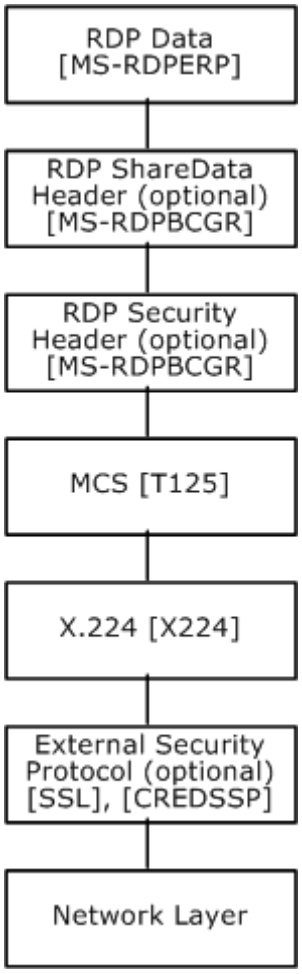

Figure 97: Remote Desktop Protocol: Remote Programs Virtual Channel protocol stack

14.3.1.13 Remote Desktop Protocol: File System Virtual Channel Logical Dependencies

The Remote Desktop Protocol: File System Virtual Channel is embedded in a static virtual channel transport, as specified in [MS-RDPBCGR]. Some of the packets described in this protocol are used by the protocols for printers (as specified in [MS-RDPEPC]), ports (as specified in [MS-RDPESP]), and smart cards (as specified in [MS-RDPESC]).

14.3.1.14 Stack View

The following diagram shows the Remote Desktop Protocol: File System Virtual Channel protocol stack.

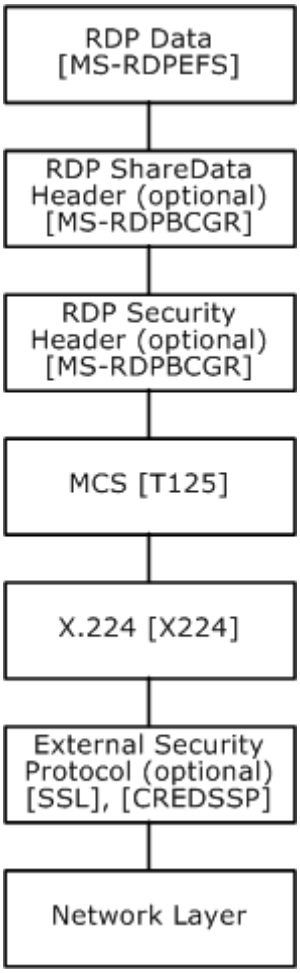

Figure 98: Remote Desktop Protocol: File System Virtual Channel protocol stack

14.3.1.15 Remote Desktop Protocol: Printing Virtual Channel Logical Dependencies

The Remote Desktop Protocol: Print Virtual Channel Extension can be considered as a subprotocol within Remote Desktop Protocol: File System Virtual Channel Extension (as specified in [MS-RDPEFS]). This protocol extends the Remote Desktop Protocol: File System Virtual Channel Extension to enable printer redirection.

This protocol operates only after the Remote Desktop Protocol: File System Virtual Channel Extension transport is fully established.

14.3.1.16 Stack View

The following diagram shows the Remote Desktop Protocol: Print Virtual Channel protocol stack.

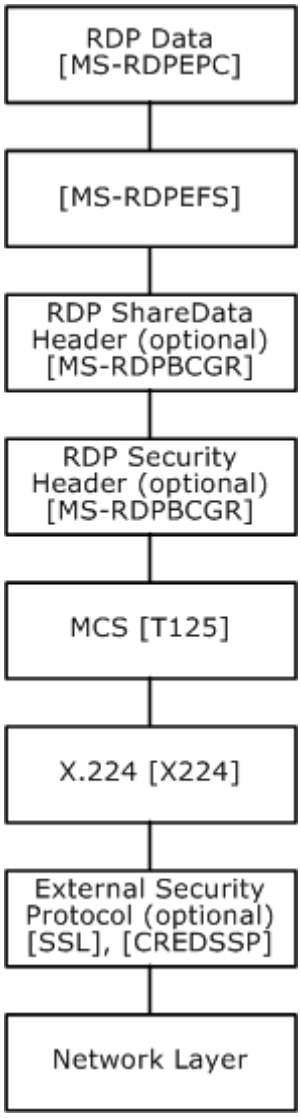

Figure 99: Remote Desktop Protocol: Print Virtual Channel protocol stack

14.3.1.17 Remote Desktop Protocol: XPS Printing Virtual Channel Logical Dependencies

The Remote Desktop Protocol: XPS Print Virtual Channel is embedded in dynamic virtual channel transport, as specified in [MS-RDPEDYC].

This protocol extension works in conjunction with the Remote Desktop Protocol: File System Virtual Channel Extension (as specified in [MS-RDPEFS]), which creates printer queues for redirected printers, and also carries the printer output from print jobs initiated by the server applications. The Remote Desktop Protocol: XPS Print Virtual Channel Extension helps redirect user-interface elements and allows users to set printing properties for the printer queues that already have been created by the Remote Desktop Protocol: File System Virtual Channel Extension.

14.3.1.18 Stack View

The following diagram shows the Remote Desktop Protocol: XML Paper Specification (XPS) Print Virtual Channel protocol stack.

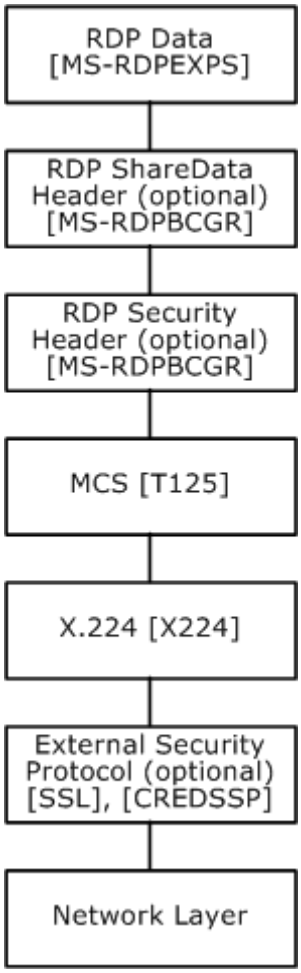

Figure 100: Remote Desktop Protocol: XPS Print Virtual Channel protocol stack

14.3.1.19 Remote Desktop Protocol: Smart Card Virtual Channel Logical Dependencies

The Remote Desktop Protocol: Smart Card Virtual Channel extension expands the functionality for Smart Cards for Windows API. RDP Device Redirection transport, as specified in [MS-RDPEFS], must be configured to redirect smart card devices. The Remote Desktop Protocol: Smart Card Virtual Channel extensions uses the Remote Desktop Protocol: File System Virtual Channel Extension as specified in [MS-RDPEFS], to transport packets between client and server.

14.3.1.20 Stack View

The following diagram shows the Remote Desktop Protocol: Smart Card Virtual Channel protocol stack.

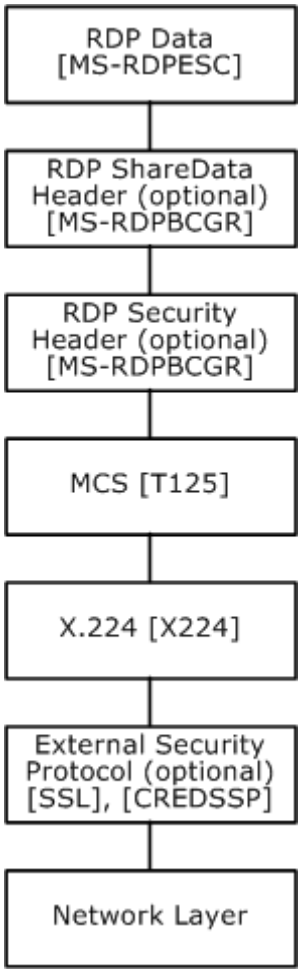

Figure 101: Remote Desktop Protocol: Smart Card Virtual Channel protocol stack

14.3.1.21 Remote Desktop Protocol: Plug and Play Virtual Channel Logical Dependencies

The Remote Desktop Protocol: Plug and Play Devices Virtual Channel is embedded in a dynamic virtual channel transport as specified in [MS-RDPEDYC].

The Remote Desktop Protocol: Plug and Play Devices Virtual Channel operates only after the dynamic virtual channel transport is fully established. If the dynamic virtual channel transport is terminated, the Remote Desktop Protocol: Plug and Play Devices Virtual Channel Extension is also terminated.

14.3.1.22 Stack View

The following diagram shows the Remote Desktop Protocol: Plug and Play Devices Virtual Channel protocol stack.

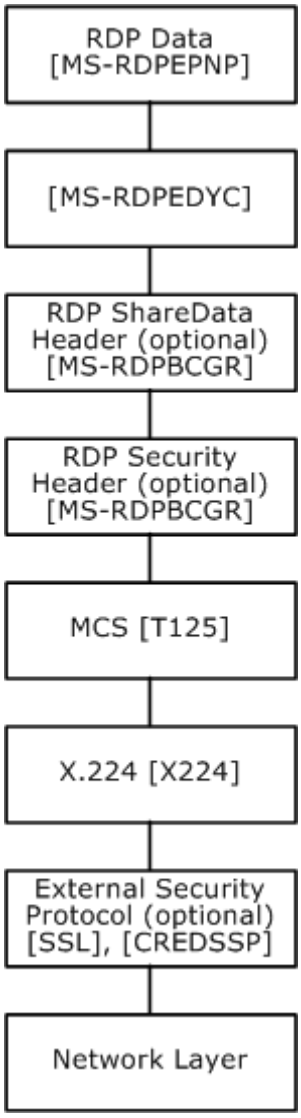

Figure 102: Remote Desktop Protocol: Plug and Play Devices Virtual Channel protocol stack

14.3.1.23 Remote Desktop Protocol: Clipboard Virtual Channel Logical Dependencies

The Remote Desktop Protocol: Clipboard Virtual Channel is embedded in a static virtual channel transport as specified in [MS-RDPBCGR].

The Remote Desktop Protocol: Clipboard Virtual Channel operates only after the static virtual channel transport (as specified in [MS-RDPBCGR]) is fully established. If the static virtual channel transport is terminated, no other communication over the Remote Desktop Protocol: Clipboard Virtual Channel occurs.

14.3.1.24 Stack View

The following diagram shows the Remote Desktop Protocol: Clipboard Virtual Channel.

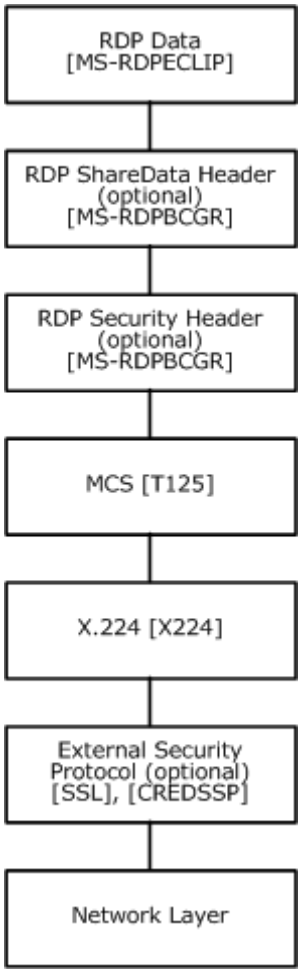

Figure 103: Remote Desktop Protocol: Clipboard Virtual Channel protocol stack

14.3.1.25 Remote Desktop Protocol: Licensing Extension Protocol Logical Dependencies

The Remote Desktop Protocol: Licensing Extension is an extension of, and adds licensing capabilities to, Remote Desktop Protocol: Basic Connectivity and Graphics Remoting, as specified in [MS-RDPBCGR].

The Remote Desktop Protocol: Licensing Extension assumes that the system already has an IP address and is thus able to communicate on the network. It also assumes that the initiator (or client) has already obtained the IP address of the server, that the server has registered a port, and that the server is actively listening for client connections on that port.

The Remote Desktop Protocol: Licensing Extension is a stateless protocol. No sequence of method calls is imposed on this protocol. When a method completes, the values returned by RPC MUST be returned unmodified to the upper layer.

All methods implemented by the TS server should enforce appropriate security measures to make sure that the TS client has required permissions to execute the method. All methods must be RPC asynchronous calls.

14.3.1.26 Stack View

The relationships between this and other protocols are best represented in terms of logical dependencies. Therefore, stack views are not provided.

14.3.1.27 Remote Desktop Protocol: Audio Output Virtual Channel Extension Relationship to Other Protocols

The Remote Desktop Protocol: Audio Output Virtual Channel Extension is embedded in a static virtual channel transport, as specified in [MS-RDPBCGR].

The Remote Desktop Protocol: Audio Output Virtual Channel Extension operates only after the static virtual channel transport (as specified in [MS-RDPBCGR]) is fully established. If the static virtual channel transport is terminated, no other communication occurs over the Remote Desktop Protocol: Audio Output Virtual Channel Extension.

14.3.1.28 Stack View

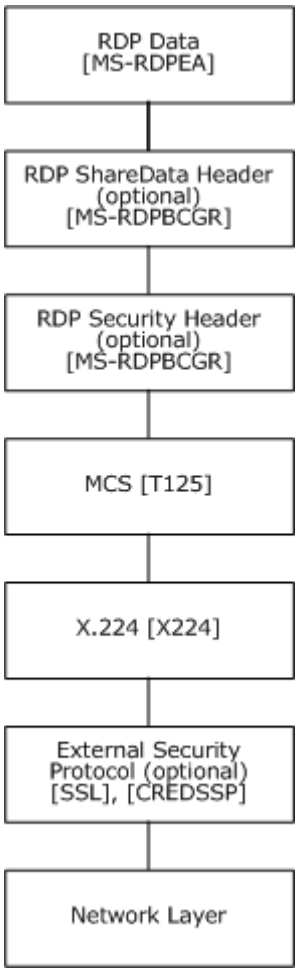

Figure 104: Remote Desktop Protocol: Audio Output Virtual Channel Extension stack

14.3.1.29 Remote Desktop Protocol: Multiparty Virtual Channel Extension Relationship to Other Protocols

The Remote Desktop Protocol: Multiparty Virtual Channel Extension is embedded in static virtual channel transport, as specified in [MS-RDPBCGR].

The Remote Desktop Protocol: Multiparty Virtual Channel Extension operates only after a static virtual channel transport, as specified in [MS-RDPBCGR], with the name "encomsp" is fully established. If the static virtual channel transport is terminated, no other communication over the Remote Desktop Protocol: Multiparty Virtual Channel Extension occurs.

14.3.1.30 Stack View

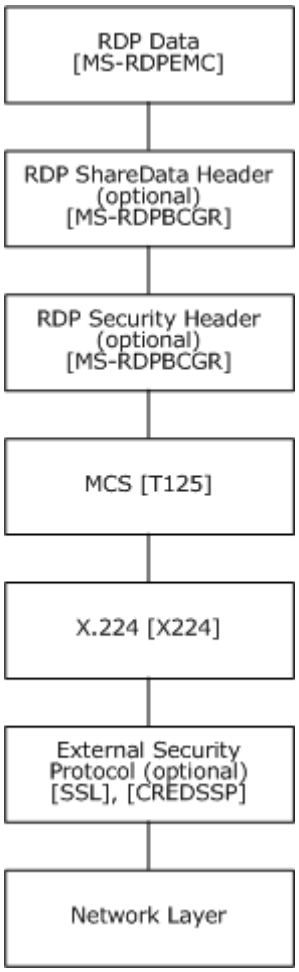

Figure 105: Remote Desktop Protocol: Multiparty Virtual Channel Extension stack

14.3.1.31 Remote Desktop Protocol: Session Selection Extension Relationship to Other Protocols

This protocol extension is a minor enhancement to the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification.

The client must have all of the information necessary to initialize the Server Preconnection PDU with values that are appropriate to the scenario.

14.3.1.32 Stack View

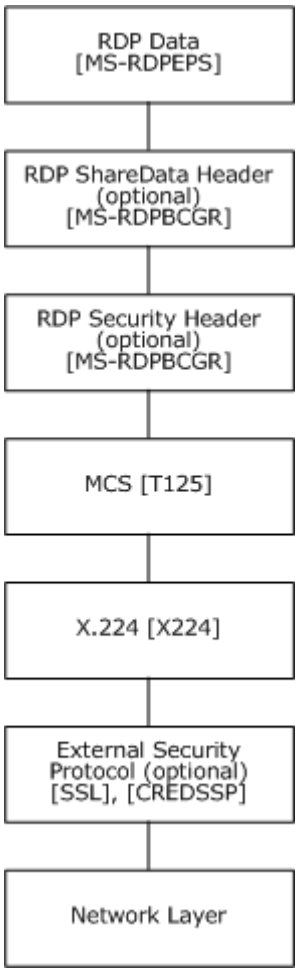

Figure 106: Remote Desktop Protocol: Session Selection Extension stack

14.3.1.33 Remote Desktop Protocol: Serial Port Virtual Channel Extension Relationship to Other Protocols

This extension can be considered as a subprotocol within Remote Desktop Protocol: File System Virtual Channel Extension as specified in [MS-RDPEFS]. This extension extends the Remote Desktop Protocol: File System Virtual Channel Extension to enable port redirection.

The Remote Desktop Protocol: Serial Port Virtual Channel Extension operates only after the Remote Desktop Protocol: File System Virtual Channel Extension transport, as specified in [MS-RDPEFS], is fully established.

14.3.1.34 Stack View

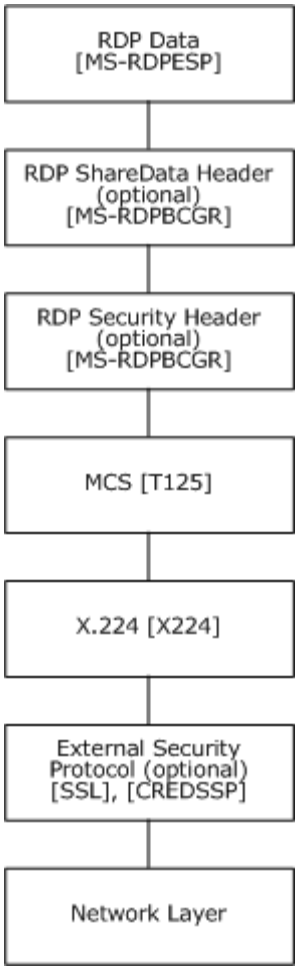

Figure 107: Remote Desktop Protocol: Serial Port Virtual Channel Extension stack

14.3.1.35 Terminal Services Terminal Server Runtime Interface (RPC) Relationship to Other Protocols

The Terminal Services Terminal Server Runtime Interface Protocol is dependent upon RPC for its transport. This protocol uses RPC over named pipes as specified in Remote Procedure Call Protocol Extensions [MS-RPCE] and Remote Procedure Call (RPC) over HTTP Protocol [MS-RPCH].

Relationships between Terminal Services Terminal Server Runtime Interface Protocol and other protocols are best represented in terms of logical dependencies. Therefore, stack views are not provided.

Terminal Services protocols can be implemented to remote a wide variety of user scenarios. The following section traces two such scenarios.

14.4 Implementation Scenarios

This section provides a scenario that illustrates common functionality that can be implemented with the protocols included in the Terminal Services. The protocols involved in this scenarios include Remote Desktop Protocol, together with other general networking protocols used in the course of standard TCP communication between the client and server. The functionality illustrated in the scenarios is:

- A Windows Vista operating system Client connecting remotely to a Windows Server 2003 operating system in Remote Administration mode.

It should be noted that the following scenarios depict processes that occur over TCP and thus assumes various TCP acknowledgment frames throughout the conversation that are not necessarily depicted in the trace details.

14.4.1 Scenario 1 - Windows Vista Client Connects Remotely to a Windows 2003 Server in Remote Administration Mode

14.4.1.1 Scenario Overview

In the following scenario, a manual logon procedure is executed by an Administrator account to initiate a terminal logon session in Remote Administration mode. The remote user then creates a text file in Notepad application, and then saves the file to the server's desktop. The user then disconnects the session.

14.4.1.2 Scenario Configuration

This scenario was implemented with a Windows Server 2003 operating system server, and a Windows Vista operating system client machine configured as follows:

Server Configuration

- Machine: Windows Server 2003 (WS200302)
- Operating system: Windows Server 2003 R2 operating system with latest patches
- IP Address: 10.0.10.1
- Subnet Mask: 255.255.255.0
- Terminal Services role

Client Configuration

- Machine: Windows Vista (ClientVista)
- Operating system: Windows Vista with latest patches
- IP Address: 10.0.10.10
- Subnet Mask: 255.255.255.0

Network Topology

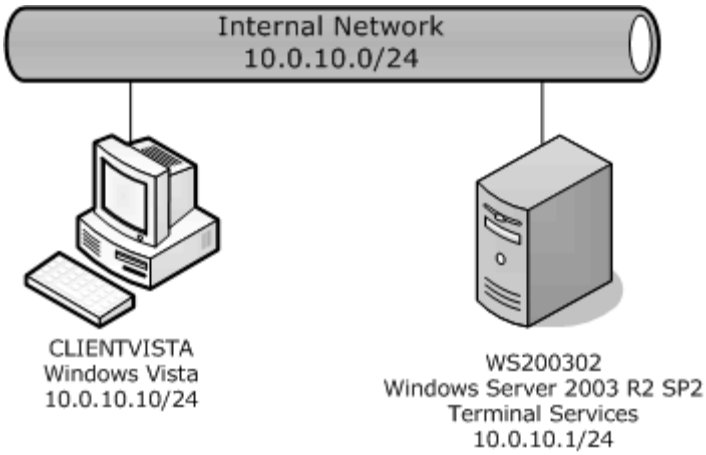

Figure 108: Network topology

14.4.1.3 Setting Up the Trace

CLIENTVISTA

1. Start NetMon 3.1.
2. BEGIN TRACE.
    1. Connect to WS200302 via RDP.
    2. Open Notepad.
    3. Type text.
    4. Save Notepad document.
    5. Log off from remote session.
3. END TRACE.

14.4.1.4 Netmon Trace Digest

- The session begins with client and server exchanging Address Resolution Protocol (ARP) requests and responses. CLIENTVISTA requests a MAC address for the server first, the server responds, and the process is repeated when the server issues an ARP request to the client:

```
WS200302 CLIENTVISTA ARP ARP: Response, 10.0.10.1 at
00-0F-1F-B2-DA-8F
CLIENTVISTA WS200302 TCP TCP: Flags=.S......, SrcPort=49174,
DstPort=HTTP(80),
Len=0, Seq=2753686059, Ack=0, Win=8192 (scale factor 2) = 32768
WS200302 CLIENTVISTA ARP ARP: Request, 192.168.54.78 asks for
192.168.54.72
CLIENTVISTA WS200302 ARP ARP: Response, 192.168.54.72 at
00-1A-92-29-C9-71
```

Connection Initiation: The client initiates the connection by sending the server an X.224 Connection Request PDU (class 0). The server responds with an X.224 Connection Confirm PDU (class 0). From this point, all subsequent data sent between client and server is wrapped in an X.224 Data PDU.

- Basic settings are exchanged between the client and server by using the MCS Connect Initial and MCS Connect Response PDUs. The Connect Initial PDU contains a GCC Conference Create Request, while the Connect Response PDU contains a GCC Conference Create Response. These two Generic Conference Control (GCC) packets contain concatenated blocks of settings data (such as core data, security data, and network data), which are read by client and server.

```
CLIENTVISTA WS200302 X224 X224: Connection Request

WS200302 CLIENTVISTA X224 X224: Connection Confirm
```

- The client sends an MCS Erect Domain Request PDU, followed by an MCS Attach User Request PDU to attach the primary user identity to the MCS domain. The server responds with an MCS Attach User Response PDU containing the user channel ID. The client then proceeds to join the user channel, the input/output (I/O) channel, and all of the static virtual channels (the I/O and static virtual channel IDs are obtained from the data embedded in the GCC packets) by using multiple MCS Channel Join Request PDUs. The server confirms each channel with an MCS Channel Join Confirm PDU. (The client only sends a Channel Join Request after it has received the Channel Join Confirm for the previously sent request.)

```
CLIENTVISTA WS200302 T125 T125: MCSConnect Initial

WS200302 CLIENTVISTA T125 T125: MCSConnect Response

CLIENTVISTA WS200302 T125 T125: Erect Domain Request, SubHeight = 0, SubInterval = 0

CLIENTVISTA WS200302 T125 T125: Attach User Request

WS200302 CLIENTVISTA T125 T125: Attach User Confirm, Result = rt-successful,
Indicator = 1008

CLIENTVISTA WS200302 T125 T125: Channel Join Request, UserID = 1008,ChannelId = 1008

WS200302 CLIENTVISTA T125 T125: Channel Join Confirm, ChannelId = 1008,
Result = rt-successful
```

- From this point, all subsequent data sent from the client to the server is wrapped in an MCS Send Data Request PDU, while data sent from the server to the client is wrapped in an MCS Send Data Indication PDU. This is in addition to the data being wrapped by an X.224 Data PDU.

```
CLIENTVISTA WS200302 RDP RDP: Encrypted MCS Channel Packet, Length = 523
```

- If standard RDP security methods are being employed and encryption is in force (this is determined by examining the data embedded in the GCC Conference Create Response packet), then the client sends a Security Exchange PDU that contains an encrypted 32-byte random number to the server. This random number is encrypted with the public key of the server (the server's public key, as well as a 32-byte server-generated random number, are both obtained

from the data embedded in the GCC Conference Create Response packet). The client and server then utilize the two 32-byte random numbers to generate session keys, which are used to encrypt and validate the integrity of subsequent RDP traffic.

```
CLIENTVISTA WS200302 RDP RDP: TsClientSecurityExchangePDU, Length = 80,
SecurityExchange LicenseEncryption

CLIENTVISTA WS200302 RDP RDP: TsClientInfoPDU, Length = 374
```

- From this point, all subsequent RDP traffic can be encrypted, and a security header is included with the data if encryption is in force (the Client Info and licensing PDUs are an exception in that they always have a security header). The Security Header follows the X.224 and MCS Headers and indicates whether the attached data is encrypted. Even if encryption is in force, server-to-client traffic may not always be encrypted, while client-to-server traffic will always be encrypted, by Microsoft RDP implementations. Encryption of licensing PDUs is optional, however.

  Secure client data (such as the username, password, and auto-reconnect cookie) is sent to the server using the Client Info PDU.

- The goal of the licensing exchange is to transfer a license from the server to the client. The client should store this license, and on subsequent connections, send the license to the server for validation. However, in some situations, the client may not be issued a license to store. In effect, the packets exchanged during this phase of the protocol depend on the licensing mechanisms employed by the server.

```
WS200302 CLIENTVISTA RDP RDP: TsServerLicenseRequest, Length = 138

CLIENTVISTA WS200302 RDP RDP: Encrypted License Packet, Length = 166

WS200302 CLIENTVISTA RDP RDP: TsServerPlatformChallenge, Length = 42
```

- The server sends the set of capabilities it supports to the client in a Demand Active PDU. The client responds with its capabilities by sending a Confirm Active PDU.

```
WS200302 CLIENTVISTA RDP RDP: DemandActivePDU, Length = 363, Descriptor = RDP
```

- The client and server send PDUs to finalize the connection details. The client-to-server and server-to-client PDUs exchanged during this phase may be sent concurrently, as long as the sequencing in either direction is maintained (there are no cross-dependencies between any of the client-to-server and server-to-client PDUs). After the client receives the Font Map PDU, it can start sending mouse and keyboard input to the server; upon receipt of the Font List PDU, the server can start sending graphics output to the client.
- In addition to input and graphics data, other data that can be exchanged between client and server after the connection has been finalized includes connection management information and virtual channel messages (exchanged between client-side plug-ins and server-side applications).

15 Virtual Private Network (VPN) Services Protocols

This section provides an overview of the Virtual Private Network (VPN) Services protocols. It provides a high-level conceptual overview of the core VPN protocols.

15.1 VPN Services Overview

The VPN protocols provide functionality for communicating between a VPN server and a VPN client, including the following data communication:

- Authentication, policy, authorization, and connection data packets as part of a private network connection.
- Secure private network data packets encoded for IPSec network address translator traversal.

15.1.1 Protocols List

The VPN Services include a set of core protocols that provides VPN functionality. These protocols are a subset of the networking protocols group described in the Introduction section of this document.

The following table lists the core protocols included in VPN services. This list does not include data structures, file structures, or algorithms that these core protocols depend on. The details for these technical dependencies are documented in the technical specifications for each core protocol.

Some non-core protocols are also listed in the following table. These protocols are industry-standard protocols that have a logical or physical relationship to one or more of the core protocols. They may be mentioned in the text but are not fully documented.

Table 1:VPN Services Core Protocols

Protocol name	Protocol description	Document short name
Secure Socket Tunneling Protocol (SSTP)	Provides a mechanism for tunneling IP packets using PPP framing on top of SSL. This protocol can be used as a VPN tunneling protocol for traversing firewalls.	[MS-SSTP]
Layer 2 Tunneling Protocol (L2TP) with IPsec	Describes the Layer 2 Tunneling Protocol (L2TP) and the utilization of IPSec for tunnel authentication, privacy protection, integrity checking, and replay protection. L2TP facilitates tunneling of PPP packets across an intervening network in a way that is as transparent as possible to both end-users and applications.	[RFC2661] [RFC3193]
Point-to-Point Tunneling Protocol (PPTP)	Specifies a method that allows Point to Point Protocol (PPP) to be tunneled through an IP network. PPTP does not specify any changes to PPP but rather describes a new vehicle for carrying PPP.	[RFC2637]

15.2 VPN Services Functionality

This section provides a general overview of how the core protocols operate as a group to provide VPN services, followed by a detailed view of each of the protocol.

The Implementation Scenarios section, later in this document, provides examples that illustrate the interaction of some of the protocols in two sample configurations.

15.2.1 VPN Services Protocols General Functionality

A VPN is an extension of a private network that encompasses links across shared or public networks like the Internet. A VPN enables the transmission of data between two endpoints across a shared or public internetwork in a manner that emulates the properties of a point-to-point private link.

To emulate a point-to-point link, data is encapsulated, or wrapped, with a header that provides routing information, allowing it to traverse the shared or public transit internetwork to reach its endpoint. To emulate a private link, the data being sent is encrypted for confidentiality. Packets that are intercepted on the shared or public network are protected from being deciphered by unauthorized parties who do not have access to the encryption keys. The tunnel is the portion of the connection in which the private data is encapsulated. The VPN connection is the portion of the connection in which the private data is encrypted.

For purposes of this task, VPN solutions would be implemented in one of the following three ways:

- **Point-to-Point Tunneling Protocol (PPTP)**—allows Internet Protocol (IP), Internet Packet eXchange (IPX), or Network Basic Input/Output System (NetBIOS) extended user interface (NetBEUI) traffic to be encrypted, and then encapsulated in an IP header to be sent across a corporate IP internetwork or a public IP internetwork such as the Internet.
- Layer Two Tunneling Protocol (L2TP)—allows IP, IPX, or NetBEUI traffic to be encrypted, and then sent over any medium that supports point-to-point datagram delivery, such as IP, X.25, Frame Relay, or Asynchronous Transfer Mode (ATM).
- Secure Sockets Tunneling Protocol (SSTP)—uses an Hypertext Transfer Protocol-over-Secure-Sockets-Layer (HTTP-over-SSL) session between VPN clients and servers to exchange encapsulated IPv4 or IPv6 packets.

15.2.1.1 VPN Services Protocol Specifics

The following section provides an overview of the core protocols used to implement VPN solutions under VPN Services.

15.2.1.1.1 PPTP

The PPTP is a Layer 2 protocol that encapsulates PPP frames in IP datagrams for transmission over an IP internetwork, such as the Internet. Microsoft's PPTP is an extension of the industry standard PPP and is used to encapsulate PPP frames within IP datagrams. PPTP can be used for remote access and router-to-router VPN connections.

PPTP uses a Transmission Control Protocol (TCP) connection for tunnel maintenance and a modified version of Generic Routing Encapsulation (GRE) to encapsulate PPP frames for tunneled data. The payloads of the encapsulated PPP frames can be encrypted and/or compressed.

15.2.1.1.2 L2TP/IPSec

L2TP/IPsec is a combination of the PPTP, a technology developed by Microsoft and other companies in the PPTP forum, and Layer 2 Forwarding (L2F), a technology proposed by Cisco Systems, Inc. Rather than maintaining two incompatible tunneling protocols that compete in the marketplace, the Internet Engineering Task Force (IETF) mandated that the two technologies be combined into a single tunneling protocol that represents the best features of PPTP and L2F. L2TP is documented in [RFC2661].

The combination of L2TP and IPsec, known as L2TP/IPsec, is an alternative to PPTP. L2TP/IPsec connections require stronger authentication by requiring two levels of authentication: a computer-

level authentication using certificates or pre-shared keys for the IPsec session and a user-level authentication using a PPP authentication protocol for the L2TP tunnel.

IPsec has been defined in a series of Requests for Comments (RFCs), notably [RFC2401], [RFC2402], and [RFC2406], which define the overall architecture, an authentication header to verify data integrity, and an encapsulation security payload for both data integrity and data encryption. IPsec was designed by the IETF as an end-to-end mechanism to ensure data security in IP-based communications.

IPsec defines two functions that ensure confidentiality: data encryption and data integrity. As defined by the IETF, IPsec uses an authentication header (AH) to provide source authentication and integrity without encryption, and the Encapsulating Security Payload (ESP) to provide authentication and integrity along with encryption. With IPsec, only the sender and recipient know the security key. If the authentication data is valid, the recipient knows that the communication came from the sender and that it was not changed in transit.

IPsec can be envisioned as a layer below the TCP/IP stack. This layer is controlled by a security policy on each computer and a negotiated security association between the sender and receiver. The policy consists of a set of filters and associated security behaviors. If a packet's IP address, protocol, and port number match a filter, the packet is subject to the associated security behavior.

The result of using PPTP or L2TP/IPsec is the creation of a virtual PPP connection between the VPN client and server. In short, the VPN connection behaves as if it was a dedicated point-to-point serial link, but packets are actually routed across the Internet.

15.2.1.1.3 PPTP Compared to L2TP/IPSec

Both PPTP and L2TP/IPsec use PPP to provide an initial envelope for the data, and then they append additional headers for transport through the internetwork. However, there are the following differences:

- With PPTP, data encryption begins after the PPP connection process (and, therefore, PPP authentication) is completed. With L2TP/IPsec, data encryption begins before the PPP connection process by negotiating an IPsec security association.
- PPTP connections use MPPE, a stream cipher that is based on the Rivest-Shamir-Aldeman (RSA) RC-4 encryption algorithm and uses 40-, 56-, or 128-bit encryption keys. Stream ciphers encrypt data as a bit stream. L2TP/IPsec connections use the Data Encryption Standard (DES), which is a block cipher that uses either a 56-bit key for DES or three 56-bit keys for 3-DES. Block ciphers encrypt data in discrete blocks (64-bit blocks, in the case of DES).
- PPTP connections require only user-level authentication through a PPP-based authentication protocol. L2TP/IPsec connections require the same user-level authentication and, in addition, computer-level authentication using computer certificates.

15.2.1.1.4 IPsec

IETF designed IPsec as an end-to-end mechanism to ensure data security in IP-based communications. IPsec has been defined in a series of RFCs, notably [RFC2401], [RFC2402], and [RFC2406], which define the overall architecture, an authentication header to verify data integrity, and an encapsulation security payload for both data integrity and data encryption.

IPsec defines two functions that ensure confidentiality: data encryption and data integrity. As defined by the IETF, IPsec uses an AH to provide source authentication and integrity without encryption, and the ESP to provide authentication and integrity along with encryption. With IPsec,

only the sender and recipient know the security key. If the authentication data is valid, the recipient knows that the communication came from the sender and that it was not changed in transit.

IPsec can be envisioned as a layer below the TCP/IP stack. This layer is controlled by a security policy on each computer and a negotiated security association between the sender and receiver. The policy consists of a set of filters and associated security behaviors. If a packet's IP address, protocol, and port number match a filter, the packet is subject to the associated security behavior.

15.2.1.1.5 Phases of L2TP Connection

For an L2TP/IPsec connection attempt to be successful, it must successfully establish the following:

1. IPsec security associations (SAs) for L2TP traffic
2. An L2TP connection
3. A PPP connection

IPsec SA negotiation for L2TP traffic consists of:

1. Main-mode negotiation—the creation of a main-mode SA by exchanging encryption key derivation information, determining the security of future main-mode packets, and authenticating the IPsec peers.
2. Quick-mode negotiation—the creation of a quick-mode SA by determining the security of the data sent between the IPsec peers.

Because the IPsec policy and filter settings for L2TP data are automatically configured, the inability to establish an IPsec SA is most likely caused by the failure of the main-mode SA negotiation because of misconfigured or missing computer certificates. Because of the secure nature of IPsec, the failure of the negotiation is not reported to the user.

15.2.1.1.6 L2TP Connection Negotiation

After the main-mode and quick-mode SAs are established, the VPN client and VPN server exchange a series of L2TP messages that are used to create an L2TP control connection and session. For more information about this process, see Section 5.0 of [RFC2661]. The following misconfigurations of the wide area network (WAN) miniport (L2TP) device of the Routing and Remote Access service are the most likely causes of a failure to create the L2TP control connection or session:

- The WAN miniport (L2TP) device is configured with too few ports. Either five or 128 L2TP ports are available by default, depending on the choices in the Routing and Remote Access Server Setup Wizard. Verify that there are enough L2TP ports for the maximum number of simultaneous L2TP connections to the VPN server.
- The WAN miniport (L2TP) device is not enabled to allow inbound remote access connections. Again, this setting is the result of choices made in the Routing and Remote Access Server Setup Wizard. Verify that the WAN miniport (L2TP) device is configured to allow inbound remote access connections.

Both the number of ports and the enabling of inbound remote access connections for the WAN miniport (L2TP) device are configured from the properties window of the Ports object in the Routing and Remote Access snap-in.

15.2.1.1.7 PPP Connection Negotiation

PPP connection negotiation for L2TP connections consists of the following phases:

1. Negotiation of the PPP link through the Link Control Protocol (LCP)
2. Authentication of the user attempting to connect
3. Negotiation and configuration of local area network (LAN) protocols, such as TCP/IP, using PPP network control protocols (NCPs)

Troubleshooting the PPP portion of the connection attempt involves its authentication and authorization, and depends on the dial-in properties of the user account, the domain and VPN server configurations, and remote access policies. For further detail on authentication and authorization, please refer to the Authentication Services overview in this document.

15.2.1.1.8 Common Configuration for a VPN Server

To deploy a VPN solution, the network administrator performs an analysis and makes design decisions regarding the network configuration, remote-access policy configuration, domain configuration, and security.

A typical Windows Server 2003 operating system VPN configuration involves the following steps:

1. Configure TCP/IP on LAN and WAN adapters.
2. Configure Routing and Remote Access service.
3. Configure the DHCP Relay Agent.
4. Configure static routes on the VPN server to reach intranet and Internet locations.
5. Configure remote access policy.
6. Configure domain users and groups.
7. Configure security policies.

For client VPN connections, configuration depends upon the protocol. The following section describes two examples.

15.2.1.1.9 PPTP-based Remote Access Client Configuration

On the Windows XP operating system remote-access client computers, the New Connection Wizard is used to create a VPN connection with the following settings:

- Network Connection Type: Connect to the network at my workplace
- Network Connection: Virtual Private Network connection
- Connection Name
- VPN Server Selection (for example: vpn.servername.example.com)
- Connection Availability: Anyone's use

15.2.1.1.10 L2TP/IPsec-based Remote Access Client Configuration

The remote access computer logs on to the domain using a LAN connection to the Contoso, Ltd intranet and receives a computer certificate through auto enrollment. Then, the New Connection Wizard is used to create a VPN connection with the following settings:

- Network Connection Type: Connect to the network at my workplace
- Network Connection: Virtual Private Network connection
- Connection Name
- VPN Server Selection: (for example: vpn.servername.example.com)
- Connection Availability: Anyone's use

On the Networking tab, Type of VPN is set to L2TP/IPsec VPN. When Type of VPN is set to Automatic, a PPTP connection is tried first. In this case, the network administrator does not want remote access clients that are capable of establishing an L2TP/IPsec connection to use PPTP.

15.2.1.1.11 HTTP-over-SSL VPN Solution Using SSTP

In Windows Server 2008 operating system, VPN connections are established by using HTTP over SSL using SSTP. SSL is also known as Transport Layer Security (TLS). HTTP over SSL on TCP port 443 is the protocol that has been used on the Internet for some time for collecting credit card numbers and other private data.

SSTP uses an HTTP-over-SSL session between VPN clients and servers to exchange encapsulated IPv4 or IPv6 packets. An HTTP-over-SSL-based remote-access VPN connection is different from the connection made by an application that uses HTTP over SSL. It should also be noted that SSTP does not support site-to-site VPN connections.

A computer running Windows Server 2008 is an SSTP-based VPN client, capable of initiating SSTP connections to an SSTP-based VPN server. The SSTP server must have a computer certificate with the Server Authentication or All-Purpose Enhanced Key Usage (EKU) property installed. The SSTP client uses this computer certificate to authenticate the SSTP server when the SSL session is established. The SSTP client validates the computer certificate of the SSTP server. To trust the computer certificate, the root certificate authority (CA) of the issuing CA of the SSTP server's computer certificate must be installed on the SSTP client.

15.2.1.1.12 SSTP

SSTP provides a mechanism for tunneling IP packets using PPP framing on top of SSL. This protocol enables users to access a private network using HTTP Secure (HTTPS). The use of HTTPS enables traversal of most firewalls and web proxies.

Many VPN services provide a way for mobile and home users to access the corporate network remotely by using PPTP and L2TP/IPsec. However, with the popularization of firewalls and web proxies, many service providers (for example, hotels) do not allow the PPTP and L2TP/IPsec traffic. This limitation results in users not receiving ubiquitous connectivity to their corporate networks. For example, GRE port blocking by many Internet service providers (ISPs) is a common problem with PPTP use.

SSTP provides an encrypted tunnel by means of the SSL/ TLS protocol. When a client establishes an SSTP-based VPN connection, the client first establishes a TCP connection to the SSTP server over TCP port 443. SSL/TLS negotiation occurs over this TCP connection. The client then validates the

server certificate it receives during the SSL authentication phase. If the server certificate is not valid or if it is not trusted by the client, the client terminates the connection.

After the successful negotiation of SSL/TLS, the client sends an HTTP request with content length encoding and a large content length on the SSL-protected connection. The server sends back an HTTP response. The HTTPS connection is now established, and the client can send and receive SSTP control packets and SSTP data packets on this connection. [SSLPROXY] specifies how to establish an HTTPS connection when a web proxy is present.

SSTP enables or performs the following functions:

- Delineation of PPP frames from the continuous stream of data that is sent by using HTTPS. For more information about PPP, see [RFC1661].
- Negotiation of parameters between two entities. Note that this feature is for future extensibility and is not completely used.
- Extensible message format to support new parameters in the future.
- Security operations to prevent a man-in-the-middle attacker from relaying PPP frames inappropriately over SSTP.

Because SSTP runs by means of an HTTPS connection, SSTP relies entirely on HTTPS for the reliable delivery of its messages. The SSTP client must use HTTPS authentication to authenticate the SSTP server. The SSTP server may use HTTPS client authentication to authenticate the SSTP client. The SSTP server must use PPP authentication to authenticate the SSTP client. Therefore, PPP authentication is required even when the SSTP server uses HTTPS authentication to authenticate the SSTP client. For more information about PPP, see [RFC1661].

15.3 VPN Services Protocols Logical Dependencies and Protocol Stack Views

This section describes the logical dependencies and protocol stack views for individual core protocols and also for core protocols as a group related to the VPN Services.

15.3.1 Logical Relationships

From an implementation perspective, the VPN core protocols that would be used to establish a VPN connection are:

- PPTP
- L2TP
- SSTP

From a task perspective, these core protocols do not depend upon each other, but each protocol depends upon other (non-core) protocols such as authentication and transport protocols.

15.3.1.1 PPTP Logical Relationships

PPTP is logically dependent upon TCP/IP (network layer), security protocols such as Internet Key Exchange (IKE), and one or more authentication protocols (such as Microsoft PPP Challenge Handshake Authentication Protocol (CHAP) Extensions [RFC2433] and Microsoft PPP CHAP Extensions Version 2 [RFC2759].

The user data carried by the PPTP protocol are PPP data packets encapsulated in GRE packets, which in turn are carried over IP.

15.3.1.2 PPTP Stack View

The arrangement described above is depicted in the following diagram of the PPTP stack.

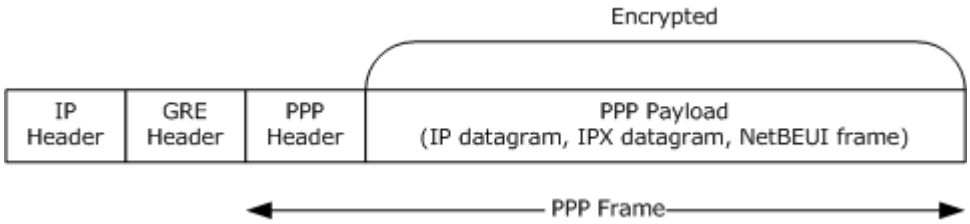

Figure 109: PPTP stack view

15.3.1.3 L2TP/IPsec Relationships

Similar to PPTP, L2TP/IPsec is logically dependent upon TCP/IP (network layer), security protocols such as Internet Key Exchange (IKE), and one or more authentication protocols (such as Microsoft PPP CHAP Extensions [RFC2433] and Microsoft PPP CHAP Extensions Version 2 [RFC2759]. It is also dependent upon IPsec. For information about IPsec, see [RFC3776] and [RFC3884].

15.3.1.4 L2TP/IPsec Stack View

The physical stack view of L2TP is depicted in the following diagram.

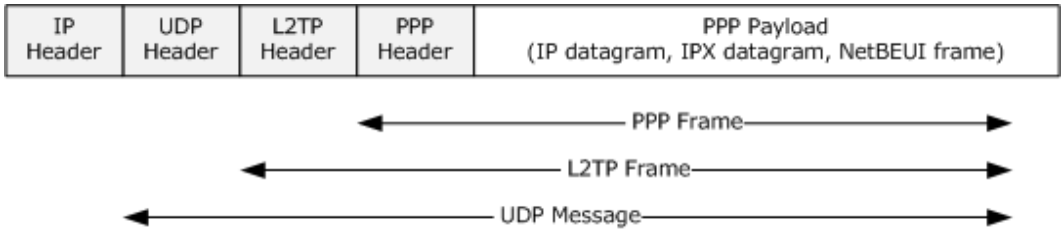

Figure 110: L2TP stack view

15.3.1.5 SSTP Logical Relationships

SSTP's relationships to other protocols is discussed in detail in section 15.2.1.1.12. As noted, SSTP allows encapsulation of PPP traffic over HTTPS, as specified in [RFC1945], [RFC2616], and [RFC2818]. For more information about PPP, see [RFC1661].

15.3.1.6 SSTP Stack View

The following diagram depicts the SSTP stack.

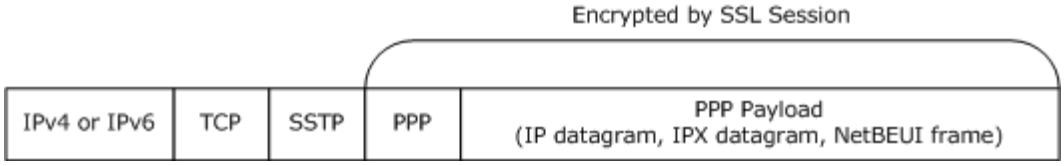

Figure 111: SSTP stack view

As stated earlier, VPN can be implemented using PPTP, L2TP, or SSTP. In the following section, two scenarios were traced to illustrate actual Windows VPN implementations.

15.4 Implementation Scenarios

This section provides two scenarios that illustrate common functionality that can be implemented with the protocols included in the VPN Services. The protocols involved in these scenarios include Point-to-Point Tunneling Protocol (PPTP), as specified in [RFC2637], together with other general authentication and networking protocols used in the course of standard TCP communication between the client and server. The functionality illustrated in the scenarios is as follows:

- Windows Vista operating system client connects via VPN to Windows Server 2003 operating system server.
- Windows Vista client connects via VPN to Windows Server 2008 operating system server.

It should be noted that the following scenarios depict processes that occur over TCP and thus assumes various TCP acknowledgment frames throughout the conversation that are not necessarily depicted in the trace details.

15.4.1 Scenario 1 - Vista Client Connects via VPN to Windows 2003 Server

15.4.1.1 Scenario Overview

In the following trace, a client computer running the Windows Vista operating system connects to a remote Windows Server 2003 operating system server computer. The server is configured to accept VPN connections through Routing and Remote Access using default settings. The following actions are then performed from the Vista Client machine:

- Map network drive.
- Open the Microsoft Word document from 2003 server on client machine.
- Print the Word document to printer attached to client machine.

15.4.1.2 Scenario Configuration

This scenario was implemented with a Windows Server 2003 operating system server and a Windows Vista operating system client machine configured as follows:

Server Configuration

- Machine: (WS200301)
- Operating system: Windows Server 2003 R2 operating system SP2
- Internal IP Address: 10.0.10.2
- Subnet Mask: 255.255.255.0
- External IP Address: 192.168.15.35
- Subnet Mask: 255.255.255.0
- Remote Access/VPN Server role configured
- Static address pool of 10.0.10.20 to 10.0.10.29 configured

- Create and share a folder

Client Configuration

- Machine: Client Vista
- Operating system: Windows Vista
- IP Address: 192.168.15.40
- Subnet Mask: 255.255.255.0
- Create VPN connection to WS200301
- Attach and install printer

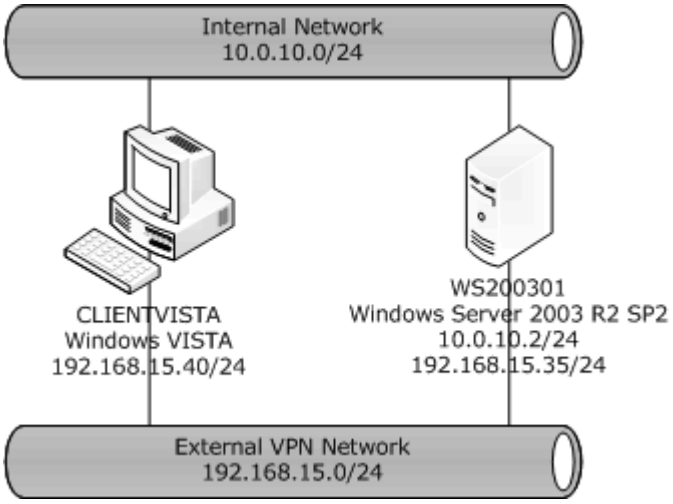

Figure 112: Network topology

15.4.1.3 Setting Up the Trace

These steps produce the VPN connection to the Windows Server 2003 operating system:

1. BEGIN TRACE.
2. Connect to the server via VPN.
3. Log in.
4. Map a network drive on the Windows Server 2003.
5. Open a Microsoft Word file from the Windows Server 2003 on the client machine.
6. Print the file to the Vista client's printer.
7. Edit and save the file.
8. Disconnect the VPN session.
9. END TRACE.

15.4.1.4 Netmon Trace Digest

- In this scenario, the client issues a request to the server for the MAC address (hardware address) using the Address Resolution Protocol (ARP). ARP is a protocol used by the IP [RFC826], specifically IPv4, to map IP network addresses to the hardware addresses used by a data link protocol. ARP is a general network protocol and is not covered in detail in this overview.

```
CLIENTVISTA WS200301 ARP ARP: Request, CLIENTVISTA asks for
WS200301
```

- The server responds with the following ARP command that contains the MAC address of the server:

```
WS200301 CLIENTVISTA ARP ARP: Response, WS200301 at
00-03-FF-92-46-6B
```

- The client and the server proceed to exchange a series of PPTP commands. PPTP specifies a series of control messages sent between a client and a server, or between a server and a server. These packets are sent and received between the client and the server to create, maintain, and disconnect a VPN across a public network.
- PPTP uses a TCP connection for tunnel maintenance and a modified version of GRE to encapsulate PPP frames for tunneled data. The payloads of the encapsulated PPP frames can be encrypted and/or compressed:

```
CLIENTVISTA WS200301 PPTP PPTP: Control Message , Start Control
Connection Request
WS200301 CLIENTVISTA PPTP PPTP: Control Message , Start Control
Connection Reply
CLIENTVISTA WS200301 PPTP PPTP: Control Message , Outgoing Call
Request
WS200301 VISTACLIENT PPTP PPTP: Control Message , Outgoing Call
Reply
CLIENTVISTA WS200301 PPTP PPTP: Control Message , Set Link Info
WS200301 CLIENTVISTA GRE GRE: Protocol = PPP,
Flags = ..K.....A....... Version 1 , Length = 0x0 , CallID = 0x92ec
```

- LCP [RFC1570] is used to exchange information about the transmission of data between client and server. This protocol makes sure that the connection is sufficient to enable a data exchange and that traffic will not commence until LCP determines the connection is adequate. If LCP determines the connection quality to be inadequate, it will drop the connection. LCP is a general purpose network protocol and is not covered in detail in this overview.

```
CLIENTVISTA WS200301 LCP LCP: Configure-Request, ID = 0,
Length = 21
WS200301 CLIENTVISTA LCP LCP: Configure-Request, ID = 0,
Length = 56
WS200301 CLIENTVISTA LCP LCP: Configure-Ack, ID = 0,
Length = 21
CLIENTVISTA WS200301 LCP LCP: Configure-Nak, ID = 1,
Length = 9
CLIENTVISTA WS200301 LCP LCP: Identification, ID = 1,
```

```
Length = 18
```

- Next, an Extensible Authentication Protocol (EAP) session is established between a client (EAP peer) and an EAP server. The EAP server and EAP peer negotiate the EAP method to use. The EAP method for Microsoft CHAP is selected. EAP methods operate within the EAP framework to provide support for a variety of authentication techniques, including EAP-TLS and EAP-MSCHAPv2. These are general purpose authentication protocols and are not covered in detail in this overview. Refer to the Authentication Services overview in this document.
- The EAP peer and EAP server continue to exchange EAP messages with MSCHAPv2 packets encapsulated in the payload. After the MSCHAPv2 packets successfully authenticate the client and the server to each other, the EAP authentication finishes:

```
WS200301 CLIENTVISTACHAP CHAP: Challenge, ID = 0, Length = 29 ,
 Name = 'WS200301'
VISTAVISTA WS200301 CHAP CHAP: Response, ID = 0, Length = 67 ,
 Name = 'administrator'
WS200301 CLIENTVISTA CHAP CHAP: Success, ID = 0, Length = 46 ,
 Message = 'S=36FA2D7B25D65FAF2279EFA1A191B474BA956A65'
```

The Microsoft implementation of PPP includes an optional callback control phase. This phase uses the Callback Control Protocol (CBCP) immediately after the authentication phase.

```
WS200301 CLIENTVISTA CBCP CBCP: Callback Request,
Identifier = 0x01
CLIENTVISTA WS200301 CBCP CBCP: Callback Response,
Identifier = 0x01
WS200301 CLIENTVISTA CBCP CBCP: Callback Ack,
Identifier = 0x01
```

- Next, PPP invokes the various network control protocols that were selected during the link establishment phase to configure protocols used by the remote client. In general terms, the IPCP configures, enables, and disables the IP protocol modules on both ends of the point-to-point link. Other protocols used in the test scenario are IPv6 Control Protocol (IPv6CP) and Compression Control Protocol (CCP).

```
CLIENTVISTA WS200301 IPCP IPCP: Code = Configure-Request,
Length = 34
CLIENTVISTA WS200301 IPv6CP IPv6CP: Code = Configure-Request,
Length = 14
CLIENTVISTA WS200301 CCP CCP: Configure-Request,Identifier=4
```

- Once the negotiation has been completed, PPP begins to forward data to and from the two peers. PPP provides a method for transmitting data packets over point-to-point links. Each transmitted data packet is wrapped in a PPP header, which is removed by the receiving system. If data compression was selected and negotiated, in-phase data is compressed before transmission. If data encryption is selected and negotiated, data is encrypted before transmission:

```
WS200301 CLIENTVISTA GRE GRE: Protocol = PPP,
Flags = ..K.....A....... Version 1 , Length = 0x0 , CallID = 0x92ec
WS200301 CLIENTVISTA PPP PPP: Compressed datagram
```

15.4.2 Scenario 2 - Vista Client Connects via VPN to Windows 2008 Server

15.4.2.1 Scenario Overview

In the following trace, a client computer running the Vista operating system connects to a remote Windows Server 2008 operating system Server computer. The server is configured to accept VPN connections through Routing and Remote Access using settings as described in the Configuration section below. The following action is then performed from the Client Vista machine:

- Map network drive.

15.4.2.2 Scenario Configuration

This scenario was implemented with a Windows Server 2003 operating system server and a Windows Vista operating system client machine configured as follows:

Server Configuration

- Machine: WS200801
- Operating system: Windows Server 2008 operating system Beta 3
- Internal IP Address: 10.0.10.4
- Subnet Mask: 255.255.255.0
- External IP Address: 192.168.15.35
- Subnet Mask: 255.255.255.0
- Network Policy and Access Services role configured
- Routing and Remote Access Services role service installed
- Static address pool of 10.0.10.20 to 10.0.10.29 configured
- Create and share a folder

CLIENT Configuration

- Machine: ClientVista
- Operating system: Windows Vista with latest updates
- IP Address: 192.168.15.40
- Subnet Mask: 255.255.255.0
- Create VPN connection to WS200801
- Attach and install printer

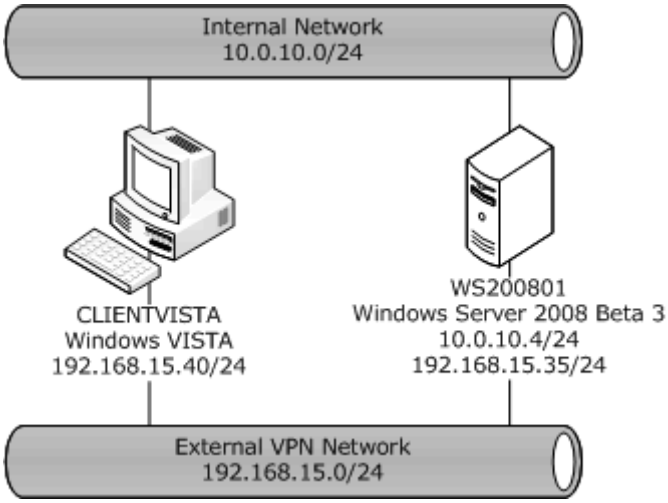

Figure 113: Network topology

15.4.2.3 Setting Up the Trace

These steps produce the VPN connection to the Windows Server 2008 operating system:

1. BEGIN TRACE.
2. Connect to the server via VPN.
3. Log in.
4. Map a network drive on the 2008 server.
5. Disconnect the VPN session.
6. END TRACE.

15.4.2.4 Netmon Trace Digest

- The client issues a request to the server for the MAC address (hardware address) using ARP, and the server responds with its address:

```
ClientVista  WS200801 ARP ARP: Request, 192.168.54.72
asks for 192.168.54.78
WS200801 ClientVista  ARP ARP: Response, 192.168.54.78 at
00-0F-1F-B2-DA-8F
```

- ARP is a protocol used by the IP [RFC826], specifically IPv4, to map IP network addresses to the hardware addresses used by a data link protocol. ARP is a general purpose protocol and is not covered in detail as part of this task. The client and server proceed to exchange a series of PPTP commands. PPTP specifies a series of control messages sent between a client and a server, or between a server and a server. These packets are sent and received between the client and server to create, maintain, and disconnect a VPN across a public network.

```
ClientVista  WS200801 PPTP PPTP: Control Message , Start Control
```

```
 Connection Request
WS200801 ClientVista  PPTP PPTP: Control Message , Start Control
Connection Reply
ClientVista  WS200801 PPTP PPTP: Control Message , Outgoing Call
 Request
WS200801 ClientVista  PPTP PPTP: Control Message , Outgoing Call
 Reply
ClientVista  WS200801 PPTP PPTP: Control Message , Set Link Info
```

- LCP [RFC1570] is used to exchange information about the transmission of data between client and server. This protocol ensures that the connection is sufficient to enable a data exchange and that traffic will not commence until LCP determines the connection is adequate. If LCP determines the connection quality to be inadequate, it will drop the connection.

```
ClientVista  WS200801 LCP LCP: Configure-Request, ID = 0,
Length = 21
WS200801 ClientVista  LCP LCP: Configure-Request, ID = 0,
Length = 56
WS200801 ClientVista  LCP LCP: Configure-Ack, ID = 0,
Length = 21
ClientVista  WS200801 LCP LCP: Configure-Reject, ID = 0,
Length = 35
WS200801 ClientVista  LCP LCP: Configure-Request, ID = 1,
Length = 25
ClientVista  WS200801 LCP LCP: Configure-Nak, ID = 1,
Length = 9
WS200801 ClientVista  LCP LCP: Configure-Request, ID = 2,
Length = 26
ClientVista  WS200801 LCP LCP: Configure-Ack, ID = 2,
Length = 26
```

- Next, an EAP session is established between a client (EAP peer) and an EAP server. The EAP server and EAP peer negotiate the EAP method to use. The EAP method for Microsoft CHAP is selected.
- The EAP peer and EAP server continue to exchange EAP messages with MSCHAPv2 packets encapsulated in the payload. After the MSCHAPv2 packets successfully authenticate the client and the server to each other, the EAP authentication finishes:

```
WS200801 ClientVista  CHAP CHAP: Challenge, ID = 0, Length = 34 ,
 Name = 'WS200801'
ClientVista  WS200801 CHAP CHAP: Response, ID = 0, Length = 67 ,
 Name = 'Administrator'
WS200801 ClientVista  CHAP CHAP: Success, ID = 0, Length = 46 ,
 Message = 'S=29322AE77CC4B0CB0259058D4F24B89A25BACF29'
```

- The Microsoft implementation of PPP includes an optional callback control phase. This phase uses the CBCP immediately after the authentication phase. If configured for callback, both the remote client and the NAS disconnect after authentication. The NAS then calls the remote client back at a specified phone number. This sequence provides an additional level of security to dial-up networking:

```
WS200801 ClientVista  CBCP CBCP: Callback Request,
Identifier = 0x01
ClientVista  WS200801 CBCP CBCP: Callback Response,
Identifier = 0x01
WS200801 ClientVista  CBCP CBCP: Callback Ack,
Identifier = 0x01
```

- Next, PPP invokes the various network control protocols that were selected during the link establishment phase to configure protocols used by the remote client. For example, during this phase, the IPCP can assign a dynamic address to the dial-in user. Other protocols used in the test scenario are IPv6CP and CCP.

```
WS200801 IPv6CP IPv6CP: Code = Configure-Request,
Length = 14
WS200801 CCPCCP: Configure-Request,Identifier=4
WS200801 IPCPIPCP: Code = Configure-Request,
Length = 34
```

- Once the negotiation has been completed, PPP begins to forward data to and from the two peers. Each transmitted data packet is wrapped in a PPP header, which is removed by the receiving system. PPTP uses a TCP connection for tunnel maintenance and a modified version of GRE to encapsulate PPP frames for tunneled data:

```
ClientVista  WS200801 GRE GRE: Protocol = PPP,
Flags = ..K.....A....... Version 1 , Length = 0x0 ,
CallID = 0x9a38
WS200801 ClientVista  TCPTCP:
Flags=....A..., SrcPort=1723, DstPort=49169, Len=0,
Seq=2767084019, Ack=3596352305, Win=255 (scale factor 8) = 65280
ClientVista  WS200801 PPP PPP: Compressed datagram
ClientVista  WS200801 PPP PPP: Compressed datagram
```

16 Web Services Protocols

This section provides an overview of Web Services protocols. It provides a high level conceptual overview of the various protocols and offers examples and explanations of how the protocols function.

16.1 Web Services Overview

The Web Services include protocols for managing and communicating with Microsoft servers that provide Internet services.

These protocols can be used to:

- Communicate with remote services.
- Provide remote services that can be accessed by existing clients.
- Provide management interfaces (for both content and administrative data) for the documented services.

These protocols do not include standard Internet protocols such as HTTP, HTTPS, SMTP, and FTP.

16.1.1 Protocols List

Web Services include two sets of protocols. The core protocols that provide the functionality related to the services and a set of protocols that are required for an implementation of the core protocols (as listed in the Normative References section of the core protocols technical specifications).

The following table lists the core protocols included in Web Services. This list does not include data structures, file structures, or algorithms that these core protocols depend on. The details for these technical dependencies are documented in the technical specifications for each core protocol.

Protocol name	Protocol description	Document short name
.NET Remoting Protocol	Provides an implementation for a sender-receiver channel that uses the TCP protocol to transmit packets. .NET Remoting allows client applications to use objects in other processes on the same computer or on a computer accessible over a network.	[MS-NRTP]
ASP.NET State Server Protocol	Enables the state management of a web application to be run out of process and on a dedicated server such as a Microsoft Windows NT service.	[MS-ASP]
FrontPage Server Remote Protocol Extensions	Enables a client to remotely author, administer, and browse a FrontPage website.	[MS-FPSE]
Internet Information Services IMSAdminBaseW Remote Protocol	Includes interfaces that provide Unicode-compliant methods for remotely accessing and administering the metabase of Internet Information Services (IIS).	[MS-IMSA]
Internet Information Services Inetinfo Remote Protocol	Defines the remote Internet Information Services (IIS) methods for calls made by the client.	[MS-IRP]

Protocol name	Protocol description	Document short name
.NET Remoting: Lifetime Services Extension	This protocol adds lifetime and remote activation capabilities to the .NET Remoting Core Protocol [MS-NRTP]. This protocol builds on the [MS-NRTP] specification.	[MS-NRLS]

The following sections provide some basic Web Services concepts and individual sections that provide different conceptual views to help visualize the physical and logical relationships among the core protocols in Web Services.

16.2 Web Services Protocols Functionality

This section describes concepts specific to web service protocols. This does not include standard Internet protocols such as HTTP; nor underlying transports such as TCP.

The primary purpose of these core protocols is to deliver web services and remote management capabilities to clients and servers; not to provide the authentication of the identity of such devices. All sections of this document presume that appropriate authentication has already taken place and that any necessary transport protocols are available to the clients and servers.

The Implementation Scenarios section, later in this document, provides examples that illustrate the interaction of some of the protocols in two sample configurations.

16.2.1 .NET Remoting Protocol

The .NET Remoting Protocol specifies a mechanism to invoke a method where the calling program and the target method are in different address spaces. Typically the calling program is invoking a method on a different machine. Arguments are passed along as part of the invocation message and return values are sent in the response.

This protocol defines two roles: client and server. The client interacts with a proxy object that represents the remote object. A formatter serializes the method invocation / message and a channel transports the serialized data. The .NET Framework includes built-in channels for HTTP and TCP for transport. The TcpChannel class streams data over TCP to a remote endpoint identified by a URI (uniform resource identifier).

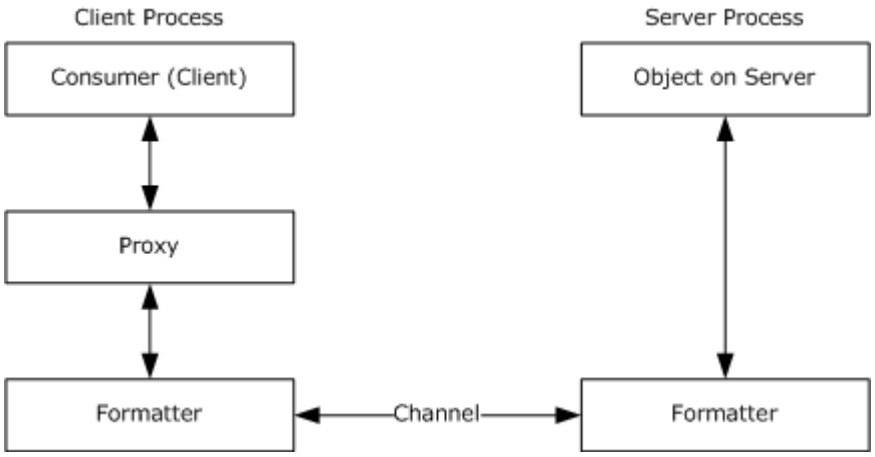

Figure 114: Conceptual remoting interaction

The client initiates communication by calling the remote method (with input arguments) using its local proxy. The server deserializes the message and responds by executing the method in an implementation-specific manner. The remote method can either be a one-way or a two-way method. If the method is one-way, no response is sent back to the client. If the method is two-way, the server sends back a response that can contain a return value and output arguments.

16.2.2 ASP.NET State Server Protocol

Web applications need to store state information associated with a specific user session. Earlier web technologies such as Active Server Pages (ASP) included a session state feature that stored state information in memory. ASP.NET also has an implementation of in-memory session state. However, in-memory session state solutions are not suitable for web farms (clusters of two or more web servers). In web farms there is no guarantee that a user session will reconnect to the same web server across multiple requests. As a result, if an in-memory session state solution is used, session data will appear to be lost when the user session connects to different servers.

The ASP.NET State Server Protocol was developed to address the use of session state in a web farm. A session state server can host an out of process session state store. Client applications such as web applications can store and retrieve session data across a web farm as long as each instance of the application is pointed at the same state server instance. The ASP.NET State Server Protocol specifies the rules for communicating session state data between a client application and a state server. The data that flows between the client application and a state server is transmitted using the Hypertext Transfer Protocol (HTTP).

When using the ASP.NET State Server Protocol, there is a client and a server component to each network conversation. The general sequence is as follows:

1. A client application runs code that requires session state. For example, a web application could process a browser request and as part of the processing, the application needs to access the session state associated with the browser session.

2. The client sends an HTTP request to the state server to retrieve session state. The request includes an identifier that correlates to the browser's user session, as well as an identifier for the specific web application making the request.

3. The state server receives the request, and based upon the user session identifier, application domain identifier, and web application identifier, retrieves the requested session data. The session data is returned in an HTTP response to the client.

4. Application code accesses session state, getting and setting values as necessary.

5. When the client finishes processing a request, it makes an HTTP request to the state server to save any changes application code has made to session state. This request contains the updated session state data, as well user session, application domain, and web application identifiers.

6. The state server accepts the HTTP request to update state data. It stores the information keyed by the user session identifier, application domain identifier, and application identifier.

16.2.3 FrontPage Server Extensions Remote Protocol

The FrontPage Server Extensions Remote Protocol specifies a set of server extensions that can be used to augment a basic HTTP server. These extensions provide file server functionality similar to WebDAV, allowing a web site to be presented as a file share. The FrontPage Server Extensions Remote Protocol is a precursor to the WebDAV protocol and may be used in similar situations. The use of WebDAV is recommended over the FrontPage Server Extensions Remote Protocol.

The FrontPage Server Extensions Remote Protocol uses HTTP version 1.1 as a transport. Requests are specialized form posts, and responses are in HTML. Despite the use of HTTP, the protocol is intended for use by a client program, not by the user directly through a web browser.

The FrontPage Server Extensions Remote Protocol is a subset of a larger protocol known as Extended FrontPage Server Extensions. The FrontPage Server Extensions Remote Protocol is used when communicating between Windows clients and Windows servers. Extended FrontPage Server Extensions is used to perform a wider array of web site administration. All implementations of the FrontPage Server Extensions Remote Protocol on Windows also implement Extended FrontPage Server Extensions.

The FrontPage Server Extensions Remote Protocol is used by client applications to display the contents of a web site as a file system. The FrontPage Server Extensions Remote Protocol provides file uploading and downloading, directory creation and listing, basic file locking, and file movement on a web server by using a set of methods. The FrontPage Server Extensions Remote Protocol has three concepts in its file system: files, directories, and services. Each of these structures MUST have a dictionary of metadata associated with it. This metadata can be used by clients or servers to store whatever information is relevant to the object. The metadata dictionary is also often used as a way for the server to communicate information about a file to the client, such as its checkout state.

Each method is an HTML form POST that accepts a set of parameters and returns a set of values as an HTML response. The method parameter defines what operation the server will perform in addition to the meanings of the other parameters and return values.

To give an example of how the protocol is used in Windows, a user might type an HTTP URL into the address bar of a file browser. The file browser contacts the server to determine if it supports the FrontPage Server Extensions Remote Protocol. If so, the file browser uses the FrontPage Server Extensions Remote Protocol to list the contents of the directory that the user entered and to display the contents just as it would display files on the local hard disk.

A typical conversation would be:

1. The HTTP OPTIONS request is sent to determine if the server supports the FrontPage Server Extensions Remote Protocol. If the response contains the MS-Author-Via header, the server supports the protocol. This value is often cached by clients.

2. The HTTP GET on _vti_inf.html will return information specifying the well-defined URLs to which the client must POST further method calls.

3. At this point the client is prepared to start making method calls against the server. The first call is a server version request whereby the client negotiates a protocol version with the server.

4. The client may make the "url to web url" request if the web site is a subweb (that is, not located at the root of the server's namespace).

5. The client may make an open service request on the web it wants to open. This request is optional but it will return information about the website's capabilities, such as support for version control.

6. The client may make any method calls against the server. The nature of any further client/server communication is determined by the specific needs of the client at the time.

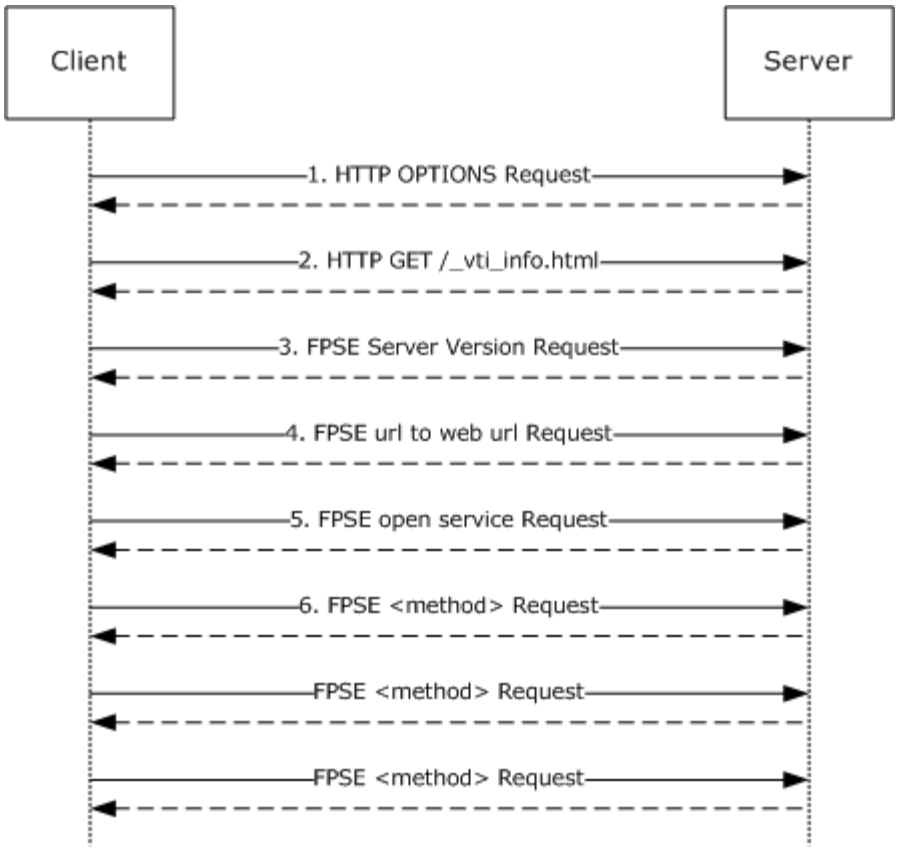

Figure 115: Generic Microsoft FrontPage Server Extensions conversation

The FrontPage Server Extensions Remote Protocol is a precursor to the WebDAV protocol and may be used in similar situations. Because it is an earlier technology, most developers will find WebDAV a more appealing option.

16.2.4 Internet Information Services IMSAdminBaseW Remote Protocol

Internet Information Services IMSAdminBase Remote Protocol provides interfaces that provide Unicode-compliant methods for remotely accessing and administering the IIS metabase associated with an application that manages IIS configuration, such as the IIS snap-in for Microsoft Management Console.

Internet Information Services IMSAdminBase Remote Protocol is a client/server protocol that is used for remotely managing a hierarchical configuration data store (metabase). A remote metabase management session begins with the client initiating the connection request to the server. If the server grants the request, the connection is established. The client can then make multiple requests to read or modify the metabase on the server by using the same session until the session is terminated.

A typical remote metabase management session involves the client connecting to the server and requesting to open a metabase node on the server. The IIS IMSAdminBase Remote Protocol relies on the DCOM protocol, which uses RPC as a transport. If the server accepts the request, it responds with an RPC context handle that refers to the node. The client uses this RPC context handle to operate on that node. This usually involves sending another request to the server specifying the

type of operation to perform and any specific parameters that are associated with that operation. If the server accepts this request, it attempts to change the state of the node based on the request and responds to the client with the result of the operation. When the client is finished operating on the server nodes, it terminates the protocol by sending a request to close the RPC context handle as shown in the figure below:

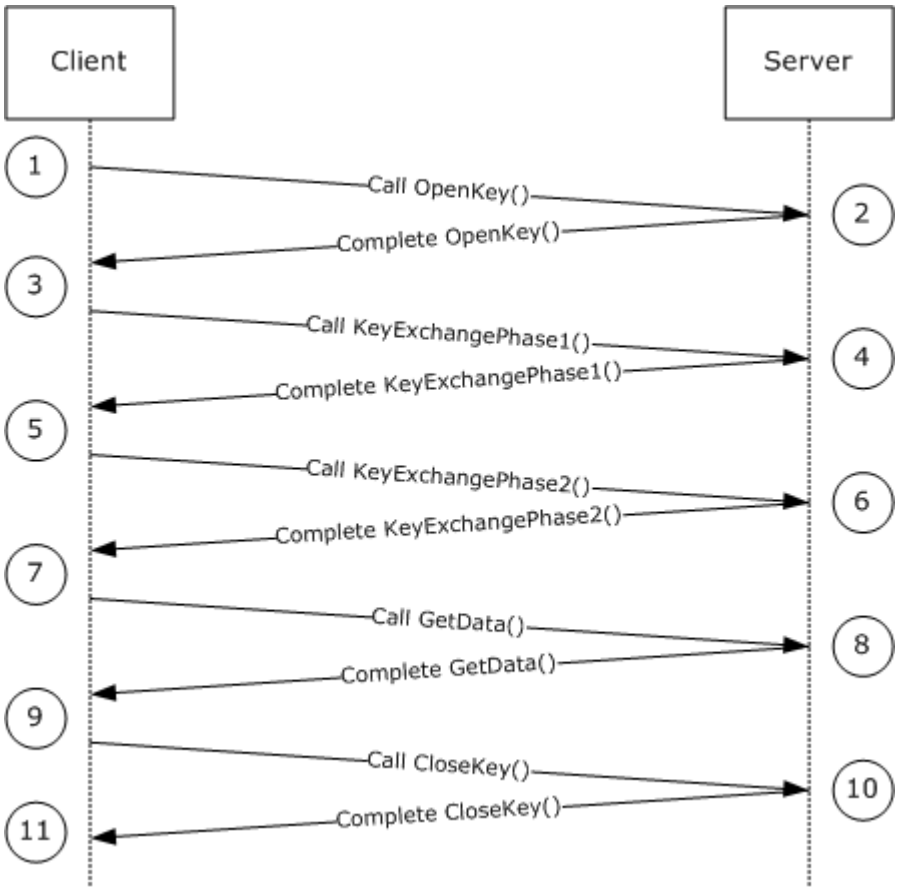

Figure 116: Sequence of messages for reading sensitive data from the server

The actual tasks that correspond to the numbers in Figure 116:

1. The client requests the server to open a node. Path location is "/mydata" and read access (METADATA_PERMISSION_READ) is requested.

2. The server checks if the "/mydata" node exists and if the connecting user is granted read access and then returns the handle back.

   Before the client sends a request to retrieve specific data from the "/mydata" location, the client will have to negotiate a secure session (that is one that was not negotiated yet)

3. The client performs the phase 1 of the handshake. Client's Key Exchange Key and Client's Signature Key are generated and public keys for both are sent to the server. Private keys for both are stored by client.

4. The server receives public keys from the client and retrieves or generates/locates its own Server's Key Exchange Key and Server's Signature Key.

It also generates the server's Session Key. It encrypts it with the client's Key Exchange Public Key. The server's Key Exchange Public Key and the server's Signature Public Key along with encrypted server's Session Key are sent back to the client.

5. The client receives Server's Key Exchange Public Key, Server's Signature Public Key and the encrypted Server's Session Key. It decrypts Server's Session Key using Client's Key Exchange Private Key.

   Client's Session Key gets generated and is encrypted with Server's Key Exchange Public Key. Also Server's Session Key Hash is generated based on Client's Session Key, Server's Session Key and constant string (HASH_TEXT_STRING_1). Encrypted Client's Session Key and the Server's Session Key Hash are sent to server.

6. The server receives the encrypted client's Session Key as well as the server's Session Key Hash from the client. It verifies the hash generated by the client to assure that the client was able to decrypt the server's session key. The server generates the client's Session Key Hash using Client's Session Key and constant string (HASH_TEXT_STRING_2). It sends it to client.

7. The client receives hash Client's Session Key Hash from the server. It verifies that the server owns the private key for key exchange key pair and was able to decrypt the client's session keys.

   At this point the server and the client exchanged session keys that will be used to encrypt sensitive data. Also the signature keys were exchanged. They will be used for message integrity checks.

   The client makes a GetData() call to retrieve sensitive data.

8. The server retrieves the requested data. It will find out if secure flag (METADATA_SECURE) is set. The server will encrypt the data value requested and build IIS_CRYPTO_BLOB message that will be sent to client.

9. The client checks the received data. It finds out that METADATA_SECURE is set and will decrypt the data and check the signature.

   The client will call CloseKey() to close the handle that was opened in step 2.

10. The server closes the handle and responds with a success code to the client.

The IMSAdminBase2W Remote Procedure Call (RPC) interface extends the IMSAdminBaseW interface, adding functionality for metabase importing and exporting, history management, and secure data encryption on backup. The IMSAdminBaseW protocol does not maintain client state information.

The IMSAdminBase3W RPC interface extends the IMSAdminBase2W interface by providing a method to return the nodes of children from a specified metabase path. The IMSAdminBase3W protocol does not maintain client state information. The protocol is stateless.

16.2.5 Inetinfo Remote Protocol

The Inetinfo Remote Protocol [MS-IRP] is an RPC-based client/server protocol that is used for managing Internet Protocol Servers such as those hosted by Microsoft Internet Information Services (IIS). Managed servers may include servers for HTTP, FTP, SMTP or other Internet protocols. The Inetinfo Remote Protocol provides functions that allow remote administration and statistics gathering from an Internet Protocol Server. The protocol provides methods for gathering statistical data about users, sites, requests, and performance.

The server does not maintain client state information. Although some client call sequences may be logically related, the protocol operation is stateless. Inetinfo Remote Protocol clients can invoke the RPC methods that are specified in this section in any order after an Inetinfo Remote Protocol session is established with the server. The outcome of the calls depends on the parameters that are passed to each of those calls and not on a particular call sequence or state maintained across method invocations.

It should also be noted that the Inetinfo Protocol has been modified between versions of Internet Information Server in ways that make interoperability between the different server versions difficult. For more information about the changes between IIS versions or the differences in Common Data Types, see the [MS-IRP] protocol documentation.

16.2.6 .NET Remoting Lifetime Services Protocol

The .NET Remoting Lifetime Services Protocol [MS-NRLS] protocol adds lifetime and remote activation capabilities to the .NET Remoting Core Protocol [MS-NRTP]. The protocol defines mechanisms for the creation of Server Objects and the invocation of Remote Methods on those Server Objects. This protocol adds a mechanism that allows clients to explicitly create Server Objects and another mechanism that allows clients and servers to control the lifetime of Server Objects.

16.2.6.1 Client Activation

This protocol introduces a new type of Server Object called Client Activated Object (CAO). A CAO can be remotely activated by a client by invoking a special Remote Method, Activate(), on a well-known SAO, passing the Server Type. The implementation of the SAO creates a new instance of the Server Type, registers it in the Server Object Table, and sends a Server Object Reference to the instance back to the client. The client receives the Server Object Reference and can use it to create a Proxy to invoke methods on the CAO.

An example of a client activating an instance and invoking a Remote Method, increment(), is depicted in the following diagram.

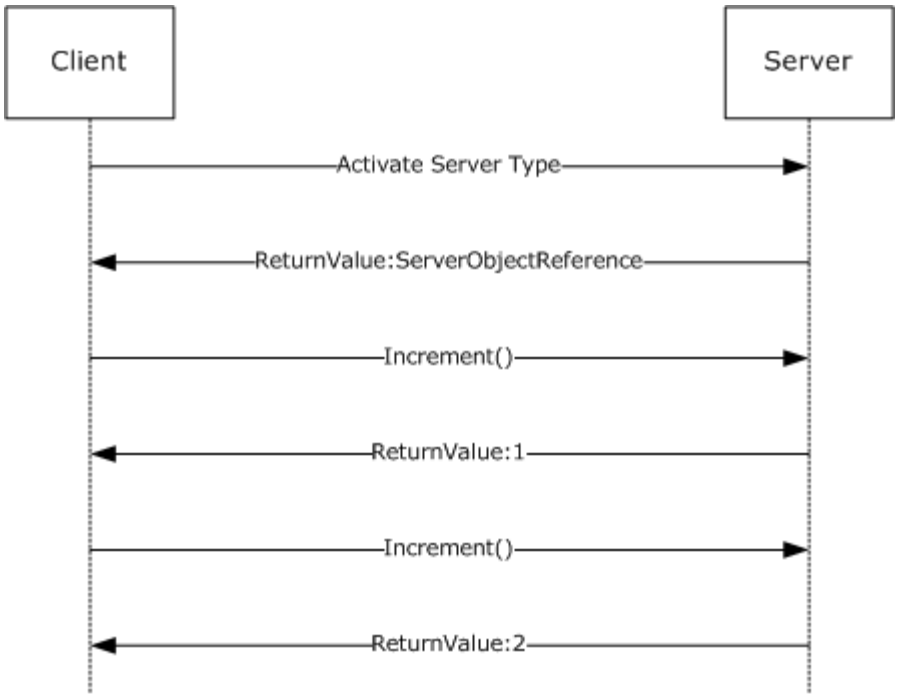

Figure 117: Client activates instance and invokes remote method

16.2.6.2 Lifetime Management

The protocol specifies a lease-based model for lifetime management of Marshaled Server Objects (MSOs) and Singleton SAO. A Lease Object is associated with each Server Object. Each Lease Object has an initial Time-To-Live (TTL) for the Server Object. For every Remote Method invocation on the Server Object the TTL is extended. If no calls are made to the Server Object for the duration of the TTL, the Server Object is considered for removal from the Server Object Table. A client can explicitly control the Server Object's lifetime through Remote Method invocations on the Server Object's Lease Object. The client gets a Server Object Reference to the Lease Object for a Server Object by calling the Server Object's GetLifetimeServiceRequest() Remote Method. The client can then invoke the Renew() Remote Method on the Lease Object to extend the TTL by a desired amount.

16.2.6.3 Sponsor

A Lease Object for a given Server Object maintains a list of Sponsors that get called when the TTL of the Server Object expires. Each Sponsor can specify if the Server Object's TTL must be extended and the duration of the extension. If there are no Sponsors associated or if none of the associated Sponsors extend the lifetime of the Server Object, then the Server Object is removed from the Server Object Table, making it unavailable to clients.

An example of a client managing the lifetime of a Server Object is shown below.

1. The client invokes a Remote Method on the Server Object:

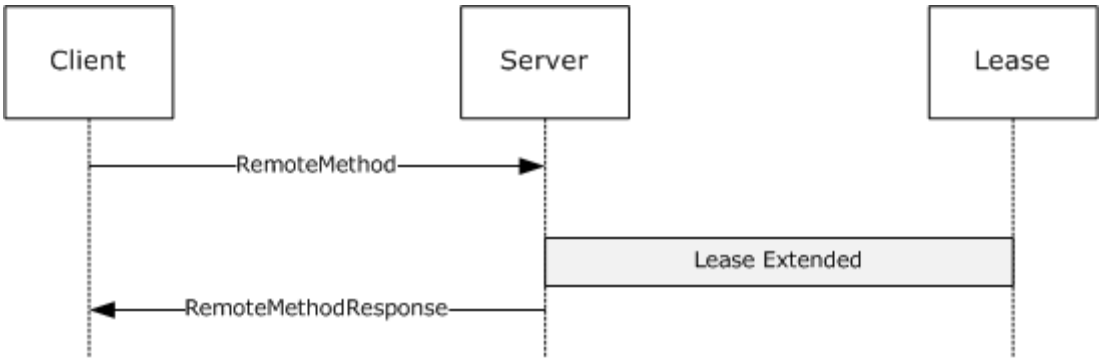

Figure 118: Remote method on server

2. The client uses the Lease Object to extend the lease-time:

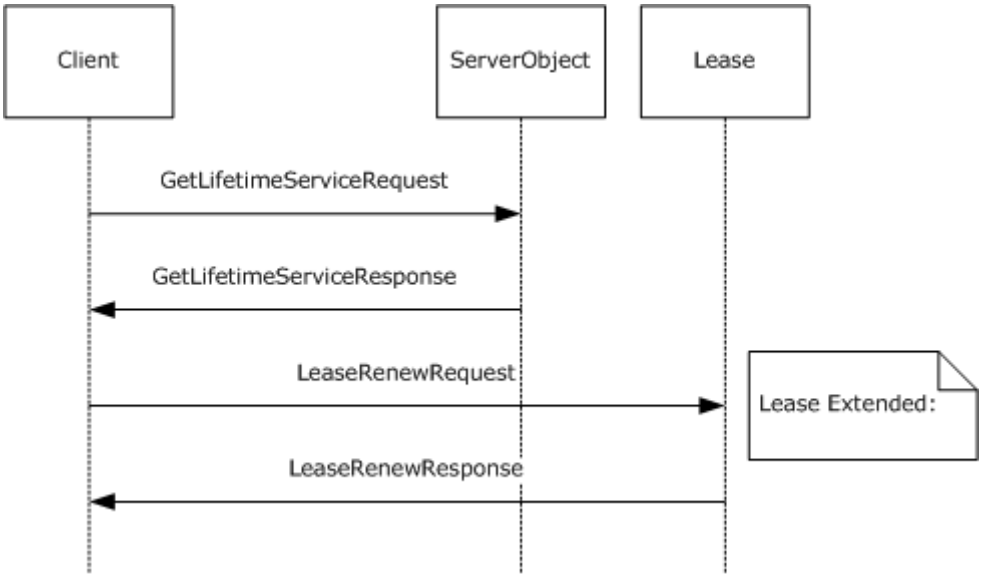

Figure 119: Lease object extends lease time

3. The client registers a Sponsor that gets invoked when the Lease Object's TTL expires.

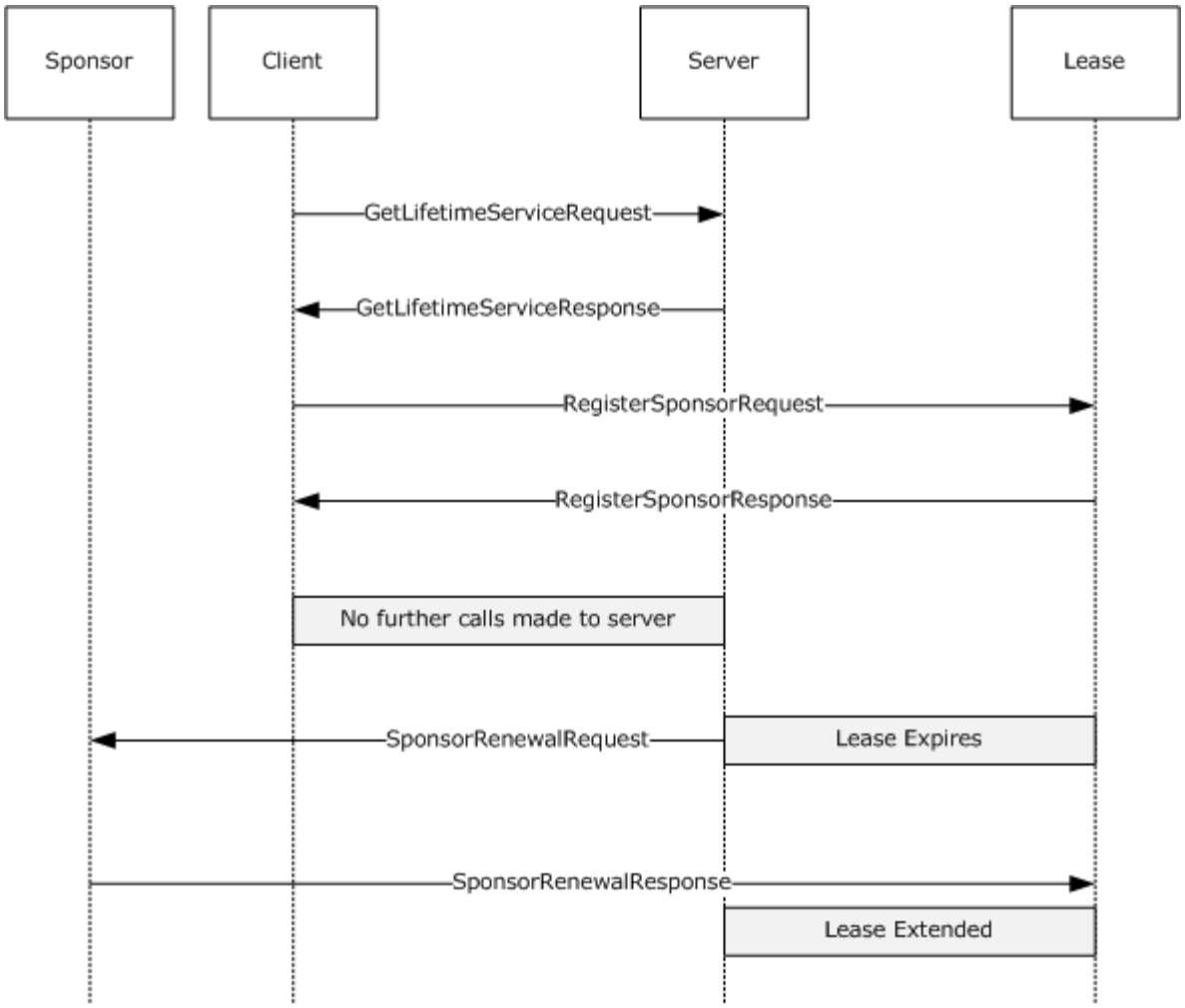

Figure 120: Client registers sponsor

16.3 Web Services Protocols Logical Dependencies

Unless otherwise specified in the individual protocol documents, the core Web Services protocols are not interdependent.

The following sections describe the dependencies of each core protocol to other protocols, including protocols that may not be core to Web Services.

16.3.1 .NET Remoting Protocols Dependencies

The .NET Remoting protocols stack is depicted in the following diagram:

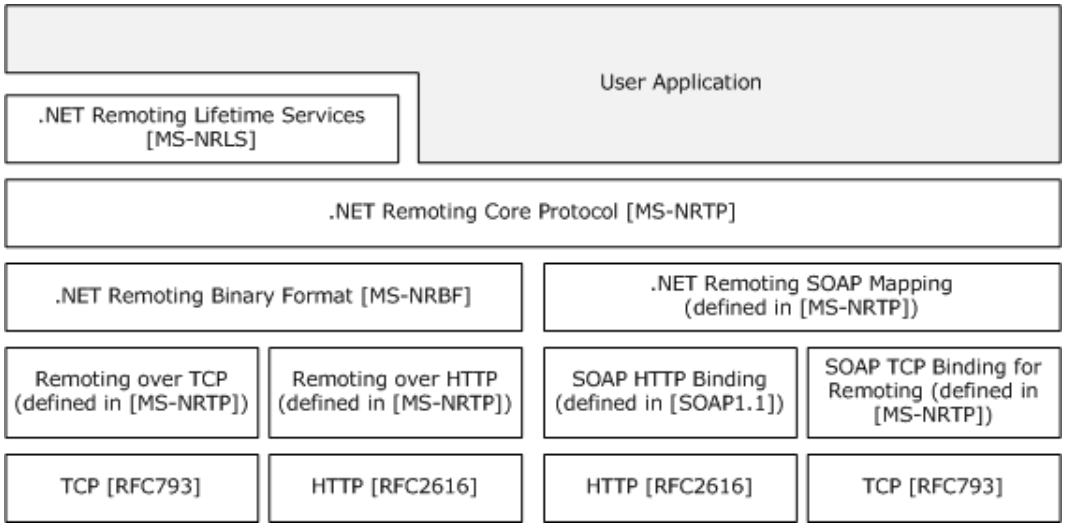

Figure 121: .NET Remoting protocols stack

The specific dependencies of each .NET Remoting protocol are described in the subsections below.

16.3.1.1 .NET Remoting Protocol Relationship to Other Protocols

This protocol depends on other structures and protocols for the encoding and transport of its messages. Further protocols may extend this protocol to provide additional services, such as .NET Remoting Lifetime Services [MS-NRLS] which defines additional message and semantics to add activation and distributed lifetime management to .NET Remoting.

.NET Remoting can be bound to TCP or HTTP. It can also use SOAP 1.1, which includes both encoding semantics and transport bindings.

If the HTTPS transport is used, a server certificate must be deployed and a client certificate may be deployed.

This protocol does not define any means for activating a server or a client. The server must be configured and begin listening in an implementation-specific way on a Channel. The client must know the format and transport used by the server (for example, Binary format over HTTP).

16.3.1.2 ASP.NET State Server Protocol Relationship to Other Protocols

The ASP.NET state server relies on HTTP. The URL for the ASP.NET state server follows the definitions in "Uniform Resource Locators (URL)" as specified in [RFC1738].

The allowable characters for strings defined in section 2.2.1.6 come from the definition of a relative URI defined in "Uniform Resource Identifiers (URI): Generic Syntax" as specified in [RFC2396].

16.3.1.3 .NET Remoting: Lifetime Services Extension Relationship to Other Protocols

The .NET Remoting: Lifetime Services Extension Protocol extends the .NET Remoting Core Protocol, adding new methods for activation and lifetime management.

16.3.2 FrontPage Server Remote Protocol Extensions Relationship to Other Protocols

The FrontPage Server Extensions Remote Protocol is transported via HTTP version 1.1 POSTs and responses.

SharePoint Team Services relies on the FrontPage Server Extensions Remote Protocol to discover the name and location of the Microsoft Office Web Services Server executable. Windows extensions to WebDAV use its error codes, as specified in [MS-WDV].

The client must know the URL of the server it wants to communicate with, which is usually passed by the user as the prompt for beginning the FrontPage Server Extensions Remote Protocol conversation. If required by the server, the client must authenticate by using the underlying HTTP mechanisms.

16.3.3 Internet Information Services IMSAdminBaseW Remote Protocol Relationship to Other Protocols

The Internet Information Services IMSAdminBaseW Remote Protocol relies on the Distributed Component Object Model (DCOM) Remote Protocol, which uses RPC as a transport, as specified in [MS-DCOM].

No other IIS protocols rely on this protocol.

16.3.4 Internet Information Services Inetinfo Remote Protocol Relationship to Other Protocols

The IIS Inetinfo Remote Protocol uses RPC as its protocol transport, as specified in [MS-RPCE].

16.4 Implementation Scenarios

This section provides two scenarios that illustrate common functionality that can be implemented with the protocols included in Web Services. The protocols involved in these scenarios include the .NET Remoting Protocol, as specified in [MS-NRTP], together with other general networking protocols used in the course of standard TCP communication between the client and server. The functionality illustrated in the scenarios is as follows:

1. Calling a Method on a Remote Object via TcpChannel
2. Viewing and Configuring Web Sites

It should be noted that the following scenarios depict processes that occur over TCP and thus assumes various TCP acknowledgment frames throughout the conversation that are not necessarily depicted in the trace details.

16.4.1 Scenario 1 - Calling a Method on a Remote Object via TcpChannel

16.4.1.1 Scenario Overview

In the following scenario, a client application calls the Test method of a remote object that uses the binary formatter and the TcpChannel class. The remote object is hosted on a web server at the endpoint tcp://192.168.200.1:9999/Server.rem.

16.4.1.2 Scenario Configuration

This scenario was implemented with a Windows Server 2003 operating system web server, and a Windows Server 2003 client machine configured as follows:

Server Configuration

- Machine: Windows Server 2003 web server (WS03WebServer)
- Operating system: Windows Server 2003 R2 operating system with latest updates
- IP Address: 192.168.200.1
- Subnet Mask: 255.255.255.0
- Application Server role
- Custom remoting code used

Client Configuration

- Machine: Windows Server 2003 as client (WS03Client)
- Operating system: Windows Server 2003 R2 with latest updates
- IP Address: 192.168.200.2
- Subnet Mask: 255.255.255.0
- Custom remoting code used
- NetMon 3.1

Network Topology

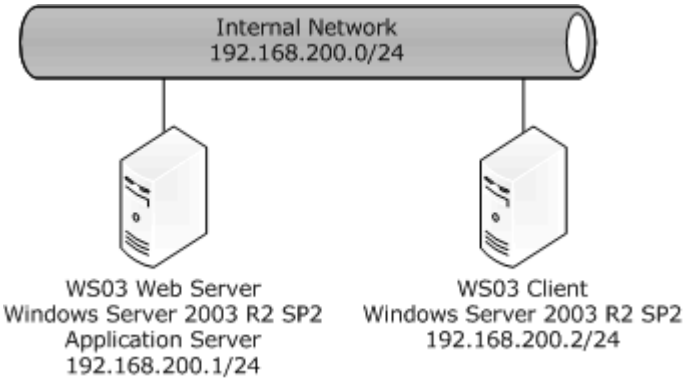

Figure 122: Network Topology

16.4.1.3 Setting Up the Trace

1. WS03 Web Server
   - Start Server.exe custom remoting code application
2. WS03 Client

1. BEGIN TRACE
    - Start Client.exe custom remoting code application
2. STOP TRACE

16.4.1.4 Netmon Trace Digest

- The method call on the remote object is sent in two frames seen in the following trace fragments:

```
WS03Client WS03WebServer DotNETRemotingTcpChannel: Request
tcp://WS03WebServer:9999/Server.rem

WS03Client WS03WebServer TCP TCP: Flags=...PA...,
SrcPort=1027, DstPort=9999,
Len=115, Seq=1462812425 - 1462812540, Ack=3294162129, Win=65535
```

- The payload of the second frame contains the fully qualified name of the method being invoked and the parameter being sent. The hexadecimal payload looks like this:

```
54 65 73 74 12 4B 52 65 6D 6F 74 65 4F 62 6A 65
63 74 2C 20 53 65 72 76 65 72 2C 20 56 65 72 73
69 6F 6E 3D 30 2E 30 2E 30 2E 30 2C 20 43 75 6C
74 75 72 65 3D 6E 65 75 74 72 61 6C 2C 20 50 75
62 6C 69 63 4B 65 79 54 6F 6B 65 6E 3D 6E 75 6C
6C 01 00 00 00 08 CD 81 01 00
```

- The last four bytes are the parameter value passed to the remote object (0x000181CD equals 98765). The text representation of the full payload is:

```
Test.KRemoteObject, Server, Version=0.0.0.0, Culture=neutral,
PublicKeyToken=null.....Í.
```

- The response from the remote object is also two frames seen in this trace segment:

```
WS03WebServer WS03Client DotNETRemotingTcpChannel: Response
WS03WebServer WS03Client TCP: Flags=...PA..., SrcPort=9999,
DstPort=1027, Len=30, Seq=3294162145 - 3294162175, Ack=1462812540,
 Win=65329
```

- The hexadecimal payload looks like this:

```
00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00
00 00 16 11 08 00 00 12 05 39 38 37 36 35
```

- The last five bytes are the characters '9', '8', '7', '6', and '5'.

16.4.2 Scenario 2 - Viewing and Configuring Web Sites

16.4.2.1 Scenario Overview

In the following scenario, the MMC snap-in for IIS is used to connect to an IIS 6.0 server to retrieve a list of websites, create a new website on the server, and restart an application pool.

16.4.2.2 Scenario Configuration

This scenario was implemented with a Windows Server 2003 operating system web server, and a Windows Server 2003 client machine configured as follows:

Server Configuration

- Machine: Windows Server 2003 web server (WS03WebServer)
- Operating system: Windows Server 2003 R2 operating system with latest updates
- IP Address: 192.168.200.1
- Subnet Mask: 255.255.255.0
- Application Server role

Client Configuration

- Machine: Windows Server 2003 as client (WS03Client)
- Operating system: Windows Server 2003 R2 with latest updates
- IP Address: 192.168.200.2
- Subnet Mask: 255.255.255.0
- NetMon 3.1

Network Topology

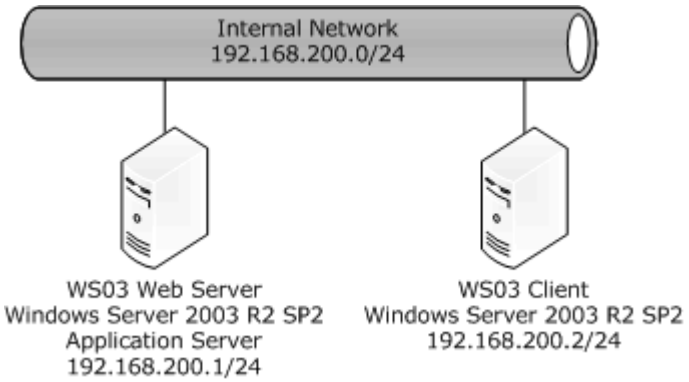

Figure 123: Network topology

16.4.2.3 Setting Up the Trace

1. WS03 Web Server

- Configure three websites all using DefaultAppPool:
  1. Default website
  2. Website1
  3. Website2

2. WS03 Client
   1. BEGIN TRACE
      1. Open IIS Manager MMC
      2. Connect to WS03 web server
      3. Create a new website called Website3
      4. Restart DefaultAppPool
   2. STOP TRACE

16.4.2.4 Netmon Trace Digest

- The client (WS03Client) binds to IMSAdminBaseW and makes requests using DCOM as shown in the following trace frames. The binary data payload of each request and response is encrypted.

```
{MSRPC:6, TCP:5, IPv4:1} WS03ClientWS03WebServerNTLMSSPNTLMSSP:
NTLM NEGOTIATE MESSAGE, Workstation Name: WS200302,
Workstation Domain: WORKGROUP

{MSRPC:6, TCP:5, IPv4:1} WS03WebServerWS03ClientNTLMSSPNTLMSSP:
NTLM CHALLENGE MESSAGE

{MSRPC:6, TCP:5, IPv4:1} WS03ClientWS03WebServerNTLMSSPNTLMSSP:
NTLM AUTHENTICATE MESSAGE, Domain: WS200302, User: Administrator,
Workstation: WS200302

{MSRPC:6, TCP:5, IPv4:1} WS03ClientWS03WebServerDCOMDCOM:
Request

{TCP:5, IPv4:1}WS03WebServer WS03ClientTCPTCP: Flags=....A...,
SrcPort=1029, DstPort=1063, Len=0, Seq=207726941, Ack=1482420964,
Win=64940 (scale factor 0) = 64940

{MSRPC:6, TCP:5, IPv4:1} WS03WebServerWS03ClientDCOMDCOM:
Response

{MSRPC:6, TCP:5, IPv4:1} WS03ClientWS03WebServerDCOMDCOM:
Request

{MSRPC:6, TCP:5, IPv4:1} WS03WebServerWS03ClientDCOMDCOM:
Response
…
…
```

17 Windows Server Update Services Protocols

This section provides an overview of the Windows Server Update Services (WSUS) protocols. It provides a high-level conceptual overview of the core WSUS protocols.

17.1 Windows Server Update Services Overview

Windows Server Update Services include protocols that communicate, via Simple Object Access Protocol (SOAP) over Hypertext Transport Protocol (HTTP), software update information, software update metadata, and XPRESS compression data between a WSUS implementation and a Windows Client configured to communicate with the Windows Update Services (WUS).

17.1.1 Protocols List

Windows Server Update Services include two sets of protocols. The core protocols provide the Windows Server Update Services functionality, and another set of protocols is required for an implementation of the core protocols (as listed in the Normative References section of the core protocols technical specifications), or to provide core networking functionality.

The following table lists the core protocols. This list does not include data structures, file structures, or algorithms that these core protocols depend on. The details for these technical dependencies are documented in the technical specifications for each core protocol.

Protocol name	Protocol description	Document short name
Background Intelligent Transfer Service (BITS) Peer-Caching: Peer Authentication Protocol	Transfers files (downloads or uploads) between a client and a server and provides progress information related to the transfers. Files may also be downloaded from a peer.	[MS-BPAU]
Background Intelligent Transfer Service (BITS) Peer-Caching: Content Retrieval Protocol	Defines methods for a network client both to query multiple servers for data associated with a given uniform resource locator (URL) and to download that data.	[MS-BPCR]
Background Intelligent Transfer Service (BITS) Peer-Caching: Peer Discovery Protocol	Used to locate networked hosts or devices that are implementing the server role of the BITS Peer-Caching: Content Retrieval Protocol. The BITS Peer-Caching: Peer Discovery Protocol provides a way for peer servers to announce their presence to connected subnets and a way for peer clients to locate servers in connected subnets.	[MS-BPDP]
Windows Server Update Services: Client-Server Protocol	Enables a computer running Microsoft Windows to communicate with the WUS and the Microsoft Software Update Services (SUS) server component.	[MS-WUSP]
Windows Server Update Services: Server-Server Protocol	Enables synchronization of updates within a hierarchical topology of Windows Server Update Servers. The WSUSSS protocol is a SOAP-based protocol that uses HTTP 1.1 as its transport.	[MS-WSUSSS]

The following sections provide some basic Windows Server Update Services protocol concepts and include individual subsections that provide different conceptual views to help visualize the physical and logical relationships among the core protocols.

17.2 Windows Server Update Services Protocols Functionality

This section describes basic Windows Server Update Services protocols concepts and provides some background for the WSUS model.

The WSUS family of protocols provides support for central publication and distribution of software components and software updates from server machines to client machines, and for the hierarchical synchronization of available software components between servers.

WSUS management infrastructure includes the following functionality:

- Microsoft Update - The Microsoft website that distributes updates for Microsoft products.
- WSUS server - The server component that is installed on a computer running Microsoft Windows Server 2003 or 2008 operating system (OS) inside a corporate firewall. WSUS server software enables administrators to manage and distribute updates through an administrative console, which can be used to manage any WSUS server in any domain with which it has a trust relationship. A WSUS server can obtain updates either from Microsoft Update or from another WSUS server, but at least one WSUS server in the network must connect to Microsoft Update to get available updates. The administrator can decide how many WSUS servers should connect directly to Microsoft Update, based on network configuration, bandwidth, and security considerations. These servers can then distribute updates to other downstream WSUS servers.
- Automatic Updates - The client computer component built into Windows OSs. Automatic Updates enables both server and client computers to receive updates either from Microsoft Update or from a WSUS server.
- Software updates - includes:

  1. Update files - The actual files that are installed on client computers.

  2. Update metadata - The information needed to perform the installation. This includes (i) update properties (title, description, Knowledge Base article, Microsoft Security Response Center number) and (ii) applicability rules (used by Automatic Updates to determine whether or not the update is needed on a particular computer).

- Installation information - Command-line options to apply when installing the updates.

Update files and Update metadata can be downloaded independently of each other. For example, WSUS can be configured so that it will not store updates locally, meaning that only update metadata (and any applicable Microsoft Software License Terms) will be downloaded to the WSUS server; clients will get their update files directly from Microsoft Update. If updates are stored locally on the WSUS server, a Windows client can either download everything at the time of synchronization, or download only the metadata during the synchronization, leaving the actual update files to be downloaded after the update is approved.

When updates are synchronized to the WSUS server, the metadata and update files are stored in two separate locations. Metadata is stored in the WSUS database. Update files can be stored either on the WSUS server or on Microsoft Update servers, depending on how the synchronization options have been configured. If the update files are stored on Microsoft Update servers, only metadata is downloaded at the time of synchronization, and updates are approved through the WSUS console. Client computers then get the update files directly from Microsoft Update at the time of installation. WSUS may be configured to deliver specific updates to target groups of computers, according to each group's update settings.

The Implementation Scenarios section, later in this document, provides examples that illustrate the interaction of some of the protocols in two sample configurations.

17.2.1 Windows Server Update Services: Server-Server Protocol

The Windows Server Update Services: Server-Server Protocol is a SOAP-based protocol that enables synchronization of updates within a hierarchical topology of update servers. The protocol operates over these transports:

- Web Services—SOAP 1.1 or SOAP 1.2 and SOAP 1.2/2 over HTTP or HTTP Secure (HTTPS) over Transmission Control Protocol/Internet Protocol (TCP/IP)
- Upstream Server (USS) Content Download—HTTP over TCP/IP

Figure 124 shows a common hierarchical topology of update servers and client computers. A USS in a hierarchy provides information about software and drivers to downstream servers (DSSs). Any update server in the hierarchy can serve simultaneously as a DSS for its USS and as a USS for its DSSs.

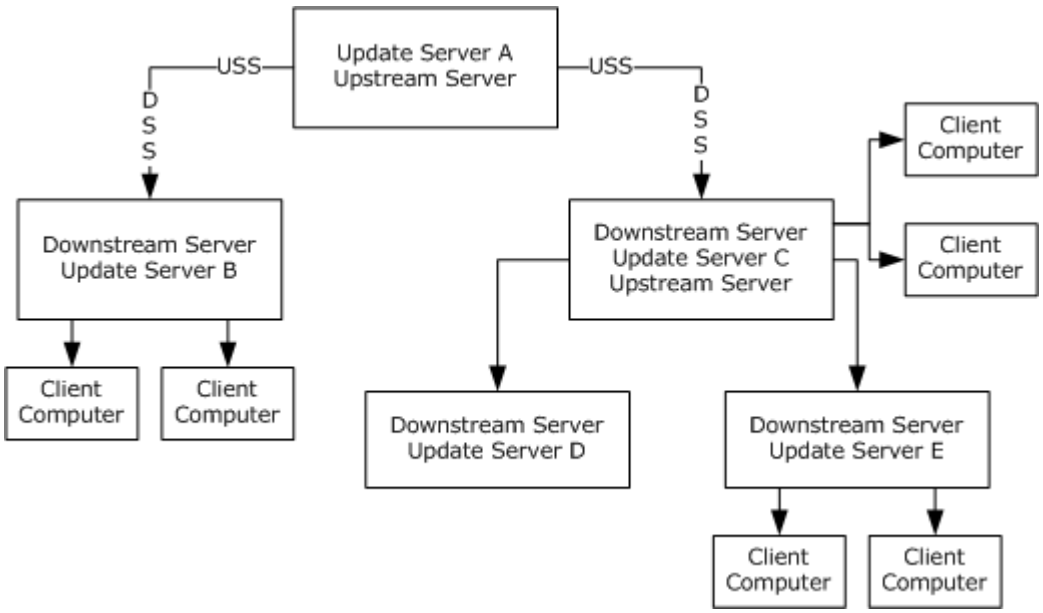

Figure 124: Hierarchical topology of update servers and client computers

For example, in figure 124, update server C acts as a DSS when communicating with its USS A and acts as a USS when communicating with its DSS D or E.

An update server groups its client computers into target groups. An update server can be configured to deploy the updates to its client computers by assigning the updates to the target groups for deployment, and optionally by specifying an installation or uninstallation deadline. This mapping of the individual update revisions to target groups is known as a deployment.

Two types of DSS are defined by this protocol: autonomous and replica. Either type of DSS gets its updates from the USS. However, only a replica DSS gets the deployments from its USS. The autonomous DSS manages its deployments independently from the USS.

17.2.1.1 Protocol DSS and USS Roles

The Windows Server Update Services: Server-Server Protocol operates between two update servers that act in these roles:

- DSS - This update server initiates all communication with the USS.
- USS - This update server responds to requests received from the DSS.

17.2.1.1.1 USS Role

An update server in the USS role provides a DSS with the ability to synchronize updates metadata, updates content, target groups, and deployments. The USS provides Web Services for DSS to communicate using SOAP message exchange. The USS can also be configured to provide update content via HTTP download requests.

17.2.1.1.2 DSS Role

DSS uses the same data model as USS. On initialization, a DSS must have some implementation-specific way to:

- Learn the **fully qualified domain name (FQDN)** or IP address and the TCP/IP port of the USS that it is configured to synchronize from.
- Learn its own DNS Name.
- Create a globally unique ID (GUID) to identify itself.

17.2.1.2 Protocol Methods

The Windows Server Update Services: Server-Server Protocol incorporates support for these capabilities:

- Discovery and synchronization of software and driver updates metadata.
- Download of content for the updates synchronized by the DSS from the USS.
- Discovery and synchronization of target groups when the DSS is configured as a replica of the USS.
- Discovery and synchronization of deployments of the updates to target groups.

When the DSS is configured as a replica of the USS, information (about updates, target groups, and deployments) is maintained on each USS, and each DSS obtains this information through a series of SOAP message exchanges that comprise the USS Web Services.

A USS can also be configured to store the content associated with updates. In this configuration, each DSS obtains content from its USS through a series of HTTP operations, which comprise the USS content download.

If the content is not available from the USS, the DSS obtains content from the Microsoft Update (MU) website through a series of HTTP operations using the file location uniform resource locators (URLs) obtained from the USS.

17.2.1.3 Protocol Message Processing Events

The DSS must perform these sequential steps (phases) every time it synchronizes with the USS:

1. Authorization - The DSS provides credentials to the USS in order to obtain a Cookie from the USS during this step.

2. Meta-Data Synchronization - The DSS uses the Cookie from the Authorization Phase and gets information about new updates from the USS during this step.

3. Deployments Synchronization - This step is applicable only if the DSS is configured as a Replica Server. The DSS gets the list of Target Groups and Deployments from the USS during this step.

4. Content Synchronization - The DSS downloads the content files associated with the updates from the USS during this step, which is performed asynchronously.

The following sequence diagram shows the details of the message exchange. Please note that the content download step is carried out asynchronously, but for simplicity it is shown in sequence.

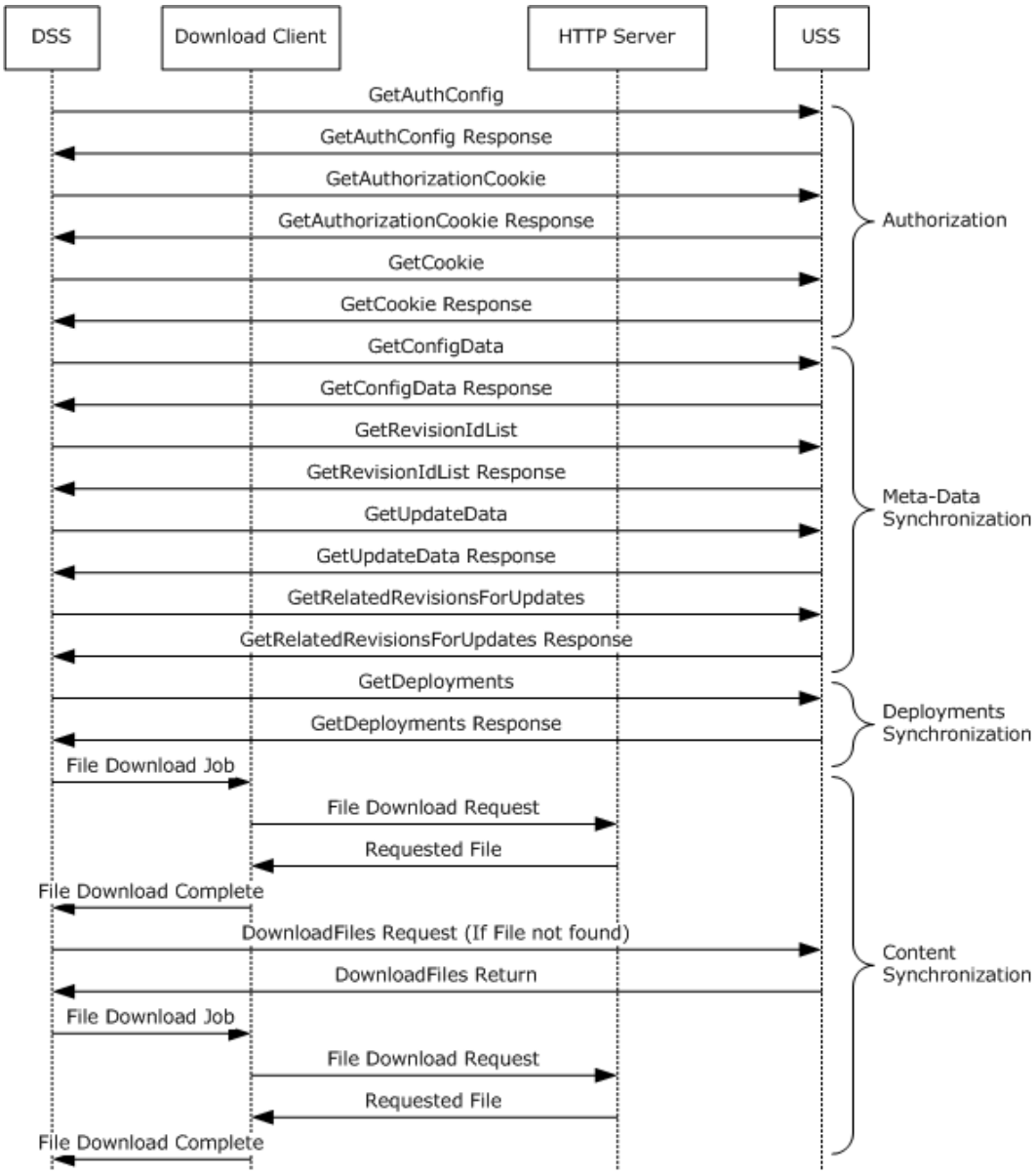

Figure 125: Windows Server Update Services: Server-Server Protocol message exchange diagram

17.2.2 Windows Server Update Services: Client-Server Protocol

The Windows Server Update Services: Client-Server Protocol enables client machines to determine available, applicable software updates, and to download those updates for installation.

The Windows Update Services: Client-Server Protocol is a SOAP-based protocol that uses HTTP 1.1 as its transport. The protocol includes four distinct phases:

1. Self-Update - The client consults the server to determine whether updated executable files are available for the client. If so, the client updates itself to operate using the updated executable files before it continues to communicate with the server.

2. Metadata Synchronization - The client synchronizes update metadata from the update server by calling a sequence of Web service methods. The metadata describes various characteristics of the update including its title, description, rules for determining whether the update is applicable to a computer, and instructions for installing the update content.

   - To reduce network overhead and increase performance, the protocol facilitates the caching of update metadata on clients.

   - To further reduce the amount of update metadata that clients must synchronize, update metadata is divided into fragments. Each client synchronizes only the fragments that it needs. In particular:

     - The client invokes the **SyncUpdates** method, which returns to the client a "core" fragment. This fragment contains sufficient update metadata for a client to evaluate whether the update content is required.

     - If the client determines that update content is required, it then invokes the **GetExtendedUpdateInfo** method to obtain additional metadata fragments.

3. Content Synchronization - The client may request update content, comprised of any files associated with the updates required by the client.

4. Reporting - The client reports events to the server that provide information about its update-related activities (for example, content download succeeded or failed, content install succeeded or failed). Reports are generated asynchronously from the rest of the protocol.

The Windows Server Update Services: Client-Server Protocol specifies how the binaries and metadata are distributed using the Client-Server Communications Protocol. It does not specify the format of the binaries or metadata themselves, but it assumes that the metadata is well-formed Extensible Markup Language (XML) and is compatible with the XPath queries for populating the server data model. In all other respects, the server treats the binaries and metadata as opaque.

Clients must be configured to obtain updates from the server. This configuration may be performed manually or, in managed environments, may be performed using any appropriate form of centralized machine configuration.

17.2.2.1 Messages and Transport

The Windows Update Services: Client-Server Protocol must be carried out over a set of Web services and virtual directories. Each Web service must support SOAP (as specified in [SOAP1.1]) and HTTP (as specified in [RFC2616]), and consists of the following set of Web services and virtual directories.

- Self-update tree - A virtual directory that contains the client "self-update" binaries, exposed at URL

```
http://serverUrl:[commonPort]/SelfUpdate
```

- Content directory - A virtual directory that contains content, exposed at URL

```
http://serverUrl:[contentPort]/Content
```

- SimpleAuth Web service - A Web service that clients consult to obtain cached state for use by servers in restricting availability of updates to groups of clients, exposed at URL

```
http[s]://serverUrl:[commonPort]/SimpleAuthWebService/SimpleAuth.asmx
```

- Windows Update Agent (WUA) - A Web service that synchronizes metadata to the client, exposed at URL

```
http[s]://serverUrl:[commonPort]/ClientWebService/Client.asmx
```

- Reporting Web service - A Web service that clients contact to report selected events that contain information about their update activity, exposed at URL

```
http[s]://serverUrl:[commonPort]/ReportingWebService/ReportingWebService.asmx
```

Each Web service should also support HTTPS for securing its communication with clients.

The following diagram illustrates the messaging sequence:

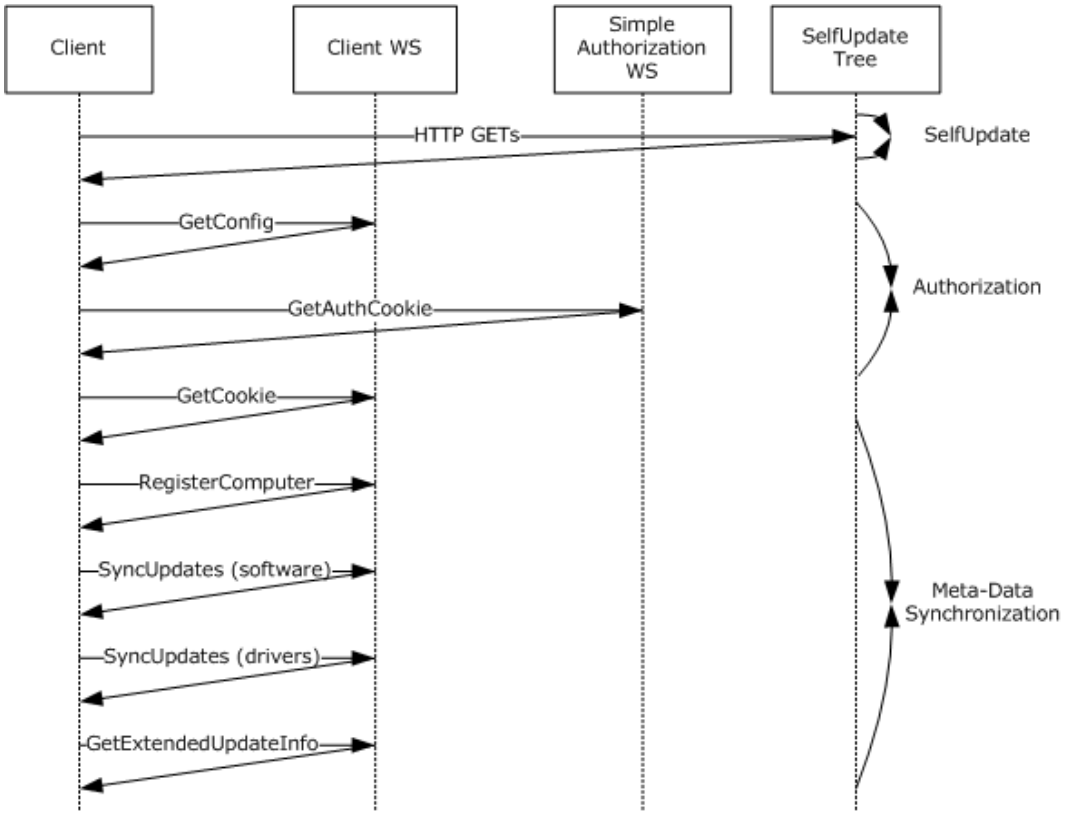

Figure 126: Messaging processing sequence

Self-update, authorization, and metadata sync must always be performed in the sequence illustrated in the diagram, although specific steps in the sequence may be omitted as an optimization.

17.2.3 Background Intelligent Transfer Service (BITS) Peer-Caching: Peer Discovery Protocol

The BITS Peer-Caching: Peer Discovery Protocol is used to locate networked hosts or devices that are implementing the server role of the Background Intelligent Transfer Service (BITS) Peer-Caching: Content Retrieval Protocol as specified in [MS-BPCR]. The BITS Peer-Caching: Peer Discovery Protocol provides a way for peer servers to announce their presence to connected subnets and a way for peer clients to locate servers in connected subnets.

Windows uses the BITS Peer-Caching: Peer Discovery Protocol to implement a distributed peer-to-peer cache of URL content for use by the BITS component. For more information about BITS, see [MSDN-BITS].

The BITS Peer-Caching: Peer Discovery Protocol is a specialization of Web Services Dynamic Discovery (WS-Discovery), as specified in [WS-Discovery], and follows its model for announcing and locating resources. The following figure depicts the message exchange model for WS-Discovery.

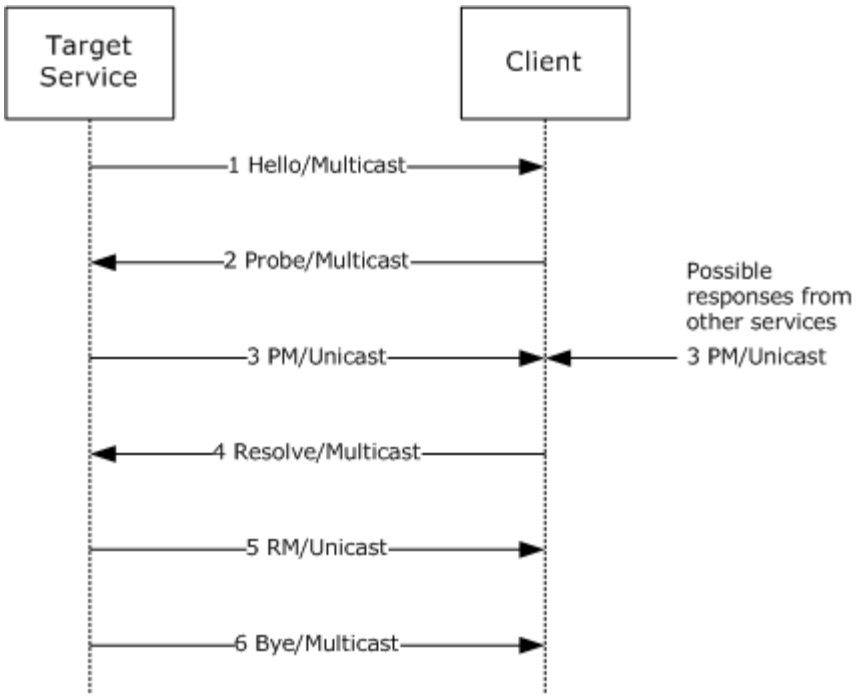

Figure 127: WS-Discovery message exchange model

The protocol defines a client role and a server role. A server announces its presence to connected IP subnets via a multicasted Hello message to UDP port 3702. A client discovers servers passively by listening for Hello messages. A client may also solicit for servers by multicasting a Probe message to the same UDP port; servers with matching characteristics reply to the client with unicast Probe-Match messages, as illustrated by step3 in the above figure.

17.2.4 Background Intelligent Transfer Service (BITS) Peer-Caching: Peer Authentication Protocol Overview

The BITS Peer-Caching: Peer Discovery Protocol, as specified in [MS-BPAU], allows hosts in an Active Directory (AD) domain to exchange self-signed X.509 certificates with enough information to associate those certificates securely with an AD identity. Peer authentication is intended for use by hosts that implement the BITS Peer-Caching: Content Retrieval Protocol.

Peer authentication uses the Kerberos security system for authentication, allowing each host to do the following:

- Verify that the peer is allowed to participate in content retrieval.
- Associate the received certificate with the peer's Kerberos identity in a trustworthy manner. The client and server must reside in Active Directory domains with bidirectional trust in order for Kerberos authentication to succeed.

This protocol is used as part of a distributed peer-to-peer cache of URL content for use by the BITS component. (For more information about BITS, see [MSDN-BITS].) Peer authentication ensures that peer clients and servers are members of the same AD domain.

17.2.5 BITS Peer-Caching: Content Retrieval Protocol Overview

The BITS Peer-Caching: Peer Authentication Protocol, as specified in [MS-BPCR], defines methods for a network client both to query multiple servers for data associated with a given URL and to download that data.

In Microsoft Windows, the BITS component uses the BITS Peer-Caching: Content Retrieval Protocol to implement a distributed peer-to-peer cache of data items based on associated HTTP and HTTPS URLs as well as Universal Naming Convention (UNC) paths. BITS discovers peer servers using the BITS Peer-Caching: Peer Discovery Protocol (for more information, see [MS-BPDP]) and authenticates them using the BITS Peer-Caching: Peer Authentication Protocol (for more information, see [MS-BPAU]). For more information on BITS, see [MSDN-BITS].

The BITS Peer-Caching: Content Retrieval Protocol does not address issues of cache management such as policies for adding and removing content or the method of storing and indexing the content. These issues are internal to the server implementation.

To start, the client uses some other protocol to identify a URL that contains a data image to be downloaded. The client uses another protocol, such as the BITS Peer-Caching: Peer Discovery Protocol, to identify hosts that may be serving some or all of the image data via the BITS Peer-Caching: Content Retrieval Protocol. The client chooses several such hosts and queries them concurrently using the BITS Peer-Caching: Content Retrieval Protocol to see whether one or more can serve the image data. Based on the responses, the client chooses whether to download the image data from the hosts or the original URL.

17.3 Windows Server Update Services Protocols Logical Dependencies and Protocol Stack Views

This section provides several conceptual WSUS areas to describe the logical dependencies among the WSUS protocols and protocol stack views as appropriate.

17.3.1 Windows Server Update Services: Server-Server Protocol Logical Relationships

The Web Services application protocol uses SOAP over HTTP or HTTPS, as specified in [SOAP1.1], for communication. The Content files are downloaded using HTTP 1.1, as specified in [RFC2616].

This protocol is closely related to the Windows Server Update Services Client-Server Protocol, as specified in [MS-WUSP], which is used for communication between Update Servers and Client Computers.

17.3.2 placeholder title

17.3.3 Windows Server Update Services: Client-Server Protocol Logical Relationships

The reporting and metadata synchronization protocols include Web services that use SOAP (as specified in [SOAP1.1]) over HTTP or HTTPS (as specified in [RFC2616]) for communication. The self-update and content synchronization protocols use HTTP 1.1 (as specified in [RFC2616]).

This protocol specification is closely related to the Windows Update Services: Server Protocol, as specified in [MS-WSUSSS], which defines mechanisms for synchronizing updates within a hierarchical configuration of update servers.

17.3.4 Windows Server Update Services: Client-Server Protocol Stack View

The following diagram depicts the Windows Server Update Services: Client Server Protocol stack view:

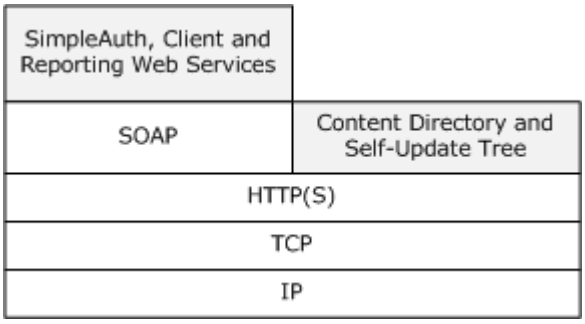

Figure 128: Windows Server Update Services: Client-Server Protocol stack

17.3.5 BITS Peer-Caching: Peer Discovery Protocol Logical Relationships

The BITS Peer-Caching: Peer Discovery Protocol does not authenticate computers to each other; Windows uses the BITS Peer-Caching: Peer Authentication Protocol specification for mutual authentication of potential peers. For more information, see [MS-BPAU].

WS-Discovery (and therefore the BITS Peer-Caching: Peer Discovery protocol) uses SOAP over UDP as its network transport. For more information, see [SOAP-UDP].

A host that implements the client or server role of the BITS Peer-Caching: Peer Discovery protocol typically also implements the same role of the BITS Peer-Caching: Content Retrieval Protocol.

17.3.6 BITS Peer-Caching: Peer Discovery Protocol Stack View

The following diagram depicts the BITS Peer-Caching: Peer Discovery Protocol stack view:

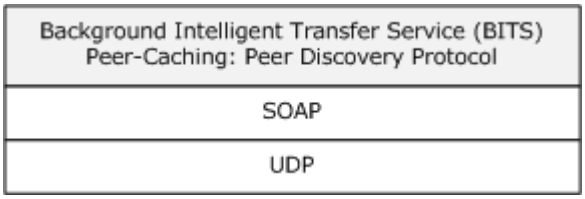

Figure 129: BITS Peer-Caching: Peer Discovery Protocol stack

17.3.7 BITS Peer-Caching: Peer Authentication Protocol Logical Relationships

A host that implements the client or server role of peer authentication typically implements the same role in the BITS Peer-Caching: Content Retrieval Protocol and the BITS Peer-Caching: Peer Discovery Protocol.

Peer authentication depends upon connection-oriented remote procedure call (RPC), as specified in [MS-RPCE], and relies on the message authentication and security features of the Kerberos protocol, as specified in [RFC1510].

This protocol is intended for use by hosts that are members of an AD domain and that use self-signed certificates for authentication during content retrieval. In an environment in which

certificates are issued from a trusted certificate authority, the content-retrieval client and server should be able to authenticate peers without using the BITS Peer-Caching: Peer Authentication Protocol.

17.3.8 BITS Peer-Caching: Peer Authentication Protocol Stack View

The following diagram depicts the BITS Peer-Caching: Peer Authentication Protocol stack view:

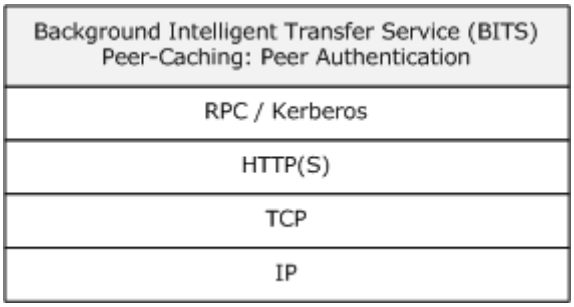

Figure 130: BITS Peer-Caching: Peer Authentication Protocol stack

17.3.9 BITS Peer-Caching: Content Retrieval Protocol Logical Relationships

The BITS Peer-Caching: Content Retrieval Protocol is a client/server protocol using HTTPS over Transport Layer Security (TLS) 1.0 as its transport. A host that implements the client side or server side of this protocol typically also implements the BITS Peer-Caching: Peer Discovery Protocol and the BITS Peer-Caching: Peer Authentication Protocol (for more information, see [MS-BPAU]) to automate the location and authentication of servers.

The consumer of this protocol may be either a top-level application or another client/server protocol.

17.3.10 BITS Peer-Caching: Content Retrieval Protocol Stack View

The following diagram depicts the BITS Peer-Caching: Content Retrieval Protocol stack view:

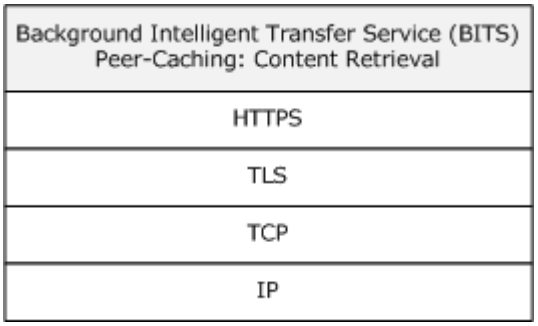

Figure 131: BITS Peer-Caching: protocol stack

The WSUS protocols could be used for a variety of update scenarios. Two scenarios are traced and analyzed in the following section.

17.4 Implementation Scenarios

This section provides two scenarios that illustrate common functionality that can be implemented with the protocols included in the Windows Server Update Server Task. The protocols involved in these scenarios include the Windows Server Update Services: Client-Server Protocol, as specified in [MS-WUSP], together with other general networking protocols used in the course of standard TCP communication between the client and server. The functionality illustrated in the scenarios is as follows:

- Windows Server 2008 operating system retrieves updates from Microsoft Public Update Server.
- Windows Server 2008 deploys updates to Vista Client.

It should be noted that the following scenarios depict processes that occur over TCP and thus assumes various TCP acknowledgment frames throughout the conversation that are not necessarily depicted in the trace details.

17.4.1 Scenario 1 - Windows Server 2008 Retrieves Updates from MS Public Update Server

17.4.1.1 Scenario Overview

In the following scenario, a Windows Server 2008 operating system (DSS) running Windows Server Update Services 3.0 connects to the Microsoft Public Update Server (www.update.microsoft.com.nsatc.net) to obtain the latest software updates.

17.4.1.2 Scenario Configuration

This scenario was implemented with Windows Server 2003 operating system and Windows Server 2008 operating system configured as follows:

Server 2003 Configuration

- Machine: WS200301.contoso.com (WS200301)
- Operating system: Windows Server 2003 R2 operating system with latest updates
- IP Address: 10.0.10.1
- Subnet Mask: 255.255.255.0
- Active Directory domain controller: contoso.com
- NetMon 3.1

Server 2008 Configuration

- Machine: WS200801.contoso.com (WS200801)
- Operating system: Windows Server 2008 Beta 3
- IP Address: 10.0.10.2
- Subnet Mask: 255.255.255.0
- Windows Server Update Services 3.0

Network Topology

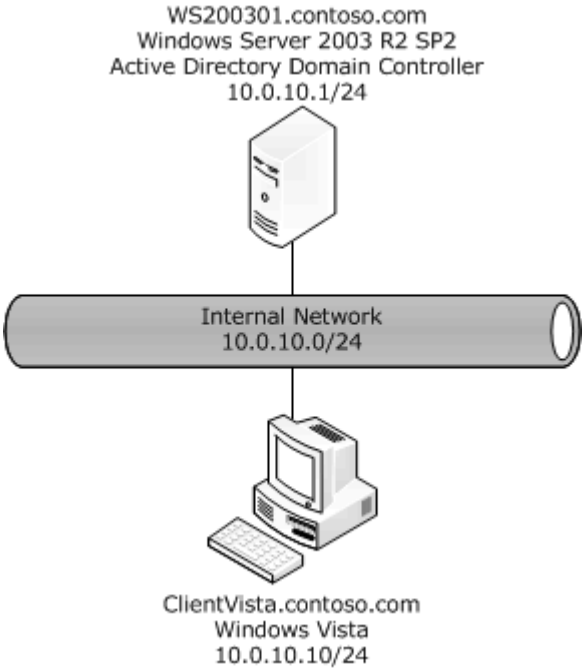

Figure 132: Network topology

17.4.1.3 Setting Up the Trace

1. Log on to all machines as CONTOSO\Administrator.
2. Turn off the firewall on all machines.
3. On WS200301, start NetMon 3.1. It should be noted that the WS200301 machine is used to capture the trace, but no packets from this machine are described in the trace analysis.
4. BEGIN TRACE.
    1. On ClientVista, press CTRL+ALT+DEL.
    2. Log on as CONTOSO\Administrator.
5. END TRACE.

17.4.1.4 Netmon Trace Digest

- The DSS connects to the USS and initiates an SSL session:

```
W200801 www.update.microsoft.com.nsatc.net TCP TCP:
Flags=.S......,
SrcPort=49249, DstPort=HTTPS(443), Len=0, Seq=3664202939, Ack=0,
Win=8192 (scale factor 8) = 2097152

W200801 www.update.microsoft.com.nsatc.net SSL SSL: Client Hello.
```

```
www.update.microsoft.com.nsatc.net W200801 SSL SSL: Server Hello.
 Certificate.

W200801 www.update.microsoft.com.nsatc.net SSL SSL: Client Key
 Exchange.
Change Cipher Spec. Encrypted Handshake Message.

www.update.microsoft.com.nsatc.net W200801 SSL SSL: Change Cipher
 Spec.
Encrypted Handshake Message.
```

- All application data is downloaded to the DSS from the USS using SSL:

```
www.update.microsoft.com.nsatc.net W200801 SSL SSL:
Application Data.
```

17.4.2 Scenario 2 - Windows 2008 Server Deploys Updates to Vista Client

17.4.2.1 Scenario Overview

In the following scenario, Windows updates from Windows Server 2003 operating system running WSUS 3.0 deploy software updates to a Windows Vista operating system client through a synchronization process.

17.4.2.2 Scenario Configuration

This scenario was implemented with a Windows Server 2003 operating system, a Windows Server 2008 operating system and a Windows Vista operating system client configured as follows:

Server 2003 Configuration

- Machine: WS200301.contoso.com (WS200301)
- Operating system: Windows Server 2003 R2 operating system with latest updates
- IP Address: 10.0.10.1
- Subnet Mask: 255.255.255.0
- Active Directory domain controller: contoso.com
- NetMon 3.1

Server 2008 Configuration

- Machine: WS200801.contoso.com (WS200801)
- Operating system: Windows Server 2008 Beta 3
- IP Address: 10.0.10.2
- Subnet Mask: 255.255.255.0
- Windows Server Update Services 3.0

Client Configuration

- Machine: Windows Vista (ClientVista)
- Operating system:Windows Vista Ultimate with latest updates
- IP Address: 10.0.10.10
- Subnet Mask: 255.255.255.0
- DNS - 10.0.10.1
- Domain member: contoso.com

Network Topology

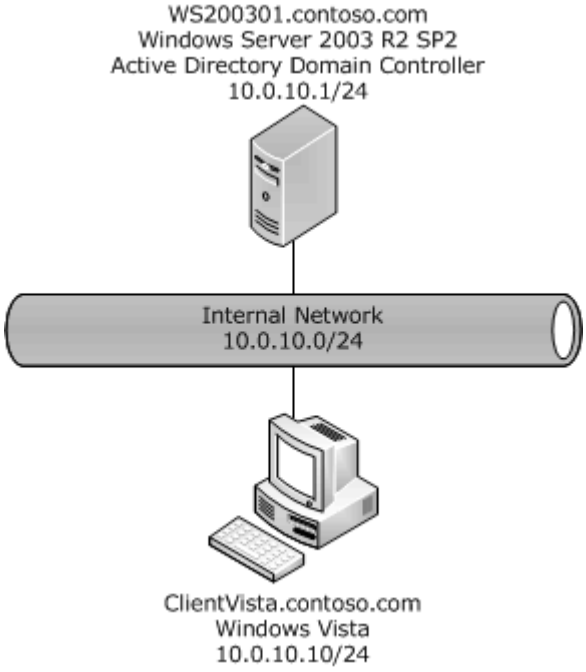

Figure 133: Network topology

17.4.2.3 Setting Up the Trace

1. Log on to all machines as CONTOSO\Administrator.
2. Turn off the firewall on all machines.
3. On WS200301, start NetMon 3.1.
4. BEGIN TRACE.
    1. On ClientVista, click Start.
    2. Log on as CONTOSO\Administrator.
5. END TRACE.

17.4.2.4 Netmon Trace Digest

- The session begins with the client and the server exchanging Address Resolution Protocol (ARP) requests and responses. The server requests a MAC (hardware) address for the client, and the client responds with its MAC address.

```
W200801 CLIENTVISTA ARP ARP: Request, W200801 asks for
CLIENTVISTA

CLIENTVISTA W200801 ARP ARP: Response, CLIENTVISTA at
00-03-FF-90-4E-1A
```

- The ReportEventBatch method is invoked by the client to report the occurrence of one or more software update-related events. The client also issues a SOAP request to the server:

```
CLIENTVISTA W200801 WinUpdV5 WinUpdV5: POST Request, Report Event
 Batch

CLIENTVISTA W200801 WinUpdV5 WinUpdV5: <soap:Envelope Request

g3oQeRfmw5Fwh2SOob2DIOgaMdRwWhl9dxHiwo/OCXO/2mlBT3yElFU+F/STlmdzw2CUe
mIwt6gPmYLYn7TfFqdQrsYxI4ujCWAzbn4lGhVV/ajqgNv0ZQx72zh3jgguD06ickuX6+
CwemrmgEHUSHTsUtzRMhDa92dsulT8Pwin+ESuITxSoqjT6UXeUDoVUCd38kNJGA==
</EncryptedData></cookie><parameters><ExpressQuery>
```

- The server response contains an embedded XML payload:

```
W200801 CLIENTVISTA TCP TCP: Flags=....A..., SrcPort=HTTP(80),
DstPort=49203, Len=0, Seq=3739279279, Ack=1389086158, Win=256
(scale factor 8) = 65536

W200801 CLIENTVISTA WinUpdV5 WinUpdV5: Response, Status Code=200,
  text/xml; charset=utf-8, Length= 352

<DataField Name = 'XmlPayload'>
<StartOffset>0x11c</StartOffset>
<EndOffset>0x27c</EndOffset>
</DataField>
<EndOffset>0x27c</EndOffset>
</DataType>
<EndOffset>0x27c</EndOffset>
</DataField>
<EndOffset>0x27c</EndOffset>
```

- The Sync Updates is the main operation that supports the synchronization of update metadata to client computers. This operation is invoked to perform both software and driver metadata synchronization. Software update synchronization MUST be performed first, using a sequence of calls to this method. Driver synchronization is performed using a single call to the method.

  If no faults occur during the operation, the server returns a SyncUpdatesResponse message to the client.

```
CLIENTVISTA W200801 WinUpdV5 WinUpdV5: POST Request, Sync Updates
```

- The server issues a response with information about the Product Code and other details about the MSP (Windows Installer Patch):

```
W200801 CLIENTVISTA WinUpdV5 WinUpdV5: Response, Status Code=/>
<m.MsiProductInstalled

Reason: ProductCode="{9019040E-6000-11D3-8CFE-0150048383C9}" />
<m.MsiProductInstalled
ProductCode="{9119040E-6000-11D3-8CFE-0150048383C9}" />
<m.MsiProductInstalled
ProductCode="{901A040E-6000-11D3-8CFE-0150048383C9}" />
<m.MsiProductInstalled
```

18 Transaction Processing Services Protocols

This section provides a technical overview of the protocols that provide transaction processing functionality in Windows. These protocols are collectively referred to as the Transaction Processing Services protocols.

18.1 Transaction Processing Services Overview

The Transaction Processing Services protocols are used to communicate with a transaction manager to perform distributed transaction coordination, driving the transaction life cycle that as described in the [MS-DTCO] section titled Transaction Lifetime.

The remainder of this section is structured as follows:

- Section 18.1.1 provides definitions of terms that are used throughout the section.
- Section 18.1.2 provides a list of references describing basic transaction concepts.
- Section 18.1.3 provides a brief description of basic transaction concepts.
- Section 18.1.4 provides a list of the Transaction Processing Services protocols and a brief description of each, with references to the protocol specifications that specify each protocol in detail.
- Section 18.2 provides a conceptual framework for Transaction Processing Services and a mapping of the protocols that are specified by MSDTC Connection Manager: OleTx Transaction Protocol and MSDTC Connection Manager: OleTx Management Protocol to this framework. This mapping provides a baseline realization of the conceptual framework that is defined in this document.
- Sections 18.3, 18.4, and 18.5 describe the remaining transaction coordination protocols and map each protocol to the conceptual framework that is described in section 18.2, describing how they vary the baseline realization of that framework.
- Sections 18.6 and 18.7 describe the dependencies among the Transaction Processing Services protocols.
- Section 18.8 provides an example that illustrates how these protocols work together to execute a single distributed transaction.

18.1.1 Definitions

The following terms are defined in [MS-GLOS]:

Abort Request
ACID
Application
Commit Request
Recovery
Subordinate Transaction Manager
Superior Transaction Manager
Transaction
Transaction Identifier
Transaction Manager
Transaction Propagation
Two-Phase Commit

The following terms are specific to this document:

Abort Outcome: One of the **outcomes** of an **atomic transaction**. The **abort outcome** indicates that the **work** performed during the lifetime of the **transaction** is discarded after the **transaction** completes. An **abort outcome** is reached when at least one **transaction participant** does not agree to commit the **transaction**.

Begin request: The action that is performed by a **root application** in order to create a new **atomic transaction**.

Commit Outcome: One of the **outcomes** of an **atomic transaction**. The **commit outcome** indicates that the **work** performed during the lifetime of the **transaction** will be retained after the **transaction** has completed, as specified by the **ACID** properties. A **commit outcome** is reached when all **transaction participants** agree to commit the **transaction**.

Enlistment: The relationship between a **participant** and a **transaction manager** in an **atomic transaction**. The term typically refers to the relationship between a **resource manager** and its **transaction manager**, or between a Subordinate **transaction manager facet** and its Superior **transaction manager facet**.

Facet: A subsystem in a **transaction manager** that maintains its own per-**transaction** state and responds to intra-**transaction manager** events from other **facets**. A **facet** can also be responsible for communicating with other **participants** of a **transaction**.

local LU: An LU 6.2 Implementation (see the LU 6.2 Implementation Details topic in [MS-DTCLU]) that communicates with a **transaction manager**.

Log: A durable store used to maintain **transaction** state.

Logical Unit (LU): In **Systems Network Architecture**, a **logical unit** is an addressable network element which serves as an access point to the network for programs and users, allowing them to access resources and communicate with other programs and users. For a more complete definition, see [SNA].

LU Name Pair: An identifier that uniquely specifies the pairing of a **local LU** and a **remote LU**.

Outcome: One of the three possible results (Commit, Abort, In Doubt) reachable at the end of a life cycle for an **atomic transaction**.

Participant: Any of the parties that are involved in an **atomic transaction** and that have a stake in the operations that are performed under the **transaction** or in the **outcome** of the **transaction**.

Push Propagation: A process where the **transaction manager** coordinating a **transaction** initiates communications with a remote **transaction manager** to enlist it in the **transaction**.

Pull Propagation: A process where the remote **transaction manager** initiates communications with the **transaction manager** coordinating a **transaction** to enlist itself in the **transaction**.

Remote LU: An LU 6.2 Implementation (see the LU 6.2 Implementation Details topic in [MS-DTCLU]) that communicates with the **local LU**.

Resource: A logical entity or unit of data whose state changes in accordance with the **outcome** of an **atomic transaction**. **Resources** are either durable or volatile.

resource manager (RM): The **participant** that is responsible for coordinating the state of a **resource** with the **outcome** of **atomic transactions**. For a specified **transaction**, a

resource manager enlists with exactly one **transaction manager** to vote on that **transaction** **outcome** and to obtain the final **outcome**. A **resource manager** is either durable or volatile, depending on its **resource**.

Rollback: Synonymous with abort.

Root Application: The **application** that is responsible for beginning and completing an **atomic transaction**. The **root application** communicates with a **root transaction manager** in order to begin and complete **transactions**.

Root Transaction Manager: The specific **transaction manager** that processes both the **Begin Request** and the **Commit Request** for a specified **transaction**. A specified **transaction** has exactly one **root transaction manager**.

Subordinate Participant: A role that is taken by a **participant** that is responsible for voting on the **outcome** of an **atomic transaction**. For a specified **transaction**, the set of **subordinate participants** is the set of all **resource managers** and the set of all **subordinate transaction managers**.

Systems Network Architecture (SNA): A data communications architecture defined by IBM. For a technical overview, see [SNA].

work: The set of state changes that are applied to **resources** inside an **atomic transaction**.

18.1.2 References

[GRAY] Gray, J. and Reuter, A., "Transaction Processing: Concepts and Techniques", San Mateo, CA: Morgan Kaufmann Publishers, 1993, ISBN: 1558601902.

[MS-DTCLU] Microsoft Corporation, "MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension".

[MS-DTCO] Microsoft Corporation, "MSDTC Connection Manager: OleTx Transaction Protocol".

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

18.1.3 Transaction Concepts

This section provides a high-level introduction to the concepts discussed in this section. For more detailed and precise descriptions, see [GRAY], the [MS-DTCO] section titled Overview, and the linked glossary entries in section 18.1.1.

18.1.3.1 Basics

Very broadly, a **transaction** is an activity that makes changes to the state of some set of **resources** so that either all the changes happen or none of the changes happen. Resources may be data, such as rows in a database, or logical entities, such as the execution state of a program. Resources changed by a transaction may be distributed across multiple systems.

Achieving this under all expected and unexpected conditions is a difficult problem with a well-established solution, described in [GRAY]. The basis of the solution is to factor the execution of the transaction into three elements (**application**, **transaction manager** and **resource manager**) that communicate with each other by using a well-defined protocol called the **Two-Phase Commit** Protocol. These elements are the **participants** in the transaction. The transaction manager and resource manager are usually provided as part of an operating system or other platform elements, such as databases, leaving most programmers with only the application itself to write.

Resource managers represent the resources involved in the transaction. A transaction manager coordinates the transaction, keeping all of the participants in step. All the changes to the resources involved in a transaction are made by applications via implementation-specific protocols outside the scope of the Two-Phase Commit Protocol. Exactly one of the applications involved in the transaction initiates and completes the transaction, through communications with its transaction manager. This application is known as the **root application**. Other participants are added to the transaction progressively. Each participant is said to be enlisted on the transaction.

The protocols involved in the Microsoft implementation of this solution are the subject of this Overview.

18.1.3.2 Transaction Trees

Multiple transaction managers and resource managers may participate in a transaction. Their individual responsibilities in the Two-Phase Commit Protocol are defined by a transaction tree, shown in the following figure.

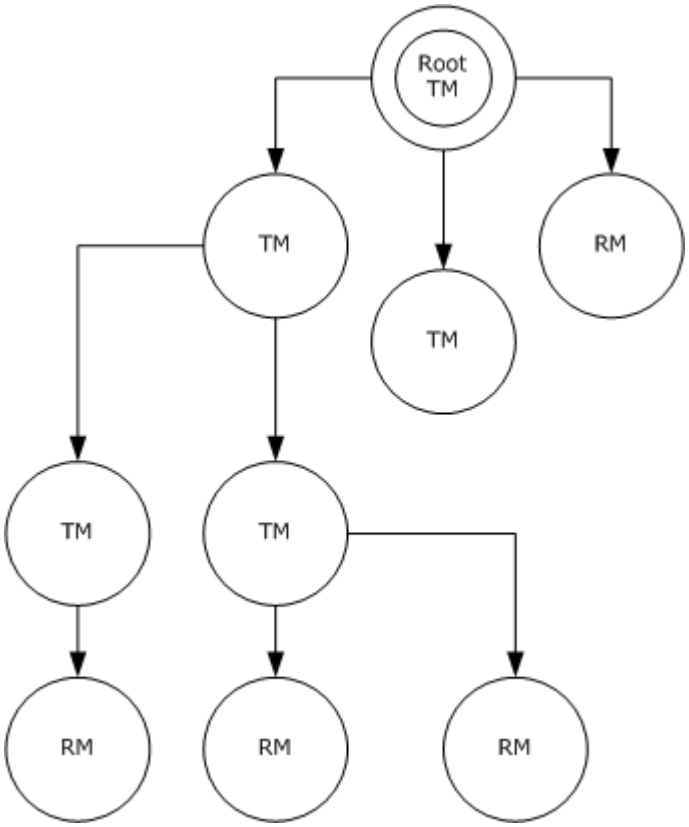

Figure 134: Transaction tree

The transaction manager at the root of the tree is the **root transaction manager**. This is the transaction manager with which the root application communicates. Any participant that enlists with a transaction manager is called a **subordinate participant**. Each transaction manager is a **superior transaction manager** if any of its subordinate participants are transaction managers. All transaction managers in the tree apart from the root transaction manager are **subordinate transaction managers**.

18.1.3.3 The Two-Phase Commit Protocol

The Two-Phase Commit Protocol described in [GRAY] defines Phase One and Phase Two. These phases might be informally described as "are you ready", and "go / no go", respectively.

Phase One begins when all the required changes of resource state have been made, and the root application asks the transaction manager to complete it. Phase One ends when the transaction manager has decided the outcome of the transaction; that is, whether all the changes should be made, or whether none of them should be made.

When the root application asks the root transaction manager to complete the transaction, it may make a **commit request**, asking that all the changes should be made, or an **abort request**, asking that none of the changes should be made. A commit request causes the root transaction manager to ask each of the subordinate participants involved in the transaction whether they are prepared to commit the changes made. This is called voting on the transaction **outcome**. Each subordinate participant must vote Read-Only, Prepared, or Aborted. Read-Only and Prepared are positive votes. Aborted is a negative vote. If any subordinate participant votes negatively, the root transaction manager decides that the final outcome of the transaction as a whole is to make no changes, called the **Abort Outcome**. If every subordinate participant votes positively, then the root transaction manager decides that the final outcome of the transaction as a whole is to make all the changes, called the **Commit Outcome**. An abort request causes the root transaction manager to notify each subordinate participant to make no changes and to notify each of their respective subordinate participants, if any, to abort the transaction.

A subordinate transaction manager determines its vote by aggregating the votes of its subordinate participants. If a subordinate transaction manager has no subordinate participants or if all of its subordinate participants vote Read-Only, then the subordinate transaction manager votes Read-Only. If at least one subordinate participant votes Prepared and after collecting all votes no subordinate participant votes Aborted, then the subordinate transaction manager votes Prepared. In all other cases, the subordinate transaction manager votes Aborted, in which case it must also notify any of its subordinate participants that had voted Prepared that the transaction has aborted.

Phase Two begins after the root transaction manager decides the outcome of the transaction. In this phase, each subordinate participant that voted Prepared is sent either a request to commit the changes if the outcome was the Commit Outcome, or a request to undo the changes (called a **rollback**) if the outcome was the Abort Outcome. The root transaction manager also sends the outcome of the transaction to the root application. A subordinate participant that voted Read-Only is not notified of the outcome of the transaction—a resource manager might vote Read-Only if it made no changes as part of the transaction. A subordinate participant that voted Abort is also not notified of outcome of the transaction. Phase Two ends when all transaction outcome notifications have been completed.

This is the Two-Phase Commit Protocol described in [GRAY]. The protocol specified by [MS-DTCO] adds an additional phase, Phase Zero. Phase Zero can be thought of as prefixed to Phase One. It begins when the root application requests completion of the transaction, and ends when all Phase Zero participants have voted that the phase is complete is complete, after which Phase One proceeds as described above. The value of the additional phase is that during Phase Zero, new participants can be enlisted on the transaction. In the protocol described in [GRAY], the set of participants is fixed from the moment that Phase One begins. Phase Zero is a very useful extension in some scenarios. For example, a caching resource manager can be placed between an application and a database resource manager such that all requested changes are held in memory, until the caching resource manager receives a request from the transaction manager to exit Phase Zero. Only then is the database resource manager itself enlisted on the transaction and the changes made to the durable store, yielding potentially significant performance gains.

18.1.3.4 Error Detection and Recovery

Transactions must either commit or abort under all circumstances, including failures in distributed communications, applications, resource managers, and transaction managers.

The strategy is to detect failure positively where possible, or through a timeout as a backstop. Recovery from errors, by default, involves assuming that any outstanding transactional work SHOULD be aborted.

For example, a resource manager that loses contact with the transaction manager coordinating a transaction, or recovers after a crash, will roll back any uncommitted changes.

18.1.4 Protocols List

The Transaction Processing Services protocols include two sets of protocols. The core protocols provide the distributed transaction coordination functionality. The communication protocols provide the underlying communications functionality that is required for an implementation of the core protocols.

The following tables list the core and communication protocols, and give brief descriptions of them.

Table 1: Transaction Processing Services core protocols

Protocol name	Protocol description	Document short name
MSDTC Connection Manager: OleTx Transaction Protocol Specification	Enables the creation of, initiation of, distributed propagation of, and participation in transactions using the OleTx Transaction Protocol over the protocol specified by MSDTC Connection Manager: OleTx Multiplexing Protocol.	[MS-DTCO]
MSDTC Connection Manager: OleTx Transaction Internet Protocol Specification	Enables the initiation of distributed transaction propagation via the TIP protocol. This protocol uses the protocol that is specified by MSDTC Connection Manager: OleTx Multiplexing Protocol to communicate.	[MS-DTCM]
Transaction Internet Protocol (TIP) Extensions	Enables distributed propagation of transactions by using the TIP protocol over TCP.	[MS-TIPP]
MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension	Enables an implementation of Logical Unit type 6.2 as defined by the IBM System Network Architecture (SNA) to participate in transactions coordinated by a transaction manager that does not implement SNA protocols. This protocol uses the protocol specified by MSDTC Connection Manager: OleTx Multiplexing Protocol for communications with the transaction manager.	[MS-DTCLU]
MSDTC Connection Manager: OleTx Management Protocol Specification	Enables management tools to obtain a list of transactions being processed by a transaction manager using the protocol specified by MSDTC Connection Manager: OleTx Multiplexing Protocol. Enables the changing of settings that are used by other Transaction Processing Services protocols using the (Remote Registry) protocol.	[MS-CMOM]

Table 2: Transaction Processing Services communication protocols

Protocol name	Protocol description	Document short name
MSDTC Connection Manager: OleTx Transports Protocol Specification	Provides negotiation of connections, and sending of variable length data over RPC for the MSDTC Connection Manager Protocols.	[MS-CMPO]
MSDTC Connection Manager: OleTx Multiplexing Protocol Specification	Enables multiplexing of multiple protocol messages in a single, MSDTC Connection Manager: OleTx Transports Protocol message. Allows multiplexing multiple logical protocol connections through a single MSDTC Connection Manager: OleTx Transports Protocol connection.	[MS-CMP]
Remote Procedure Call Protocol Extensions	Provides the ability to invoke methods on remote processes.	[MS-RPCE]

The dependencies between these protocols are detailed in sections 18.6 and 18.7.

18.2 Transaction Processing Services Protocol Concepts

The conceptual framework for the Transaction Processing Services protocols is defined in terms of the roles that are involved in the transaction life cycle that is specified in the [MS-DTCO] section titled Transaction Lifetime and described in section 18.1.3. An implementation of the transaction life cycle involves the following roles:

Application: A client application that wants to perform transacted work on a number of Resource Managers. The application creates a transaction, and thus, only it has the right to commit the transaction.

Application Service: A service that accepts requests to perform transacted work on local Resource Managers. An Application Service does not have the right to commit transactions.

Transaction Manager: A service that coordinates the lifetime of transactions, providing functionality for Resource Managers to enlist in these transactions, and provides functionality to enlist in transactions that are coordinated by remote Transaction Managers.

Resource Manager: A participant that is responsible for coordinating the state of a resource with the outcome of atomic transactions. For a specified transaction, a resource manager enlists with exactly one transaction manager to vote on that transaction outcome and to obtain the final outcome.

Management Tool: An application that monitors the health of a Transaction Manager and configures settings related to transaction coordination.

The communication between these roles is depicted in the following figures.

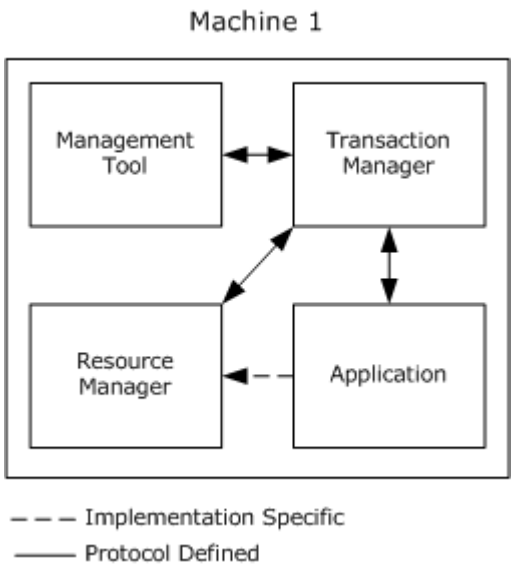

Figure 135: Basic communication between the roles defined in the transaction life cycle

In the most basic scenario, an application wants to perform work on a local Resource Manager. There is no propagation necessary, because the Resource Manager and application share a common local Transaction Manager.

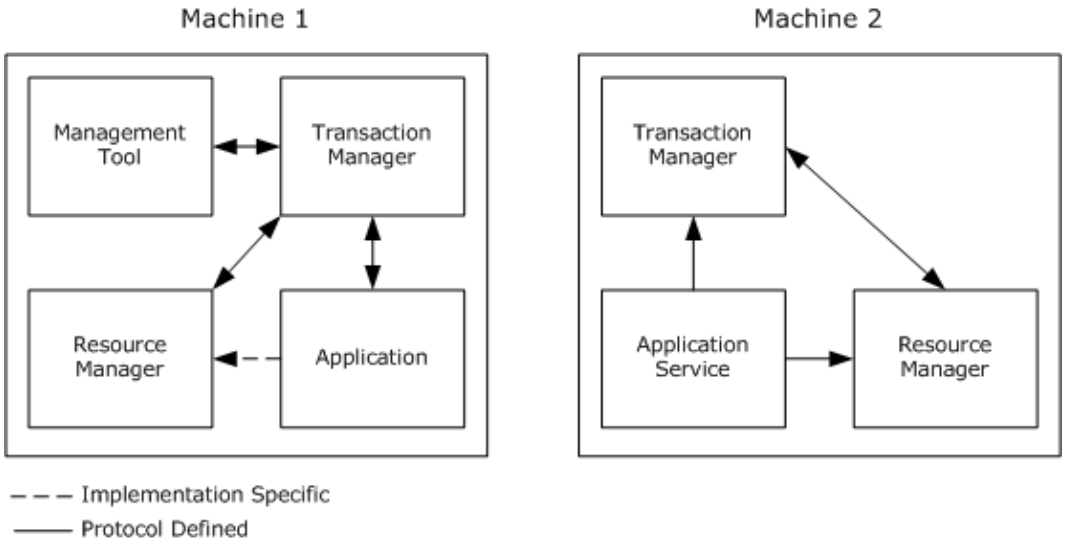

Figure 136: Distributed communication between the roles defined in the transaction life cycle

In the distributed scenario, an application wants to perform work on a local Resource Manager and a remote Resource Manager. It is necessary for the transaction to be propagated from the application's local Transaction Manager to the remote Resource Manager's Transaction Manager.

Each core protocol that is listed in section 18.6 is used to support some part of the communications between the roles that are shown in the previous figures. The protocol that is most complete in this regard is the protocol that is specified by MSDTC Connection Manager: OleTx Transaction Protocol,

which supports all the communications between the roles shown except between the Management Tool and the Transaction Manager, between the application and the Application Service, between the application and the Resource Manager, and between the Application Service and the Resource Manager.

The communication between the Application and Application Service, the application and Resource Manager, and the Application Service and Resource Manager is implementation specific. The expectation is that this will consist of a request for work to be done, along with all information that is necessary to enlist in the transaction, including the transaction identifier.

Except where noted in section 18.6, the protocols that are used for communication between these roles depend on the protocol that is specified by MSDTC Connection Manager: OleTx Multiplexing Protocol. This protocol supports both multiplexing multiple logical sessions over a single MSDTC Connection Manager: OleTx Transports Protocol session, and multiplexing multiple protocol messages into a single MSDTC Connection Manager: OleTx Transports Protocol message. The protocol that is specified by MSDTC Connection Manager: OleTx Transports Protocol in turn depends on RPC.

The abstract state machine that drives the transaction life cycle that is described in th [MS-DTCO] section titled Transaction Lifetime is defined only in [MS-DTCO]. An implementation of this state machine is necessary for any implementation of a Transaction Manager, and thus any implementation of MSDTC Connection Manager: OleTx Transaction Internet Protocol, Transaction Internet Protocol (TIP) Extensions, MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension, or MSDTC Connection Manager: OleTx Management Protocol requires an MSDTC Connection Manager: OleTx Transaction Protocol implementation.

A common concept shared by all the roles is the transaction identifier that is used by the Transaction Manager to identify transactions in its Transaction Table. This data is used by all protocols to refer to a transaction, and to associate actions that are performed on the Transaction Manager with that transaction.

18.2.1 Role Descriptions

This section describes each role in terms of the conceptual actions in the transaction life cycle that it performs. Therefore, an action may represent the multiple roundtrip communications that are specified in a protocol.

The following role descriptions map the protocols that are specified by the MSDTC Connection Manager: OleTx Transaction Protocol to the communications between roles; however, the MSDTC Connection Manager: OleTx Management Protocol is mapped to the communications between the Management Tool role and the Transaction Manager role.

18.2.1.1 Application

The application role is performed by a client application that wants to perform transacted work on a number of resource managers. To perform this function, it takes responsibility for the initial creation of a transaction at the start of the transaction life cycle, as defined in the [MS-DTCO] section titled Transaction Lifetime. It performs the following actions:

- **Creates a Transaction**: Requests that the local transaction manager create a transaction and provide the application with the transaction identifier of the created transaction.
- **Requests a Push Transaction**: Requests that the local transaction manager perform a Push Transaction action to propagate the transaction associated with the transaction identifier to a remote transaction manager.

- **Completes the Transaction**: Indicates to the transaction manager that it can commit the transaction using the Two-Phase Commit protocol.
- **Performs Transacted Work**:
  - Serializes the information necessary to identify the transaction for transport, including the transaction identifier.
  - Sends the serialized information, along with the work request information, to a resource manager or an application service.

The Creates a Transaction, Requests a Push Transaction, and Completes a Transaction actions map to the application role of the protocol specified in the MSDTC Connection Manager: OleTx Transaction Protocol. The Performs Transacted Work action is implementation-specific. The transaction manager's communications with the application role are defined by the application facet specified in the MSDTC Connection Manager: OleTx Transaction Protocol.

The Request Push Transaction action of an application using the protocol specified in [MS-DTCO] requests that the local transaction manager perform a Push Transaction action that uses the transaction manager-to-transaction manager communication protocol specified in [MS-DTCO].

18.2.1.2 Application Service

The application service role represents an application that accepts requests to perform transacted work on local resource managers. This is to allow an Application to perform work on remote resource managers. It performs the following actions:

- **Requests a Pull Transaction**: Requests that the local transaction manager perform a Pull Transaction action to propagate the transaction associated with the transaction identifier from a remote transaction manager.
- **Requests a Push Transaction**: Requests that the local transaction manager perform a Push Transaction action to propagate the transaction associated with the transaction identifier to a remote transaction manager.
- **Deserializes a Transaction**: Requests that the local transaction manager provide the transaction identifier of the transaction associated with the serialized information if it is present in the Transaction Table, or return an error if it is not.
- **Performs Transacted Work**:
  - Serializes the information necessary to identify the transaction for transport, including the transaction identifier.
  - Sends the serialized information, along with the work request information, to a resource manager or an application service.

The Requests a Pull Transaction, Requests a Push Transaction, and Deserializes a Transaction actions map to the application role of the protocol specified in [MS-DTCO]. The Performs Transacted Work action is implementation-specific. The transaction manager's communications with the application service role are defined by the application facet specified in the MSDTC Connection Manager: OleTx Transaction Protocol.

The Request Pull Transaction action of an application service using the protocol specified in [MS-DTCO] requests that the local transaction manager initiate a Pull Transaction action that uses the transaction manager-to-transaction manager communication protocol specified in [MS-DTCO].

The Request Push Transaction action of an application service using the protocol specified in [MS-DTCO] requests that the local transaction manager initiate a Push Transaction action that uses the transaction manager-to-transaction manager communication protocol specified in [MS-DTCO].

18.2.1.3 Transaction Manager

The transaction manager's role is to coordinate the lifetime of transactions and to provide functionality to enlist in transactions coordinated by a remote transaction manager. This allows applications to enlist local or remote resource managers in transactions coordinated by their local transaction managers. It maintains a Transaction Table of all currently incomplete transactions, as defined in section 18.2. It communicates with other transaction managers to perform the following transaction propagation actions:

- **Push Transaction**: Requests that the remote transaction manager add the transaction to its Transaction Table and enlist in the transaction. This allows a subsequent Deserialize Transaction action by an application service local to the remote transaction manager to retrieve the propagated transaction's transaction identifier.
- **Pull Transaction**: Requests that the remote transaction manager enlist the transaction manager in the transaction. If the request is successful, the transaction is added to the transaction manager's Transaction Table so that a subsequent Deserialize Transaction action by a local application service will retrieve the propagated transaction's transaction identifier. If the request is not successful, an error is returned.

When performing a Push Transaction, this role maps to the superior transaction manager facet, as specified in [MS-DTCO], and the receiving transaction manager maps to the subordinate transaction manager facet, as specified in [MS-DTCO].

When performing a Pull Transaction, this role maps to the subordinate transaction manager facet, as specified in [MS-DTCO], and the receiving transaction manager maps to the superior transaction manager facet, as specified in [MS-DTCO].

When a transaction manager receives a transaction, it is enlisted in the transaction, as described in the Two-Phase Commit protocol and receives notifications, coordinates the outcome with its enlistments, and votes on the outcome of the transaction. When the protocol specified by the MSDTC Connection Manager: OleTx Transaction Protocol is used to propagate a transaction, the subsequent Two-Phase Commit communication occurs through the protocol that was used to propagate the transaction.

18.2.1.4 Resource Manager

The resource manager role represents a resource that can accept requests to perform transacted work. It does this to ensure consistency between its resources and other resources enlisted in a transaction. The canonical example of a resource manager is a database. It communicates with the local transaction manager to perform the following actions:

- **Register**: Registers the resource manager with the local transaction manager. This is necessary to ensure durability, because when recovering from a transient failure the local transaction manager must be able to identify the resource manager.
- **Enlist**: Requests that the transaction manager register the resource manager to participate in the transaction using the Two-Phase Commit protocol.

This role maps to the resource manager role of the protocol, as specified in [MS-DTCO]. The transaction manager's communications with this role are defined by the resource manager facet, as specified in [MS-DTCO].

18.2.1.5 Management Tool

The management tool role represents an application that wants to monitor the health of a transaction manager and configure settings related to transaction coordination. It communicates with a transaction manager to perform the following actions:

- **Subscribe to Transaction Information**: The transaction manager periodically provides a list of all transactions in its Transaction Table that are not currently complete, along with information about the state of the transactions returned.
- **Configure Settings**: Through the use of the protocols specified by the Windows Remote Registry Protocol and the Service Control Manager Remote Protocol, the management tool can configure the transaction manager's settings.

This role maps to the management client role specified in [MS-CMOM]. The transaction manager's communications with this role are defined by the management server role specified in [MS-CMOM].

18.2.2 Example

The following figure illustrates the interaction between the transaction life-cycle roles:

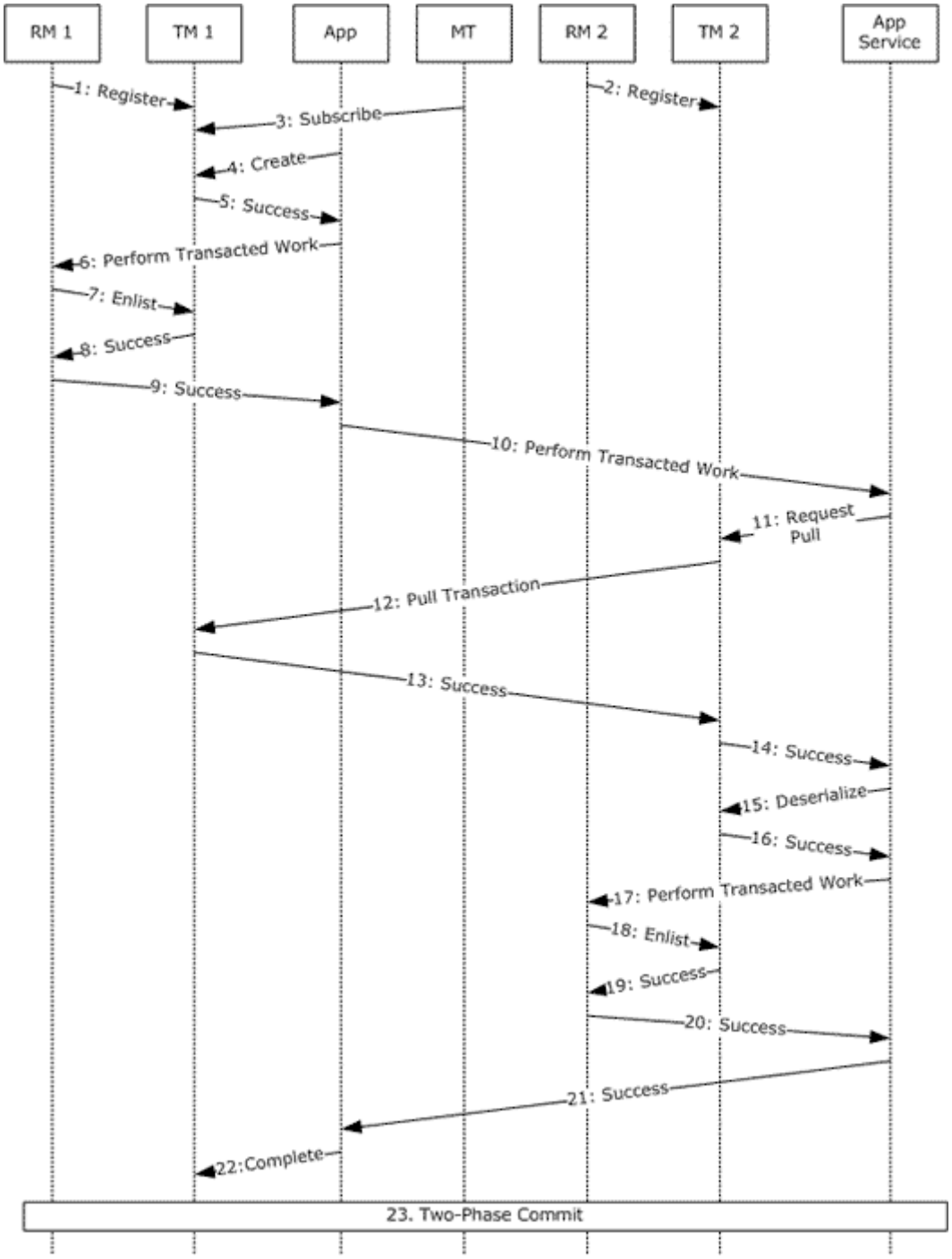

Figure 137: Example communication between the roles defined in the transaction life cycle

1. Resource manager 1 performs a Register action against transaction manager 1.

2. Resource manager 2 performs a Register action against transaction manager 2.

3. The management tool performs a Subscribe for transaction information action against transaction manager 1 to monitor the progress of the Two-Phase Commit protocol and to resolve the transaction if it reaches an error state.

4. The application initiates a Create transaction action against transaction manager 1.

5. Transaction manager 1 creates the transaction and returns the transaction identifier to application, completing the Create transaction action initiated in Step 4.

6. The application initiates a Perform Transacted Work action against resource manager 1.

7. Resource manager 1 initiates an Enlist action against transaction manager 1.

8. Transaction manager 1 acknowledges that resource manager 1 is enlisted in the transaction, completing the Enlist action initiated in Step 7.

9. Resource manager 1 reports successful completion of transacted work, completing the Perform Transacted Work action initiated in Step 6.

10. The application initiates a Perform Transacted Work action against Application Service.

11. Application Service initiates a Request Pull Transaction action against transaction manager 2.

12. Transaction manager 2 initiates a Pull Transaction action against transaction manager 1, specifying the serialized transaction identifier.

13. Transaction manager 1 acknowledges that transaction manager 2 is now enlisted in the transaction, completing the Pull Transaction action initiated in Step 12.

14. Transaction manager 2 returns success to the application service, completing the Request Pull transaction action initiated in Step 11.

15. The application service initiates a Deserialize transaction action against transaction manager 2.

16. Transaction manager 2 provides the transaction identifier to the application service, completing the Deserialize transaction action initiated in Step 15.

17. The application service initiates a Perform Transacted Work action against resource manager 2.

18. Resource manager 2 initiates an Enlist action against transaction manager 2.

19. Transaction manager 2 acknowledges that resource manager 2 is enlisted in the transaction, completing the Enlist action initiated in Step 18.

20. Resource manager 2 reports successful completion of transacted work, completing the Perform Transacted Work action initiated in Step 17.

21. The application service responds to the application, completing the Perform Transacted Work action initiated in Step 10.

22. The application performs a Complete transaction action against transaction manager 1.

23. The transaction now begins Phase 1 of the Two-Phase Commit protocol.

18.3 MSDTC Connection Manager: OleTx Transaction Internet Protocol Specification Overview

The MSDTC Connection Manager: OleTx Transaction Internet Protocol, as specified in [MS-DTCM], provides an alternative application to transaction manager and application service to transaction manager communication protocol. The protocol provides some of the same functionality as the MSDTC Connection Manager: OleTx Transaction Protocol implementation, but the required transaction manager actions are performed using a Transaction Internet Protocol (TIP) implementation. This protocol provides an alternative protocol used by the following actions from the following roles:

- **Application Role**: Request a Push Transaction.
- **Application Service Role**:
  - Request a Pull Transaction.
  - Request a Push Transaction.

Both of these roles map to the TIP interoperability application role of the protocol specified by [MS-DTCM]. When communicating with these roles, the transaction manager role maps to the TIP interoperability provider specified in [MS-DTCM].

If a transaction manager Push Transaction action is required by an application or application service using the protocol specified in [MS-DTCM] to perform a Request Push Transaction action, the transaction manager uses the TIP transaction manager-to-transaction manager communication protocol.

If a transaction manager Pull Transaction action is required by an application service using the protocol specified in [MS-DTCM] to perform a Request Pull Transaction action, the transaction manager uses the TIP transaction manager-to-transaction manager communication protocol.

18.4 TIP Extensions Overview

The TIP extensions, as specified in [MS-TIPP], provide an alternative transaction manager-to-transaction manager communication protocol. The protocol provides the same functionality as the transaction manager-to-transaction manager protocol specified by an MSDTC Connection Manager: OleTx Transaction Protocol implementation, but the communication is performed using TIP messages. This protocol provides an alternative protocol used by the following actions from the following roles:

- **Transaction Manager Role**:
  - Push Transaction
  - Pull Transaction

When performing the Push Transaction action the local transaction manager role maps to the TIP superior transaction manager facet, as specified in [MS-TIPP], and the remote transaction manager maps to the TIP transaction manager role, as specified in [MS-TIPP].

When performing the Pull Transaction action, the local transaction manager role maps to the TIP subordinate transaction manager facet, as specified in [MS-TIPP], and the remote transaction manager maps to the TIP transaction manager role, as specified in [MS-TIPP].

The protocol interchanges defined by [MS-TIPP] are performed using TIP over TCP.

18.5 MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension

IBM's System Network Architecture (SNA) defines Logical Unit type 6.2 (LU 6.2) to support peer communications between LU 6.2 programs. LU 6.2 includes support for coordinating transactions between the LU 6.2 programs so that either both complete or both abort. The MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension as specified in [MS-DTCLU] allows an implementation of LU 6.2 to delegate much of the responsibility for managing a local transaction and coordinating its outcome with a transaction owned by a remote LU 6.2 implementation to a Transaction Manager. The Transaction Manager manages and coordinates transaction state, while the LU 6.2 implementation remains responsible for all LU to LU communications on the wire. To achieve this behavior, the protocol specified by [MS-DTCLU] provides an alternate resource manager to Transaction Manager communication protocol. The protocol provides the same conceptual functionality as the MSDTC Connection Manager: OleTx Transaction Protocol (as specified by [MS-DTCO]) implementation, though its implementation differs. This corresponds to the following actions from the following roles.

- **Resource Manager Role**:
  - Register
  - Enlist
- **Application Role**:
  - Perform Transacted Work

The resource manager role maps to the LU 6.2 Implementation role specified in [MS-DTCLU]. When communicating with this role, the Transaction Manager role is mapped to the Transaction Manager communicating with an LU 6.2 Implementation facet. This has the effect of enlisting the LU 6.2 transacted work in the transaction as a resource. This in turn means that in the course of a Perform Transacted Work action, a transaction identifier can be serialized in a form understandable by an LU 6.2 Application Service.

This allows an LU 6.2 Application Service to perform work in the same transactional context as any other application that uses the protocols specified by [MS-DTCO].

18.6 Protocol Stack

The following figure represents the wire protocol dependencies of the Transaction Processing Services protocols. For instance, the Transaction Manager to Transaction Manager communication specified in the Transaction Internet Protocol (TIP) Extensions, is performed over TCP.

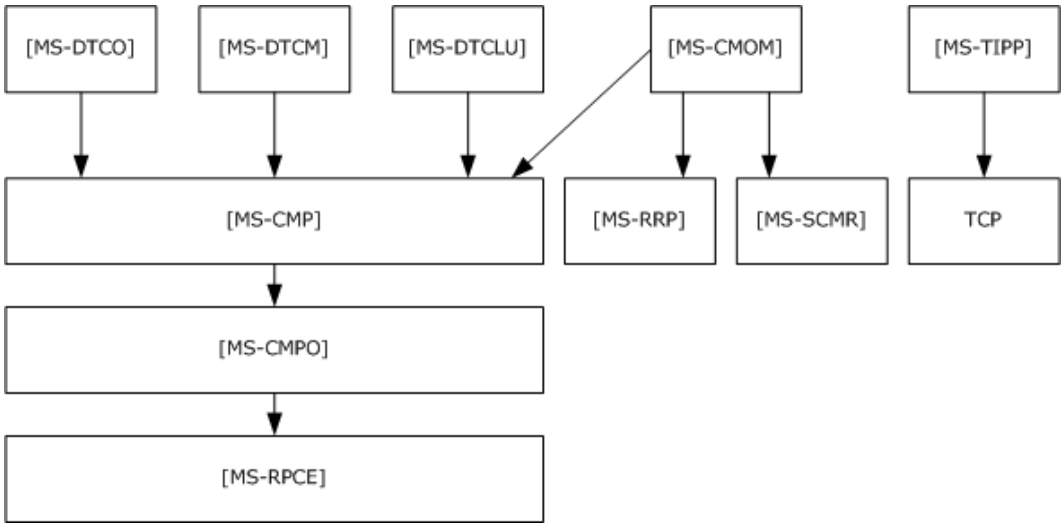

Figure 138: Transaction Processing Services Protocol stack

All communication with Transaction Managers via the protocols specified by the MSDTC Connection Manager: OleTx Transaction Protocol, MSDTC Connection Manager: OleTx Transaction Internet Protocol, and MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension depend upon the protocol specified by the MSDTC Connection Manager: OleTx Multiplexing Protocol, which in turn depends on the protocol specified by the MSDTC Connection Manager: OleTx Transports Protocol, which depends on RPC.

The Subscribe for Transaction Information action of the Management Tool is performed by the protocol specified by the MSDTC Connection Manager: OleTx Management Protocol using the protocol specified by the MSDTC Connection Manager: OleTx Multiplexing Protocol. The Configure Settings action is performed through the use of the protocols specified by the Windows Remote Registry Protocol and the Service Control Manager Remote Protocol.

18.7 Logical Dependencies

The following figure represents the logical dependencies of the Transaction Processing Services protocols. For instance, the Transaction Internet Protocol (TIP) Extensions (as specified in [MS-TIPP]) requires an implementation of the abstract state machine specified in the MSDTC Connection Manager: OleTx Transaction Protocol.

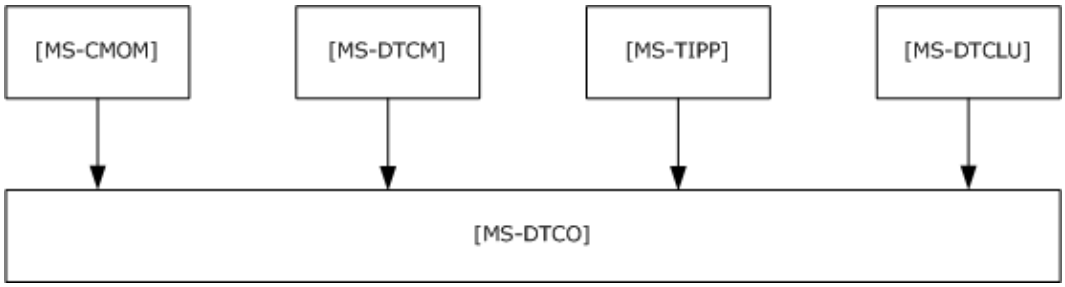

Figure 139: Transaction Processing Services logical dependencies

As described in section 18.2, the abstract state machine that drives the Two Phase Commit protocol is specified in [MS-DTCO]. An implementation of this is required to support the protocols specified by [MS-CMOM], [MS-DTCM], [MS-TIPP], and [MS-DTCLU].

The protocol specified by [MS-DTCM] also requires that an implementation of the Transaction Internet Protocol (TIP) is present, as described in section 18.3. An implementation of the protocol specified by [MS-TIPP] is the only such implementation included in the Transaction Processing Services protocols.

18.8 Implementation Scenario

The following scenario demonstrates how the Transaction Processing Services protocols are able to interoperate with one another.

18.8.1 Overview

The scenario includes the following participants.

- Three Transaction Managers:
  - A Root Transaction Manager (TM1) that implements the protocols as specified in [MS-DTCO], [MS-CMOM], [MS-DTCM], and [MS-TIPP].
  - A TIP Transaction Manager (TM2) that implements the Transaction Internet Protocol (TIP) as specified in [RFC2371].
  - A Subordinate Transaction Manager (TM3) that implements the protocols as specified in [MS-DTCO], and [MS-DTCLU].
- Two Resource Managers:
  - A Resource Manager (RM1) that implements the Resource Manager Role as specified in [MS-DTCO].
  - An LU 6.2 Implementation (LU1) that implements the LU 6.2 Implementation Role as specified in [MS-DTCLU] and the Application Role as specified in [MS-DTCO]. LU1 also performs the Application Service and Resource Manager roles defined in this document.
  - A Management Tool (MT) that implements the Management Client Role as specified in [MS-CMOM].
- An Application (APP) that implements the Application Role as specified in [MS-DTCO] and the TIP Interoperability Application Role as specified in [MS-DTCM].

The participants are arranged as shown in the following figure. The scenario is described as a series of message exchanges between participants that occur before, during, or after the lifetime of a single transaction. Some message exchanges interleave with other message exchanges, but are described separately to illustrate and clarify their function and purpose.

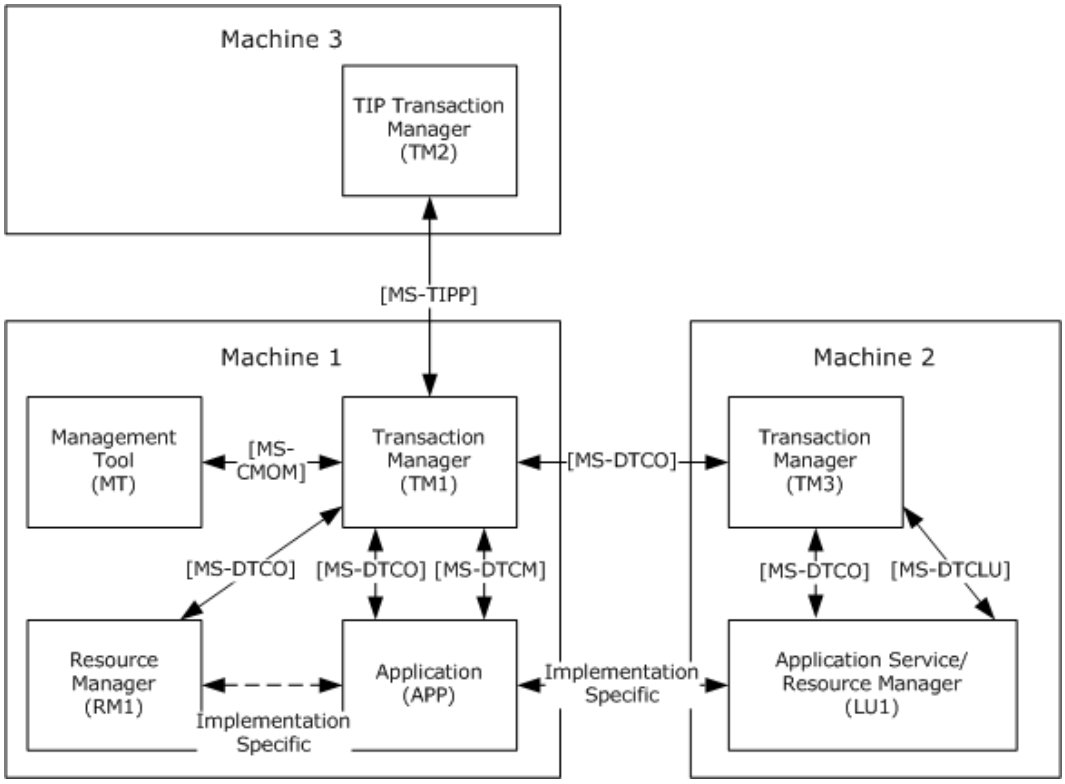

Figure 140: Implementation scenario architecture

Participants that implement the protocols specified in [MS-DTCO] communicate with each other using the MSDTC Connection Manager: OleTx Multiplexing Protocol (as specified in [MS-CMP]) connections that are in turn layered on top of the MSDTC Connection Manager: OleTx Transports Protocol (as specified in [MS-CMPO]) sessions. It is assumed that [MS-CMPO] sessions are established as required. Messages are sent from one participant to another by submitting a MESSAGE_PACKET to the underlying [MS-CMP] layer.

TIP messages are exchanged between the TIP Transaction Manager (TM2) and the Root Transaction Manager (TM1) using TIP connections that are created on top of TCP transport connections.

18.8.2 Precursory Message Exchanges

In these precursory message exchanges, a Management Tool (MT) subscribes for transaction information with its Transaction Manager (TM1), a Resource Manager (RM1) registers with its Transaction Manager (TM1), and an LU 6.2 Implementation (LU1) registers with its Transaction Manager (TM3).

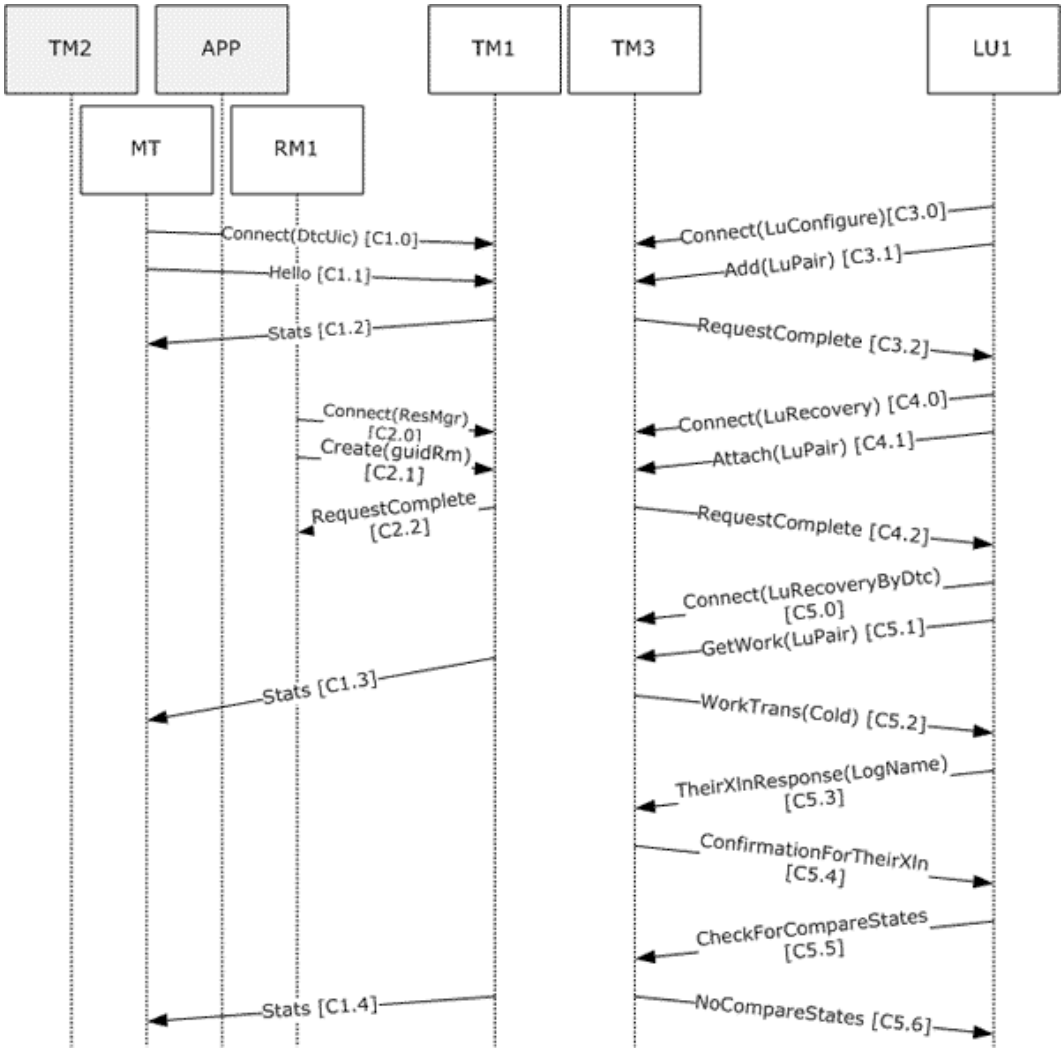

Figure 141: Precursory message exchange

18.8.2.1 Subscribing a Management Tool for Transaction Information

This message exchange demonstrates how a Management Tool (MT) subscribes for transaction information from a Transaction Manager (TM1) using protocols specified in the MSDTC Connection Manager: OleTx Management Protocol, as specified in [MS-CMOM].

- **[C1.0] Connect(DtcUic)**: MT initiates a CONNTYPE_TXUSER_DTCUIC connection on an MSDTC Connection Manager: OleTx Management Protocol session with TM1.
- **[C1.1] Hello**: MT sends an MTAG_HELLO message to TM1.
- **[C1.2] Stats**: When TM1 receives the connection request, its starts a timer (if one does not exist) and adds MT to its list of Management Client Role connections. Each time the timer expires, TM1 sends a MSG_DTCUIC_STATS message to MT. If TM1 is tracking any active transactions, then TM1 also sends an MSG_DTCUIC_TRANLIST message. In this example, no MSG_DTCUIC_TRANLIST message is sent.

MT continues to receive these messages from TM1 until it closes the connection by initiating the disconnect sequence [C1.3; C1.4].

18.8.2.2 Registering a Resource Manager

This message exchange demonstrates how a Resource Manager (RM1) registers with its Transaction Manager (TM1) using protocols as specified in [MS-DTCO].

1. **[C2.0]Connect(ResMgr)**: RM1 initiates a CONNTYPE_TXUSER_RESOURCEMANAGER connection on an MSDTC Connection Manager: OleTx Transports Protocol (as specified in [MS-CMPO]) session with TM1.
2. **[C2.1]Create(guidRm)**: RM1 sends a TXUSER_RESOURCEMANAGER_MTAG_CREATE message specifying a GUID that uniquely identifies the Resource Manager (guidRm) to TM1.
3. **[C2.2]RequestComplete**: When TM1 receives the TXUSER_RESOURCEMANAGER_MTAG_CREATE message, TM1 adds RM1 to its list of registered Resource Managers and sends a TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE message to RM1.

RM1 continues to maintain this connection to enable the creation of new enlistments in transactions and its participation in two-phase commit processing.

18.8.2.3 Registering an LU 6.2 Implementation

This message exchange demonstrates how an LU 6.2 Implementation (LU1) registers with its Transaction Manager (TM3) using protocols specified in the MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension, as specified in [MS-DTCLU]. Registering involves three separate message exchanges.

1. Configuring an LU Name Pair
2. Registering as the recovery process for the LU Name Pair
3. Performing cold recovery for the LU Name Pair

18.8.2.3.1 Configuring an LU Name Pair

This message exchange demonstrates how LU1 configures an LU Name Pair with TM3.

- **[C3.0]Connect(LuConfigure)**: LU1 initiates a CONNTYPE_TXUSER_DTCLUCONFIGURE connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM3.
- **[C3.1]Add(LuPair)**: LU1 sends a TXUSER_DTCLURMCONFIGURE_MTAG_ADD message specifying the LU Name Pair (LuPair) to TM3.
- **[C3.2]RequestComplete**: When TM3 receives the TXUSER_DTCLURMCONFIGURE_MTAG_ADD message, TM3 adds the LU Name Pair to its table of LU Name Pairs, and sends a TXUSER_DTCLURMCONFIGURE_MTAG_REQUEST_COMPLETED message to LU1.

When LU1 receives the TXUSER_DTCLURMCONFIGURE_MTAG_REQUEST_COMPLETED response from TM3, LU1 initiates the disconnect sequence.

18.8.2.3.2 Registering as the Recovery Process for an LU Name Pair

This message exchange demonstrates how LU1 registers as the recovery process for an LU Name Pair with TM3.

- **[C4.0]Connect(LuRecovery)**: LU1 initiates a CONNTYPE_TXUSER_DTCLURECOVERY connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM3.
- **[C4.1]Attach (LuPair)**: LU1 sends a TXUSER_DTCLURMRECOVERY_MTAG_ATTACH message to TM3 specifying an LU Name Pair (LuPair) which was previously configured with TM3.
- **[C4.2]RequestComplete**: When TM3 receives the TXUSER_DTCLURMRECOVERY_MTAG_ATTACH message, TM3 registers the connection's MS-CMPO session for all Recovery Processing associated with the LU Name Pair, and sends a TXUSER_DTCLURMRECOVERY_MTAG_REQUEST_COMPLETED message to LU1.

When LU1 receives the TXUSER_DTCLURMRECOVERY_MTAG_REQUEST_COMPLETED message from the TM3, LU1 continues to maintain the connection to enable recovery processes to be initiated and to enable the creation of new enlistments in the transactional work associated with the LU Name Pair.

18.8.2.3.3 Performing Cold Recovery for an LU Name Pair

This message exchange demonstrates how LU1 performs cold recovery for an LU Name Pair with TM3.

- **[C5.0]Connect(LuRecoveryByDtc)**: LU1 initiates a CONNTYPE_TXUSER_DTCLURECOVERYINITIATEDBYDTC connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM3.
- **[C5.1]GetWork (LuPair)**: LU1 sends a TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_GETWORK message to TM3 specifying the LU Name Pair (LuPair) for which LU1 registered as the recovery process.
- **[C5.2]WorkTrans (Cold)**: When TM3 receives the TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_GETWORK message, TM3 determines that it needs to perform cold recovery (Cold) for the LU Name Pair, and sends a TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_WORK_TRANS message to LU1.
- **[C5.3]TheirXlnResponse (LogName)**: When LU1 receives the TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_WORK_TRANS message, LU1 exchanges log names with the Remote LU, and sends a TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_THEIR_XLN_RESPONSE message specifying the Remote LU Log Name (LogName) to TM3.
- **[C5.4]ConfirmationForTheirXln**: When TM3 receives the TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_THEIR_XLN_RESPONSE message, TM3 verifies that the reported state of the Remote LU is consistent with TM3's state, and sends a TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_CONFIRMATION_FOR_THEIR_XLN message to LU1.
- **[C5.5]CheckForCompareStates**: When LU1 receives the TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_CONFIRMATION_FOR_THEIR_XLN message, LU1 sends a TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_CHECK_FOR_COMPARESTATES message to TM3 to query whether recovery work is required for any LU 6.2 transactional work involving the LU Name Pair.

- **[C5.6]NoCompareStates**: When TM3 receives the TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_CHECK_FOR_COMPARESTATES message, TM3 checks the local and remote transactional state, and sends a TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_NO_COMPARESTATES message to LU1.

When LU1 has received the TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_NO_COMPARESTATES message, no further messages are to be sent using this connection, and LU1 initiates the disconnect sequence.

18.8.3 Application-Driven Transactional Message Exchanges

In these sets of message exchanges: an Application (APP) begins a transaction, enlists a Resource Manager (RM1), pushes the transaction to a TIP Transaction Manager (TM2), and propagates the transaction to an LU 6.2 Implementation (LU1) which enlists the transaction, all in preparation for committing the transaction. These message exchanges induce related message exchanges between the three Transaction Managers (TM1, TM2, and TM3), and the effects of these message exchanges are reflected in the message exchanges between Transaction Manager (TM1) and the Management Tool (MT).

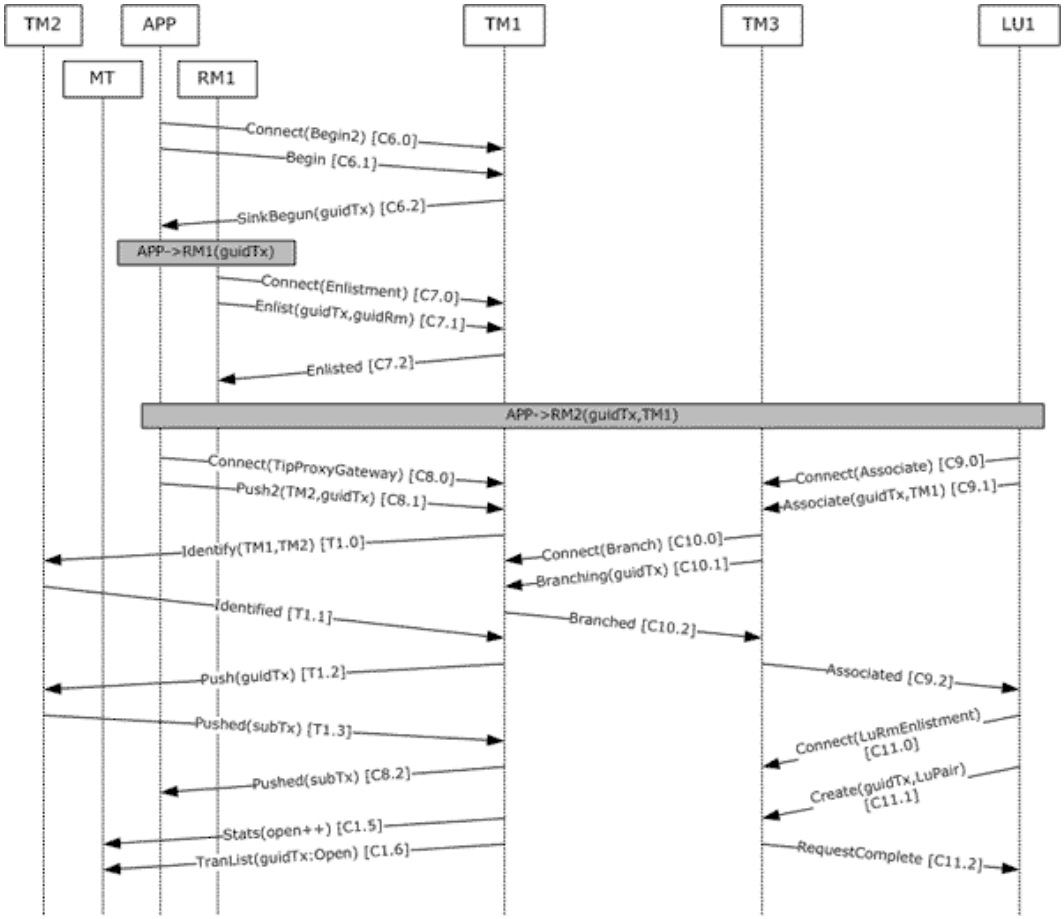

Figure 142: Transactional message exchanges

18.8.3.1 Beginning an MSDTC Connection Manager: OleTx Transaction Protocol Transaction

This message exchange demonstrates how an Application (APP) creates a transaction with its Transaction Manager (TM1) using protocols specified in [MS-DTCO].

- **[C6.0]Connect(Begin2)**: APP initiates a CONNTYPE_TXUSER_BEGIN2 connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM1.
- **[C6.1]Begin**: APP sends a TXUSER_BEGIN2_MTAG_BEGIN message to TM1 specifying the isolation level, timeout, transaction description, and isolation flag.
- **[C6.2]SinkBegun(guidTx)**: When TM1 receives the TXUSER_BEGIN2_MTAG_BEGIN message, TM1 creates a transaction object with a globally unique identifier (guidTx) and sends a TXUSER_BEGIN2_MTAG_SINK_BEGUN message to APP, and adds the transaction to its list of known transaction objects.

When APP receives the TXUSER_BEGIN2_MTAG_BEGUN message from TM1, the transaction (guidTx) has begun. APP is now free to propagate this transaction to Transaction Managers, Resource Managers and Application Services to perform work as part of the transaction, as long as it maintains the CONNTYPE_TXUSER_BEGIN2 connection. Eventually, APP decides whether to commit or abort the transaction. If APP disconnects the connection before committing or aborting the transaction, then TM1 will abort the transaction.

18.8.3.2 Enlisting a Resource Manager using protocols specified in [MS-DTCO]

This message exchange demonstrates how a Resource Manager (RM1) enlists on a transaction with its Transaction Manager (TM1) using protocols specified in [MS-DTCO]. This message exchange assumes that APP has used an out-of-band mechanism (for example, application API) to request RM1 to perform work as part of the transaction [APP->RM1(guidTx)]. This message exchange also assumes that RM1 has registered with TM1 as shown in section 18.8.2.2.

- **[C7.0]Connect(Enlistment)**: RM1 initiates a CONNTYPE_TXUSER_ENLISTMENT connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM1.
- [**C7.1]Enlist(guidTx)**: RM1 sends a TXUSER_ENLISTMENT_MTAG_ENLIST message to TM1 specifying the transaction GUID (guidTx), and the GUID that uniquely identifies itself (guidRm).
- **[C7.2]RequestComplete**: When TM1 receives the TXUSER_ENLISTMENT_MTAG_ENLIST message, TM1 enlists RM1 in the requested transaction, adds RM1 to its list of subordinates for the transaction, and sends a TXUSER_ENLISTMENT_MTAG_ENLISTED message to RM1.

RM1 continues to maintain the connection and waits for two-phase commit notifications from TM1.

18.8.3.3 Propagating the Transaction to a TIP Transaction Manager using protocols specified in [MS-DTCM] and [MS-TIPP] via Push Semantics

This message exchange demonstrates how an Application (APP) requests its Transaction Manager (TM1) using protocols specified in [MS-DTCM] to propagate the transaction from the Transaction Manager (TM1) to the TIP Transaction Manager (TM2) using the TIP protocol.

This message exchange assumes that the Application (APP) has acquired through some out-of-band mechanism (for example, application API) the TIP URL of the TIP Transaction Manager (TM2); in this example "tip://Machine2:3372/".

- **[C8.0]Connect(TipProxyGateway)**: APP initiates a CONNTYPE_TXUSER_TIPPROXYGATEWAY connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM1.
- **[C8.1]Push2(TM2,guidTx)**: APP sends a TXUSER_TIPPROXYGATEWAY_MTAG_PUSH2 user message specifying the transaction GUID (guidTx), and the TIP URL of TM2.
- **[T1.0]Identify(TM1,TM2)**: When TM1 receives the TXUSER_TIPPROXYGATEWAY_MTAG_PUSH2 message, TM1 locates the transaction and creates a new TIP connection with TM2 in the INITIAL state using the protocol specified in [MS-TIPP]. TM1 uses the TIP URL specified in the message to create the TIP connection over the TCP transport session established with TM2, and sends an IDENTIFY command to TM2 specifying TM1's primary Transaction Manager address and TM2's secondary Transaction Manager address.
- **[T1.1]Identified**: When TM2 receives the IDENTIFY command, TM2 sends an IDENTIFIED command to TM1, and the state of the TIP connection is changed to IDLE.
- **[T1.2]Push(guidTx)**: When TM1 receives the IDENTIFIED command, TM1 sends a PUSH command to TM2 specifying the Primary's Transaction Identifier (guidTx).
- **[T1.3]Pushed(subTx)**: When TM2 receives the PUSH command, TM2 adds the transaction to its list of transaction objects with a newly created transaction identifier (subTx), sends a PUSHED command to TM1, and the state of the TIP connection changes to ENLISTED.
- **[C8.2]Pushed(subTx)**: When TM1 receives the PUSHED command, TM1 sends a TXUSER_TIPPROXYGATEWAY_MTAG_PUSHED message APP specifying TM2's transaction Identifier (subTx).

When APP receives the TXUSER_TIPPROXYGATEWAY_MTAG_PUSHED message, APP initiates the disconnect sequence on the CONNTYPE_TXUSER_TIPPROXYGATEWAY connection.

18.8.3.4 Propagating the Transaction to an MSDTC Connection Manager: OleTx Transaction Protocol Application Service using Pull Semantics

This message exchange demonstrates how an Application (APP) propagates the transaction to an LU 6.2 Implementation (LU1) using protocols specified in [MS-DTCO] via pull semantics; this enables the LU1 to subsequently enlist on the transaction using protocols specified in [MS-DTCLU].

APP starts this exchange by sending an out-of-band message (for example, application API) to LU1 requesting it to pull the transaction (guidTx) from TM1 [APP->LU1(guidTx,TM1)].

- **[C9.0]Connect(Associate)**: LU1 initiates a CONNTYPE_TXUSER_ASSOCIATE connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM3.

  [C9.1]Associate(guidTx,TM1): LU1 sends a TXUSER_ASSOCIATE_MTAG_ASSOCIATE message to TM3 specifying the transaction Identifier (guidTx) and sufficient information (TM1's Machine Name, Endpoint GUID, and supported COM protocols) to establish an MSDTC Connection Manager: OleTx Transports Protocol session with TM1, if a session is not already established.

  [C10.0]Connect(Branch): When TM3 receives the TXUSER_ASSOCIATE_MTAG_ASSOCIATE message, TM3 attempts to locate the transaction in its list of transaction objects using the transaction Identifier (guidTx). Since the transaction object is not located, TM3 attempts to pull the transaction from the TM1 using information contained in the message, and thus TM3 initiates a CONNTYPE_PARTNERTM_BRANCH connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM1.

  [C10.1]Branching(guidTx): TM3 sends a PARTNERTM_BRANCH_MTAG_BRANCHING message with the transaction identifier (guidTx) of the requested transaction to TM1.

[C10.2]Branched: When TM1 receives the PARTNERTM_BRANCH_MTAG_BRANCHING message, TM1 creates a subordinate branch and sends a PARTNERTM_BRANCH_MTAG_BRANCHED message to TM3.

[C9.2]Associated: When TM3 receives the PARTNERTM_BRANCH_MTAG_BRANCHED message, TM3 keeps the connection open in order to process two-phase commit notifications from TM1, and sends a TXUSER_ASSOCIATE_MTAG_ASSOCIATED message to LU1 on the CONNTYPE_TXUSER_ASSOCIATE connection to inform LU1 that the pull operation was successful. TM3 continues to maintain the CONNTYPE_PARTNERTM_BRANCH connection with TM1, and waits for two-phase commit processing.

When LU1 receives the TXUSER_ASSOCIATE_MTAG_ASSOCIATED message, LU1 initiates the disconnect sequence on the CONNTYPE_TXUSER_ASSOCIATE connection.

18.8.3.5 Enlisting an LU 6.2 Implementation in a Transaction using protocols specified in MS-DTCLU

After the LU 6.2 Implementation (LU1) has pulled the transaction to its Transaction Manager (TM3), it enlists on the transaction using protocols specified in [MS-DTCLU].

- **[C11.0]Connect(LuRmEnlistment)**: LU1 initiates a CONNTYPE_TXUSER_DTCLURMENLISTMENT connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM3.
- **[C11.1]Create(guidTx,LuPair)**: LU1 sends a TXUSER_DTCLURMENLISTMENT_MTAG_CREATE message to TM3 specifying the transaction identifier (guidTx) and the LU Name Pair (LuPair).
- **[C11.2]RequestComplete**: When TM3 receives the TXUSER_DTCLURMENLISTMENT_MTAG_CREATE message, TM3 creates an LU enlistment on the transaction, and sends a TXUSER_DTCLURMENLISTMENT_MTAG_REQUEST_COMPLETED message to LU1.

When LU1 receives the TXUSER_DTCLURMENLISTMENT_MTAG_REQUEST_COMPLETED message, LU1 continues to maintain the connection and waits for two-phase commit processing.

18.8.3.6 Monitoring the Transaction Using Protocols Specified in [MS-CMOM]

This message exchange demonstrates TM1 broadcasting transaction information at this point in the lifecycle of the transaction (guidTx).

- **[C1.5]Stats(open++)**: Because the transaction (guidTx) is active but not yet committing or aborting, TM1 sends a MSG_DTCUIC_STATS message to MT with the number of open transactions incremented by one (open++).
- **[C1.6]TranList(guidTx:Open)**: TM1 sends a MSG_DTCUIC_TRANLIST message to MT listing the transaction (guidTx) in the open state (XACTSACT_OPEN).

18.8.4 Two-Phase Commit Transactional Message Exchanges

In these sets of message exchanges, the Application (APP) commits the transaction started in section 18.8.3.1 by using protocols specified in [MS-DTCO]. At this point, the message exchanges are driven by the Root Transaction Manager (TM1), which has three (3) subordinates – the Resource Manager (RM1), the TIP Transaction Manager (TM2), and the Subordinate Transaction Manager (TM3) – and all message exchanges follow the two-phase commit protocol. All messages occur on the existing TIP connection and existing connections over, MSDTC Connection Manager: OleTx Transports Protocol sessions.

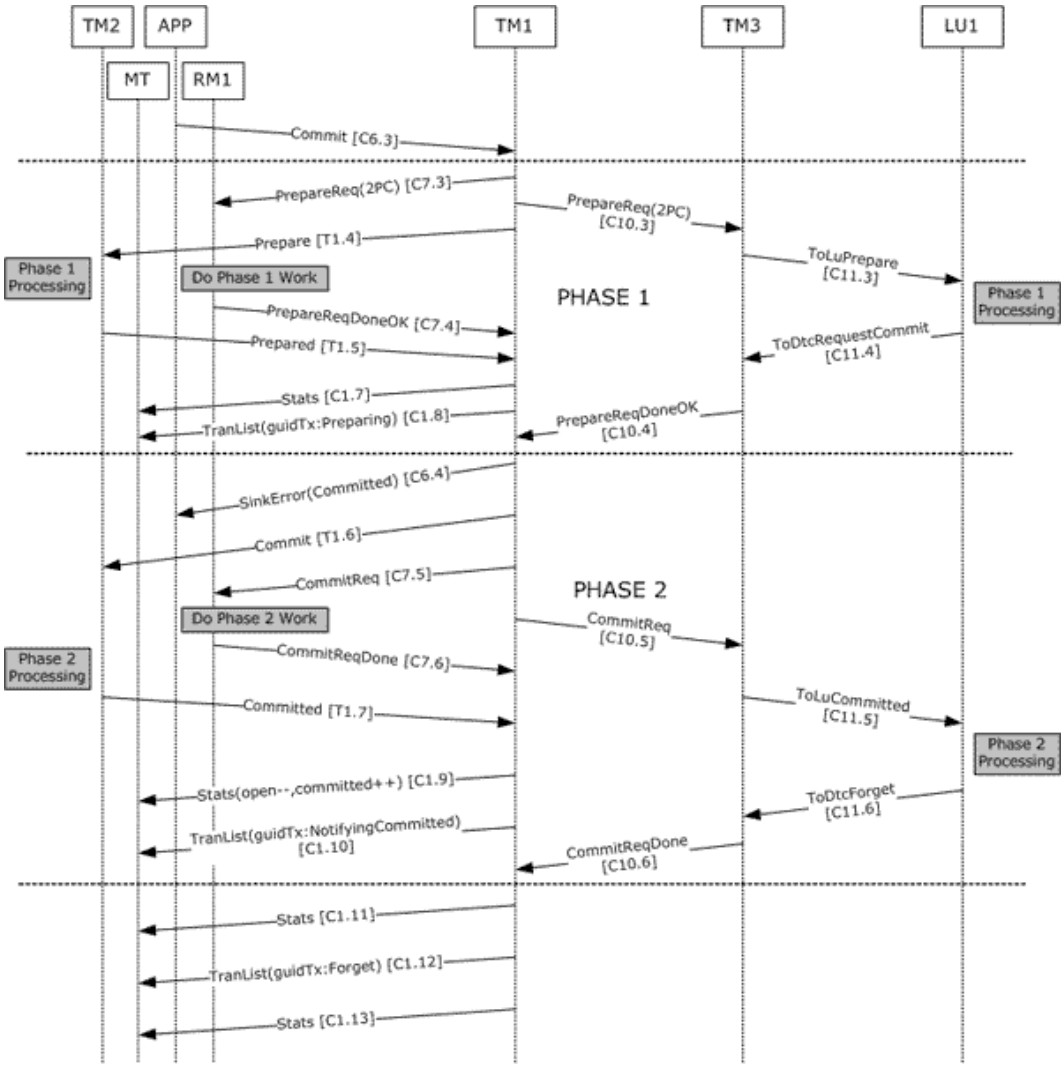

Figure 143: Two-phase commit transactional message exchanges

18.8.4.1 Committing the Transaction Using Protocols Specified in MS-DTCO

This message exchange demonstrates the Application (APP) committing the transactions (guidTx).

- **[C6.3]Commit**: APP sends a TXUSER_BEGIN2_MTAG_COMMIT message to TM1.

APP maintains the connection and waits for the outcome of the transaction from TM1.

18.8.4.2 Phase One

When TM1 receives the TXUSER_BEGIN2_MTAG_COMMIT message, the transaction's state enters phase one and TM1 attempts to commit the transaction by iterating through its list of subordinate branches, notifying the subordinates that phase one processing has begun. TM1 then waits for reply notifications from each of the subordinates before determining the outcome of the transaction.

18.8.4.2.1 Phase One - Subordinate Resource Manager

- **[C7.3]PrepareReq(2PC)**: TM1 sends a TXUSER_ENLISTMENT_MTAG_PREPAREREQ message to RM1 over the CONNTYPE_TXUSER_ENLISTMENT connection, indicating that this is a two-phase commit (2PC).
- **[C7.4]PrepareReqDoneOK**: When RM1 has successfully completed its phase one work, RM1 sends a TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE message to TM1 specifying TXUSER_ENLISTMENT_PREPAREREQDONE_OK.

RM1 maintains the connection and waits for the transaction outcome from TM1.

18.8.4.2.2 Phase One - Subordinate TIP Transaction Manager

- **[T1.4]Prepare**: TM1 sends a PREPARE command over the TIP connection associated with the transaction to TM2.
- **[T1.5]Prepared**: When TM2 has successfully completed its phase one processing, TM2 sends a PREPARED command to TM1 over the TIP connection.

The state of the TIP connection is now PREPARED and TM2 waits for transaction outcome from TM1.

18.8.4.2.3 Phase One - Subordinate Transaction Manager

- **[C10.3]PrepareReq(2PC)**: TM1 sends a PARTNERTM_PROPAGATE_MTAG_PREPAREREQ message to TM3 over the CONNTYPE_PARTNERTM_BRANCH connection, indicating that this is a two-phase commit (2PC).
- **[C11.3]ToLuPrepare**: When TM3 receives the PARTNERTM_PROPAGATE_MTAG_PREPAREREQ message, TM3 iterates through each of its subordinate branches to send out phase one notifications and sends a TXUSER_DTCLURMENLISTMENT_MTAG_TO_LU_PREPARE message to LU1 over the CONNTYPE_TXUSER_DTCLURMENLISTMENT connection.
- **[C11.4]ToDtcRequestCommit**: When LU1 receives the TXUSER_DTCLURMENLISTMENT_MTAG_TO_LU_PREPARE message, LU1 completes its phase one work, sends a TXUSER_DTCLURMENLISTMENT_MTAG_TO_DTC_REQUESTCOMMIT message to TM3, and waits for the transaction outcome from TM3.
- **[C10.4]PrepareReqDoneOK**: When TM3 receives the TXUSER_DTCLURMENLISTMENT_MTAG_TO_DTC_REQUESTCOMMIT message, TM3 sends a PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE message to TM1.

TM3 maintains the connection and waits for transaction outcome from TM1.

18.8.4.2.4 Phase One - Monitoring the Transaction Using Protocols Specified in MS-CMOM

This message exchange demonstrates TM1 broadcasting transaction information during phase one of the transaction (guidTx).

- **[C1.7]Stats**: Because the transaction's outcome is not yet known, TM1 sends a MSG_DTCUIC_STATS message to MT with no changes from its previous message [C1.5] related to this transaction.
- **[C1.8]TranList(guidTx:Preparing)**: TM1 sends a MSG_DTCUIC_TRANLIST message to MT listing the transaction (guidTx) in the preparing state (XACTSACT_PREPARING).

18.8.4.3 Phase Two

When TM1 receives successful phase-one notification replies from each of its subordinate branches, TM1 commits the transaction and begins phase-two processing by sending commit verification to APP and commit notifications to each of its subordinate branches (RM1, TM2, and TM3).

18.8.4.3.1 Phase Two - Application

- **[C6.4]SinkError(Committed)**: TM1 sends a TXUSER_BEGIN2_MTAG_SINK_ERROR message to APP over the CONNTYPE_TXUSER_BEGIN2 connection, specifying that the transaction has committed (TRUN_TXBEGIN_ERROR_NOTIFY_COMMITTED).

When APP receives the TXUSER_BEGIN2_MTAG_SINK_ERROR message, APP initiates the disconnect sequence.

18.8.4.3.2 Phase Two - Subordinate Resource Manager

- **[C7.5]CommitReq**: TM1 sends a TXUSER_ENLISTMENT_MTAG_COMMITREQ message to RM1 over the CONNTYPE_TXUSER_ENLISTMENT connection.
- **[C7.6]CommitReqDone**: When RM1 has completed its commit work, RM1 sends a TXUSER_ENLISTMENT_MTAG_COMMITREQDONE message to TM1 and initiates the disconnect sequence on the CONNTYPE_TXUSER_ENLISTMENT connection with TM1.

18.8.4.3.3 Phase Two - Subordinate TIP Transaction Manager

- **[T1.6]Commit**: TM1 sends a COMMIT command over the TIP connection associated with the transaction to TM2.
- **[T1.7]Committed**: When TM2 has successfully completed its phase two processing, TM2 sends a COMMITTED command to TM1 over the TIP connection.

The state of the TIP connection with TM1 is now IDLE, and the current transaction of the TIP connection is cleared.

18.8.4.3.4 Phase Two - Subordinate Transaction Manager

- **[C10.5]CommitReq**: TM1 sends a PARTNERTM_PROPAGATE_MTAG_COMMITREQ message to TM3 over the CONNTYPE_PARTNERTM_BRANCH connection.
- **[C11.5]ToLuCommitted**: When TM3 receives the PARTNERTM_PROPAGATE_MTAG_COMMITREQ message, TM3 iterates through each of its subordinate branches to send out commit notifications and sends a TXUSER_DTCLURMENLISTMENT_MTAG_TO_LU_COMMITTED message to LU1 over the CONNTYPE_TXUSER_DTCLURMENLISTMENT connection.
- **[C11.6]ToDtcForget**: When LU1 receives the TXUSER_DTCLURMENLISTMENT_MTAG_TO_LU_COMMITTED message, LU1 completes its phase-two processing, sends a TXUSER_DTCLURMENLISTMENT_MTAG_TO_DTC_FORGET message to TM3, and initiates the disconnect sequence.
- **[C10.6]CommitReqDone**: When TM3 receives the TXUSER_DTCLURMENLISTMENT_MTAG_TO_DTC_FORGET message, TM3 sends a PARTNERTM_PROPAGATE_MTAG_COMMITREQDONE user message to TM1, and initiates the disconnect sequence.

18.8.4.3.5 Phase Two - Monitoring the Transaction Using Protocols Specified in [MS-CMOM]

This message exchange demonstrates TM1 broadcasting transaction information during phase two of the transaction (guidTx).

- **[C1.9]Stats(open--,committed++)**: Since the transaction is now committed, TM1 sends a MSG_DTCUIC_STATS message to MT with the number of open transactions decremented by one (open--) and the number of committed transactions incremented by one (committed++).
- **[C1.10]TranList(guidTx:NotifyingCommitted)**: TM1 sends a MSG_DTCUIC_TRANLIST message to MT listing the transaction (guidTx) in the notifying committed state (XACTSACT_NOTIFYING_COMMITTED).

18.8.4.3.6 Phase Two - The Root Transaction Manager

After TM1 receives all phase-two replies from each of its subordinates, the transaction life cycle is complete and it removes the transaction from its list of known transactions.

18.8.4.3.7 Post-Phase Two Messages - Monitoring the Transaction Using Protocols Specified in MS-CMOM

This message exchange demonstrates TM1 broadcasting transaction information after phase two of the transaction (guidTx) is completed.

- **[C1.11]Stats**: TM1 sends a MSG_DTCUIC_STATS message to MT with no changes from its previous message [C1.9] related to this transaction.
- **[C1.12]TranList(guidTx:Forget)**: TM1 sends a MSG_DTCUIC_TRANLIST message to MT listing the transaction (guidTx) in the forget state (XACTSACT_FORGET). Any future MSG_DTCUIC_TRANLIST messages will not include this transaction.
- **[C1.13]Stats**: TM1 sends a MSG_DTCUIC_STATS message to MT with no changes from its previous message [C1.11] related to this transaction. Since there are no active transactions that TM1 is tracking, no MSG_DTCUIC_TRANLIST message is sent.

19 Networking and Transports

This section includes a general-purpose set of protocols that are not mapped to a specific service. These protocols are networking and transport protocols that are implemented in the covered Windows-based servers.

19.1 Networking and Transports Protocols Overview

An overview of the following networking and transports protocols is provided in the following sections:

- RPC over HTTP
- Distributed Component Object Model (DCOM)
- EAP MS-CHAPv2
- PEAPv0
- Internet Protocol Security Protocols Extensions
- Windows Management Instrumentation

Other networking and transport protocols that are previously described in this document include:

- Passport
- Event log
- ExtendedError Remote Data Structure
- ICertPassage
- Messenger Service Remote Protocol
- Network Time Protocol (NTP) Authentication Extensions
- W32Time Remote Protocol
- RPC, Secure RPC
- Windows Remote Registry Protocol
- Workstation Service Remote Protocol
- Authenticated Internet Protocol

19.2 RPC over HTTP Protocol

The RPC over HTTP protocol [MS-RPCH] specifies the use of either HTTP [RFC2616] or HTTPS [RFC2818] as a transport for the Microsoft Remote Procedure Calls Extensions [MS-RPCE]. In particular, it specifies provisions for using HTTP request/response streams as virtual channels, encoding rules for transporting **RPC PDUs** with HTTP requests and responses, and roles for participants in the protocol.

19.2.1 Protocol Stack

The following diagram illustrates the protocol stack resulting from the combined roles and encoding rules for the RPC over HTTP protocol [MS-RPCH].

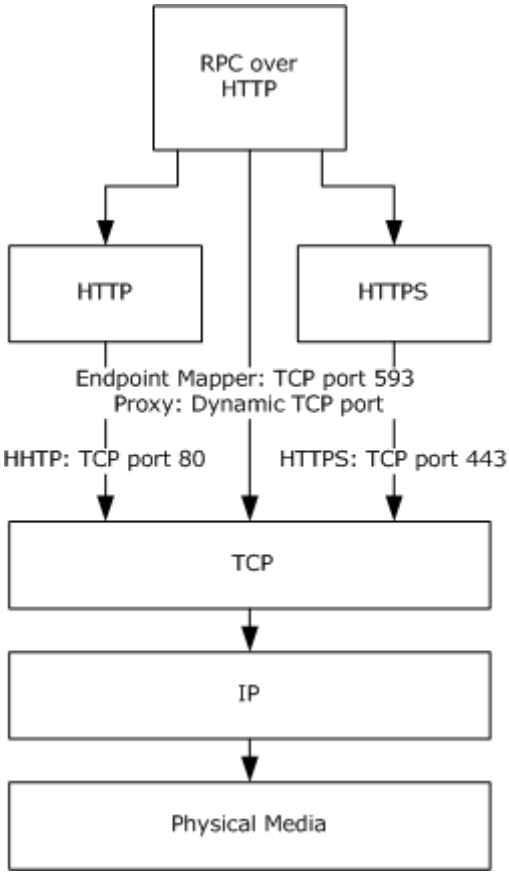

Figure 144: RPC over HTTP protocol stack

As shown in the preceding diagram, the RPC over HTTP protocol is layered directly over the HTTP and HTTPS protocols using their standard TCP port numbers.

The protocol stack for this section does not contain any other protocols.

19.2.2 Logical Dependencies

The RPC over HTTP protocol [MS-RPCH] is not logically dependent on any protocols other than HTTP, HTTPS, and TCP, which are its underlying transports.

19.3 Distributed Component Object Model

The Distributed Component Object Model (DCOM) Remote Protocol, as specified in [MS-DCOM], provides mechanisms for exposing application objects in distributed systems via remote procedure calls. It consists of a set of RPC interfaces that may be implemented over any RPC transport by using any RPC protocol sequence that is supported by the Remote Procedure Call Protocol Extensions, as specified in [MS-RPCE]. This layering is illustrated in the following diagram.

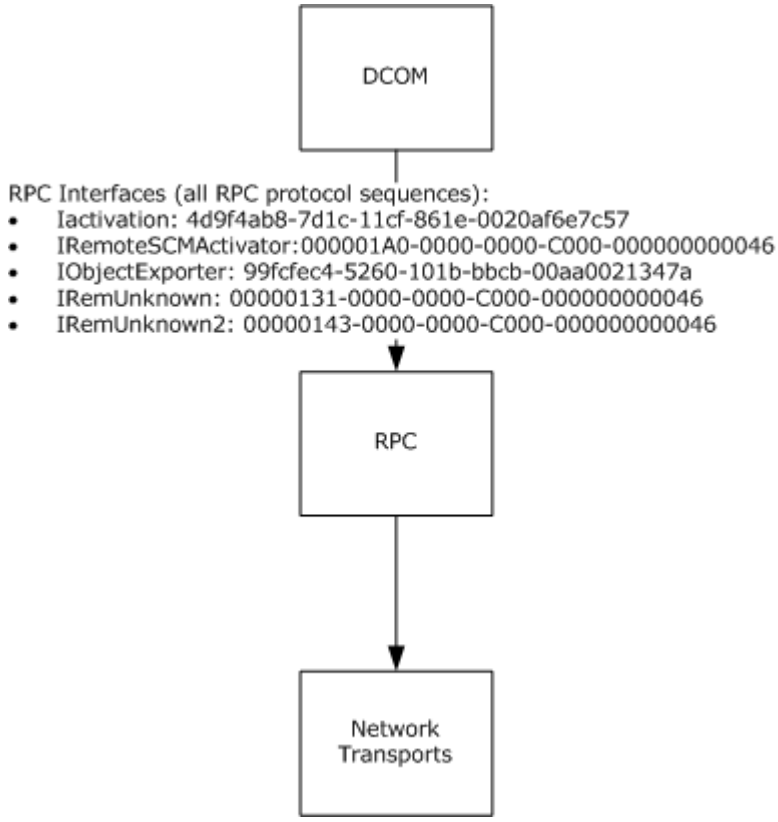

Figure 145: Remote Procedure Call Protocol Extensions stack layers

The DCOM Remote Protocol, as specified in [MS-DCOM], is not logically dependent on any protocols other than the Remote Procedure Call Protocol Extensions. The dependencies for the latter are specified in the Remote Procedure Calls section.

19.4 EAP MS-CHAPv2

This section describes the relationship among protocols and protocol extensions that are used for EAP MS-CHAPv2 authentication.

19.4.1 Protocol Stack

An EAP MS-CHAPv2 peer passes the MS-CHAPv2[RFC2759] protocol over EAP [RFC2284] [RFC3748] and a lower-layer protocol such as PPP [RFC1661] or IEEE 802.1X to authenticate to an EAP server using a Network Access Server (also known as the EAP authenticator) as a pass-through. The EAP authenticator passes through EAP packets communicated between the EAP MS-CHAPv2 peer and server, utilizing RADIUS/EAP protocol [RFC3579]; keying material is passed from the RADIUS/EAP server to the authenticator using the Microsoft RADIUS VSA extensions [RFC2548].

In turn, the Microsoft RADIUS server uses the Net Logon Remote protocol [MS-NRPC] to communicate to a domain controller and validate the user credentials provided.

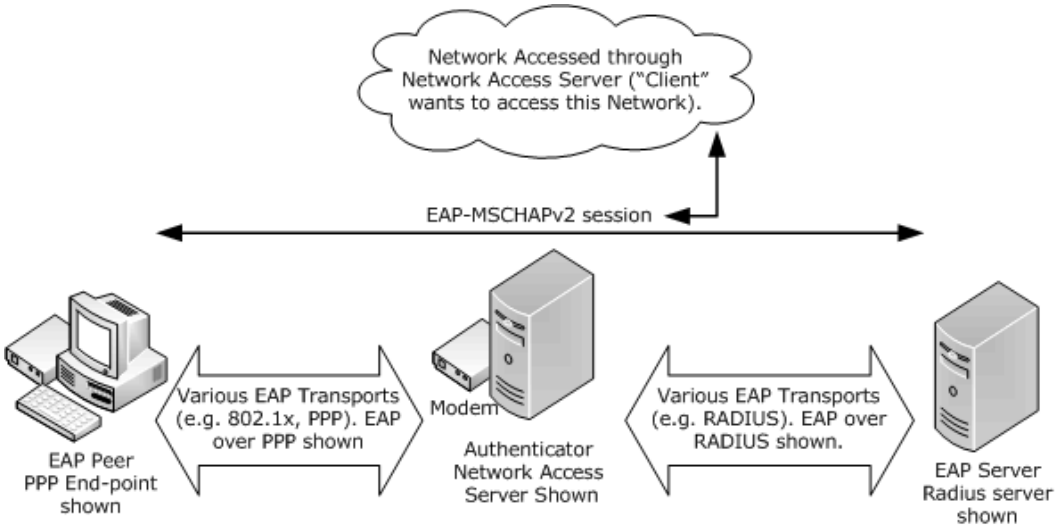

Figure 146: Relationship between the EAP peer, the EAP authenticator/Network Access Server and the EAP/RADIUS server

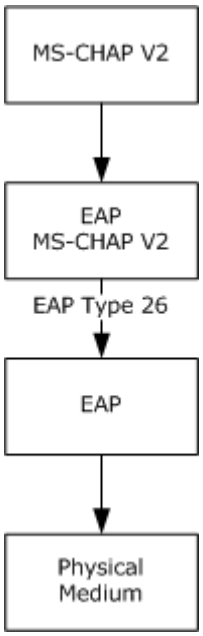

Figure 147: Depicts the protocol stack used by the EAP MS-CHAPv2 client

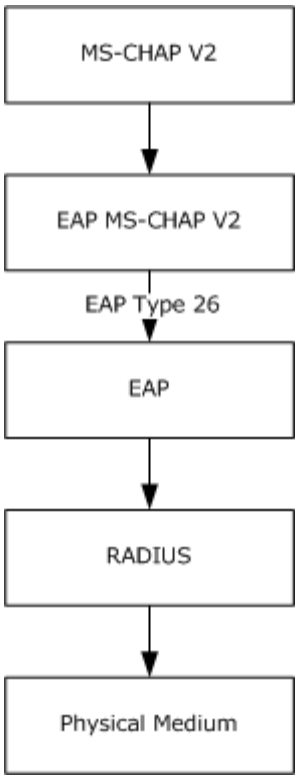

Figure 148: Depicts the protocol stack used by a Microsoft RADIUS server

19.5 PEAPv0

This section describes the relationship among protocols and protocol extensions used for PEAPv0 authentication.

19.5.1 Protocol Stack

A PEAPv0 peer uses the TLS protocol [RFC2246] run over **EAP** [RFC2284] [RFC3748] and a lower-layer protocol such as PPP [RFC1661] or IEEE 802.1X to establish a server-authenticated **TLS** tunnel to an **EAP** server using a **network access server (NAS)** (also known as the **EAP authenticator**) as a pass-through. Within the **TLS** tunnel, an inner **EAP** method (such as EAP MS-CHAP v2) then runs to enable the peer to authenticate to the server. The **EAP authenticator** passes through **EAP** packets communicated between the PEAPv0 peer and server, utilizing RADIUS/EAP protocol [RFC3579]; keying material is passed from the RADIUS/EAP server to the **authenticator** using the Microsoft RADIUS VSA extensions [RFC2548].

The relationship between the **EAP peer**, the **EAP authenticator**/**NAS** and the EAP/**RADIUS server** is shown in the following Figure "Relationship between EAP peer, Network Access Server, and EAP/RADIUS server".

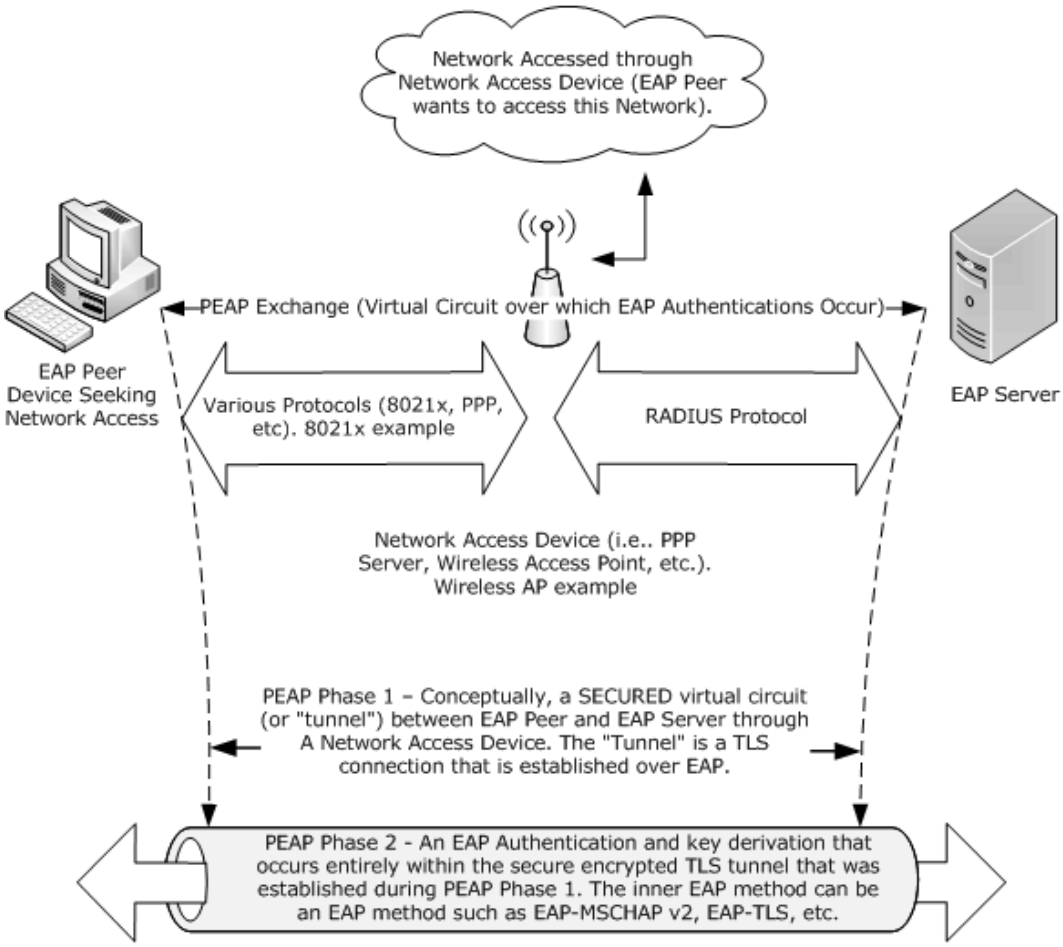

Figure 149: Relationship between EAP peer, Network Access Server, and EAP/RADIUS server

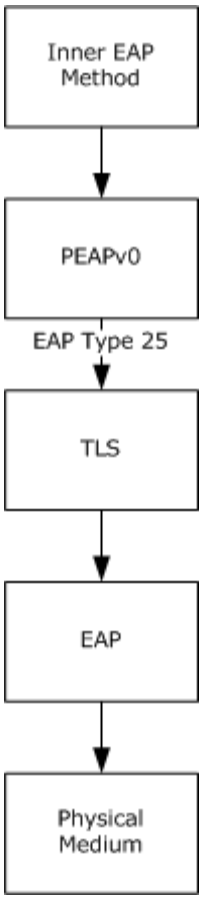

Figure 150: Depicts the protocol stack used by the PEAPv0 client

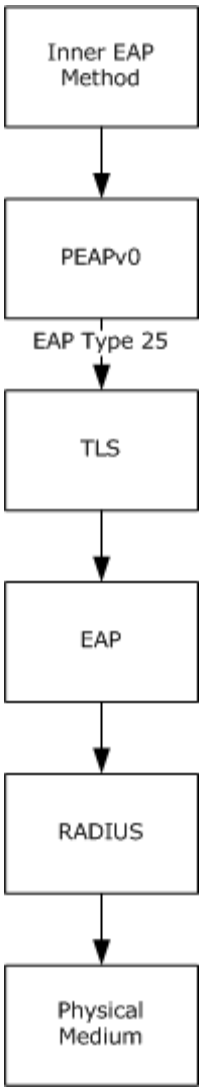

Figure 151: Depicts the protocol stack used by a Microsoft RADIUS server

19.6 Windows Management Instrumentation

The Windows Management Instrumentation Remote Protocol is as specified in [MS-WMI]. The Windows Management Instrumentation Encoding Version 1.0 Protocol, as specified in [MS-WMIO], describes the encoding of object types that are used in the protocol.

The Windows Management Instrumentation Remote Protocol is used to access and manipulate remote management objects that are implemented according to the Common Information Model (CIM), which is defined by the Distributed Management Task Force. For more information, see [DMTF-DSP0004].

19.6.1 Protocol Stack

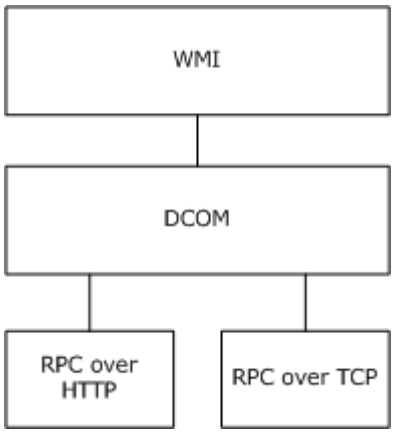

Figure 152: WMI protocol stack security

The Windows Management Instrumentation Remote Protocol uses different default security levels in different versions of Windows, as specified in [MS-WMI]. For compatibility with all clients, the user can specify the NTLM Authentication Protocol (as specified in [MS-NLMP]), the Kerberos protocol, or the SPNEGO Protocol Extensions (as specified in [MS-SPNG]).

The Windows Management Instrumentation Remote Protocol communicates exclusively through DCOM. The DCOM Remote Protocol is actually the foundation for the Windows Management Instrumentation Remote Protocol and is used to establish the protocol, secure the commutation channel, authenticate clients, and to implement a reliable communication between clients and servers.

19.7 Remote Data Services (RDS) Transport

The Remote Data Services (RDS) Transport Protocol facilitates communication for distributed applications. Operating at the application level, the RDS Transport Protocol allows querying and manipulation of data on remote servers. To facilitate data operations, the RDS Transport Protocol specifies how the remote method and its parameters are presented in an RDS message, which is then transmitted in an HTTP request to a server for execution. It also specifies how the results of an invoked method are represented in a message for transmission back to the client in the HTTP response. Finally, it specifies the data-centric messages used for data query and manipulation as well as their associated RecordSets.

The core data-centric messages used in an RDS Transport Protocol server are as follows.

- Execute - Provides a method to execute a complex SQL command and return a RecordSet.
- Query - Provides a method to execute a simple parameterless SQL query command and RecordSet.
- Synchronize - Provides a method for an RDS Transport Protocol client to synchronize data changes from the client to the server.

19.7.1 Protocol Stack

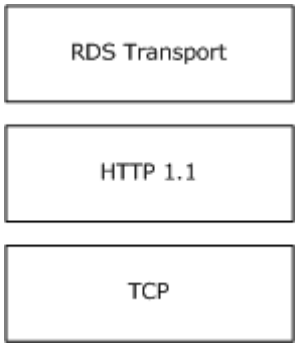

Figure 153: Protocol Stack

The RDS Transport Protocol depends on HTTP 1.1 as defined in [RFC2616], and uses HTTP as its underlying transport. The functionality supplied by RDS Transport Protocol has been superseded by SOAP and DCOM. For more information, see [SOAP1.1] and [MS-DCOM].

20 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

21 Index

O

P

R

S

T

V

W