[MS-GRVSSTP]:

Simple Symmetric Transport Protocol (SSTP)

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **Copyrights**. This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft Open Specifications Promise or the Microsoft Community Promise. If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **License Programs**. To see all of the protocols in scope under a specific license program and the associated patents, visit the Patent Map.
- **Trademarks**. The names of companies and products contained in this documentation might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than as specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications documentation does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments, you are free to take advantage of them. Certain Open Specifications documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Support. For questions and support, please contact dochelp@microsoft.com.

Revision Summary

Date	Revision History	Revision Class	Comments
4/4/2008	0.1	New	Initial Availability
6/27/2008	1.0	Major	Revised and edited the technical content
12/12/2008	1.01	Editorial	Revised and edited the technical content
7/13/2009	1.02	Major	Revised and edited the technical content
8/28/2009	1.03	Editorial	Revised and edited the technical content
11/6/2009	1.04	Editorial	Revised and edited the technical content
2/19/2010	2.0	Minor	Updated the technical content
3/31/2010	2.01	Editorial	Revised and edited the technical content
4/30/2010	2.02	Editorial	Revised and edited the technical content
6/7/2010	2.03	Editorial	Revised and edited the technical content
6/29/2010	2.04	Editorial	Changed language and formatting in the technical content.
7/23/2010	2.05	Minor	Clarified the meaning of the technical content.
9/27/2010	2.05	None	No changes to the meaning, language, or formatting of the technical content.
11/15/2010	2.05	None	No changes to the meaning, language, or formatting of the technical content.
12/17/2010	2.06	Major	Significantly changed the technical content.
3/18/2011	2.06	None	No changes to the meaning, language, or formatting of the technical content.
6/10/2011	2.06	None	No changes to the meaning, language, or formatting of the technical content.
1/20/2012	2.7	Minor	Clarified the meaning of the technical content.
4/11/2012	2.7	None	No changes to the meaning, language, or formatting of the technical content.
7/16/2012	2.7	None	No changes to the meaning, language, or formatting of the technical content.
10/8/2012	2.7	None	No changes to the meaning, language, or formatting of the technical content.
2/11/2013	2.7	None	No changes to the meaning, language, or formatting of the technical content.
7/30/2013	2.7	None	No changes to the meaning, language, or formatting of the technical content.
11/18/2013	2.7	None	No changes to the meaning, language, or formatting of the technical content.
2/10/2014	2.7	None	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
4/30/2014	2.7	None	No changes to the meaning, language, or formatting of the technical content.
7/31/2014	2.7	None	No changes to the meaning, language, or formatting of the technical content.
10/30/2014	2.7	None	No changes to the meaning, language, or formatting of the technical content.
6/23/2016	2.7	None	No changes to the meaning, language, or formatting of the technical content.
9/14/2016	2.7	None	No changes to the meaning, language, or formatting of the technical content.
9/19/2017	3.0	Major	Significantly changed the technical content.
12/12/2017	3.1	Minor	Clarified the meaning of the technical content.

Table of Contents

1 Introduction

The Simple Symmetric Transport Protocol (SSTP) is a TCP-based, message-oriented, application-layer binary protocol. It enables two higher-level applications to engage in bi-directional communication and to exchange data over multiple logical sessions on a single network connection.

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

account: A collection of data and settings for a SharePoint Workspace or Groove identity that represents a user. This includes shared spaces, messages, and preferences that are associated with a user's identity. An account can reside on one or more devices.

connection: (1) A link between two devices that uses the Simple Symmetric Transport Protocol (SSTP). Each connection can support one or more SSTP sessions.

(2) A link that two physical machines or applications share to pass data back and forth.

device: A client or server computer that uses a **device URL** to identify itself as an endpoint for synchronizing account data.

device URL: A unique identifier for a client device, as described in [RFC3986].

device-targeted message: A message with an intended destination of a specific resource handler, identity, and client device combination. A device-targeted message is sent over a session addressed by a tuple of resource URL, identity URL, and client device URL.

fanout: The process of transmitting a message from a client device to a relay server for replication and distribution to multiple recipients.

identity: A digital persona that is associated with two key pairs, one for encrypting data and another for signing data.

identity URL: A string of characters that uniquely identifies an identity and conforms to the syntax of a URI, as described in [RFC3986].

identity-targeted message: A message that is destined for a specific resource handler and identity combination, regardless of the client device. The message address includes a resource URL, identity URL, and client device URL, where the client device URL is empty.

little-endian: Multiple-byte values that are byte-ordered with the least significant byte stored in the memory location with the lowest address.

message sequence: A command sequence "message/data… data…data/endmessage" that comprises a data transmission during a Simple Symmetric Transport Protocol (SSTP) session.

multi-drop fanout: A type of message fanout in which a client sends a message, addressed to multiple recipients, to a common recipient relay server.

presence: A status indicator on a client device that is transmitted by using the Wide Area Network Device Presence Protocol (WAN DPP).

relay server: A server application that provides message transmission services for Simple Symmetric Transport Protocol (SSTP) messages.

relay URL: A string of characters that uniquely identifies a relay server and conforms to the syntax of a URI, as described in [RFC3986].

session: A unidirectional communication channel for a stream of messages that are addressed to one or more destinations. A destination is specified by a resource URL, an identity URL, and a device URL. More than one session can be multiplexed over a single connection.

Simple Symmetric Transport Protocol Security Protocol (SSTP) security: An independent sub-protocol that is exchanged within defined Simple Symmetric Transport Protocol (SSTP) messages, and is used for mutual authentication between a relay server and a client device or an account.

single-hop fanout: A form of message fanout, where a client sends a single message with multiple recipients to the sender's relay server. The sender's relay server then groups copies of the message according to recipient relays and distributes the message to those target relay servers.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as defined in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[MS-GRVSSTPS] Microsoft Corporation, "Simple Symmetric Transport Protocol (SSTP) Security Protocol".

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

1.2.2 Informative References

[IANAPORT] IANA, "Service Name and Transport Protocol Port Number Registry", http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

[MS-GRVDYNM] Microsoft Corporation, "Groove Dynamics Protocol".

[MS-GRVHENC] Microsoft Corporation, "HTTP Encapsulation of Simple Symmetric Transport Protocol (SSTP)".

[MS-GRVSPMR] Microsoft Corporation, "Management Server to Relay Server Groove SOAP Protocol".

[MS-GRVWDPP] Microsoft Corporation, "Wide Area Network Device Presence Protocol (WAN DPP)".

[RFC2616] Fielding, R., Gettys, J., Mogul, J., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.rfc-editor.org/rfc/rfc2616.txt

[RFC3986] Berners-Lee, T., Fielding, R., and Masinter, L., "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005, http://www.rfc-editor.org/rfc/rfc3986.txt

1.3 Protocol Overview (Synopsis)

The Simple Symmetric Transport Protocol (SSTP) is an application-layer protocol that enables two **devices** to engage in bi-directional, asynchronous communication. This protocol supports multiple overlapping message transmissions over a single network **connection (2)**. This protocol depends on an established and presumed reliable transport-layer connection (2).

SSTP connections (1) are established by one SSTP device to another SSTP device listening on the preferred Internet Assigned Numbers Authority (IANA)-assigned SSTP port 2492/TCP [IANAPORT]. However, as an application-layer protocol, this protocol can operate on any TCP connection (2) established between two SSTP-compatible devices on any mutually agreed-upon listening port.

This protocol employs tunneling and encapsulation methods to navigate firewalls and adapt to different port requirements. If the preferred port 2492/TCP is not available, this protocol can use HTTP Tunneling via the HTTP Connect handshake method to operate over port 443/TCP, as described in [MS-GRVHENC]. If firewalls block both ports 2492 and 443, this protocol can be encapsulated in standard HTTP [RFC2616] over port 80/TCP via any of the SSTP Encapsulation protocols, as described in [MS-GRVHENC].

SSTP communication begins when a device sends an SSTP Connect command to a target device over a TCP connection (2) and receives an affirmative connection response in return. Over this established bi-directional SSTP connection (1), either the initiating or target device can use SSTP commands to open SSTP **sessions** and send messages. SSTP sessions are the uni-directional communications channels through which participating devices send and receive messages.

This protocol is intended for use by client and **relay server** devices. SSTP clients are end-user devices with messages to exchange, typically containing user or application-generated data. Relay servers are essentially store and forward devices that facilitate delivery of client messages.

This protocol supports authentication of client devices via the **Simple Symmetric Transport Protocol Security Protocol (SSTP) security** sub-protocol, described in [MS-GRVSSTPS]. This protocol does not encrypt messages, but expects payload to be encrypted by higher-layer protocols.

In conjunction with the Wide Area Network Device Presence Protocol (WAN DPP), this protocol also supports client presence notification. This capability depends on presence servers that inform subscribing clients of each other's published device addresses and online status. WAN DPP is described separately in [MS-GRVWDPP].

The following sections provide more detail about SSTP commands, connections (1), sessions, messages, and roles.

1.3.1 SSTP Commands

SSTP commands control all levels of SSTP communications. These commands enable devices to establish, respond to, and close SSTP **connections (1)** and **sessions**, and to send and acknowledge receipt of SSTP messages.

Some SSTP commands are role-dependent; how and when they can be used depends on whether the issuing device is a client or a relay server. For example, the Attach command is valid only if issued by a client with an established connection to a relay server. Conversely, the AttachResponse command is only valid if issued by a relay server responding to a client Attach request.

Most SSTP commands are processed within an SSTP session, delimited by Open (or FanoutOpen) and Close commands. The following Connect and connection-related commands are exceptions, processed outside the session, at the connection level:

- Connect
- ConnectResponse

- ConnectAuthenticate
- ConnectClose
- Noop

The following list summarizes the SSTP command set:

- Connect: Initiates an SSTP connection to another device.
- ConnectResponse: Acknowledges the initial connection attempt.
- ConnectAuthenticate: Validates the connection authentication sequence.
- Open: Opens an SSTP session, targeting a single device.
- FanoutOpen: Opens an SSTP **fanout** session to a relay server, targeting multiple devices. When in **single-hop fanout**, opens a proxied session through the home relay to a foreign relay, for one or more devices.
- OpenResponse: Acknowledges the status of an Open attempt.
- Message: Begins the **message sequence** on an SSTP session.
- Data: The payload or payload portion of a message sequence.
- EndMessage: Ends the message sequence.
- Noop: Acknowledges receipt of SSTP messages and ensures that the underlying transport connection remains open.
- Close: Ends an SSTP session.
- SessionStatus: Indicates the removal of some destinations from a fanout session.
- ConnectClose: Closes the SSTP connection between devices.
- Attach: Initiates authentication of a client **account** when a client connects to a relay server.
- AttachResponse: Provides a security challenge from a relay server to a client Attach attempt.
- AttachAuthenticate: Answers the security challenge from an AttachResponse, and ends the client account authentication sequence.
- Register: Initiates the registration of a client account/device combination or **identity** list when a client connects to a relay server.
- RegisterResponse: Indicates the status of a Register attempt, as reported the relay server.

1.3.2 SSTP Connections

This protocol provides a full-duplex application-level **connection (1)** between two computer devices that have established a network-layer connection via TCP or another comparable reliable network transport. Clients initiate SSTP connections to other clients and to servers; servers initiate SSTP connections to other servers but not to clients.

To establish an SSTP connection, an initiating device sends an SSTP Connect command to a specified recipient. If delivery is successful, the recipient responds by sending a ConnectResponse command that contains a ResponseId field with a value of Ok. This two-way handshake results in an established SSTP connection. All subsequent application-level communication between these devices is

multiplexed over this connection. Either device can close the connection at any time by sending a ConnectClose command.

1.3.3 SSTP Sessions

Sessions are opened over an established SSTP **connection (1)**. SSTP sessions are uni-directional channels that transport messages between two devices over an established SSTP connection. Multiple SSTP sessions can be opened in both directions simultaneously on a single connection. SSTP sessions can be opened by either a client or server.

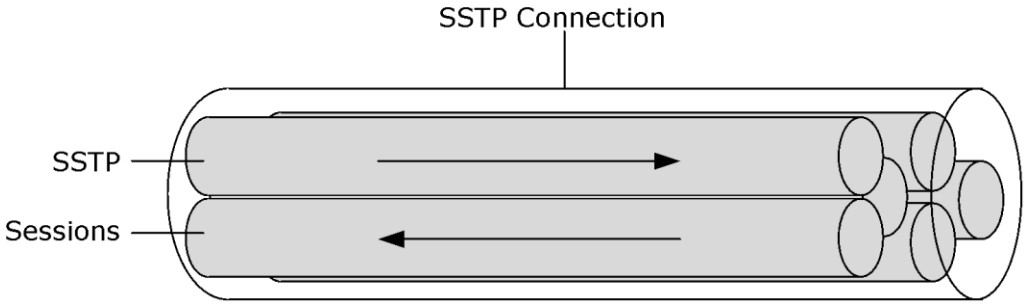


Figure 1: SSTP sessions carried within a single connection

To open a new session on an established SSTP connection between two devices, one device sends an SSTP Open command to the other. When the second device returns an OpenResponse command to confirm receipt, the session is considered opened and application-layer messaging can begin.

A device indicates that it has finished sending its last message for an SSTP session by sending an SSTP Close command to close the SSTP session. Either device sender or recipient devices, can close a session at any time; there is no requirement that one device wait for the other device to stop sending data before a session can be closed.

Each SSTP session directs messages to a specific resource handler, a higher-layer designation for a specific message type (such as instant messages or data updates). Resource handlers contribute to an addressing combination that includes identity, and client device. The address entity is identified in the Open command by the following entry: resource URL, **identity URL**, and **device URL**, forming the core addressing model of SSTP.

A message can be sent to resource handlers for multiple destination identities and devices by opening a fanout session over an SSTP connection to a relay server. The FanoutOpen command extends the addressing model to include a **relay URL**, which identifies the relay server associated with the destination client. The relay server receiving data on a fanout session takes the role of a data multicaster, forwarding data to multiple destinations. The method for associating a relay server with a client is application-determined.

SSTP session addressing accommodates two types of destinations: those addressed to a specific user on a specific device, and those addressed to a specific user regardless of the device. **Device-targeted messages** are those addressed to a specific resource-user-device combination, and the addressing entry of resource URL, identity URL, and device URL is used. **Identity-targeted messages** are those addressed to a specific resource handler-identity combination, regardless of the user's device, and only the session's resource URL and identity URL are used; the device URL is empty.

1.3.4 SSTP Messages

SSTP messages are transmitted over an open **session** via a sequence of SSTP commands. Each message consists of a Message command, followed by one or more Data commands, and a final

EndMessage command. Large blocks of application-level data are transmitted via multiple Data commands. The Data commands together comprise the message content (or payload).

The contents of SSTP messages on a session are delivered to the higher-layer resource handler for the identity and device specified in the session Open command. In the case of message fanout, the FanoutOpen command specifies this information.

Message acknowledgments are handled at the **connection (1)** level. Accurate message acknowledgments depend upon the guaranteed message order and transmission afforded by the underlying TCP transport. Recipients acknowledge successfully delivered messages by including a message count in a returning Noop, Message, or ConnectClose command.

Recipients can acknowledge multiple messages simultaneously by specifying the count of all messages received on a connection to-date.

1.3.5 SSTP Client and Server Roles

This protocol supports communications between client and relay server devices over TCP **connections (1)**. Because client devices might not be able to directly connect to each other across the Internet, they rely on relay servers to enable their data exchanges. Client devices initiate connections to relay servers and all application-level data originates on client devices. Relay servers are message store and forward devices that facilitate delivery of client messages, temporarily holding messages for offline or otherwise inaccessible users.

The relationship between SSTP clients and relay servers is as follows:

- Each client is assigned to one or more relay servers, as determined by a higher-layer application.
- Clients initiate connections to relay servers to deposit or retrieve addressed messages over SSTP **sessions**.
- The relay server application provides a resource handler that accepts and stores client messages addressed to other clients.

Relay server resource handlers deliver client messages to corresponding resource handlers on destination clients by opening sessions on client-initiated SSTP connections to the relay server.

1.3.5.1 Client Role

Client devices use SSTP to communicate user or application-generated data. SSTP clients connect to relay servers to deposit (or enqueue) messages, and to retrieve (or dequeue) messages. Any SSTP client can fulfill one or both of the following roles:

- Send Role: Clients package user or application-generated data into messages, address messages to target clients, initiate **connections (1)** to relay servers, and enqueue messages to a relay server message store.
- Receive Role: Clients initiate connections to relay servers, dequeue messages from the relay server store, acknowledge message receipt, and present message data to a user interface.

1.3.5.2 Server Role

Using this protocol as a transport, SSTP relay servers accept and efficiently deliver client messages. Relay servers support the following key SSTP message handling services (each of which is further explained in subsequent sections):

- Message storage and delivery to local addresses (those destined for clients assigned to that relay).
- Multicasting of a single message to multiple local addressees (**multi-drop fanout**).

- Message forwarding to remote relays (**single-hop fanout**).

SSTP clients connect to relay servers to enqueue or dequeue messages. SSTP relay servers do not initiate **connections (1)** with clients.

The following diagram illustrates SSTP client and relay server roles:

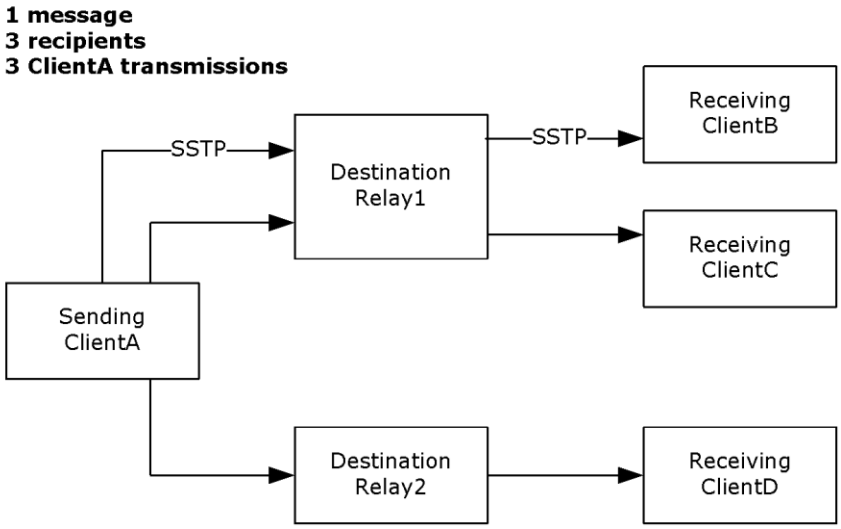


Figure 2: Basic SSTP client-server roles

SSTP relay servers can provide the following additional functionality in conjunction with cooperating protocols.

- Client authentication: Utilizing the SSTP Security protocol, relay servers can assume the role of client authenticator. When SSTP Security is used, the Register command permits client devices to register with an authenticating relay server upon initial connection; the Attach command permits registered clients present valid authentication credentials to dequeue messages. SSTP Security is described separately in [MS-GRVSSTPS].
- Device Presence: With the Wide Area Network Device Presence Protocol (WAN DPP), described in [MS-GRVWDPP], relay servers can assume the role of presence servers that enable client devices to find each other on the Internet. Presence servers collect and disseminate client device identification and online status information via WAN DPP-specific SSTP messages.

1.3.5.2.1 Relay Server Role - Message Store and Forward

Storing and forwarding messages is a primary role of an SSTP relay server. Message store and forward functionality is an integral part of successful SSTP message delivery, providing a means of storing messages while target recipients are offline or unavailable. Each client is assigned to a relay to access these services. Both the mechanism for assigning relay servers to clients and for distributing that information as part of an addressing entry, is application-dependent.

Resource handlers defined on the relay server support ordered enqueuing and dequeuing of message sequences by participating clients. This permits proper handling of data updates in client applications that depend upon sequential message delivery. The implementation details of a relay server message store, where clients can deposit and collect properly addressed SSTP messages, is application-dependent.

To enqueue data in a relay server store, a client opens a **session** specifying an address that contains a resource handler resident on a target client assigned to the relay server. The relay server retains the subsequent message sequences for this session in the local store. When the target client attaches to

(and optionally authenticates with) the relay server, and the relay server has data for any identity or device registered by that client, the relay server opens a session to the client and delivers the messages from the local store.

1.3.5.2.2 Relay Server Role - Fanout

When transmitting large amounts of data to multiple recipients or transmitting over a slow network link, this protocol uses the relay server's fanout capability to expedite communications. Fanout is a process for conveying a stream of data from a client to a relay server for replication and distribution to multiple recipients.

For example, if fanout is employed to expedite delivery of a single message to multiple recipients, a client sends a single copy of the message to each of the relay servers associated with the recipients. Each relay server, similar to a multi-cast router, distributes copies of the message to target clients. Because some recipients share a common relay server, this process helps reduce the number of communications links and reduces bandwidth usage. The client application determines when fanout is applied.

Depending upon the relay server to which the recipients are assigned, fanout can be designated as either multi-drop or single-hop.

1.3.5.2.2.1 Relay Server Role - Multi-Drop Fanout

When a message is addressed to multiple recipients that share the same relay server, the client sends a single copy of the message to the shared destination relay server, which then deposits a copy of the message in a local store for each addressed recipient. The relay server forwards messages from the store to destination clients when they connect to their relay servers. This distribution method is referred to as multi-drop fanout.

The following diagram illustrates multi-drop fanout through Relay1:

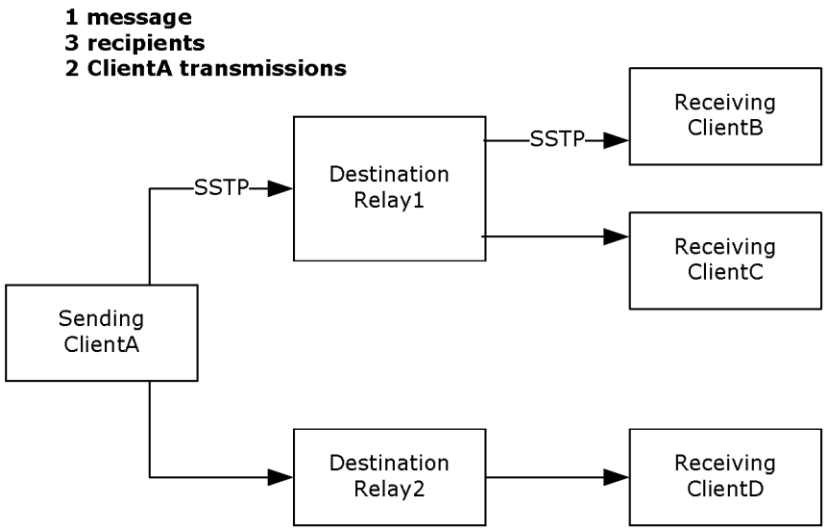


Figure 3: Multi-drop fanout

1.3.5.2.2.2 Relay Server Role - Single-Hop Fanout

When a message is addressed to multiple recipients assigned to different relay servers, the client can send a single copy of the message to its (the sending client's) assigned relay server which then forwards a copy of the message to each destination relay server. The remote relay servers then deposit a copy of the message in a local store for each addressed recipient. The relay servers forward

messages from the store to the destination clients when they connect to their relay servers. This distribution method is referred to as single-hop fanout.

Single-hop fanout extends the multi-drop functionality to include multiple relay servers in a single message delivery. SSTP applications can employ single-hop fanout to reduce network usage by client devices.

The following diagram illustrates single-hop fanout through Relay1:

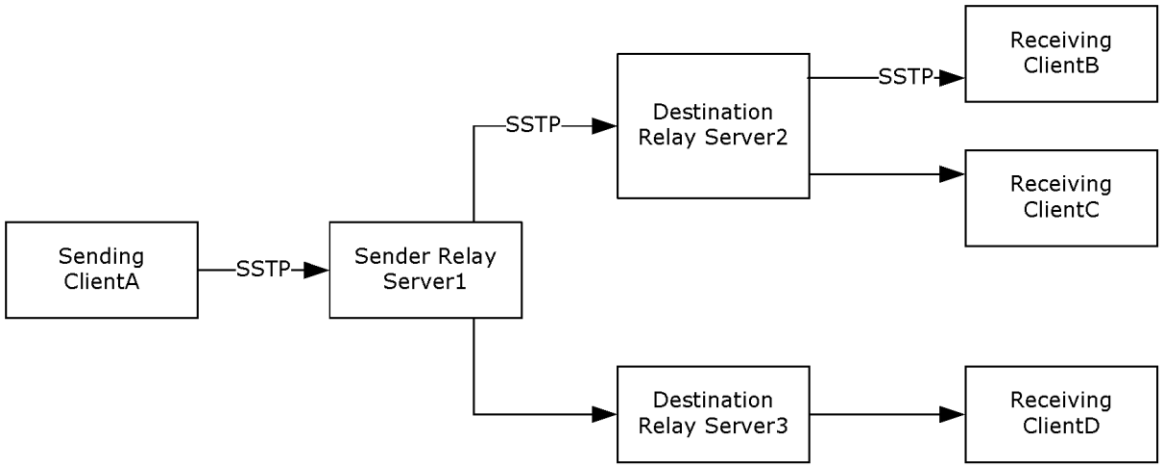


Figure 4: Single-hop fanout

1.3.6 SSTP Communication Examples

The following sections describe a basic SSTP message exchange between two peer devices and a fanout **session** that involves a relay server.

1.3.6.1 Basic SSTP Message Exchange

A basic SSTP **connection (1)** scenario involves two devices communicating over a direct network connection. In this case, if Device1 intends to send a message to a Device2, Device1 begins by establishing an SSTP connection with Device2 and upon receipt of a response from Device2 opens an SSTP **session**. After the expected response from Device2, Device1 sends the message to Device2. Device2 acknowledges receipt and eventually closes the SSTP session. After the Device2 user exits the active application, Device2 closes the SSTP connection.

The following schematic shows message flow over a direct SSTP connection between two peer devices.

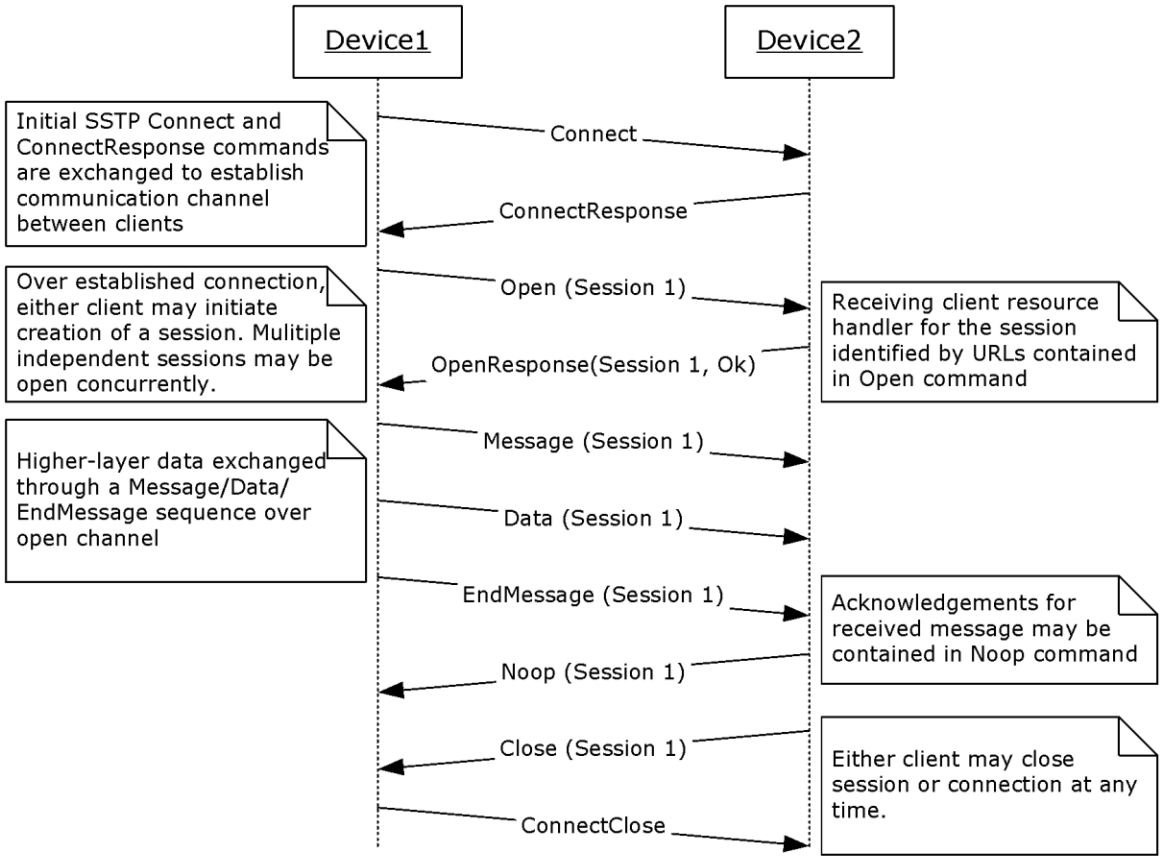


Figure 5: Basic connection and data exchange sequence

1.3.6.2 SSTP Multi-Drop Fanout Message Exchange

The following schematic shows message flow when a relay server is employed to perform multi-drop message storage and distribution, where one of the addressed recipients has connected and authenticated with the relay server:

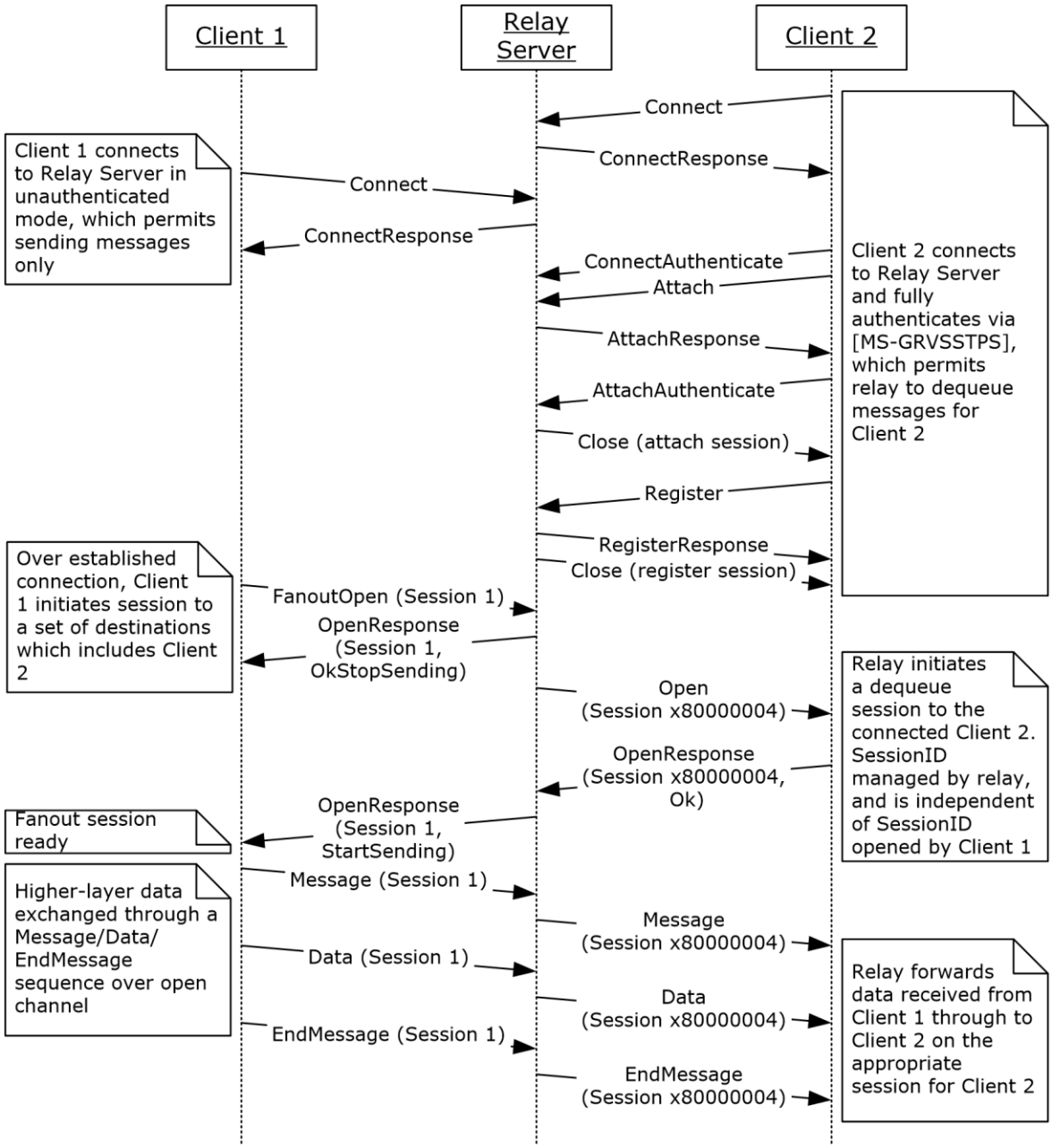


Figure 6: Multi-drop fanout - store and forward to authenticated client

1.3.6.3 SSTP Single-Hop Fanout Message Exchange

The following schematic shows message flow when a relay server is employed to perform single-hop message storage and distribution to a remote relay:

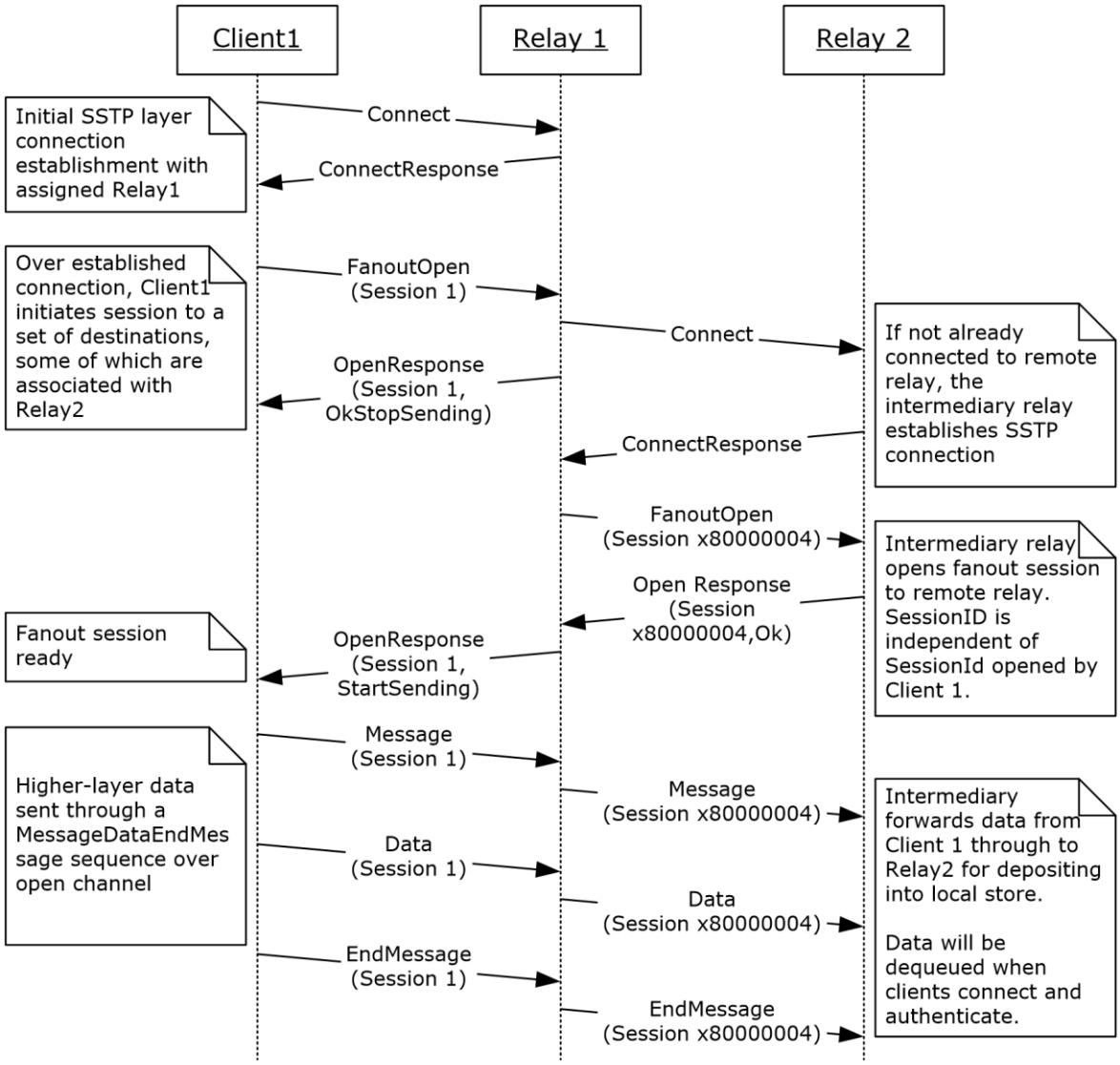


Figure 7: Single-hop fanout - store and forward to a remote relay

1.4 Relationship to Other Protocols

SSTP is designed to augment standard transport protocols with features such as multiplexed messaging to multiple destinations over a single **connection (1)** and application-level message acknowledgments over a stream transport. SSTP operates over TCP, using IANA-registered port 2492/TCP [IANAPORT]. SSTP can also be encapsulated to operate over HTTP, via any registered HTTP port, as described in [MS-GRVHENC].

SSTP Security [MS-GRVSSTPS] is a sub-protocol that supports client authentication in conjunction with SSTP **sessions**.

WAN Device Presence Protocol [MS-GRVWDPP] is a client presence protocol that uses SSTP as its transport.

Groove Dynamics [MS-GRVDYNM] is a synchronization protocol that uses SSTP as its transport.

The following diagram shows the SSTP protocol stack:

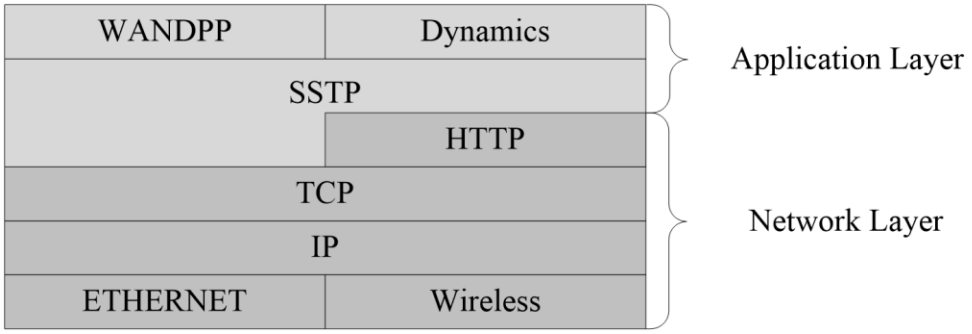


Figure 8: SSTP protocol stack

1.5 Prerequisites/Preconditions

SSTP communications require established TCP **connections (1)** between participating devices.

The IP addresses of destination devices are required to create underlying transport connections.

1.6 Applicability Statement

SSTP is designed to augment standard transport protocols (such as TCP) with the capabilities of sending message acknowledgments over a stream transport, processing messages, and multicasting single messages to multiple application endpoints over a single **connection (1)**. An SSTP client connection (1) to an intermediary SSTP relay server is required to use multicasting features.

1.7 Versioning and Capability Negotiation

Supported Transports: SSTP uses TCP as its transport, as described in section 2.1.

Protocol Versions: The SSTP protocol versions currently specified are versions 1.5 and 1.6; SSTP 1.5 is indicated by MajorVersionNumber of 1 and MinorVersionNumber of 5, and SSTP 1.6 is indicated by MajorVersionNumber of 1 and MinorVersionNumber of 6. SSTP 1.6 introduces several optimization mechanisms for fanout and **session** status commands. To accommodate future version updates, SSTP supports limited capability negotiation via the MajorVersionNumber and MinorVersionNumber fields, as specified in sections 2.2.1.1 and 2.2.2.1.

Security and Authentication Methods: SSTP relies on the SSTP Security protocol for security and authentication, as described in [MS-GRVSSTPS].

Capability Negotiation: SSTP incorporates fields to permit applications to negotiate end-to-end supported device capabilities via inclusion of vendor extensible fields, as described in sections 2.2.1.1 and 2.2.2.1. The SSTP protocol itself does not impose or perform any specific capability negotiations.

Devices with the same major version but different minor versions are compatible.

1.8 Vendor-Extensible Fields

SSTP has the following vendor-extensible fields, as described in section 2.2.1.1:

- Peer Product Version
- Peer Product Capabilities

1.9 Standards Assignments

SSTP operates over TCP on the Internet Assigned Numbers Authority (IANA) registered port 2492 [IANAPORT].

2 Messages

2.1 Transport

SSTP is an application-layer stream-oriented binary protocol, exchanged over a reliable transport layer. All multiple-byte elements in SSTP MUST be treated as **little-endian**, unless otherwise specified.

Designed for operation over TCP via the official IANA-assigned port 2492 [IANAPORT], SSTP can also operate over port 443/TCP or encapsulated over HTTP on port 80/TCP [MS-GRVHENC].

2.2 Message Syntax

SSTP commands consist of a combination of fixed-length and variable-length fields. Each command is limited to a specified length. There are no prescribed limits on the lengths of the variable-length fields in these commands.

Every SSTP message MUST start with a 3-byte message header that contains a one-byte CommandId field indicating the command type, followed by a two-byte CommandLength field indicating the total length of the SSTP message. The maximum length of a command depends on the CommandId, described subsequently.

CommandId values and SSTP command length MUST comply with the specifications in the following table:

Mnemonic	Value	SSTP Command Length in Bytes
Connect	0x01	2055 maximum
ConnectResponse	0x02	2055 maximum
ConnectAuthenticate	0x03	2055 maximum
ConnectClose	0x04	8 or 12
Open	0x05	2055 maximum
FanoutOpen	0x06	65535 maximum
OpenResponse	0x07	Exactly 8
SessionStatus	0x12	2055 maximum
Close	0x11	Exactly 8
Message	0x0d	2055 maximum
Data	0x0e	2055 maximum
EndMessage	0x0f	Exactly 7
Noop	0x10	Exactly 7
Attach	0x08	2055 maximum
AttachResponse	0x09	2055 maximum
AttachAuthenticate	0x0a	2055 maximum

Mnemonic	Value	SSTP Command Length in Bytes
Register	0x0b	8192 maximum
RegisterResponse	0x0c	2055 maximum

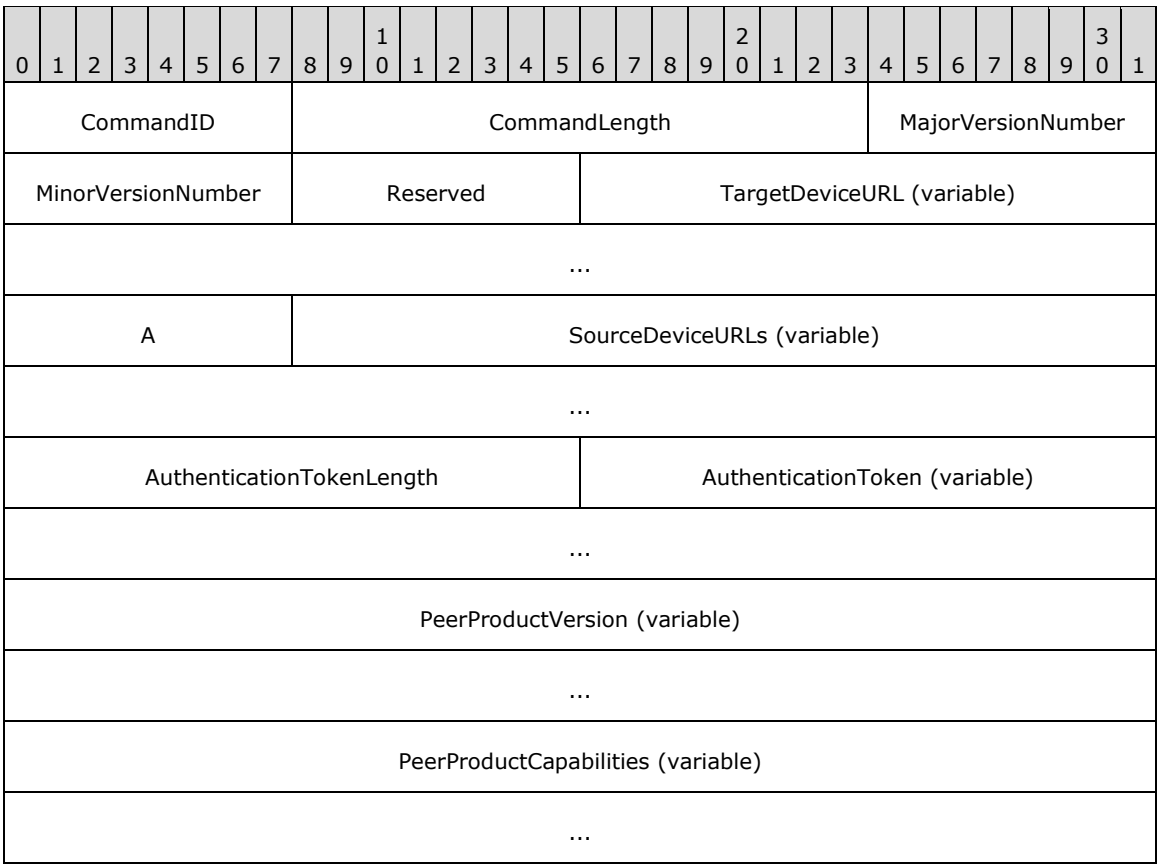
2.2.1 Connect

The Connect command initiates an SSTP **connection (1)** and contains basic connection (1) information about the sending device.

The total length of this command MUST NOT exceed 2055 bytes.

2.2.1.1 Connect Command Fields

Connect command fields are defined in the following table.



CommandID (1 byte): The command identifier. This field MUST be set to 0x01.

CommandLength (2 bytes): The total length of the command, in bytes.

MajorVersionNumber (1 byte): The SSTP major version of the sending device. This field MUST be set to 0x01.

MinorVersionNumber (1 byte): The SSTP minor version number of the sending device. If the sending device supports SSTP 1.6, this field MUST be set to 0x06; otherwise, this field MUST be set to 0x05.

Reserved (1 byte): A reserved field that MUST be set to 0x00.

TargetDeviceURL (variable): A variable-length ASCII string terminated by the byte 0x00 containing the expected **DeviceURl** of the device that is receiving this message.

A - NumSourceDeviceURLs (1 byte): The number of device URLs in the **SourceDeviceURLs** field of this command.

SourceDeviceURLs (variable): A list of variable-length ASCII strings, each terminated by the byte 0x00, that identify the sending device. The number of strings MUST be as specified in the **NumSourceDeviceURL** field.

AuthenticationTokenLength (2 bytes): The length in bytes of the **AuthenticationToken** field of the **Connect** command.

AuthenticationToken (variable): A byte sequence that is used to authenticate the connecting device. The length of this byte sequence MUST be as specified in the **AuthenticationTokenLength** field. The receiving device interprets the contents of this token as an SSTP Security Protocol message, as specified in [MS-GRVSSTPS].

PeerProductVersion (variable): A variable-length ASCII string terminated by 0x00 containing the sender's product version.<1> This field is informational and usage is application-dependent. The field SHOULD contain a series of application-defined tokens, where each token is separated by a space, and MUST contain at least one token.

PeerProductCapabilities (variable): A variable length ASCII string terminated by 0x00 containing the sender's product capabilities.<2> This field is informational, and usage is application-dependent The format of this string is "<*token1*>;<*token2*>;…;<*tokenN*>" where each of <*token1*> through <*tokenN*> MUST be separated by a semicolon (;). Each token is a string and MUST NOT contain leading or trailing spaces. Tokens SHOULD be short capitalized abbreviations and SHOULD NOT contain spaces. An example of this string is as follows:

```
ABC;MDF
```

2.2.2 ConnectResponse

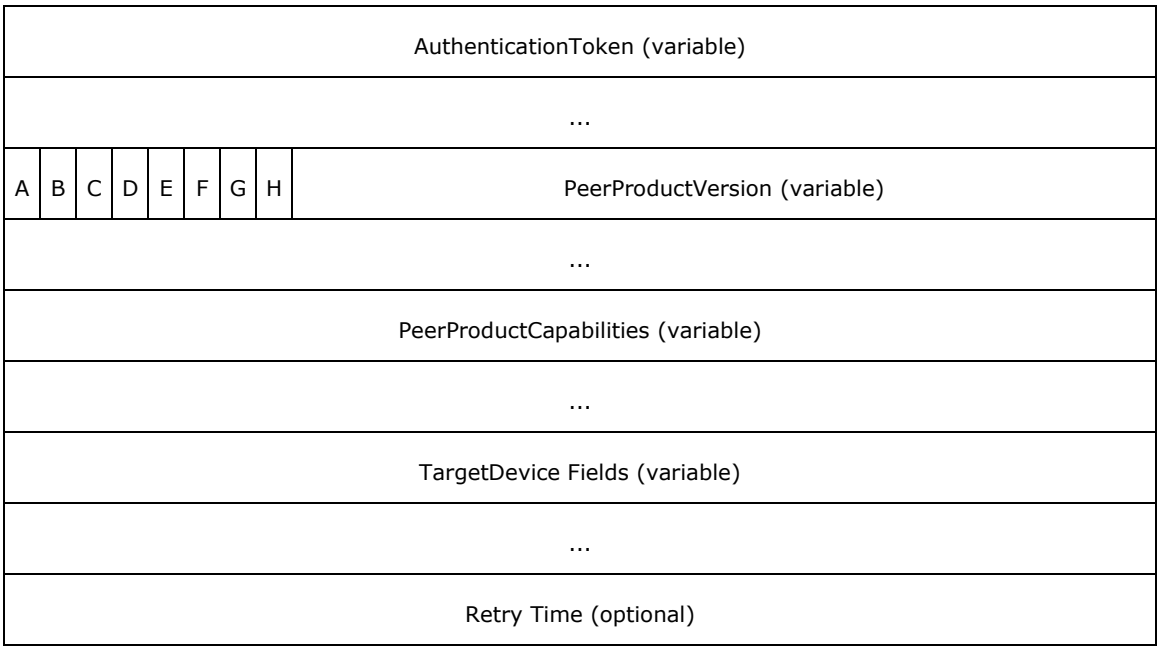
The **ConnectResponse** command indicates the success or failure of an SSTP **connection (1)** request and conveys basic device information. A device sends a **ConnectResponse** command upon receipt of a **Connect** command from another device.

The total length of this message MUST NOT exceed 2055 bytes.

2.2.2.1 ConnectResponse Command Fields

ConnectResponse command fields are defined as shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
CommandID								CommandLength																MajorVersionNumber							
MinorVersionNumber								ResponseId												AuthenticationTokenLength											



CommandID (1 byte): The command identifier. This field MUST be set to 0x02.

CommandLength (2 bytes): The total length of the command, in bytes.

MajorVersionNumber (1 byte): The SSTP major version number of the sending device. This field MUST be set to 0x01.

MinorVersionNumber (1 byte): The SSTP minor version number of the sending device. If the sending device supports SSTP 1.6 then this field MUST be set to 0x06; otherwise this field MUST be set to 0x05.

ResponseId (1 byte): A value indicating success or failure of the **Connect** request, as described in the following table.

Value	Mnemonic	Description
0x00	Ok	The **connection (1)** attempt is successful.
0x01	WrongDevice	The device URL of the sending device is not the same as the device URL specified in the **TargetDeviceURL** field of the corresponding **Connect** command.
0x02	TryLater	The sending device is requesting that the receiving device close the connection (1) and SHOULD NOT attempt to reconnect for the amount of time specified in the **RetryTime** field of this command.
0x03	WillUpgrade	The sending device is running a lower and potentially incompatible version of SSTP than the receiving device.
0x04	WontUpgrade	The sending device is running a version of SSTP that is incompatible with the SSTP version on the receiving device.
0x05	NewVersionRequired	The sending device is running a higher SSTP major version number than the receiving device.
0x06	AuthenticationFailed	The sending device was unable to authenticate the receiving device using the information provided in the **AuthenticationToken** of the **Connect** command.

Value	Mnemonic	Description
0x09	ConnectRejected	The connection (1) from the receiving device is blocked because of a security lockout as defined in [MS-GRVSPMR].

AuthenticationTokenLength (2 bytes): The length in bytes of the **AuthenticationToken** field.

AuthenticationToken (variable): A byte sequence that is used to authenticate the connecting device. The length of this byte sequence MUST be as specified in the **AuthenticationTokenLength** field. The receiving device interprets the contents of this token as an SSTP Security Protocol message, as specified in [MS-GRVSSTPS].

A - r1 (1 bit, variable): This field is unused and MUST be set to zero. If the ResponseId is NewVersionRequired, this field does not exist in the ConnectResponse message.

B - r2 (1 bit, variable): This field is unused and MUST be set to zero. If the ResponseId is NewVersionRequired, this field does not exist in the ConnectResponse message.

C - r3 (1 bit, variable): This field is unused and MUST be set to zero. If the ResponseId is NewVersionRequired, this field does not exist in the ConnectResponse message.

D - r4 (1 bit, variable): This field is unused and MUST be set to zero. If the ResponseId is NewVersionRequired, this field does not exist in the ConnectResponse message.

E - r5 (1 bit, variable): This field is unused and MUST be set to zero. If the ResponseId is NewVersionRequired, this field does not exist in the ConnectResponse message.

F - C (1 bit, variable): This field is unused and MUST be set to zero. If the ResponseId is NewVersionRequired, this field does not exist in the ConnectResponse message.

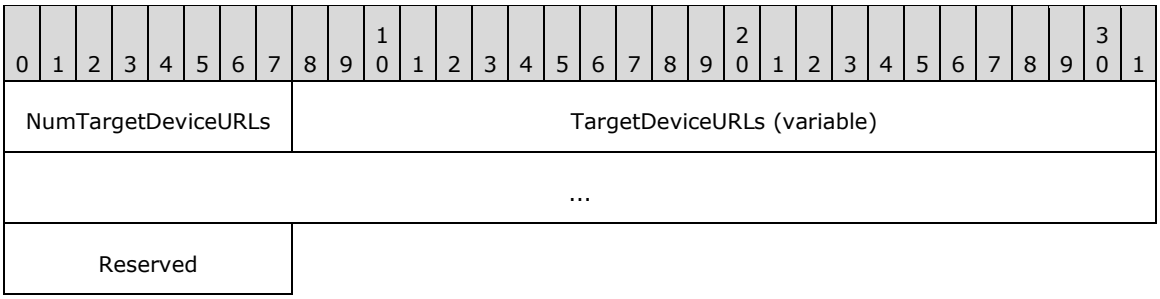
G - S (1 bit, variable): If set to 1, the sending device supports single-hop fanout via **sessions** opened with the **FanoutOpen** command. An alternate name for this field is **SingleHopFanout**. If the ResponseId is NewVersionRequired, this field does not exist in the ConnectResponse message.

H - M (1 bit, variable): If set to 1, the sending device supports multi-drop fanout via sessions opened with the **FanoutOpen** command. An alternate name for this field is **Multi-dropFanout**. If the ResponseId is NewVersionRequired, this field does not exist in the ConnectResponse message.

PeerProductVersion (variable): See section 2.2.1.1.

PeerProductCapabilities (variable): See section 2.2.1.1.

TargetDevice Fields (variable): If the **ResponseID** field contains a value of "Ok", the **NumTargetDeviceURLs**, **TargetDeviceURLs** and **Reserved1** fields MUST be present. Otherwise, these fields MUST NOT be present. These fields are specified as shown in the following table.



NumTargetDeviceURLs (1 byte): The number of device URLs in the **TargetDeviceURLs** field of this command.

TargetDeviceURLs (variable): A list of variable-length ASCII strings, each terminated by the byte 0x00, that identify the sending device.

Reserved (1 byte): This field is reserved and MUST be 0x00.

Retry Time (4 bytes, optional): The time, in seconds, that the receiving device SHOULD wait before attempting another connection (1) to the sending device. If the **ResponseID** field contains a value of **TryLater** or **WillUpgrade**, the **RetryTime** field MUST be present. Otherwise, this field MUST NOT be present.

2.2.3 ConnectAuthenticate

The **ConnectAuthenticate** command is used to validate that the sender can respond correctly to the security challenge provided in the previous **ConnectResponse** command, as specified in [MS-GRVSSTPS].

The total length of this message MUST NOT exceed 2055 bytes.

2.2.3.1 ConnectAuthenticate Command Fields

The **ConnectAuthenticate** command fields are defined as shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
CommandId								CommandLength												A											
...								AuthenticationToken (variable)																							
...																															

CommandId (1 byte): The command identifier. This field MUST be set to 0x03.

CommandLength (2 bytes): The total length of the command, in bytes.

A - AuthenticationTokenLength (2 bytes): The length in bytes of the **AuthenticationToken** field.

AuthenticationToken (variable): A byte sequence that is used to authenticate the connecting device. The length of this byte sequence MUST be as specified in the AuthenticationTokenLength field. The receiving device interprets the contents of this token as an SSTP Security protocol message, as specified in [MS-GRVSSTPS].

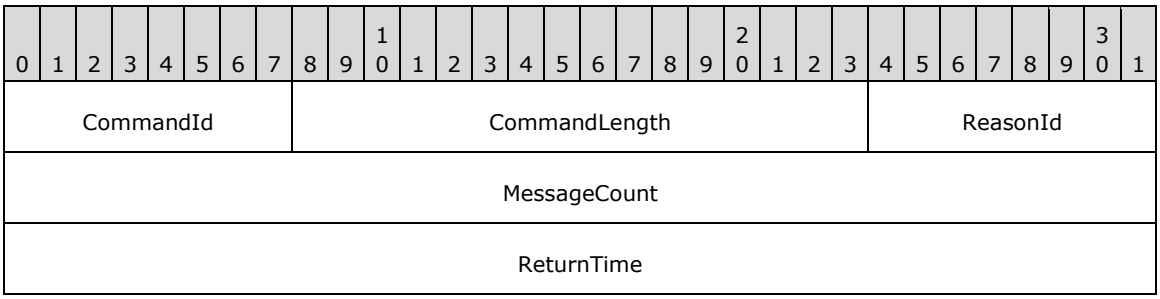
2.2.4 ConnectClose

The **ConnectClose** command is used to close the SSTP **connection (1)**.

If the **ReasonId** field is set to the value 0x01 (Resting), the **ReturnTime** field MUST be present, and the total length of this message MUST be 12 bytes. Otherwise, the total length of this message MUST be 8 bytes.

2.2.4.1 ConnectClose Command Fields

ConnectClose command fields are defined as follows:



CommandId (1 byte): The command identifier. This field MUST be set to 0x04.

CommandLength (2 bytes): The total length of the command, in bytes.

ReasonId (1 byte): This field MUST be set to one of the values in the following table indicating the reason for closing the **connection (1)**.

Value	Mnemonic	Description
0x00	NoReason	This is the default value.
0x01	Resting	The sending device cannot accept any more data and is closing the connection (1). The receiving device SHOULD NOT attempt to reconnect for the amount of time specified in the **ReturnTime** field.
0x02	Idle	The sending device higher-layer has detected that the connection (1) is idle and is closing the connection (1).
0x03	ProtocolError	The sending device has detected that the receiving device has generated an SSTP protocol violation and is closing the connection (1).
0x04	DeviceAuthenticationFailed	The sending device was unable to authenticate the receiving client device.
0x05	UserAuthenticationFailed	The sending device was unable to authenticate the account specified in a previous **Attach** command.
0x06	StaleConnectAuthenticate	The receiving client device did not respond with the correct **ConnectAuthenticate** command to the security challenge provided by the sending device in a previous **ConnectResponse** command.
0x07	StaleAttachAuthenticate	The receiving client device did not respond with a correct **AttachAuthenticate** command to the security challenge provided in a previous **AttachResponse** command.
0x08	ResponseTimeout	The sending device has not received an expected response to a previous SSTP command. This **ReasonId** MAY be used by a higher-layer which implements optional connection (1) and **session** timeouts, as specified in section 3.1.2.
0x09	Rejected	The connection (1) from the receiving client device is blocked because of a security lockout as defined in [MS-GRVSPMR].

Value	Mnemonic	Description
0x0a	DecryptionFailed	The connection (1) from the receiving client device is blocked because of a cryptographic incompatibility as specified in [MS-GRVSSTPS].
0x0c	CrossedConnections	The sending device has detected that an SSTP connection (1) already exists between the sending and the receiving devices and is closing the connection (1).
0x0d	InternalError	The sending device encountered an unexpected condition, and is closing the connection (1).
0x0e	Upgrade	The sending device is closing the connection (1) because of SSTP version incompatibility.
0x0f	TooManyUnknownSessionCmds	The sending device has detected that the receiving device has sent commands on a session that is not currently open.
0x10	NewVersionRequired	The sending device has detected that the receiving device is running an incompatible version of SSTP, and is closing the connection (1).

MessageCount (4 bytes): The number of SSTP messages that the sending device acknowledges it has successfully received and processed.

ReturnTime (4 bytes): The time in seconds that the receiving device SHOULD wait before reconnecting to the sending device. This optional field MUST be present only if the **ReasonId** field, described previously, is set to 0x01 (Resting).

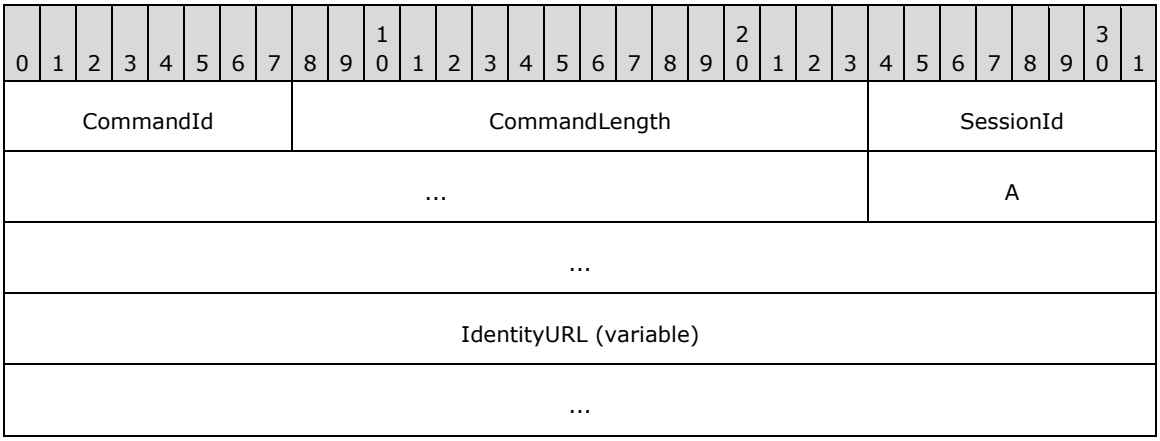
2.2.5 Open

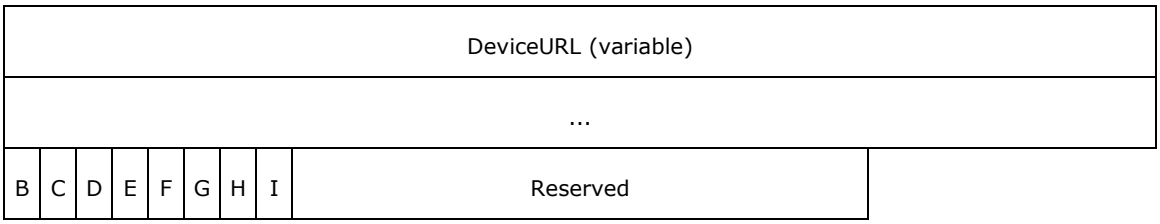
The Open command is used to open an SSTP **session** between two connected SSTP devices. The Open command identifies the destination using an addressing entry that contains the target resource URL, identity URL, and device URL.

The total length of this message MUST NOT exceed 2055 bytes.

2.2.5.1 Open Command Fields

Open command fields are defined as shown in the following table.





CommandId (1 byte): The command identifier. This field MUST be set to 0x05.

CommandLength (2 bytes): The total length of the command, in bytes.

SessionId (4 bytes): The identifier for the SSTP **session**, as specified in section 3.1.1.3.

A - ResourceURL (variable): A variable length ASCII string terminated by 0x00 containing the URL of the resource handler for all messages on this SSTP session. This field MUST NOT be an empty string.<3>

IdentityURL (variable): A variable length ASCII string terminated by 0x00 containing the identity URL of the destination for all messages on this SSTP session. For any session other than a WAN DPP session, this field MUST NOT be an empty string.<4>

DeviceURL (variable): A variable length ASCII string terminated by 0x00 containing the device URL of the destination for all messages on this SSTP session. For identity-targeted sessions, this field MUST be an empty string.

B - r1 (1 bit): This field is reserved and MUST be set to zero.

C - r2 (1 bit): This field is reserved and MUST be set to zero.

D - r3 (1 bit): This field is reserved and MUST be set to zero.

E - r4 (1 bit): This field is reserved and MUST be set to zero.

F - r5 (1 bit): This field is reserved and MUST be set to zero.

G - r6 (1 bit): This field is reserved and MUST be set to zero.

H - r7 (1 bit): This field is reserved and MUST be set to zero.

I - I (1 bit): The field is unused, and SHOULD be zero<5>. A receiver MUST ignore this bit.

Reserved (2 bytes): The field is reserved and MUST be 0x0000.

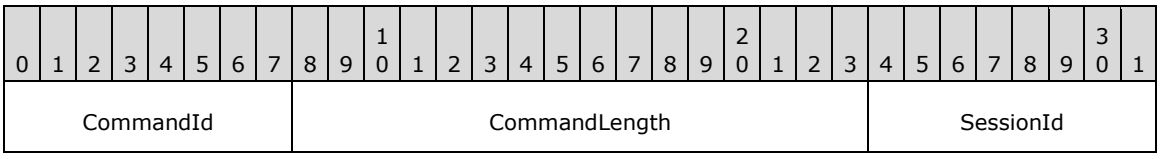
2.2.6 FanoutOpen

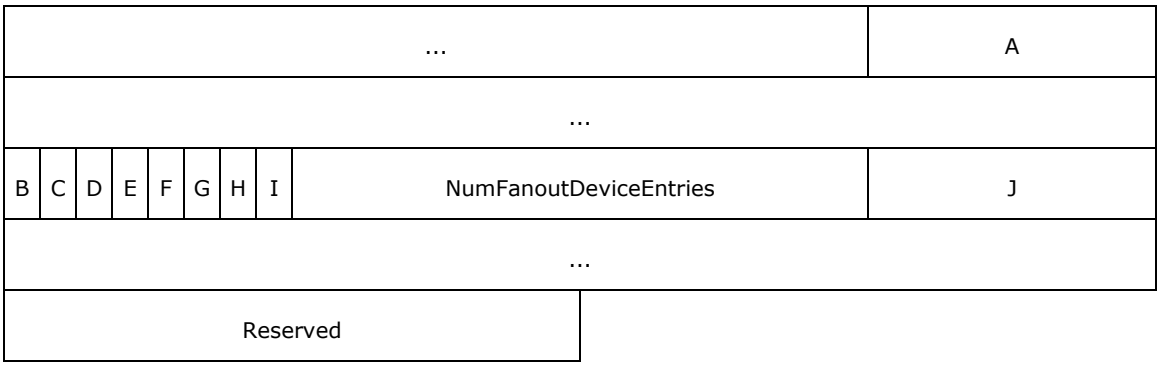
The **FanoutOpen** command opens a **fanout** SSTP **session** from a device to a relay server.

The total length of this message MUST NOT exceed 65535 bytes.

2.2.6.1 FanoutOpen Command Fields

FanoutOpen command fields are defined as follows:





CommandId (1 byte): The command identifier. This field MUST be set to 0x06.

CommandLength (2 bytes): The total length of the command, in bytes.

SessionId (4 bytes): The identifier for the SSTP **session** as specified in section 3.1.1.3.

A - ResourceURL (variable): A variable length ASCII string terminated by 0x00 containing the URL of the resource handler for all messages on this SSTP session. This field MUST NOT be an empty string.

B - r1 (1 bit): This field is reserved and MUST be set to zero.

C - r2 (1 bit): This field is reserved and MUST be set to zero.

D - r3 (1 bit): This field is reserved and MUST be set to zero.

E - r4 (1 bit): This field is reserved and MUST be set to zero.

F - r5 (1 bit): This field is reserved and MUST be set to zero.

G - r6 (1 bit): This field is reserved and MUST be set to zero.

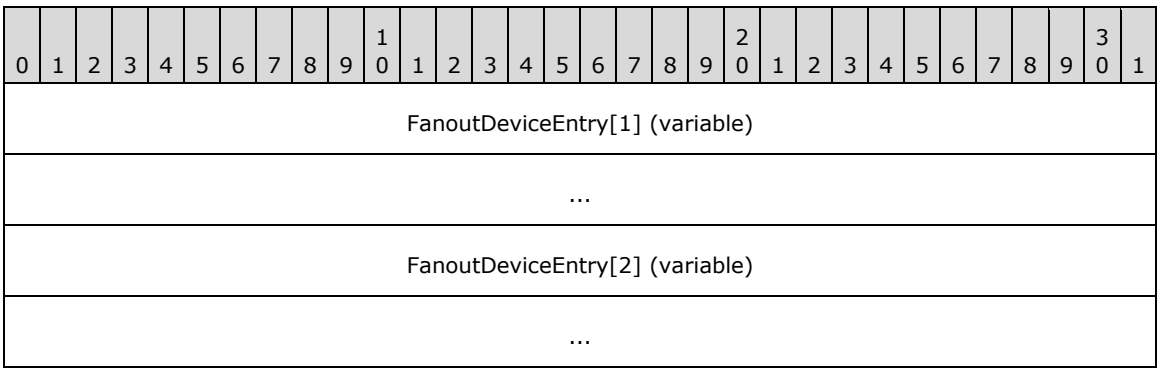
H - r7 (1 bit): This field is reserved and MUST be set to zero.

I - I (1 bit): This field is reserved and SHOULD<6> be zero. A receiver MUST ignore this bit.

NumFanoutDeviceEntries (2 bytes): The total number of entries in the **FanoutDeviceEntries** field.

J - FanoutDeviceEntries (variable): A variable length list of fanout device entries. The **FanoutDeviceEntries** list MUST contain exactly as many entries as indicated in the preceding **NumFanoutDeviceEntries** field.

FanoutDeviceEntries command variable fields are defined as shown in the following table.



FanoutDeviceEntry[n] (variable)
...

FanoutDeviceEntry[1] (variable): First **FanoutDeviceEntry**.

FanoutDeviceEntry[2] (variable): Second **FanoutDeviceEntry**.

FanoutDeviceEntry[n] (variable): Last **FanoutDeviceEntry**.

In SSTP 1.5, each fanout device entry is composed of the three variable-length ASCII strings, each terminated by 0x00: identity URL, device URL and relay URL. Each string indicates (to the relay server) the destination resource handler that will process sent messages received on the fanout session. The **IdentityURL** field of each entry MUST NOT be empty. The **DeviceURL** field of each entry MAY<7> be empty. If the resource handler associated with the recipient relay server is the destination of messages on this session, the **RelayURL** field<8> MUST be empty. If the **RelayURL** field is empty, a resource handler associated with the recipient relay server is the destination of messages on this session.

In SSTP 1.6, the fanout device entry is composed of four variable-length ASCII strings, each terminated by 0x00. A new **FailoverDeviceURLs** is included, in addition to the **IdentityURL**, **DeviceURL**, and **RelayURL** fields described in the preceding paragraph. The **FailoverDeviceURLs** field MUST be set to 0x00.

The SSTP 1.5 **FanoutDeviceEntries** consisting of **IdentityURL**, **DeviceURL**, and **RelayURL** fields are defined as shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FanoutDeviceEntry[i].IdentityURL (variable)																															
...																															
FanoutDeviceEntry[i].DeviceURL (variable)																															
...																															
FanoutDeviceEntry[i].RelayURL (variable)																															
...																															

FanoutDeviceEntry[i].IdentityURL (variable): ASCII string terminated by 0x00.

FanoutDeviceEntry[i].DeviceURL (variable): ASCII string terminated by 0x00.

FanoutDeviceEntry[i].RelayURL (variable): ASCII string terminated by 0x00.

The SSTP 1.6 **FanoutDeviceEntries** consisting of **IdentityURL**, **DeviceURL**, **RelayURL**, and **FailoverDeviceURLs** fields are defined as shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
FanoutDeviceEntry[i].IdentityURL (variable)																															
...																															
FanoutDeviceEntry[i].DeviceURL (variable)																															
...																															
FanoutDeviceEntry[i].RelayURL (variable)																															
...																															
FanoutDeviceEntry[i].FailoverDeviceURLs (variable)																															
...																															

FanoutDeviceEntry[i].IdentityURL (variable): ASCII string terminated by 0x00.

FanoutDeviceEntry[i].DeviceURL (variable): ASCII string terminated by 0x00.

FanoutDeviceEntry[i].RelayURL (variable): ASCII string terminated by 0x00.

FanoutDeviceEntry[i].FailoverDeviceURLs (variable): ASCII string terminated by 0x00.

Reserved (2 bytes): This field is reserved and MUST be 0x0000.

2.2.7 OpenResponse

The OpenResponse command is sent in response to an Open or FanoutOpen command to notify the receiving device of the **session** status. OpenResponse commands are also sent to notify of session status changes.

The total length of this message MUST be 8 bytes.

2.2.7.1 OpenResponse Command Fields

OpenResponse command fields are defined as shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
CommandId								CommandLength																SessionId							
...																								ResponseId							

CommandId (1 byte): The command identifier. This field MUST be set to 0x07.

CommandLength (2 bytes): The total length of the command, in bytes.

SessionId (4 bytes): The identifier for the SSTP **session**, as specified in section 3.1.1.3.

ResponseId (1 byte): The **ResponseId** field MUST have one of the values in the following table, indicating success or failure of the **Open** request.

Value	Mnemonic	Description
0x00	Ok	The SSTP session has been successfully opened and the responder is ready to receive messages.
0x04	NoResource	The fanout session request to a WAN DPP presence server is rejected.
0x05	Unknown	The session could not be opened because of an undefined error.
0x08	NoFanoutEntries	The fanout session request did not contain valid fanout address list entries.
0x09	StartSending	The responding device is ready to receive messages on the session.
0x0a	StopSending	The responding cannot receive messages on the session.
0x0b	OkStopSending	The session can be open but messages cannot be received.
0x0c	FanoutNotSupported	The sending device does not support single-hop fanout, and a remote relay was specified in the **FanoutDeviceEntries** list of a **FanoutOpen** command.

2.2.8 SessionStatus

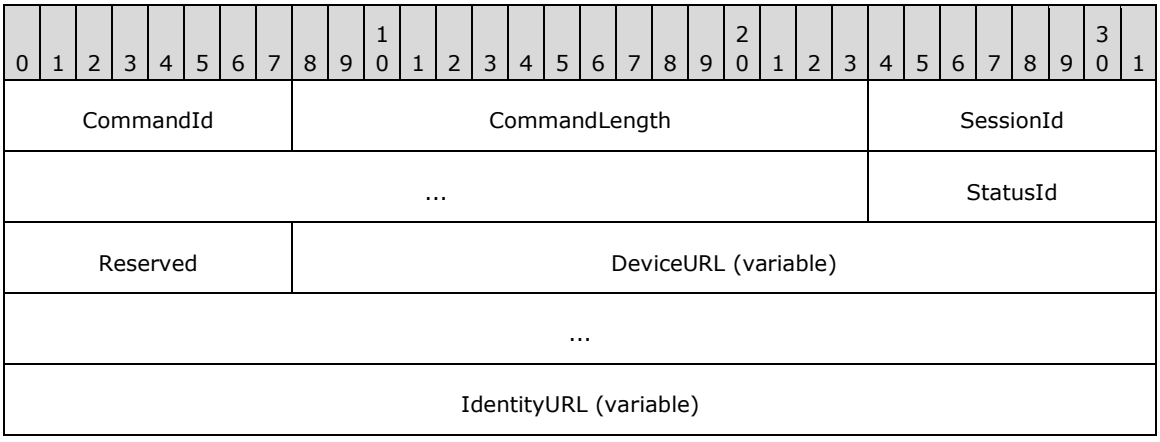
The **SessionStatus** command indicates that the specified destinations are no longer part of the fanout **session**.

The total length of this command MUST NOT exceed 2055 bytes.

There are two different types of **SessionStatus** commands, where the first type is supported by both SSTP 1.5 and 1.6, whereas the second type is only supported by SSTP 1.6.

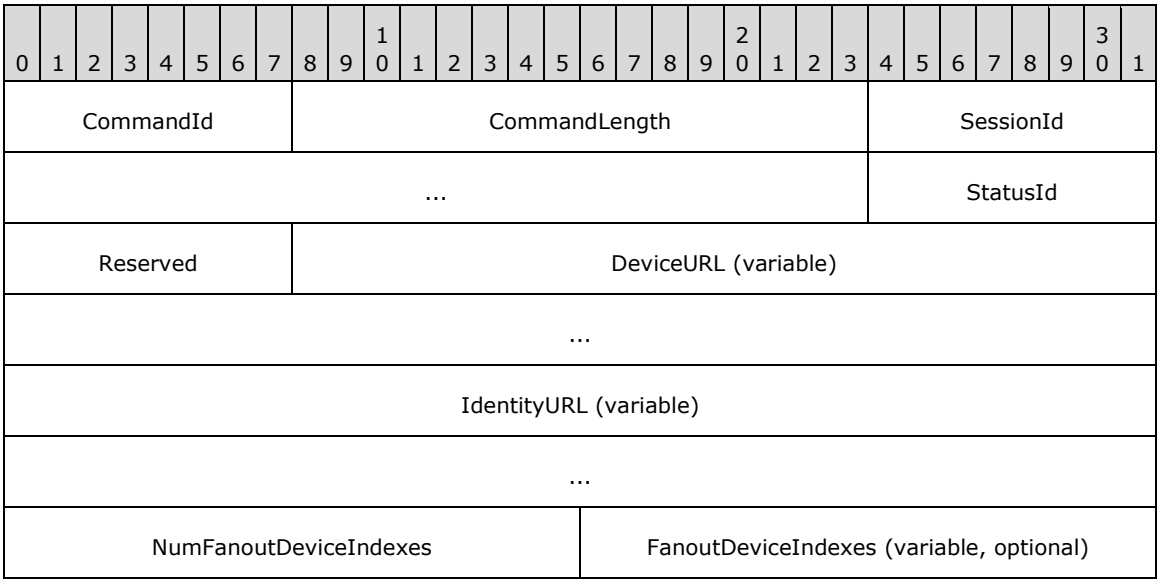
2.2.8.1 SessionStatus Command Fields

SSTP 1.5 and SSTP 1.6 SessionStatus command fields with a single fanout device are defined as shown in the following table.



...

SSTP 1.6 SessionStatus command fields with multiple fanout devices are defined as shown in the following table.



CommandId (1 byte): The command identifier. This field MUST be set to 0x12.

CommandLength (2 bytes): The total length of the command, in bytes.

SessionId (4 bytes): The identifier for the SSTP **session**, as specified in section 3.1.1.3.

StatusId (1 byte): This field MUST be one of the values described in the following table indicating the cause of the status change.

Value	Mnemonic	Description
0x01	DNSLookupFailed	The DNS lookup of the relay server specified in the **DeviceURL** field failed.
0x02	HostNotReachable	A TCP **connection (2)** to the remote relay server specified in the **DeviceURL** field could not be established.
0x03	ConnectionClosed	An existing forwarding connection (1) to the remote relay specified in the **DeviceURL** field was lost.
0x04	QuotaWouldBeExceeded	The specified recipient no longer has space available to store messages for this session.
0x05	LockedOut	The specified recipient is no longer permitted to store messages for this session.

Reserved (1 byte): This field is reserved, and MUST be 0x00.

DeviceURL (variable): A variable length ASCII string terminated by 0x00 containing the device URL of the device affected by this command.

In SSTP 1.5, if the **StatusId** is 0x01, 0x02, or 0x03 this MUST be a relay URL and indicates that all fanout destinations on this relay server are no longer part of this session.

In SSTP 1.5, if the **StatusId** is one of 0x04 or 0x05 this MUST be a client device URL and indicates that the destination specified by the combination of this **DeviceURL** field and the **IdentityURL** field is no longer part of this session.

In SSTP 1.6, for optimization, a single **SessionStatus** command can specify multiple devices that have all undergone the same status change. Multiple devices are specified by the optional **NumFanoutDeviceIndexes** and **FanoutDeviceIndexes** fields. If only one fanout target has failed, the **DeviceURL** field MUST be a client device URL and indicates that the destination specified by the combination of this **DeviceURL** field and the **IdentityURL** field is no longer part of this session. In addition, **NumFanoutDeviceIndexes** MUST be set to 0x0000 and the **FanoutDeviceIndexes** field MUST be omitted.

In SSTP 1.6, if more than one fanout device has failed, both the **DeviceURL** and **IdentityURL** fields MUST be set to an empty string, and the **NumFanoutDeviceIndexes** and **FanoutDeviceIndexes** fields MUST be used to specify the devices.

IdentityURL (variable): A variable length ASCII string terminated by 0x00 containing the identity URL of a recipient.

In SSTP 1.5, if the **StatusId** is one of 0x01, 0x02, or 0x03 this MUST be set to an empty string.

In SSTP 1.5, if the **StatusId** is one of 0x04 or 0x05 this MUST NOT be an empty string.

In SSTP 1.6, for optimization, a single **SessionStatus** command can specify multiple devices that have all undergone the same status change. Multiple devices are specified by the optional **NumFanoutDeviceIndexes** and **FanoutDeviceIndexes** fields. If only one fanout target has failed, the **IdentityURL** field MUST NOT be an empty string. In addition, **NumFanoutDeviceIndexes** MUST be set to 0x0000 and the **FanoutDeviceIndexes** field MUST be omitted.

In SSTP 1.6, if more than one fanout device has failed, both the **DeviceURL** and **IdentityURL** fields MUST be set to the empty string, and the **NumFanoutDeviceIndexes** and **FanoutDeviceIndexes** fields MUST be used to specify the devices.

NumFanoutDeviceIndexes (2 bytes): The total number of fanout device indices in the list **FanoutDeviceIndexes** field. This is an optional field introduced in SSTP 1.6 **SessionStatus** messages. In SSTP 1.5, this field MUST NOT exist.

In SSTP 1.6, if there is only one fanout device on which to report status (as was the case with all SSTP 1.5 **SessionStatus** messages), the **DeviceURL** and **IdentityURL** fields MUST be used instead. In this case, the **NumFanoutDeviceIndexes** field MUST be set to 0x0000 and the **FanoutDeviceIndexes** field MUST NOT exist.

FanoutDeviceIndexes (variable, optional): A variable length list of encoded fanout device entries. This is an optional field introduced in SSTP 1.6 **SessionStatus** messages, for optimization, that allows a single **SessionStatus** command to specify the status of multiple fanout devices. In SSTP 1.5, this field MUST NOT exist.

If the field is present, the **FanoutDeviceEntries** list MUST contain exactly as many entries as indicated in the preceding **NumFanoutDeviceEntries** field.

The **FanoutDeviceIndexes** field is a zero-based index. A **FanoutDeviceIndexes** entry is a 2-byte unsigned integer that is used as an index to locate a fanout device entry in the original **FanoutOpen** command. An index MUST be less than the size of the fanout device entries in the associated **FanoutOpen** command and SHOULD NOT<9> be sent in redundant **SessionStatus** commands for the same associated **FanoutOpen** command.

If **NumFanoutDeviceIndexes** is set to 0x0000, this indicates that there is only one fanout device specified in this **SessionStatus** command. In this case, the **IdentityURL** field MUST be used instead, the **DeviceURL** field MUST be NULL, and the **FanoutDeviceIndexes** field MUST NOT exist.

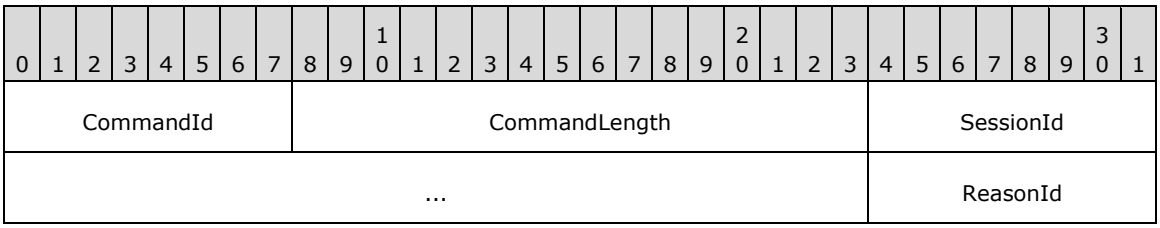
2.2.9 Close

The **Close** command ends an SSTP **session** on a **connection (1)**.

The total length of this message MUST be 8 bytes.

2.2.9.1 Close Command Fields

The **Close** command fields are defined as shown in the following table.



CommandId (1 byte): The command identifier. This field MUST be set to 0x11.

CommandLength (2 bytes): The total length of the command, in bytes.

SessionId (4 bytes): The identifier for the SSTP **session**, as specified in section 3.1.1.3.

ReasonId (1 byte): This field MUST be one of the reason identifiers described in the following table indicating the reason for closing the session.

Value	Mnemonic	Description
0x00	NoReason	This is the default value.
0x02	Idle	The sending device higher-layer has detected that the session is idle and is closing the session.
0x03	ProtocolError	The sending device has detected an SSTP Security message protocol violation and is closing the session.
0x04	DeviceAuthenticationFailed	The sending device was unable to authenticate the receiving client device.
0x05	UserAuthenticationFailed	The sending device was unable to authenticate the account specified in a previous Attach command.
0x07	StaleAttachAuthenticate	The receiving client device did not respond with a correct **AttachAuthenticate** command to the security challenge provided in a previous **AttachResponse** command.
0x0b	QuotaWouldBeExceeded	The identity's message quota has been exceeded on the relay server.
0x0d	InternalError	The sending device encountered an unexpected condition and is closing the **connection (1)**.

Value	Mnemonic	Description
0x15	EmptySession	The relay server will not accept data sent by the receiving device for this session because all the remote relay servers or local recipients of the fanout session are either unavailable or can no longer accept data.

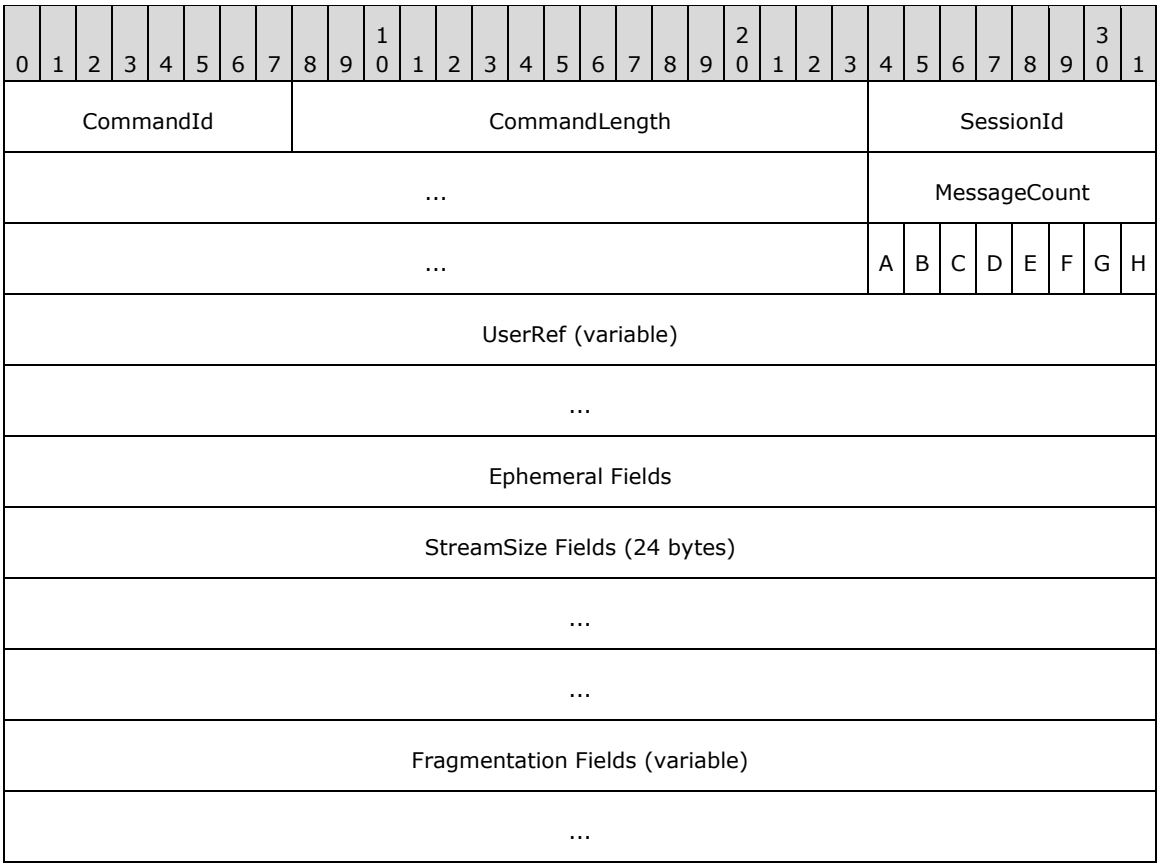
2.2.10 Message

The Message command begins a Message/Data**...**Data**...**Data/End Message sequence and provides specific details about the message sequence.

The total length of this message MUST NOT exceed 2055 bytes.

2.2.10.1 Message Command Fields

The **Message** command fields are defined as shown in the following table.



CommandId (1 byte): The command identifier. This field MUST be set to 0x0d.

CommandLength (2 bytes): The total length of the command, in bytes.

SessionId (4 bytes): The identifier for the SSTP **session**, as specified in section 3.1.1.3.

MessageCount (4 bytes): The number of SSTP messages that the sending device acknowledges it has successfully received and processed. Although this field is contained within a session message, this is a connection-level acknowledgment.

A - r1 (1 bit): This field is reserved and MUST be set to zero.

B - F (1 bit): If set to 1, this value indicates that the **Fragmentation Fields** are present.

C - G (1 bit): If set to 1, this value indicates that the receiving application SHOULD track this message and report message status to the resource handler.

D - S (1 bit): If set to 1, this value indicates that the **StreamSize Fields** are present.

E - r2 (1 bit): This field is reserved and MUST be set to zero.

F - A (1 bit): If set to 1, this value indicates that the sending device is requesting that the receiving device acknowledge this message immediately. An alternate name for this field is **AcknowledgeImmediately**.

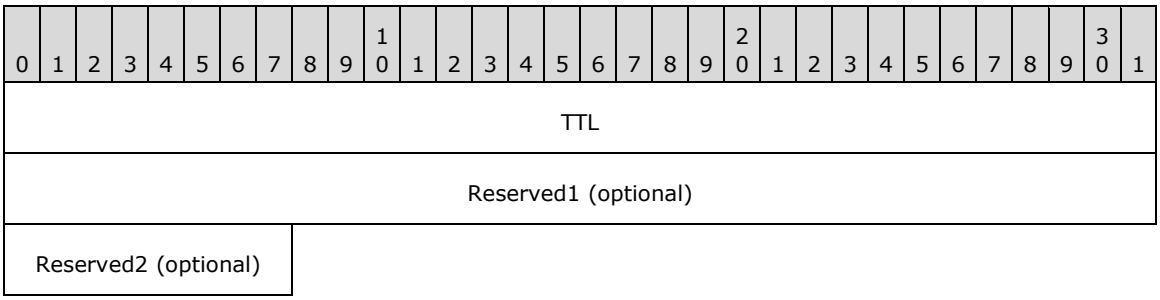
G - E (1 bit): If set to 1, this value indicates a message has **Ephemeral Fields** present. An alternate name for this field is **Ephemeral**.

H - D (1 bit): If set to 1, this value indicates that the sending device is requesting that the message be discarded by the relay server if the destination device is offline. An alternate name for this field is **DoNotDeliverIfOffline**.

UserRef (variable): A variable-length ASCII string terminated by 0x00 containing an optional arbitrary string containing a higher-layer application-defined identifier for the message.

Ephemeral Fields (4 bytes): If the **E** bit is set to 1, these MUST be present. Otherwise, these fields MUST NOT be present.

These fields are defined as shown in the following table.



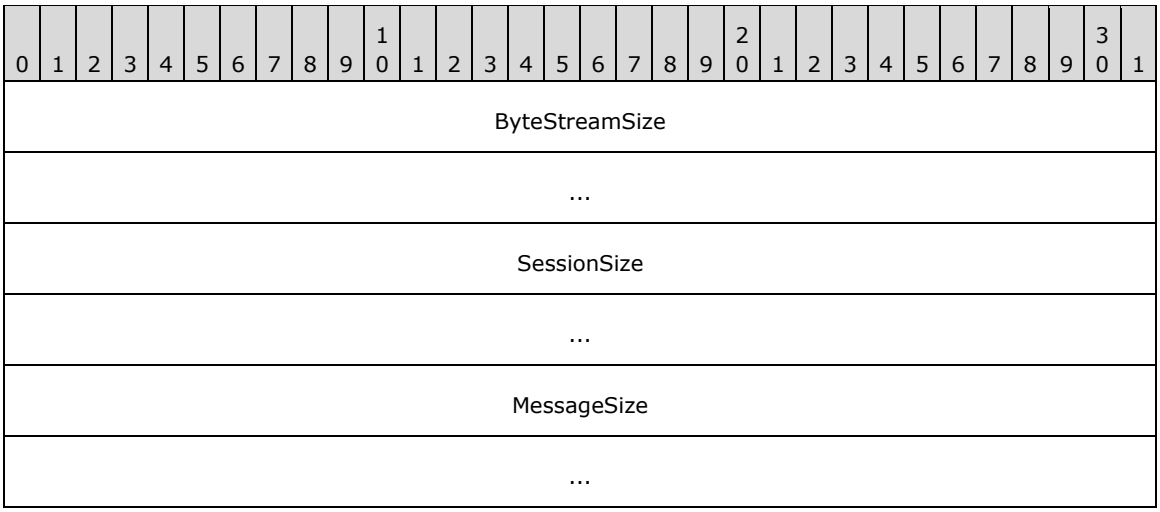
TTL (4 bytes): The message time to live, in seconds. Only relay servers SHOULD act on this field. A TTL field value of 0 indicates an infinite time to live for the message. Setting the TTL field to 0 is functionally equivalent to clearing the **E** bit, and not including the **Ephemeral Fields** in the **Message** command.

Reserved1 (4 bytes, optional): MUST be set to 0x00000000 if present.

Reserved2 (1 byte, optional): MUST be set to 0x00 if present.

StreamSize Fields (96 bytes): If the **S** bit is set to 1, these MUST be present. Otherwise, these fields MUST NOT be present.

These fields are defined as shown in the following table.



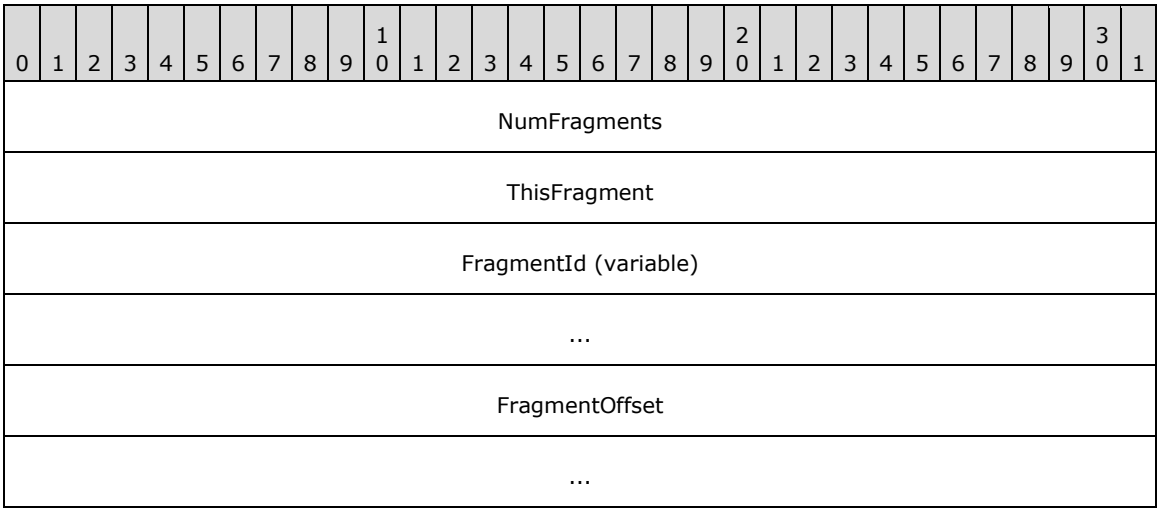
ByteStreamSize (8 bytes): The total number of bytes in the message in the higher-layer byte stream. An application can divide a single byte stream into multiple SSTP message sequences. This field MUST be the size of the original byte stream. A value of zero indicates the length of the byte stream is unknown.

SessionSize (8 bytes): The total number of bytes of the higher-layer byte stream that remain to be sent, including the size of the current message sequence byte stream. A value of zero indicates that the amount of data waiting to be sent is unknown.

MessageSize (8 bytes): The size of the message sequence, in bytes, before being divided up for transmission via individual **Data** commands. A value of zero indicates that the length of the message sequence is unknown.

Fragmentation Fields (variable): If the **F** bit is set to 1, these MUST be present. Otherwise, these fields MUST NOT be present.

These fields are defined as shown in the following table.



NumFragments (4 bytes): The total number of fragments in the fragmented message.

ThisFragment (4 bytes): The number of this fragment.

FragmentId (variable): A variable-length ASCII string terminated by 0x00 containing a unique identifier that is common to all the message sequences that are part of the same higher-layer byte stream.

FragmentOffset (8 bytes): The offset in the fragmented byte stream.

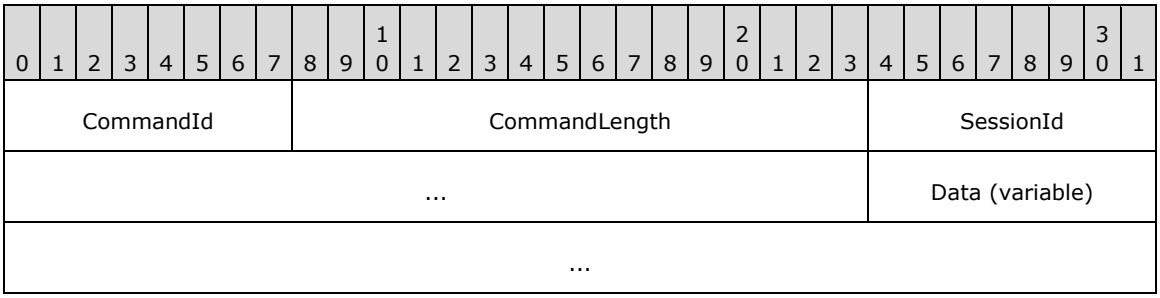
2.2.11 Data

The **Data** command contains a portion of the data in the SSTP message sequence.

The total length of this message MUST NOT exceed 2055 bytes.

2.2.11.1 Data Command Fields

The **Data** command fields are defined as shown in the following table.



CommandId (1 byte): The identifier of the command. This field MUST be set to 0x0e.

CommandLength (2 bytes): The total length of the command, in bytes.

SessionId (4 bytes): The identifier for the SSTP **session**, as specified in section 3.1.1.3.

Data (variable): A variable length byte sequence containing the payload. The number of bytes in this field MUST be equal to the **Data** Command message length minus the length of the **CommandId**, **CommandLength**, and **SessionId** fields.

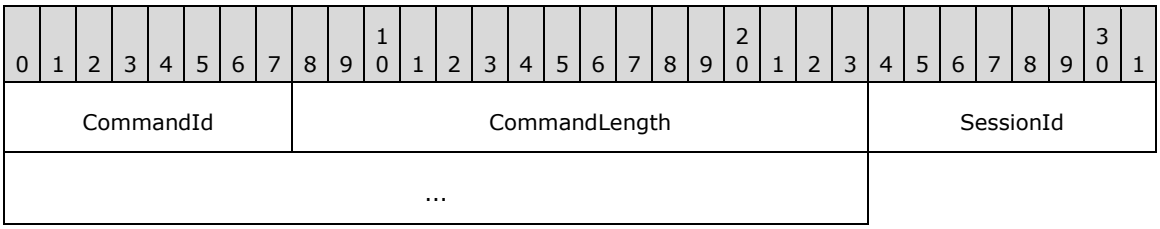
2.2.12 EndMessage

The **EndMessage** command denotes the end of an SSTP message sequence.

The total length of this message MUST be 7 bytes.

2.2.12.1 EndMessage Command Fields

The **EndMessage** command fields are defined as shown in the following table.



CommandId (1 byte): The command identifier. This field MUST be set to 0x0f.

CommandLength (2 bytes): The total length of the command, in bytes.

SessionId (4 bytes): The identifier for the SSTP **session**, as specified in section 3.1.1.3.

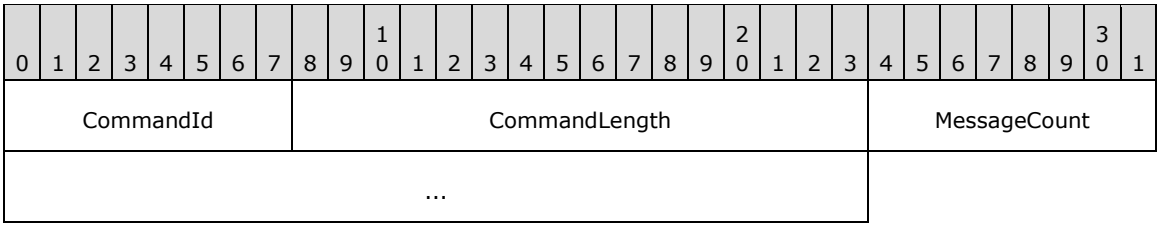
2.2.13 Noop

The **Noop** command is used to acknowledge commands and validate the underlying transport **connection (2)**.

The total length of this message MUST be 7 bytes.

2.2.13.1 Noop Command Fields

The **Noop** command fields are defined as shown in the following table.



CommandId (1 byte): The command identifier. This field MUST be set to 0x10.

CommandLength (2 bytes): The total length of the command, in bytes.

MessageCount (4 bytes): The number of SSTP messages the sending device acknowledges it has successfully received and processed.

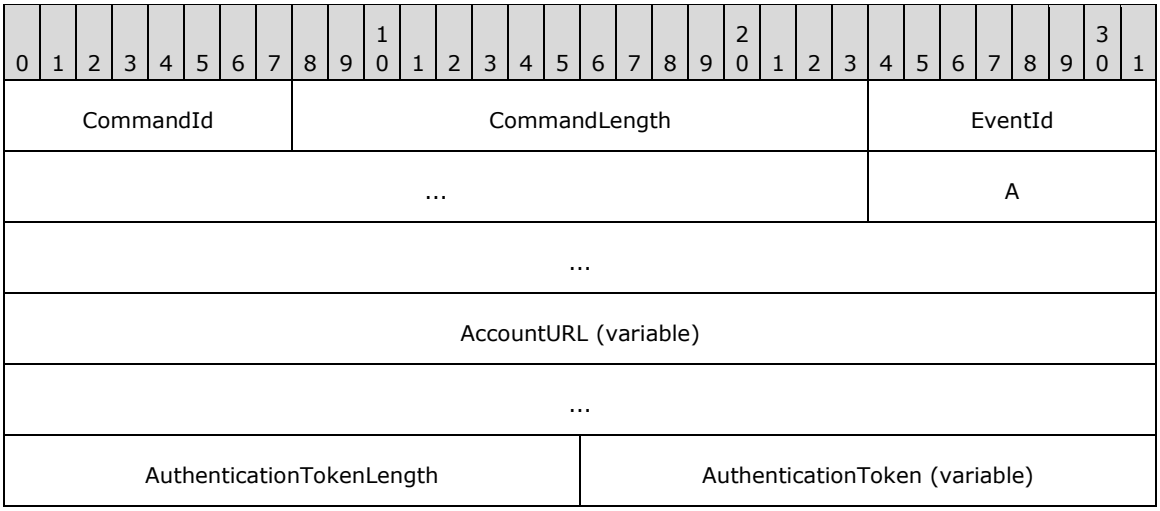
2.2.14 Attach

The **Attach** command is used to authenticate an account to a relay server.

The total length of this message MUST NOT exceed 2055 bytes.

2.2.14.1 Attach Command Fields

The **Attach** command fields are defined as shown in the following table.



...

CommandId (1 byte): The command identifier. This field MUST be set to 0x08.

CommandLength (2 bytes): The total length of the command, in bytes.

EventId (4 bytes): The **session** identifier for this attach session, as specified in section 3.2.4.1.2.

A - ResourceURL (variable): A variable-length ASCII string terminated by 0x00 containing the relay URL of the relay server authenticating the account. This field MAY be an empty string.

AccountURL (variable): A variable-length ASCII string terminated by 0x00 containing the URL of the account to be authenticated. This field MUST NOT be empty.

AuthenticationTokenLength (2 bytes): The length in bytes of the **AuthenticationToken** field.

AuthenticationToken (variable): A byte sequence that is used to authenticate the sender of the **Attach** command.

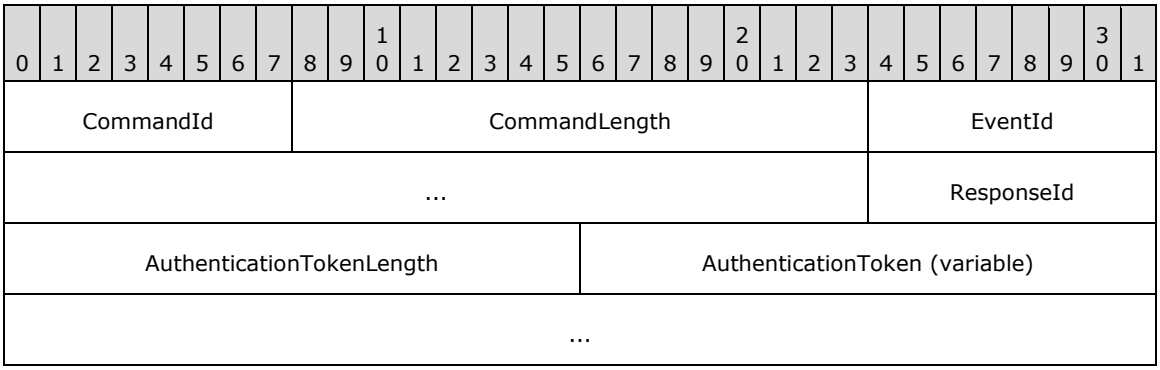
2.2.15 AttachResponse

The **AttachResponse** command is used to provide a security challenge to the sender of the Attach command.

The total length of this message MUST NOT exceed 2055 bytes.

2.2.15.1 AttachResponse Command Fields

The **AttachResponse** command fields are defined as shown in the following table.



CommandId (1 byte): The command identifier. This field MUST be set to 0x09.

CommandLength (2 bytes): The total length of the command, in bytes.

EventId (4 bytes): The identifier of the attach **session**.

ResponseId (1 byte): This field MUST be one of the values described in the following table.

Value	Mnemonic	Description
0x00	Ok	The first phase of the Attach authentication finished successfully.
0x01	AttachRejected	The Attach was rejected for the account indicated

Value	Mnemonic	Description
		in the Attach command.
0x02	AccountUnknown	The Attach was rejected because an unknown account was presented in the Attach command.
0x03	AwaitingRegister	The account presented in the Attach command is unknown to this relay server, but the account can be registered with the relay server by sending a Register command on the same session as the Attach command.

AuthenticationTokenLength (2 bytes): The length in bytes of the **AuthenticationToken** field.

AuthenticationToken (variable): A byte sequence that is used to authenticate the sender of the **AttachResponse** command.

The length of this byte sequence MUST be as specified in the **AuthenticationTokenLength** field. The contents of this token are as specified in [MS-GRVSSTPS].

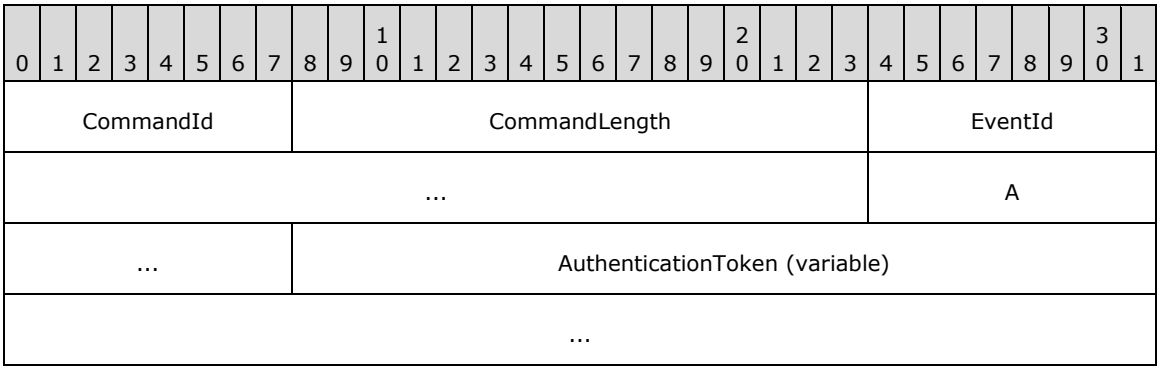
2.2.16 AttachAuthenticate

The **AttachAuthenticate** command is used to validate that the sender can respond correctly to a challenge provided in the previous **AttachResponse** command.

The total length of this message MUST NOT exceed 2055 bytes.

2.2.16.1 AttachAuthenticate Command Fields

The **AttachAuthenticate** command fields are defined as shown in the following table.



CommandId (1 byte): The command identifier. This field MUST be set to 0x0a.

CommandLength (2 bytes): The total length of the command, in bytes.

EventId (4 bytes): The identifier of the attach **session**.

A - AuthenticationTokenLength (2 bytes): The length in bytes of the **AuthenticationToken** field.

AuthenticationToken (variable): A byte sequence that is used to complete account authentication.

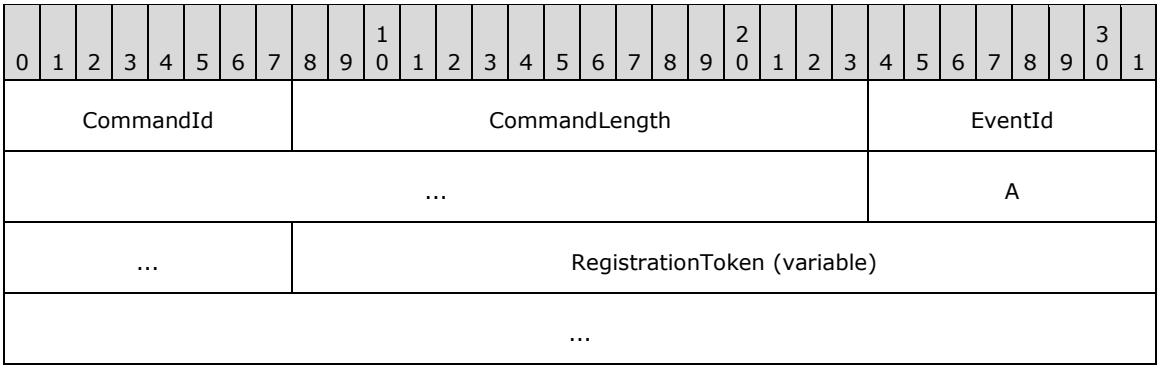
The length of this byte sequence MUST be as specified in the **AuthenticationTokenLength** field. The contents of this token are as specified in [MS-GRVSSTPS].

2.2.17 Register

The **Register** command is used in both the **Attach** sequence and the identity registration sequence as specified in [MS-GRVSSTPS]. This command MUST NOT exceed 8192 bytes.

2.2.17.1 Register Command Fields

The **Register** command fields are defined as shown in the following table.



CommandId (1 byte): The command identifier. This field MUST be set to 0x0b.

CommandLength (2 bytes): The total length of the command, in bytes.

EventId (4 bytes): A **session** identifier for the register-session as specified in section 3.2.4.1.3.

A - RegistrationTokenLength (2 bytes): The length in bytes of the **RegistrationToken** field of the **Register** command

RegistrationToken (variable): A byte sequence that is used to request the registration of a device-account combination, or list of identities.

The length of this byte sequence MUST be as specified in the **RegistrationTokenLength** field. The contents of this token are as specified in [MS-GRVSSTPS].

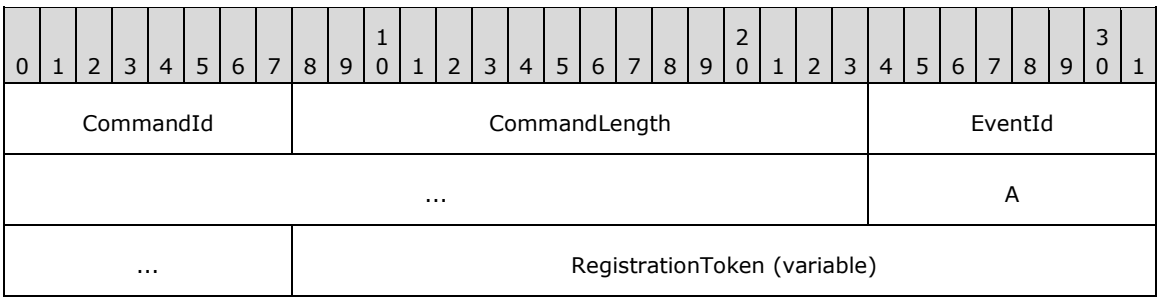
2.2.18 RegisterResponse

The **RegisterResponse** command is used to indicate the status of a registration attempt.

This command MUST NOT exceed length of 2055 bytes.

2.2.18.1 RegisterResponse Command Fields

The **RegisterResponse** command fields are defined as shown in the following table.



...

CommandId (1 byte): The command identifier. This field MUST be set to 0x0c.

CommandLength (2 bytes): The total length of the command, in bytes.

EventId (4 bytes): The identifier of the register-session.

A - RegistrationTokenLength (2 bytes): The length in bytes of the **RegistrationToken** field.

RegistrationToken (variable): A byte sequence that is used to verify the success or failure of an attempt to register a device/ account combination, or list of identities.

The length of this byte sequence MUST be as specified in the **RegistrationTokenLength** field. The contents of this token are as specified in [MS-GRVSSTPS].

3 Protocol Details

Details common to the client and relay server are specified in section 3.1. Details specific to the client are specified in section 3.2. Details specific to the relay server are specified in section 3.3.

3.1 Common Details

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an application can maintain to participate in this protocol. This information is provided to facilitate understanding of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior conforms to the specified normative behavior.

The following sections describe common protocol details in the context of SSTP **connections (1)**, **sessions**, and data exchange.

3.1.1.1 Global Configuration

The implementation maintains the following state variables:

SSTPPort: The TCP port on which to listen for incoming SSTP **connections (1)**. The assigned port for SSTP over TCP is 2492 [IANAPORT].

Version: The supported version of the SSTP protocol. For implementations of SSTP 1.5, the version is 1.5 (MajorVersion 1, MinorVersion 5). For implementations of SSTP 1.6, the version is 1.6 (MajorVersion 1, MinorVersion 6). SSTP 1.6 implementations are able to handle SSTP 1.5 messages as well.

LocalDeviceURLs: Each SSTP device is assigned one or more unique DeviceURLs. The mechanism for assignment of this URL is application-dependent.

NumberOfConcurrentDeviceConnections: An optional application limit to the number of SSTP connections (1) between two devices.

MultidropSupported: A Boolean configuration value, which if set to true, indicates that the local application can accept incoming fanout **sessions** targeting multiple local resource handlers which are implemented on the receiving device. This MUST be false for a client device.

SingleHopSupported: A Boolean configuration value, which if set to true, indicates that the local application can accept incoming fanout sessions targeting resource handlers which are accessible only through a remote relay servers. This MUST be false for a client device.

3.1.1.2 SSTP Connection

The following state variables MUST be maintained for each transport layer **connection (2)**:

Inbound/Outbound: A Boolean value to indicate whether the device has originated or accepted the underlying transport connection (2). A value of true indicates that the device originated the connection (2), and false indicates that the device accepted the connection (2).

ConnectionState: An enumeration indicating the state of the SSTP connection (1). The states are 'transport layer connected', 'connecting', 'established', and 'aborting'.

OutboundMessageList: A table consisting of a time-ordered list of message sequences which have been sent, containing a reference to the originating **session** and the higher-layer data.

Version: A value to indicate the version of the SSTP protocol, which is to be used for all transactions on this connection (1) while in the 'established' state. This version in use on a specific SSTP connection (1) is not necessarily identical to the globally configured Version supported by the device, as specified in section 3.1.1.1.

InboundMessageList: A table consisting of a time-ordered list of message sequences which have been received, containing a reference to the receiving session, the higher-layer data, additional received meta-data for the message sequence, and the processing disposition for the received data. The processing states are: 'processing', 'complete'. The additional meta-data to be retained as part of an InboundMessageList entry comprises the Message command F, G, S, A, E, and D bits, Ephemeral fields, StreamSize fields, and Fragmentation fields.

The following state transition diagram shows the state relationships for an SSTP connection (1):

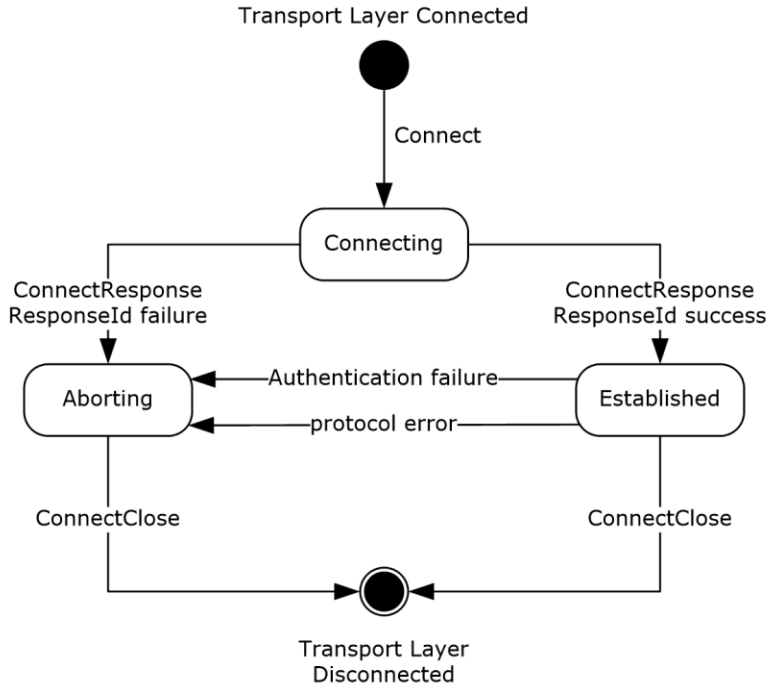


Figure 9: SSTP connection state diagram

3.1.1.3 SSTP Sessions

The device that sends the Open or FanoutOpen is the **session** originator. The device that receives the Open or FanoutOpen is the session receiver. Both devices MUST maintain session state.

The following state variables MUST be maintained by both the session originator and session receiver for each session:

SessionId: The identifier for the SSTP session.

SessionState: An enumeration indicating the state of the session. The states are: 'opening', 'suspended', 'ready', and 'blocked'.

The following state transition diagram shows the state relationships for an SSTP session:

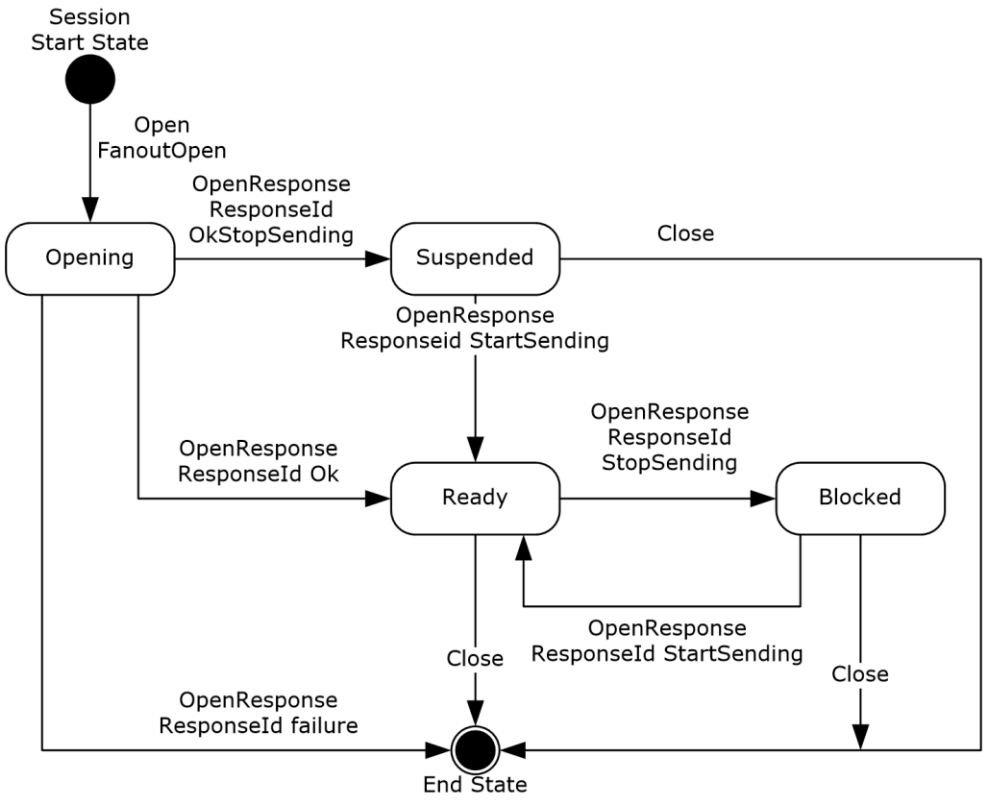


Figure 10: Session state diagram

The following state variables MUST be maintained by the session originator:

OutboundAddressingEntry: The destination addressing entry for a session, consisting of ResourceURL, IdentityURL<10>, and DeviceURL, identifying the recipient resource handler. When creation of a session is requested by the higher-layer, these are supplied by the higher-layer application.

OutboundFanoutAddressingList: The list of addressing entries for a fanout session, consisting of a common ResourceURL, and a list of IdentityURL, DeviceURL, and RelayURLs identifying the recipient resource handlers. In SSTP 1.6, there is an additional FailoverDeviceURLs field, which MUST be set to 0x00. When creation of a fanout session is requested by the higher-layer, these are supplied by the higher-layer application. The list order is important and MUST be maintained, because the SessionStatus command references a particular entry in the list using its zero-based index number.

The following state variables MUST be maintained by the session receiver:

InboundAddressingEntry: The destination addressing entry for a received session, consisting of ResourceURL, IdentityURL, and DeviceURL, identifying the recipient resource handler. A ResourceHandlerAvailability state is associated with a valid InboundAddressingEntry.

InboundFanoutAddressingList: The list of addressing entries for a fanout session, consisting of a common ResourceURL, and a list of IdentityURL, DeviceURL, and RelayURLs identifying the recipient resource handlers. In SSTP 1.6, there is an additional FailoverDeviceURLs field, which MUST be set to 0x00. The list order is important and MUST be maintained, because the SessionStatus command references a particular entry in the list using its zero-based index number.

ResourceHandlerAvailability: A variable maintained by the higher-layer which indicates the readiness of a specific resource handler, as identified by an addressing entry, to receive data. The states are 'ready', 'notReady', or in a 'fault' state.

MessageReceiver State: An enumeration indicating the state of the incoming session message sequence transfer. The states are 'waiting', 'ready', 'buffering'.

The following state transition diagram shows the state relationships for the MessageReceiver for an SSTP session:

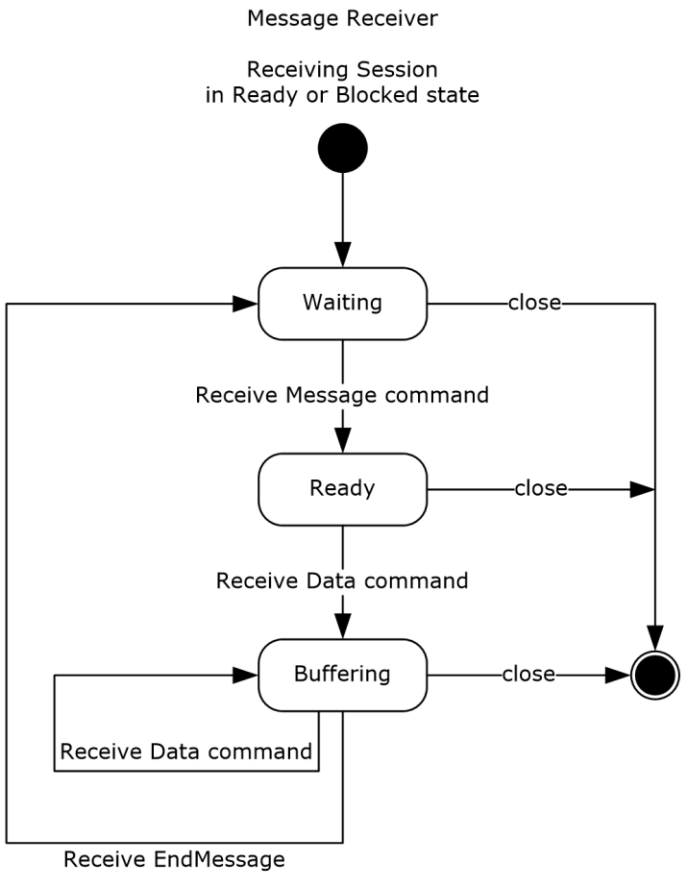


Figure 11: MessageReceiver state diagram

3.1.2 Timers

Although not mandated by the SSTP protocol, an application MAY implement additional timers to ensure timely **connection (1)** and **session** transitions.<11>

3.1.2.1 Message Acknowledgment Timer

A timer used to ensure that messages are acknowledged within a fixed time. There is one timer for each SSTP **connection (1)**. The default value is 5 seconds.<12>

3.1.3 Initialization

The device specific settings specified in section 3.1.1.1 MUST be initialized with the locally configured values.

Connection (1) and **Session** state MUST be removed.

The SSTP device MUST open a listener at the designated SSTPPort, and wait for establishment of an incoming network layer connection (2) and delivery of the first detected SSTP Connect message.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Establishing an SSTP Connection

To establish an SSTP **connection (1)**, the higher-layer MUST first establish a reliable transport connection (2) to the destination device. When the transport connection (2) is established, the connection (1) state MUST be set to the initial state of 'transport layer connected'.

The device MUST then send a Connect command over the transport connection (1).

- The sender MUST set the TargetDeviceURL field to an expected application defined LocalDeviceURL of the device to which the transport has connected, as supplied by the higher-layer.
- The sender MUST set the MajorVersionNumber and MinorVersionNumber fields to the values for the version of SSTP which it supports, as specified in the device configuration in section 3.1.1.1.
- The sender MUST set the SourceDeviceURLs field to the list of its locally configured LocalDeviceURLs.

If a client device is connecting to a relay server, it SHOULD include device authentication information in the Connect command, as specified in section 3.2.4.1.1.

Upon sending the Connect command the connection (1) state MUST be set to 'connecting'.

3.1.4.2 Closing an SSTP Connection

The higher-layer can close any established SSTP **connection (1)**. The device MUST send a ConnectClose command, with a ReasonId value as supplied by the higher-layer.

The MessageCount field MUST be set to the number of processed inbound message sequences entries, as specified in section 3.1.4.2.1.

Upon sending the ConnectClose, connection (1) and **session** state for all sessions on the connection (1) MUST be removed, and the underlying transport MUST be terminated.

3.1.4.2.1 Determining the MessageCount Field Value

An acknowledgment count for processed message sequences MUST be determined by examining the oldest entries in the InboundMessageList for the **connection (1)**, and counting the number of sequential entries with a state of 'complete'. Those entries MUST be removed from the InboundMessageList. That number of entries is set in the MessageCount field.

3.1.4.3 Opening a Session

The higher layer can originate a **session** by sending either an Open command, as specified in section 3.1.4.3.2 or a FanoutOpen command as specified in section 3.1.4.3.3.

The higher-layer can open a session on a **connection (1)** in the 'established' state.

The Open command MUST be used to open a session with a client.

3.1.4.3.1 Selecting a SessionId

The **session** originator MUST set the SessionId field to a valid and unused session identifier for the **connection (1)**.

The session identifier is a 4 byte integer in which the most significant bit indicates which device initiated the underlying SSTP connection (1), as determined by the connection (1) inbound/outbound state variable. If the Inbound/Outbound state variable is true, the most significant bit of the session identifier selected by the session originator MUST be set. Valid session identifiers for the connection (1) initiator are 0x00000000 to 0x7FFFFFFF. Valid session identifiers for the device which accepted the connection (1) are 0x80000000 to 0xFFFFFFFF.

The session identifier selected by the session originator MUST be contained in the SessionId field of all subsequent commands pertaining to this session for the lifetime of this session. The session identifier MUST NOT be used for any other session until the original session ends and session state is removed.

3.1.4.3.2 Sending an Open Command

The higher-layer MUST supply the OutboundAddressingEntry for the **session**. The sending device MUST set the ResourceURL, IdentityURL and DeviceURL fields of the Open command to the values provided by the higher-layer<13>. It is valid to specify the DeviceURL field as the empty string (0x00) to indicate an identity-targeted session.

Upon sending the Open command, the sender's session MUST be set to the 'connecting' state.

3.1.4.3.3 Sending a FanoutOpen Command

The higher-layer MUST supply the OutboundFanoutAddressingList for the **session**. The sending device MUST set the common ResourceURL, and the IdentityURL, DeviceURL, and RelayURL fields of the FanoutDeviceEntries in the FanoutOpen command to the values provided by the higher-layer<14>. It is valid to specify a DeviceURL field as the empty string (0x00) to indicate an identity-targeted session. The RelayURL field of a FanoutDeviceEntry can be set to the empty string (0x00) for recipients that are associated with the connected relay server.

If the **connection (1)** Version state variable indicates that the sending connection (1) version is SSTP 1.6, then the higher-layer MUST also supply the FailoverDeviceURLs field of the FanoutDeviceEntries in the FanoutOpen command. The FailoverDeviceURLs field MUST be set to the empty string (0x00).

If the connection (1) Version state variable indicates that the sending connection (1) version is SSTP 1.5, the FailoverDeviceURLs field MUST NOT exist in the FanoutOpen command.

The higher-layer MUST maintain the OutboundFanoutAddressingList for the life of the session as the SessionStatus command returns state change notifications based on an index into the OutboundFanoutAddressingList.

Upon sending the FanoutOpen command, the sender's session MUST be set to the 'connecting' state.

3.1.4.4 Closing a Session

The higher layer can close any existing **session**. The device MUST send a **Close** command. The higher layer MUST supply a **ReasonId**, as specified in section 2.2.9.1.

Upon sending a **Close**, the sending device MUST remove all session state.

3.1.4.5 Changing Resource Handler Availability State

The higher-layer can change the **ResourceHandlerAvailability** state for the resource handler for any existing inbound **session**.

The higher-layer MUST provide notifications of changes in resource availability, and set the associated **ResourceHandlerAvailability** state as follows:

ResourceHandlerAvailability state	Description
ready	Can accept and process application data.
notReady	Temporarily inaccessible, or not ready to accept and process application data.
fault	Inaccessible, or has failed, or access has been denied by the higher-layer.

When the higher-layer indicates the loss of the resource handler because of a fault, it MUST supply an application defined **ReasonId** for the resultant **Close** command to be sent to the session originator. The **ReasonId** MUST be one of NoReason (0x00), QuotaWouldBeExceeded (0x0b), or InternalError (0x0d).

The receiving session MUST process higher-layer notifications of changes in resource handler availability, as shown in the following table.

Session state (session receiver)	New ResourceHandlerAvailability	Session receiver action
suspended	ready	MUST send **OpenResponse** with **ResponseId** of StartSending (0x09). Session state MUST be set to "ready".
suspended	notReady	None.
suspended	fault	MUST send **Close** with higher-layer supplied ReasonId. Session state MUST be removed.
ready	ready	None.
ready	notReady	MUST send **OpenResponse** with ResponseId of StopSending (0x0a). Session state MUST be set to "blocked".
ready	fault	MUST send **Close** with higher-layer supplied **ReasonId**. Session state MUST be removed.
blocked	ready	MUST send **OpenResponse** with **ResponseId** of StartSending (0x09). Session state MUST be set to "ready".
blocked	notReady	None.
Blocked	fault	MUST send **Close** with a higher-layer supplied **ReasonId**. Session state MUST be removed.

3.1.4.6 Sending Data

The higher layer can send data on an outgoing **session** in the "ready" state.

The device MUST first send a **Message** command. The **MessageCount** field MUST be set to the number of processed inbound message sequences entries, as specified in section 3.1.4.2.1. If the Message Acknowledgment Timer is started, it MUST be stopped.

Usage of the **Message** command A, F, G, S, E, or D bits, with their associated fields, as specified in section 2.2.10.1, is determined by the higher layer.

The device MUST then send one or more **Data** commands containing the higher-layer data. If the higher-layer data is larger than 2048 bytes, it MUST be split into multiple **Data** commands, where each contains up to 2048 bytes of application data.

The device MUST then send an **EndMessage** command. The **OutboundMessageList** table (in section 3.1.1.2) for the underlying **connection (1)** MUST be updated to contain the data as the most recently sent message sequence.

3.1.4.7 Notifying of Resource Handler Completion

The higher layer can notify of resource handler completion for any message in the **InboundMessageList** table in the "processing" state. When notified, the **InboundMessageList** table state for the associated entry MUST be set to "complete".

The device MUST consider the set of all messages in the **InboundMessageList** table that arrived before the first message that is not in the "complete" state.

If one of those messages was sent as part of a sequence for which the **Message** command had specified the A (AcknowledgeImmediately) bit when received as specified in section 3.1.5.10, the device SHOULD<15> send a **Noop** command with the **MessageCount** field set to the value as determined by the process specified in section 3.1.4.2.1.

If the set is not empty, and the Message Acknowledgment Timer (as specified in section 3.1.2.1) is not currently active, then it MUST be started.

3.1.4.8 Sending a Noop

The higher-layer can send a Noop command on a **connection (1)** in the 'established' state. The MessageCount field MUST be set to the number of processed inbound message sequences entries, as specified in section 3.1.4.2.1. If the Message Acknowledgment Timer is started then it MUST be stopped.

3.1.5 Message Processing Events and Sequencing Rules

Every command to be sent MUST be formatted with the appropriate 3 byte SSTP message header as specified in section 2.2 of this document. The CommandLength field MUST contain the exact length in bytes of the command data, including the header.

The receiving device MUST parse every byte of the stream received on the underlying transport **connection (2)**. The type of the message is determined by the first byte of the message. The remainder of the byte stream MUST be parsed as specified in section 2.2. The message MUST be processed as specified in the appropriate following subsection.

If the message is not a valid SSTP message as specified in section 2.2, the recipient MUST send a ConnectClose command as specified in section 3.1.4.2, with ReasonId of ProtocolError (0x03).

3.1.5.1 Receiving a Connect Command

If the recipient's **connection (1)** state is not 'transport layer connected', then the state MUST be set to 'aborting', and the recipient MUST send a ConnectClose command as specified in section 3.1.4.2. Otherwise, the connection (1) state MUST be set to 'connecting'.

The recipient MUST send a ConnectResponse command. The ResponseId and RetryTime fields MUST be determined as follows:

- If the supplied TargetDeviceURL field is not the same as one of the locally configured LocalDeviceURLs, the ResponseId MUST be set to WrongDevice (0x01).
- If the supplied MajorVersionNumber and MinorVersionNumber are incompatible with the recipient's locally configured version, the ResponseId MUST be set to:
  - One of WillUpgrade (0x03) or WontUpgrade (0x04) if the sender's major version is higher than that of the recipient. The sender's minor version is used to determine whether WillUpgrade or WontUpgrade is sent.
  - NewVersionRequired (0x05) if the sender's major version is less than that of the recipient.
- If the recipient imposes connection (1) restrictions based upon the originating device URL, and one of the supplied SourceDeviceURLs is subject to an application defined security lockout, as specified in [MS-GRVSPMR], the ResponseId MUST be set to ConnectRejected (0x09).
- If the recipient has a local configured limit to the NumberOfConcurrentDeviceConnections, and the number of connections (1) from the supplied SourceDeviceURL exceeds this limit, the ResponseId MUST be set to Ok (0x00). After responding with the required ConnectResponse, the recipient MUST send a ConnectClose command with a ReasonId of CrossedConnections (0x0c).
- If the recipient is not able to handle a new connection (1) for any other reason, it MUST set the ResponseId to TryLater (0x02), and the higher-layer MUST specify a suggested RetryTime field.
- If the recipient is a relay server, additional processing of an embedded Security message as specified in section 3.3.5.1 MUST be performed to determine the ResponseId.
- If none of the preceding restricting conditions are detected, the ResponseId MUST be set to Ok (0x00).

The remaining fields in the ConnectResponse MUST be set as follows:

- The recipient MUST set the MajorVersionNumber and MinorVersionNumber fields to its version of SSTP, as specified in section 3.1.1.1.
- If the ResponseId is Ok, the recipient MUST set the TargetDeviceURLs field to the list of its locally configured LocalDeviceURLs, as specified in section 3.1.1.1.
- The M (Multi-dropFanout) bit MUST be set to the configured value for MultidropSupported, as specified in section 3.1.1.1.
- The S (SingleHopFanout) bit MUST be set to the configured value for SingleHopSupported, as specified in section 3.1.1.1.

If the contained ResponseId is Ok, the connection (1) state MUST be set to 'established'. The SSTP connection (1) state variable Version maintained by the recipient, as specified in section 3.1.1.2, MUST be set to the lesser of the received MinorVersion value, or the receiver's configured MinorVersion value, as specified in section 3.1.1.1.

If the contained ResponseId is not Ok, the connection (1) state MUST be set to 'aborting', and a ConnectClose MUST be sent as the next command with a higher-layer determined ReasonId.

3.1.5.2 Receiving a ConnectResponse Command

If the recipient's **connection (1)** state is not 'connecting', then the recipient MUST set the connection (1) state to 'aborting', and the recipient MUST send a ConnectClose command as specified in section 3.1.4.2, with ReasonId of ProtocolError (0x03).

If the ResponseId is TryLater (0x02), the recipient SHOULD pass the value of the RetryTime field to the higher-layer, to permit it to wait the given number of seconds before attempting to re-establish a transport layer connection (1) to the target device.

If the ResponseId is not Ok, the connection (1) state MUST be set to 'aborting'. If the ResponseId is Ok, the connection (1) state MUST be set to 'established'. The SSTP connection (1) state variable Version maintained by the recipient, as specified in section 3.1.1.2, MUST be set to the lesser of the received MinorVersion value, or the receiver's configured MinorVersion value, as specified in section 3.1.1.1.

3.1.5.3 Receiving a ConnectAuthenticate Command

If the recipient is a client, it MUST process the command as specified in section 3.2.5.3.

If the recipient is a relay server, it MUST process the command as specified in section 3.3.5.3.

3.1.5.4 Receiving a ConnectClose Command

If the ConnectClose command contains a nonzero MessageCount field value, the OutboundMessageList acknowledgment handling MUST be performed, as specified in section 3.1.5.19.

Upon receipt of a ConnectClose, all state for the **connection (1)** and **sessions** on the connection (1) MUST be removed.

If the ReasonId is Resting (0x01), the receiver SHOULD pass the value of the ReturnTime field to the higher-layer, to permit it to wait the given number of seconds before attempting to re-establish a transport layer connection (2) to the target device.

3.1.5.5 Receiving an Open Command

If the **connection (1)** is not in the 'established' state, or a **session** identified by the contained SessionId field already exists, the recipient MUST send a ConnectClose command as specified in section 3.1.4.2, with a ReasonId of TooManyUnknownSessionCmds (0x0f).

Upon receipt of an Open command the receiving session state MUST be set to 'opening'.

The receiving device MUST set the session InboundAddressingEntry to the session addressing entry in the ResourceURL, IdentityURL, and DeviceURL fields.

The higher-layer MUST provide the ResourcHandlerAvailability state for the addressing entry.

The recipient MUST send an OpenResponse command. The ResponseId MUST be determined based on the ResourcHandlerAvailability state as follows:

If the state is:

- 'fault', the ResponseId MUST be set to Unknown (0x05), and session state MUST be removed.
- 'notReady', the ResponseId MUST be set to OkStopSending (0x0b), and the session state MUST be set to the 'suspended' state.
- ''ready'', the ResponseId MUST be set to Ok (0x00), and the session state MUST be set to the 'ready' state.
- Upon sending the OpenResponse for a session, the receiving session MessageReceiver state MUST be set to 'waiting'.

3.1.5.6 Receiving a FanoutOpen Command

If the recipient is a client, it MUST process the command as specified in section 3.2.5.6.

If the recipient is a relay server, it MUST process the command as specified in section 3.3.5.6.

3.1.5.7 Receiving an OpenResponse Command

If an OpenResponse command is received with a SessionId field identifying a **session** for which there is no state, the recipient MUST send a ConnectClose command as specified in section 3.1.4.2 with a ReasonId of TooManyUnknownSessionCmds (0x0f).

If an OpenResponse command is received on a session which was not originated by the recipient, the recipient MUST send a ConnectClose command as specified in section 3.1.4.2 with a ReasonId of ProtocolError (0x03).

For an OpenResponse received on a valid open session, the action to be taken depends on the session state at time of receipt.

Session state(session originator)	OpenResponse ResponseId	Recipient Action
opening	(0x00) Ok	Session state MUST be set to 'ready'
	(0x04) NoResource	Session state MUST be removed
	(0x05) Unknown	
	(0x08) NoFanoutEntries	
	(0x0c) FanoutNotSupported	
	(0x09) StartSending	MUST close **connection (1)** as specified in section 3.1.4.2 with ReasonId of ProtocolError (0x03)
	(0x0a) StopSending	
	(0x0b) OkStopSending	Session state MUST be set to 'suspended'
suspended	(0x00) Ok	MUST close connection (1) as specified in section 3.1.4.2 with ReasonId of ProtocolError (0x03)
	(0x04) NoResource	
	(0x05) Unknown	
	(0x08) NoFanoutEntries	
	(0x0c) FanoutNotSupported	
	(0x09) StartSending	Session state MUST be set to 'ready'
	(0x0a) StopSending	MUST close connection (1) as specified in section 3.1.4.2 with ReasonId of ProtocolError (0x03)
	(0x0b) OkStopSending	
ready	(0x00) Ok	MUST close connection (1) as specified in section 3.1.4.2 with ReasonId of ProtocolError (0x03)
	(0x04) NoResource	
	(0x05) Unknown	
	(0x08) NoFanoutEntries	

Session state(session originator)	OpenResponse ResponseId	Recipient Action
	(0x0c) FanoutNotSupported	
	(0x09) StartSending	None
	(0x0a) StopSending	Session state MUST be set to 'blocked'
	(0x0b) OkStopSending	MUST close connection (1) as specified in section 3.1.4.2 with ReasonId of ProtocolError (0x03)
blocked	(0x00) Ok	MUST close connection (1) as specified in section 3.1.4.2 with ReasonId of ProtocolError (0x03)
	(0x04) NoResource	
	(0x05) Unknown	
	(0x08) NoFanoutEntries	
	(0x0c) FanoutNotSupported	
	(0x09) StartSending	Session state MUST be set to 'ready'
	(0x0a) StopSending	None
	(0x0b) OkStopSending	MUST close connection (1) as specified in section 3.1.4.2 with ReasonId of ProtocolError (0x03)

If the OpenResponse is received by a relay-server on a relay originated single-hop session, additional processing MUST be performed as specified in section 3.3.5.7.

3.1.5.8 Receiving a SessionStatus Command

If a SessionStatus is received on a **session** which has not been originated by a FanoutOpen, the recipient MUST send a ConnectClose command as specified in section 3.1.4.2 with ReasonId of ProtocolError (0x03).

If a SessionStatus command is received with a SessionId field identifying a session for which there is no state, the recipient MUST send a ConnectClose command as specified in section 3.1.4.2 with a ReasonId of TooManyUnknownSessionCmds (0x0f).

Upon receipt of a SessionStatus the fanout participant identified either by the DeviceURL and IdentityURL fields or in the FanoutDeviceIndexes list MUST be marked as no longer active in the session OutboundFanoutAddressingList.

If the IdentityURL field is empty (0x00), then the DeviceURL indicates the RelayURL of a relay server which is no longer available for participation in the fanout session. All fanout participants with that RelayURL MUST be marked as no longer active in the session OutboundFanoutAddressingList.

If the SessionStatus is received by a relay-server on an outbound single-hop session, additional processing MUST be performed as specified in section 3.3.5.8.

The SessionStatus command MUST be parsed as specified in section 2.2.8 based upon the **connection (1)** Version state variable of the receiving connection (1). If the version-specific parsing does not process exactly the number of bytes indicated by the SessionStatus CommandLength field,

then the recipient MUST send a ConnectClose as specified in section 3.1.4.2 with a ReasonId of ProtocolErrors (0x03).

If the connection (1) Version state variable indicates that the receiving connection (1) version is SSTP 1.6 and if both DeviceURL and IdentityURL fields are empty (0x00), then the SessionStatus command is used to notify status changes for multiple fanout devices, whose indices are identified in the FanoutDeviceIndexes field. Each index in the FanoutDeviceIndexes field corresponds to a fanout entry in the associated FanoutOpen command, which specifies the DeviceURL and IdentityURL of the fanout device. The process described in this section MUST be followed for each fanout device. While processing a SessionStatus command, FanoutDeviceEntries MUST NOT be removed from the OutboundFanoutAddressingList to avoid skewing index-to-FanoutDeviceEntry mappings.

3.1.5.9 Receiving a Close Command

If the **connection (1)** is not in the 'established' state, the recipient MUST send a ConnectClose command as specified in section 3.1.4.2, with a ReasonId of TooManyUnknownSessionCmds (0x0f). If a Close command is received with a SessionId field identifying a **session** for which there is no state and which is not in AttachSessionIds or RegisterSessionIds (as specified in sections 3.2.1 and 3.3.1), the recipient MUST ignore the command.

Upon receipt of a Close command, all session state for the indicated session MUST be removed.

If the Close is received by a relay-server on an outbound single-hop session, additional processing MUST be performed as specified in section 3.3.5.9.

3.1.5.10 Receiving a Message Command

If a Message command is received with a SessionId field identifying a **session** for which there is no state, the recipient MUST send a ConnectClose as specified in section 3.1.4.2 with a ReasonId of TooManyUnknownSessionCmds (0x0f).

If a Message command is received on a session for which the MessageReceiver is not in the 'waiting' state, the recipient MUST send a ConnectClose as specified in section 3.1.4.2 with a ReasonId of ProtocolError (0x03).

Upon receiving the Message command, the MessageReceiver state MUST be set to 'ready'.

If the Message command contains a nonzero MessageCount field value, the OutboundMessageList acknowledgment handling MUST be performed, as specified in section 3.1.5.19.

3.1.5.11 Receiving a Data Command

If a Data command is received with a SessionId field identifying a **session** for which there is no state, the recipient MUST close the **connection (1)** as specified in section 3.1.4.2 with a ReasonId of TooManyUnknownSessionCmds (0x0f).

If a Data command is received on a session for which the MessageReceiver is not in the 'ready' or 'buffering' state, the recipient MUST send a ConnectClose as specified in section 3.1.4.2 with a ReasonId of ProtocolError (0x03).

Upon receiving the Data command, the MessageReceiverstate MUST be set to 'buffering'.

3.1.5.12 Receiving an EndMessage Command

If an EndMessage command is received with a SessionId field identifying a **session** for which there is no state, the recipient MUST send a ConnectClose as specified in section 3.1.4.2 with a ReasonId of TooManyUnknownSessionCmds (0x0f).

If an EndMessage command is received on a session for which the MessageReceiver is not in the 'buffering' state, the recipient MUST send a ConnectClose as specified in section 3.1.4.2 with a ReasonId of ProtocolError (0x03).

Upon receiving the EndMessage command, the MessageReceiver state MUST be set to 'waiting'. If the Message Acknowledgment Timer specified in section 3.1.2.1 is not active, it MUST be started.

The application data block assembled by the MessageReceiver while in the 'buffering' state MUST be passed to the application defined resource handler for asynchronous processing, and the message information MUST be added to the InboundMessageList table as the most recently received message sequence. The InboundMessageList state for this entry MUST be set to 'processing'.

Additional server-specific processing of a message sequence received by a relay server on a fanout session MUST be performed as specified in section 3.3.5.12.

3.1.5.13 Receiving a Noop Command

If the Noop command contains a nonzero MessageCount field value, the outbound MessageList acknowledgment handling MUST be performed, as specified in section 3.1.5.19.

3.1.5.14 Receiving an Attach Command

As specified in sections 3.2.5.14 and 3.3.5.14.

3.1.5.15 Receiving an AttachResponse Command

As specified in sections 3.2.5.15 and 3.3.5.15.

3.1.5.16 Receiving an AttachAuthenticate Command

As specified in sections 3.2.5.16 and 3.3.5.16.

3.1.5.17 Receiving a Register Command

As specified in sections 3.2.5.17 and 3.3.5.17.

3.1.5.18 Receiving a RegisterResponse Command

As specified in sections 3.2.5.18 and 3.3.5.18.

3.1.5.19 Receiving Acknowledgments in a MessageCount Field

Upon receipt of a Noop command (as specified in section 3.1.5.13), Message command (as specified in section 3.1.5.10), or ConnectClose command (as specified in section 3.1.5.4) containing a nonzero MessageCount field, the indicated number of oldest entries in the OutboundMessageList table for the underlying **connection (2)** MUST be removed. The higher-layer application can discard any associated data.

Additional server-specific processing of acknowledgments received by a relay server on a fanout **session** MUST be performed as specified in section 3.3.5.19.

3.1.6 Timer Events

3.1.6.1 Message Acknowledgment Timer Event

Upon expiry of this timer, the local device MUST send to the remote device a Noop command with the MessageCount field set to the value as determined by the method specified in section 3.1.4.2.1.

3.1.7 Other Local Events

3.1.7.1 Transport Loss

Upon loss of the underlying transport layer, the application MUST remove **connection (1)** state and **session** state for all sessions associated with the lost transport. For any open outbound sessions, message sequences are considered to be successfully delivered if a delivery acknowledgment has been received. Delivery acknowledgment is indicated if a previously received Message or Noop command contained a nonzero MessageCount, as specified in 3.1.5.10 or 3.1.5.13. Any message sequences remaining in the OutboundMessageList table, defined in 3.1.1.2, after MessageCount processing is finished, as specified in 3.1.5.19, are not considered to be successfully delivered.

With loss of the transport, the application MUST discard all connection (1) authentication information as specified in [MS-GRVSSTPS] associated with the device which was communicating via the transport.

3.2 Client Role

This section specifies the requirements to fulfill the client role.

3.2.1 Abstract Data Model

A client MUST maintain appropriate state variables to support SSTP **connections (1)** and **sessions**, and acknowledgment message lists as specified in section 3.1.1.

For a client connection (1) to a relay server that performs SSTP Security protocol authentication, the following state variables MUST be maintained for the transport layer connection (2):

AttachSessionIds: A set of SessionIds used by attach sessions.

RegisterSessionIds: A set of SessionIds used by register sessions.

Additional state variables as specified in [MS-GRVSSTPS] are required for a client to authenticate on a connection (1) with a relay server.

3.2.2 Client Specific Timers

None.

3.2.3 Initialization

As specified in section 3.1.3.

3.2.4 Higher-Layer Triggered Events

3.2.4.1 Authenticating to a Relay Server

When a client connects to a relay server which implements a local store for the client, the client SHOULD request retrieval of the stored messages from the relay server. To do this, the client MUST perform both device and account authentication and device and identity registration with the relay server by initiating the SSTP Security protocol exchanges as specified in [MS-GRVSSTPS].

Once a client has authenticated with the relay server, the authentication remains valid for the duration of the SSTP **connection (1)**.

The client higher-layer retains all application-assigned identification information, such as account, device, and identity URLs. For clients that have finished an initial authentication registration exchange with a relay server, the client retains the authentication token assigned by the relay server to the account/device and account/identity pairs.

3.2.4.1.1 Performing Device Authentication

The Connect, ConnectResponse, ConnectAuthenticate commands are used to carry SSTP Security protocol messages to exchange and verify authentication information for a device.

The following table shows the type of SSTP Security protocol messages that are carried in these SSTP commands, as specified in [MS-GRVSSTPS]:

SSTP Command	SSTP Security Message	Direction
Connect	SecConnect	Client to server
ConnectResponse	SecConnectResponse, SecConnectResponseDeviceRegistrationNeeded, SecConnectResponseDeviceAuthenticationFailed	Server to client
ConnectAuthenticate	SecConnectAuthenticate	Client to server

When connecting to a relay server, if the client wants to request delivery of messages stored in the relay server local store, the client MUST embed a SecConnect message in the AuthenticationToken field of the Connect command. The format of the SecConnect message is as specified in [MS-GRVSSTPS].

3.2.4.1.2 Performing Account Authentication

The Attach, AttachResponse, AttachAuthenticate commands are used to carry SSTP Security protocol messages to exchange and verify authentication information for an account.

The following table shows the types of SSTP Security protocol messages that are carried in these SSTP commands, as specified in [MS-GRVSSTPS]:

SSTP Command	SSTP Security Message	Direction
Attach	SecAttach	Client to server

SSTP Command	SSTP Security Message	Direction
AttachResponse	SecAttachResponse, SecAttachResponseAccountRegistrationNeeded, SecAttachResponseNewDeviceRegistrationNeeded, SecAttachResponseAccountAuthenticationFailed	Server to client
AttachAuthenticate	SecAttachAuthenticate	Client to server

Following successful device authentication, the client MUST initiate account authentication by sending an Attach command on a **connection (1)** in the 'established' state for each account to be authenticated.

The client MUST set the EventId field to a valid and unused **session** identifier for the connection (1), as specified in section 3.1.4.3.1. The session identifier MUST be added to the AttachSessionIds.

The Attach command IdentityURL field identifies the account which is being submitted for authentication, and is supplied by the higher-layer. The ResourceURL field MUST be set to the relay URL of the relay server to which the client is connected. The client MUST embed a SecAttach message in the AuthenticationToken field of the Attach command.

The format of the SecAttach message is as specified in [MS-GRVSSTPS].

3.2.4.1.3 Performing Device and Identity Registration

The Register and RegisterResponse commands are used to carry SSTP Security messages to associate an identity or a device with an account.

The following table shows the types of SSTP Security protocol messages that are carried in these SSTP commands, as specified in [MS-GRVSSTPS]:

SSTP Command	SSTP Security Message	Direction
Register	SecDeviceAccountRegister, SecIdentityRegister	Client to server
RegisterResponse	SecDeviceAccountRegisterResponse	Server to client

The sequence of Register commands to be sent is as determined by [MS-GRVSSTPS], depending on whether or not the previously received AttachResponse contained an SSTP Security message requesting device registration.

If device registration was requested by the server Security protocol response, the client MUST embed a SecDeviceAccountRegister message in the RegistrationToken field of the Register command, add the EventId field to the AttachSessionIds, and send it to the relay server. The format of the SecDeviceAccountRegister message is as specified in [MS-GRVSSTPS].

Following successful account authentication, the client MUST register each identity assigned by the higher-layer to the account. The client MUST embed a SecIdentityRegister message in the RegistrationToken field of the Register command. The format of the SecIdentityRegister message is as specified in [MS-GRVSSTPS].

In a Register command containing a SecIdentityRegister message, the client MUST set the EventId field to a valid and unused **session** identifier for the **connection (1)**, as specified in section 3.1.4.3.1, and send it to the relay server. The session identifier MUST be added to the RegisterSessionIds.

3.2.5 Message Processing Events and Sequencing Rules

This section specifies the requirements to fulfill the client role in processing events.

3.2.5.1 Receiving a Connect Command

A client MUST process a Connect as specified in section 3.1.5.1.

3.2.5.2 Receiving a ConnectResponse Command

A client MUST process a ConnectResponse as specified in section 3.1.5.2.

If the ConnectResponse contains an AuthenticationToken field, it MUST be extracted and processed as specified by the SSTP Security protocol [MS-GRVSSTPS]. If [MS-GRVSSTPS] specifies a Security message to be returned to the sender, it MUST be embedded in the AuthenticationToken field of a ConnectAuthenticate command, and sent to the sender.

3.2.5.3 Receiving a ConnectAuthenticate Command

If a ConnectAuthenticate command is received by a client, the recipient MUST send a ConnectClose command as specified in section 3.1.4.2 with ReasonId of ProtocolError (0x03).

3.2.5.4 Receiving a ConnectClose Command

A client MUST process a ConnectClose as specified in section 3.1.5.4.

3.2.5.5 Receiving an Open Command

A client MUST process an Open as specified in section 3.1.5.5.

3.2.5.6 Receiving a FanoutOpen Command

If a FanoutOpen command is received by a client, the recipient MUST send a ConnectClose command as specified in section 3.1.4.2, with ReasonId of ProtocolError (0x03).

3.2.5.7 Receiving an OpenResponse Command

A client MUST process an OpenResponse as specified in section 3.1.5.7.

3.2.5.8 Receiving a SessionStatus Command

A client MUST process a SessionStatus as specified in section 3.1.5.8.

If the OutboundFanoutAddressingList for the **session** has no remaining members after completion of the processing specified in section 3.1.5.8, then the receiver SHOULD<16> send a Close command for that session.

3.2.5.9 Receiving a Close Command

A client MUST process a Close as specified in section 3.1.5.9.

If the SessionId field in the received Close command is in AttachSessionIds or RegisterSessionIds, it MUST be removed.

3.2.5.10 Receiving a Message Command

A client MUST process a Message as specified in section 3.1.5.10.

3.2.5.11 Receiving a Data Command

A client MUST process a Data as specified in section 3.1.5.11.

3.2.5.12 Receiving an EndMessage Command

A client MUST process an EndMessage as specified in section 3.1.5.12.

3.2.5.13 Receiving a Noop Command

A client MUST process a Noop as specified in section 3.1.5.13.

3.2.5.14 Receiving an Attach Command

If an Attach command is received by a client, the recipient MUST send a ConnectClose command as specified in section 3.1.4.2, with ReasonId of ProtocolError (0x03).

3.2.5.15 Receiving an AttachResponse Command

If the EventId field in a received AttachResponse is not in AttachSessionIds then the recipient MUST send a ConnectClose command as specified in section 3.1.4.2 with a ReasonId of ProtocolError (0x03).

The contained AuthenticationToken MUST be extracted and submitted to the SSTP Security protocol for validation and processing, as specified by the SSTP Security protocol [MS-GRVSSTPS]. The recipient MUST respond to the server with the command and message contents as determined by [MS-GRVSSTPS]. This MUST be either a Register, an AttachResponse, or a Close command, to be sent on the Attach **session**.

3.2.5.16 Receiving an AttachAuthenticate Command

If an AttachAuthenticate command is received by a client the recipient MUST send a ConnectClose command as specified in section 3.1.4.2, with ReasonId of ProtocolError (0x03).

3.2.5.17 Receiving a Register Command

If a Register command is received by a client the recipient MUST send a ConnectClose command as specified in section 3.1.4.2, with ReasonId of ProtocolError (0x03).

3.2.5.18 Receiving a RegisterResponse Command

If the EventId field in a received RegisterResponse is not in AttachSessionIds or RegisterSessionIds, the recipient MUST send a ConnectClose command as specified in section 3.1.4.2 with a ReasonId of ProtocolError (0x03).

If the RegistrationTokenLength field is nonzero, then the contained RegistrationToken MUST be extracted and submitted to the SSTP Security protocol for validation and processing, as specified by the SSTP Security protocol [MS-GRVSSTPS].

3.2.6 Client Specific Timer Events

None.

3.2.7 Other Local Events

None.

3.3 Relay Server Role

This section specifies the requirements to fulfill the relay server role.

3.3.1 Abstract Data Model

For relay server **connection (1)** with a client, on which the client is performing SSTP Security protocol authentication, the following state variables MUST be maintained for each transport layer connection (2):

AttachSessionIds: A set of SessionIds used by Attach **sessions**.

RegisterSessionIds: A set of SessionIds used by a Register sessions.

Additional state variables as specified in [MS-GRVSSTPS] are required for a relay server to authenticate a connecting client.

In addition to the common state variables specified in section 3.1.1, a session receiver on a relay server MUST maintain the following state variables for each inbound fanout session:

MessageProcessingState: A variable associated with a specific message sequence received for a fanout session, which tracks the processing state of the resource handler in handling of submitted application data. The message processing state is either 'processing' or 'complete'.

FanoutAddressingList: The list of addressing entries for a received fanout session, consisting of a common ResourceURL, and a list of IdentityURL, DeviceURL, and RelayURLs identifying the recipient resource handlers. In SSTP 1.6, each fanout addressing entry has an additional FailoverDeviceIndexes field. For each entry in the list, there is an associated:

- ResourceHandlerAvailability state
- Reference to an outbound single-hop session
- MessageProcessingState for each message sequence received by the fanout session

The FanoutAddressingList is maintained for the life of the session. The original list entry order MUST be maintained with no entries added or deleted.

AggregateResourceHandlerAvailability: A variable which indicates the aggregate readiness of the collection of all ResourceHandlerAvailability states for all identified resource handlers contained in the FanoutAddressingList. The aggregate state is either 'ready' or 'notReady'.

3.3.2 Server Specific Timers

3.3.2.1 Offline Device Delivery Data TTL Timer

A timer used to detect excessive delays in the delivery of a message sequence for which the D (DoNotDeliverIfOffline) bit is set in the corresponding Message command. This timer specifies the maximum permissible time for delivery of this data to the **session** recipient. The default value is 5 minutes.<17>

3.3.2.2 Ephemeral Data Delivery Timer

A timer used to detect the inability to deliver a message sequence for which the E (Ephemeral) bit is set in the corresponding Message command. The timer is set to the value in seconds contained in the TTL field of the Message command.

3.3.3 Server Initialization

As specified in section 3.1.3.

3.3.4 Higher-Layer Triggered Events

3.3.4.1 Changing Resource Handler Availability State

3.3.4.1.1 Changing Resource Handler Availability State for a Non-fanout Session

This event MUST be processed as specified in section 3.1.4.5.

3.3.4.1.2 Changing Resource Handler Availability State for a Fanout Session

As specified in section 3.3.5.6, the receiving **session** has a ResourceHandlerAvailability state variable for each addressed fanout device entry contained in the FanoutAddressingList collection, and an AggregateResourceHandlerAvailability state variable representing the overall readiness of all fanout resource handlers.

The higher-layer can change the ResourceHandlerAvailability state for a resource handler which is the destination of a local multi-drop session. For such destinations, the higher-layer MUST provide notifications of changes in resource availability, and set the associated ResourceHandlerAvailability state as follows:

ResourceHandlerAvailability state	Description
ready	Can accept and process application data
notReady	Temporarily inaccessible, or not ready to accept and process application data
fault	Inaccessible, or has failed, or access has been denied by the higher-layer

When the higher-layer indicates the loss of the resource handler because of a fault, it MUST supply an application defined StatusId for the resultant SessionStatus command to be sent to the session originator. The StatusId MUST be one of QuotaWouldBeExceeded (0x04) or LockedOut (0x05).

For those FanoutAddressingList entries which correspond to outbound single-hop sessions, as specified in section 3.3.5.6.1, the associated ResourceHandlerAvailability state variable indicates the state of the locally originated outbound single-hop session. For those sessions, the ResourceHandlerAvailability change notifications are generated by the relay server, as follows:

ResourceHandlerAvailability state	Description
ready	Generated as specified in section 3.3.5.7 when the outbound session state is set to 'ready'
notReady	Generated as specified in section 3.3.5.7 when the outbound session state is set to 'opening', 'suspended', or 'blocked'
fault	Generated as specified in: ▪ section 3.3.5.7 when the session state is removed ▪ section 3.3.5.8 when a SessionStatus command is received ▪ section 3.3.5.9 when session state is removed ▪ section 3.3.7.1 when session state is removed

When the specified sections indicate the loss of a single-hop session because of a fault, the StatusId for the resultant SessionStatus command to be sent to the session originator is supplied as specified in the corresponding section. If the Version state variable of the SSTP **connection (1)** to the session originator indicates that the originator's connection (1) Version is SSTP 1.6, then the higher-layer MAY aggregate the list of faulting resources into a single SessionStatus reply to the session originator by supplying multiple devices in the FanoutDeviceIndexes field of the SessionStatus command rather than sending individual SessionStatus replies for each device<18>.

The receiving session MUST process the notification of change to a ResourceHandlerAvailability state as follows:

New ResourceHandlerAvailability state	Session Receiver Action
ready	Set the AggregateResourceHandlerAvailability state to 'ready' if the ResourceHandlerAvailability state for all entries in the FanoutAddressingList is 'ready'.
notReady	Set the AggregateResourceHandlerAvailability state to 'notReady' if the ResourceHandlerAvailability state for any entry in the FanoutAddressingList is 'notReady'.
fault	▪ MUST send a SessionStatus command to the session originator. The StatusId field MUST contain the StatusId provided with the fault notification. ▪ If the StatusId is QuotaWouldBeExceeded (0x04) or LockedOut (0x05), the device URL and identity URL of the failed resource handler as provided by the higher-layer MUST be set in the DeviceURL and IdentityURL fields.

New ResourceHandlerAvailability state	Session Receiver Action
	▪ If the StatusId is one of DNSLookupFailed (0x01), HostNotReachable (0x02), or ConnectionClosed (0x03), the RelayURL of the failed remote relay as provided by the originator of the notification MUST be set in the DeviceURL field. ▪ MUST mark the resource handler(s) from the FanoutAddressingList as unavailable for the receiving session. ▪ If all entries in the FanoutAddressingList are marked unavailable then the session receiver MUST send a Close command with ReasonId of EmptySession (0x15) to the session originator, and remove all session state.

If session state still exists, and these actions have caused a change to the AggregateResourceHandlerAvailability state for the receiving session, the session MUST then process the new AggregateResourceHandlerAvailability as follows:

Session state(session receiver)	New AggregateResourceHandlerAvailability	Session Receiver Action
suspended	ready	MUST send OpenResponse with ResponseId of StartSending (0x09). Session state MUST be set to 'ready'
suspended	notReady	None
ready	ready	None
ready	notReady	MUST send OpenResponse with ResponseId of StopSending (0x0a). Session state MUST be set to 'blocked'
blocked	ready	MUST send OpenResponse with ResponseId of StartSending (0x09). Session state MUST be set to 'ready'
blocked	notReady	None

3.3.4.2 Notifying of Resource Handler Completion

3.3.4.2.1 Notifying of Resource Handler Completion for Non-fanout Sessions

This event MUST be processed as specified in section 3.1.4.7.

3.3.4.2.2 Notifying of Resource Handler Completion for Fanout Sessions

When a resource handler completes processing of data submitted as specified in section 3.3.5.12.1, the MessageProcessingState for the associated FanoutAddressingList entry MUST be set to 'complete'.

If the MessageProcessingState for all FanoutAddressingList entries is 'complete', the MessageProcessingState for the associated message sequence in the InboundMessageList MUST also be set to 'complete'. At that time, the processing specified in section 3.1.4.7 MUST be performed.

3.3.5 Message Processing Events and Sequencing Rules

3.3.5.1 Receiving a Connect Command

A relay server MUST process a Connect as specified in section 3.1.5.1.

Additionally, if the Connect contains an AuthenticationToken field, it MUST be extracted and processed as specified by the SSTP Security protocol [MS-GRVSSTPS]. The ResponseId MUST be set to the value returned by the SSTP Security Protocol server processing. If [MS-GRVSSTPS] processing encounters an authentication error, it MUST provide a failure ResponseId of AuthenticationFailed (0x06) to be sent to the **connection (1)** originator. If [MS-GRVSSTPS] specifies a Security message to be returned to the connection (1) originator, it MUST be embedded in the AuthenticationToken field of the ConnectResponse.

3.3.5.2 Receiving a ConnectResponse Command

A relay server MUST process a ConnectResponse as specified in section 3.1.5.2.

If a ConnectResponse with a ResponseId of Ok (0x00) is received from a remote relay-server to which the **connection (1)** has been established in response to handling a single-hop fanout request, as specified in section 3.3.5.6.1, the recipient MUST open a fanout **session** to that server, as specified in section 3.1.4.3.3, for each requested single-hop fanout session to the remote relay. The OutboundFanoutAddressingList supplied to the fanout open request MUST be the subset of the FanoutAddressingList whose RelayURL matches the sender.

If a ConnectResponse with a ResponseId field containing any value other than Ok (0x00), the recipient MUST examine the set of all fanout sessions requested for the sending relay-server, as specified in section 3.3.5.6.1, and for each requested session the receiving relay server MUST generate an event indicating a ResourceHandlerAvailability state of 'fault', specifying a StatusId of ConnectionClosed (0x03), and the RelayURL of the remote relay. The generated ResourceHandlerAvailability notification MUST be processed as specified in section 3.3.4.1.2.

3.3.5.3 Receiving a ConnectAuthenticate Command

If the recipient's **connection (1)** state is not 'established', then the recipient MUST send a ConnectClose command as specified in section 3.1.4.2 with ReasonId of ProtocolError (0x03).

The contents of the AuthenticationToken field MUST be extracted and processed as specified by the SSTP Security protocol [MS-GRVSSTPS]. If [MS-GRVSSTPS] processing encounters an authentication error, the relay MUST send a ConnectClose command as specified in section 3.1.4.2 with a ReasonId of StaleConnectAuthenticate (0x06).

3.3.5.4 Receiving a ConnectClose Command

A relay server MUST process a ConnectClose as specified in section 3.1.5.4.

If the ConnectClose is received on a **connection (1)** to a remote relay server, the set of all outbound fanout **sessions** with that relay server are to be examined, and for each fanout session the receiving relay server MUST generate an event indicating a ResourceHandlerAvailability state of 'fault', specifying a StatusId of ConnectionClosed (0x03), and the RelayURL of the remote relay. The

generated ResourceHandlerAvailability notification MUST be processed as specified in section 3.3.4.1.2.

If the ConnectClose is received on a connection (1) with an inbound fanout session, where the fanout session had associated outbound single-hop sessions to remote relay servers in the FanoutAddressingList, the relay server MUST send to each of those relay servers a Close command with ReasonId EmptySession (0x15).

3.3.5.5 Receiving an Open Command

A relay server MUST process an Open as specified in section 3.1.5.5.

3.3.5.6 Receiving a FanoutOpen Command

If a FanoutOpen command is received with a SessionId field identifying a **session** for which there is no state, the recipient MUST send a ConnectClose as specified in section 3.1.4.2 with a ReasonId of TooManyUnknownSessionCmds (0x0f).

Upon receipt of a FanoutOpen command the receiving session state MUST be set to 'opening'. The recipient MAY send an OpenResponse command<19>. The ReasonId MUST be determined based on the fields in the Open command as follows:

- If the NumFanoutDeviceEntries field contains a value of 0 the fanout session MUST set the ResponseId to OK (0x00), and session state MUST be removed.
- If the locally configured value for Multi-dropSupported is false, and the FanoutOpen destination addresses specify a FanoutDeviceEntry RelayURL consisting of an empty string (0x00), or matching one of the locally configured LocalDeviceURLs, then the fanout session MUST be rejected by setting the ResponseId to NoFanoutEntries (0x08), and session state MUST be removed.
- If the locally configured value for SingleHopSupported is false, and the FanoutOpen destination addresses specify a FanoutDeviceEntry RelayURL containing a non-empty RelayURL which is not the same as one of the locally configured LocalDeviceURLs, then the fanout session MUST be rejected by setting the ResponseId to FanoutNotSupported (0x0c), and session state MUST be removed.
- If the ResourceURL is "grooveWanDPP" the fanout session MUST be rejected by setting the ResponseId to NoResource (0x04), and session state MUST be removed.
- The receiving device MUST validate the IdentityURL, DeviceURL, and RelayURL for each FanoutDeviceEntry found in the FanoutDeviceEntries list. If the validation of the IdentityURL, DeviceURL, and RelayURL fails, then the fanout session entry is not recognized by the receiver as a valid destination resource handler and the ResponseId MUST be set to Unknown (0x05), and session state MUST be removed.
- Otherwise, the ResponseId MUST be set to OkStopSending (0x0b), and session state MUST be set to 'suspended'.

For each valid fanout session entry, an entry MUST be added to the session FanoutAddressingList collection. The ResourceHandlerAvailability for each addressed resource in the FanoutAddressingList is determined as follows:

- For those FanoutAddressingList entries which identify a local relay server multi-drop target, the ResourceHandlerAvailability for the addressed resource MUST be set to 'notReady'. The higher-layer can notify of changes as specified in section 3.3.4.1.2.
- For those FanoutAddressingList entries which identify a remote relay server single-hop target, the ResourceHandlerAvailability for the addressed resource MUST be set to 'notReady', and subsequent processing as specified in section 3.3.5.6.1 MUST be performed.

Upon sending the OpenResponse for a session, the receiving session MessageReceiver state MUST be set to 'waiting'.

3.3.5.6.1 Single-Hop Processing

For each FanoutAddressingList entry containing a remote relay RelayURL, the higher-layer MUST determine if there is a suitable SSTP **connection (1)** with the designated remote relay server that is in the 'established' state.

The FanoutDeviceEntries field of the received FanoutOpen MUST be parsed as specified in section 2.2.6.1 based upon the connection (1) Version state variable of the receiving connection (1). If the version-specific parsing does not process exactly the number of bytes indicated by the FanoutOpen CommandLength field, then the recipient MUST send a ConnectClose as specified in section 3.1.4.2 with a ReasonId of ProtocolError (0x03).

If a connection (1) in the 'established' state to the remote relay server is located, the relay server MUST open a fanout **session** to that server, as specified in section 3.1.4.3.3. The OutboundFanoutAddressingList supplied to the outbound single-hop session MUST be those addressing entries from the FanoutAddressingList for the receiving session of section 3.3.5.6, where the RelayURL matches the URL of the remote relay on the SSTP connection (1).

If no such connection (1) exists, the server MUST attempt to establish an underlying transport to the destination and establish an SSTP connection (1), as specified in section 3.1.4.1. If a transport connection (2) to a relay server cannot be established, the higher-layer MUST generate a ResourceHandlerAvailability fault notification for the specific FanoutAddressingList entry, specifying a failure code of either DNSLookupFailed (0x01), or HostNotReachable (0x02), and the RelayURL of the unreachable relay, to be processed as specified in section 3.3.4.1.2.

3.3.5.7 Receiving an OpenResponse

A relay server MUST process an OpenResponse as specified in section 3.1.5.7.

Additionally, if the OpenResponse is received for an outbound single-hop **session**, and results in a change of session state, the relay server MUST generate an event indicating the ResourceHandlerAvailability state for all FanoutAddressingList entries using this session as follows:

Session state	ResourceHandlerAvailability state
opening	notReady
suspended	notReady
ready	ready
blocked	notReady
(session end-state)	fault

If the OpenResponse processing results in removal of session state, then the generated ResourceHandlerAvailability MUST indicate a 'fault' state, specifying a StatusId of ConnectionClosed (0x03), and the RelayURL of the unreachable relay. The generated ResourceHandlerAvailability notification MUST be processed as specified in section 3.3.4.1.2.

3.3.5.8 Receiving a SessionStatus Command

A relay server MUST process a SessionStatus as specified in section 3.1.5.8.

Additionally, if a SessionStatus command is received from an outbound single-hop **session** indicating the loss of a remote resource handler, the receiving relay server MUST generate an event indicating a ResourceHandlerAvailability state of 'fault', and include the StatusId, DeviceURL and IdentityURL fields of the received SessionStatus command in the event. The generated ResourceHandlerAvailability notification MUST be processed as specified in section 3.3.4.1.2.

3.3.5.9 Receiving a Close Command

A relay server MUST process a Close as specified in section 3.1.5.9.

Additionally, if the Close command is received for an outbound single-hop **session**, the relay server MUST generate a ResourceHanlderAvailability event indicating a 'fault' state for all FanoutAddressingList entries using this session, specifying a failure code of ConnectionClosed (0x03), and the RelayURL of the remote relay. The generated ResourceHandlerAvailability notification MUST be processed as specified in section 3.3.4.1.2.

If the Close is received for an inbound fanout session, where the fanout session had associated outbound single-hop sessions to remote relay servers in the FanoutAddressingList, the relay server MUST send to each of those relay servers a Close command with ReasonId EmptySession (0x15).

3.3.5.10 Receiving a Message Command

A relay server MUST process a Message as specified in section 3.1.5.10.

3.3.5.11 Receiving a Data Command

A relay server MUST process a Data as specified in section 3.1.5.11.

3.3.5.12 Receiving an EndMessage Command

A relay server MUST process an EndMessage as specified in section 3.1.5.12.

Upon receipt of the EndMessage command for a message sequence for which the D (DoNotDeliverIfOffline) bit was set in the corresponding Message command, the relay server MUST process the message sequence in the following manner:

- If client device online **presence** is published to the relay server as described in [MS-GRVWDPP], then:
- If the client resource handler is not online, the data MUST be discarded.
- If the client resource handler is online, the relay server MUST start an instance of the Offline Device Delivery Data TTL timer (section 3.3.2) specific to the received message sequence, and attempt to deliver the data to the client as specified in section 3.1.4.6. If the relay server is able to deliver the message sequence to the targeted client resource handler before expiry of this timer, the timer MUST be cancelled.
- If client device online presence is not published to the relay server, the relay server MUST start an instance of the Offline Device Delivery Data TTL timer (section 3.3.2) specific to the received message sequence, and attempt to deliver the data to the client as specified in section 3.1.4.6. If the relay server is able to deliver the message sequence to the targeted client resource handler before expiry of this timer, the timer MUST be cancelled.

Upon receipt of the EndMessage command for a message sequence for which the E (Ephemeral) bit was set in the corresponding Message command in the Message command, the relay server MUST process the message sequence in the following manner:

- The relay server MUST start an instance of the Ephemeral Data Delivery Timer (section 3.3.2) specific to the received message sequence, where the timer is set to the value of the TTL field from the corresponding Message command, and attempt to deliver the data to the client as specified in section 3.1.4.6. If the relay server is able to deliver the message sequence to the targeted client resource handler before expiry of this timer, the timer MUST be cancelled.

3.3.5.12.1 Fanout Session

For each entry in the FanoutAddressingList a MessageProcessingState for this message sequence MUST be created and set to the 'processing' state. The message information MUST be added to the InboundMessageList table for the underlying **connection (2)** as the most recently received message sequence. The InboundMessageList state for this entry MUST be set to 'processing'.

3.3.5.12.1.1 Multi-drop Resource Handler Processing

The data assembled by the MessageReceiver while in the 'buffering' state MUST be submitted to the resource handler associated with each FanoutAddressingList entry in the fanout **session**.

3.3.5.12.1.2 Single-hop Resource Handler Processing

For those target resource handlers that reside on a remote relay server (single-hop fanout), the message sequence MUST be processed by forwarding it to the remote relay server on the associated outbound fanout **session**. The message sequence MUST be sent on the single-hop fanout session as specified in section 3.1.4.6.

3.3.5.13 Receiving a Noop Command

A relay server MUST process a Noop as specified in section 3.1.5.13.

3.3.5.14 Receiving a Attach Command

If the **connection (1)** is not in the 'established' state, or a **session** with the session identifier in the EventId field already exists, or the AttachSessionId state variable (as specified in section 3.3.1) for this connection (1) already exists, the recipient MUST close the connection (1) as specified in section 3.1.4.2 with a ReasonId of TooManyUnknownSessionCmds (0x0f). Otherwise, if the recipient does not close the connection (1) with a ReasonId of TooManyUnknownSessionCmds (0x0f), the value of the EventId field MUST be added to the AttachSessionIds.

The recipient MUST extract the contained AuthenticationToken field, and submit it to SSTP Security protocol validation as specified in [MS-GRVSSTPS]. The recipient MUST respond to the client with the command and message contents as determined by [MS-GRVSSTPS], to be sent on the attach session.

3.3.5.15 Receiving an AttachResponse Command

If an AttachResponse command is received by a relay server, the recipient MUST send a ConnectClose command as specified in section 3.1.4.2 with ReasonId of ProtocolError (0x03).

3.3.5.16 Receiving an AttachAuthenticate Command

If the **connection (1)** is not in the 'established' state, or the value of the EventId field is not in the AttachSessionIds, the recipient MUST close the connection (1) as specified in section 3.1.4.2 with a ReasonId of TooManyUnknownSessionCmds (0x0f).

The recipient MUST extract the contained AuthenticationToken field, and submit it to SSTP Security protocol validation as specified in [MS-GRVSSTPS]. The recipient MUST respond to the client with the command and message contents as determined by [MS-GRVSSTPS], to be sent on the attach **session**.

3.3.5.17 Receiving a Register Command

The recipient MUST extract the contained RegistrationToken field, for submission to SSTP Security protocol validation as specified in [MS-GRVSSTPS].

If the contained Security message is a [MS-GRVSSTPS] SecIdentityRegister message, if a **session** with the session identifier in the EventId field already exists, or the EventId field is in the AttachSessionIds or the RegisterSessionIds, the recipient MUST close the **connection (1)** as specified in section 3.1.4.2 with a ReasonId of TooManyUnknownSessionCmds (0x0f). Otherwise the value of the EventId field MUST be added to the RegisterSessionIds.

For any other contained Security message, if a session with the session identifier in the EventId field already exists, or the value of the EventId is not in the AttachSessionIds for this connection (1), the recipient MUST close the connection (1) as specified in section 3.1.4.2 with a ReasonId of TooManyUnknownSessionCmds (0x0f).

The recipient MUST submit the extracted Security message to SSTP Security protocol validation as specified in [MS-GRVSSTPS]. The relay MUST respond to the client with the command and message contents as determined by [MS-GRVSSTPS]. This MUST be either a RegisterResponse or a Close command, to be sent on the same session as the incoming Register command. [MS-GRVSSTPS] specifies the conditions under which the RegisterResponse command is sent and specifies the conditions under which the Close command is sent.

3.3.5.18 Receiving a RegisterResponse Command

If a RegisterResponse command is received by a relay server, the recipient MUST send a ConnectClose command as specified in section 3.1.4.2 with ReasonId of ProtocolError (0x03).

3.3.5.19 Receiving Acknowledgments in a MessageCount Field

A relay server MUST process an acknowledgment in a MessageCount field as specified in section 3.1.5.19.

Additionally, when an acknowledgment is received for messages sent on an outbound fanout **session**, the MessageProcessingState for the corresponding FanoutAddressingList entries MUST be set 'complete'. Completion notifications MUST be generated as specified in section 3.1.4.7.

3.3.6 Server Specific Timer Events

3.3.6.1 Offline Device Delivery Data TTL Timer Timeout Event

Occurs when the time to deliver data contained in a message sequence marked with the D (DoNotDeliverIfOffline) bit has been exceeded. The relay server MUST discard the data, and cancel any other timers associated with the message sequence.

3.3.6.2 Ephemeral Data Delivery Timer Timeout Event

Occurs when the time to deliver data contained in a message sequence marked with the E (Ephemeral) bit has been exceeded. The relay server MUST discard the data, and cancel any other timers associated with the message sequence.

3.3.7 Other Local Events

3.3.7.1 Transport Loss and Fanout Sessions

If the **connection (1)** for an outbound single-hop fanout **session** to a remote relay server fails, the relay server MUST generate an event indicating a 'fault' state for the outbound session, specifying a failure code of ConnectionClosed (0x03), and the RelayURL of the remote relay. The generated fault event MUST be processed as specified in section 3.3.4.1.2.

If the connection (1) with the originator of a fanout session is lost, where the fanout session had associated outbound single-hop sessions to remote relay servers in the FanoutAddressingList for the receiving session, the relay server MUST send to each of those relay servers a Close command with ReasonId EmptySession (0x15).

Connection (1) state and any session state associated with the lost underlying transport MUST be removed.

4 Protocol Examples

In the following protocol examples, establishment of the underlying transport layer **connection (2)** between any two participating devices occurs prior to transmission of an initial SSTP Connect command. Similarly, dropping of the transport layer connection (2) between participating devices occurs following a ConnectClose, where the transport can be dropped by either device.

For each of the following message exchange scenarios, only those message fields which are most relevant to the particular use case are explicitly called out in the diagram message annotations

4.1 Initial SSTP Connection Establishment Examples

4.1.1 Version Negotiation

Protocol version negotiation is implicit in accepting and returning a ConnectResponse containing a ResponseId Ok. Although the currently supported versions of SSTP are 1.5 and 1.6, the following illustrates a possible future scenario in which an updated version of SSTP has been specified as version 3.6.

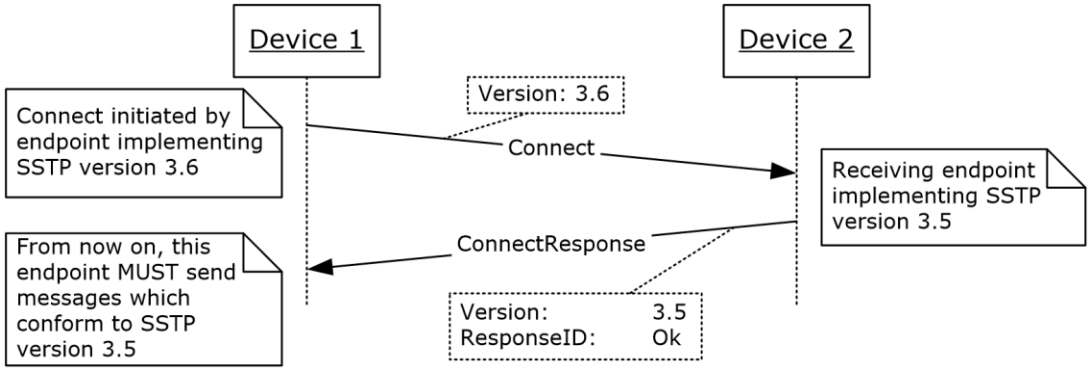


Figure 12: Connect accepted: implicit version negotiation

Following this initial **connection (1)**, both devices exchange message conformant to SSTP version 3.5 only. If either device subsequently detects an incoming byte stream which does not comply with the version 3.5 specification, the receiving device treats it as a protocol error.

4.1.2 Incorrect Target

Before accepting an SSTP **connection (1)** request, the receiving device ensures that it is the device with which the originator intended to connect, by comparing the supplied TargetDeviceURL with the recipient's local device URL. The local DeviceURLs of both Device 1 and Device 2 are maintained by each device as persistent initialization data.

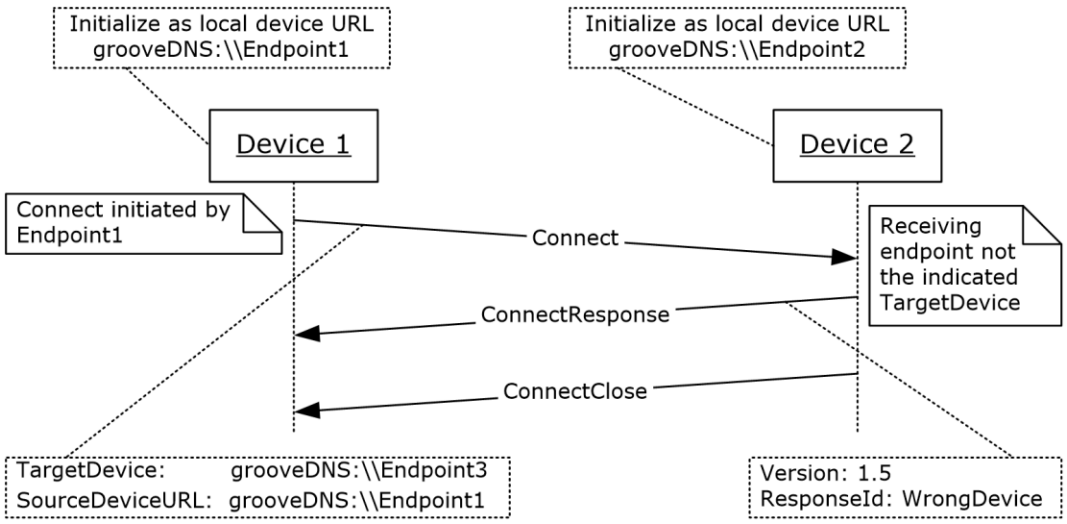


Figure 13: Connect rejected: incorrect destination URL

4.1.3 Connection Authentication

See [MS-GRVSSTPS] for a full specification of the underlying SSTP Security protocol, its usage within SSTP **connections (1)**, attach and registration sequences, and authentication protocol samples.

4.2 Session Management Examples

The following sections present contrasting SSTP **session** scenarios, involving client-to-client message exchange, client message exchange with a relay server, and client message exchange with fanout.

4.2.1 Device to Device Session

This scenario involves two client devices with an established SSTP **connection (1)** where client A is to send a message to client B. To begin, client A sends an Open command to client B, addressing the Open to an address entry of URLs on B's device: B's identity URL (client B), resource handler URL (apphandler), and device URL (device2). When client B responds with an OpenResponse, client A sends a data message to client B in several Data commands. Upon receipt of a Noop command from client B, client A returns a Close command to close the **session**. This illustrates the basic exchanges involved in opening a session and performing data transfer from one device to another.

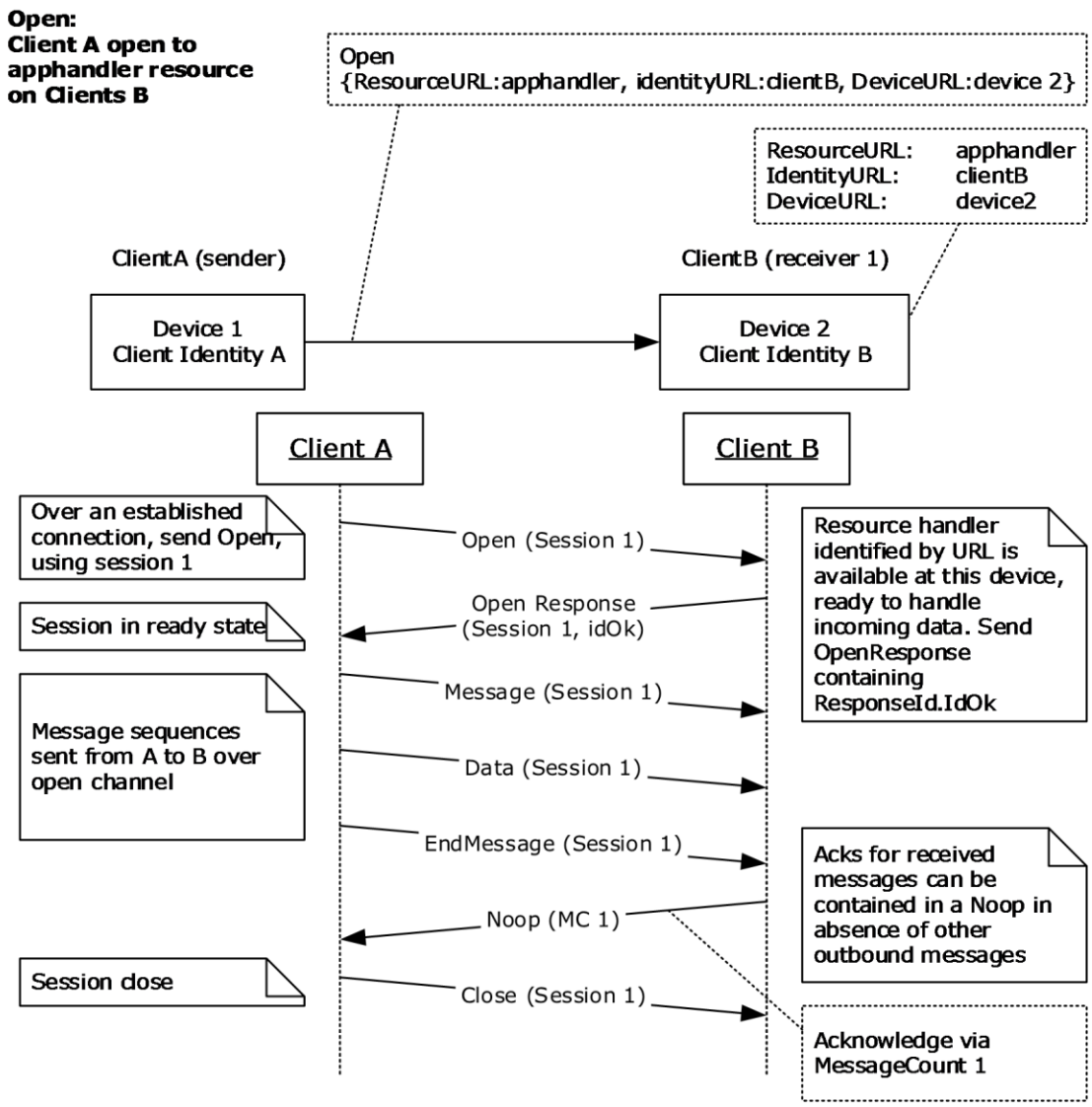


Figure 14: SSTP session between client devices

4.2.2 Device to Device Bi-directional Session Open

This scenario illustrates the opening of **sessions** by both SSTP devices on a **connection (1)**, resulting in bidirectional data transfer over independent sessions. In this scenario, client B also opens a session to client A, where client B selects a new SessionId from its valid range. Both participating devices can send data as message sequences simultaneously over their outbound sessions. The message sequence recipient acknowledges receipt and processing of the incoming data in a subsequent Message or Noop command.

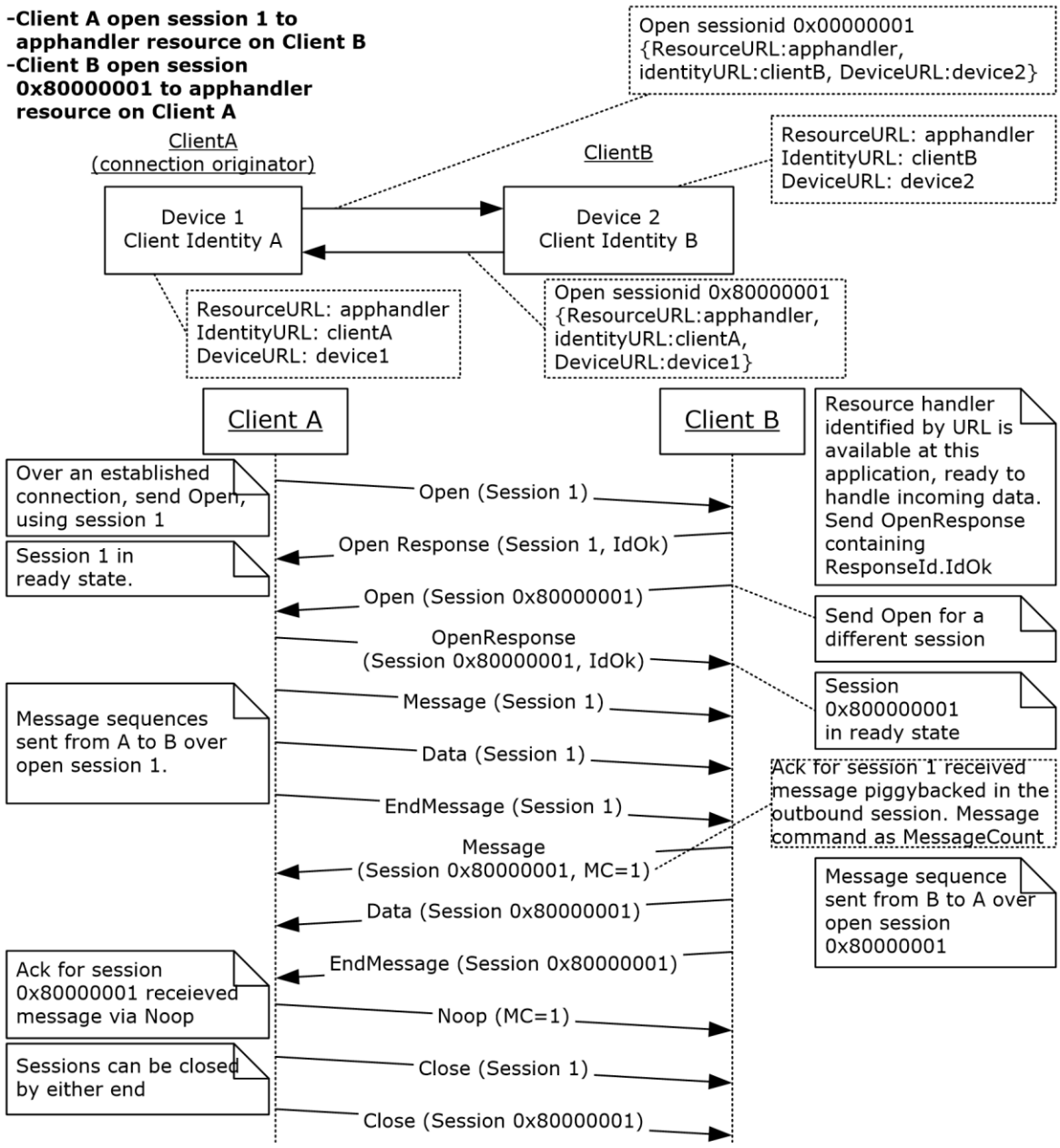


Figure 15: Bi-directional sessions between devices

4.2.3 Device to Relay Server Session Open

In the previous example, client A had a direct SSTP **connection (1)** to client B, and A was able to open a **session** directly to the desired resource handler on B. If this is not possible, client A can instead establish an SSTP connection (1) to the relay server associated with client B, and send the data to the relay server.

In this scenario, client A connects to client B through Relay1, an intermediary relay server which supports client authentication and provides message store-and-forward functionality.

The process of transmitting data from client A to client B through a relay server involves two essentially independent SSTP conversations: Client A transmits a message sequence to the addressing entry of client B on the relay server, which the server retains in a local store. Client B establishes an authenticated connection (1) and registers, which causes the relay server to open a dequeue session to the authenticated client B, and deliver the contents of the local store to client B.

The following diagram illustrates this scenario:

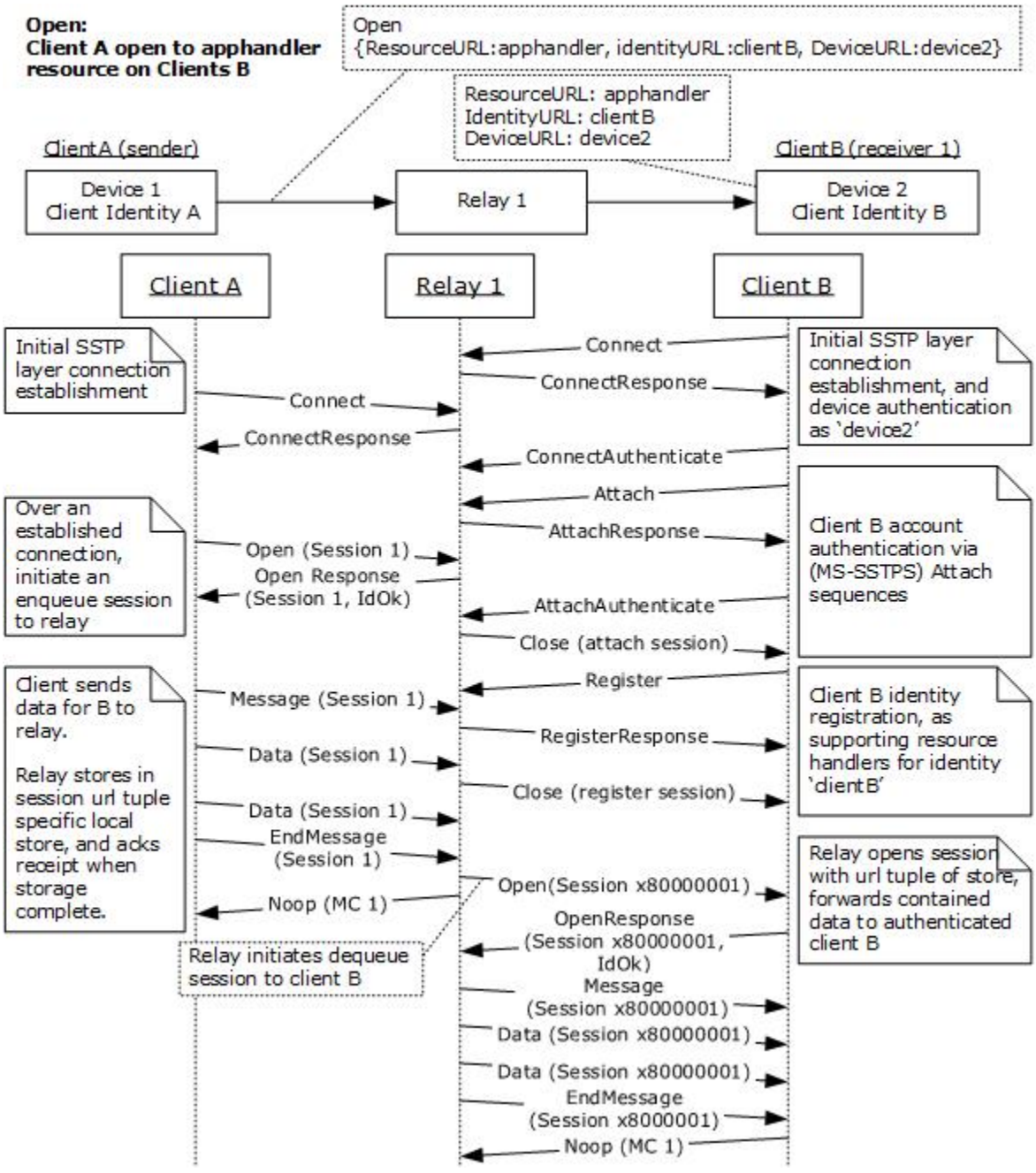


Figure 16: SSTP session via authenticating relay

4.2.4 Fanout Open

If a relay does not support single-hop fanout, this is indicated to the **connection (1)** initiator in the ConnectResponse command. There can be cases in which the client can nevertheless send a FanoutOpen command containing single-hop destination addresses in the fanout device list to an application which does not support this functionality. In this case, the relay denies the **session**, as illustrated in the following diagram.

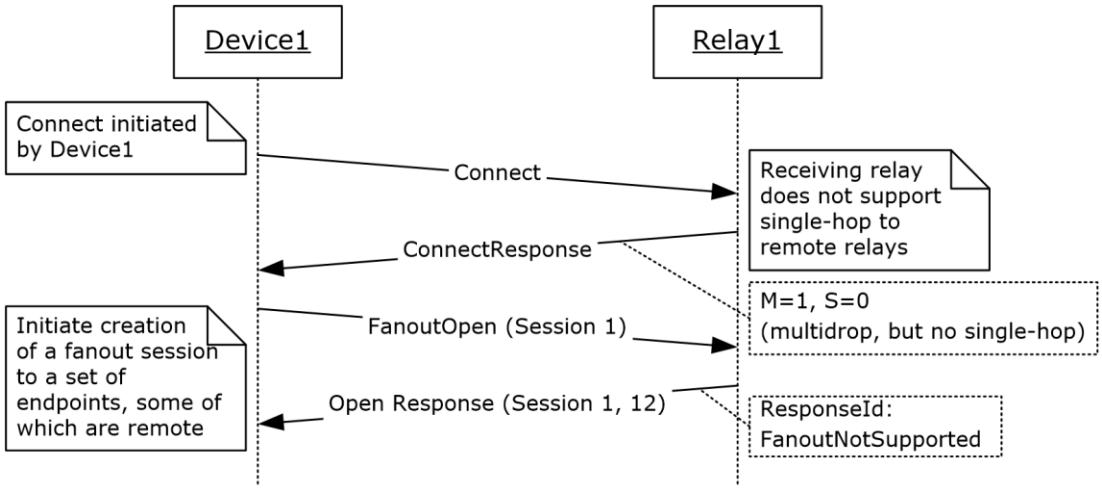


Figure 17: Fanout denied

4.2.5 Multi-drop Fanout

In the following multi-drop fanout scenario<20>, client A is about to send a message to clients B and C which both have the resource handler identified by the URL 'apphandler'. These two clients are accessible through the same relay server to which the sending client A is connected. Client A therefore uses the FanoutOpen command to open a fanout **session**, specifying the two target clients in the FanoutOpen FanoutDeviceEntries list and the desired ResourceURL. Because the sending and receiving clients share the same relay server, the RelayURL field of the fanout device entries can be the empty string.

In handling a multi-drop fanout session, the relay server forwarding the messages provides message acknowledgment to the originating client (client A) only when it has successfully delivered the message to all target resource handlers. Client A is assured of message delivery only when it receives this message acknowledgment.

To properly determine the fanout addressing entry contained in the fanout device entry list, the originating device has to have access to the relay URLs for each destination.

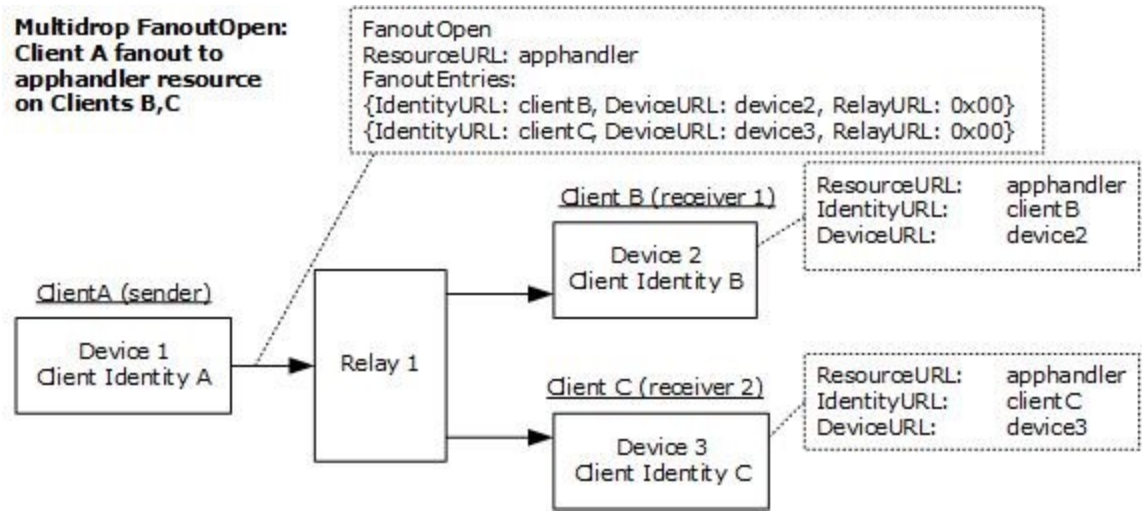


Figure 18: Fanout session

4.2.6 Single-Hop Fanout

In the following SSTP 1.5 single-hop fanout scenario, client A is about to send a message to the resource handler identified by the URL 'apphandler' on all of the target clients B, C, D, and E, where B and C are associated with Relay1, but D and E are associated with Relay2.

Relay1 processes the addressing entries for clients B and C as local multi-drop targets, and D and E as remote targets on Relay2. Relay1 establishes an independent SSTP fanout **session** to Relay2.

While Relay1 is establishing the single-hop fanout session to Relay2, the originating session with Client A is in the 'suspended' state. The session with Client A is set to the 'ready' state only after the fanout session to the remote relay is set to the 'ready' state.

When message sequences are received from client A, Relay1 forwards copies of the messages to local multi-drop recipients B and C, and also forwards a copy of incoming message sequences received from A to Relay2 over the outbound single-hop session. Relay2 forwards copies of the received message sequences to the local multi-drop recipients D and E.

When providing message acknowledgment to client A, Relay1 also processes acknowledgments received from Relay2, in addition to acknowledgments from the local resource handlers for the local multi-drop recipients B, C. When an acknowledgment is returned to Client A, it is Relay1's assurance of delivery of the message sequence to all fanout recipients B, C, D, and E.

From the perspective of Relay2, the incoming session originated by Relay1 is handled as a local multi-drop session for the Relay2 local clients D and E as described in the in the multi-drop fanout example in section 4.2.5.

The following diagram illustrates this SSTP 1.5 single-hop fanout session:

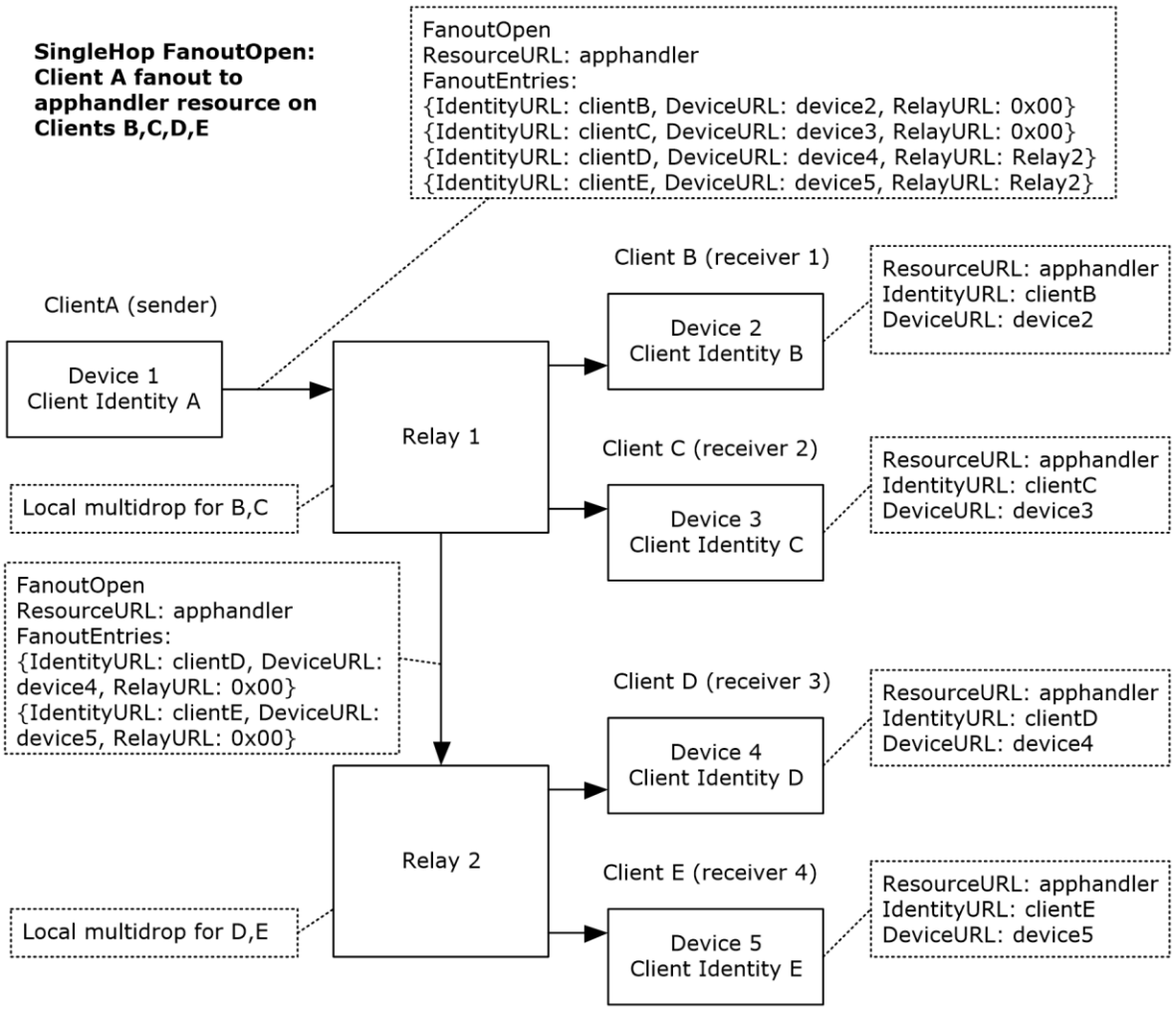


Figure 19: Single-hop fanout session

The following diagram, based on SSTP 1.5, shows the sequence of exchanged SSTP messages for this single-hop fanout session, from opening the session to delivery of the first message sequence. The session identifier used between client A and Relay1 is selected by client A when it originates the fanout session to Relay1. The session identifier used between Relay1 and Relay2 is selected by Relay1 when it originates the single-hop fanout session to Relay2:

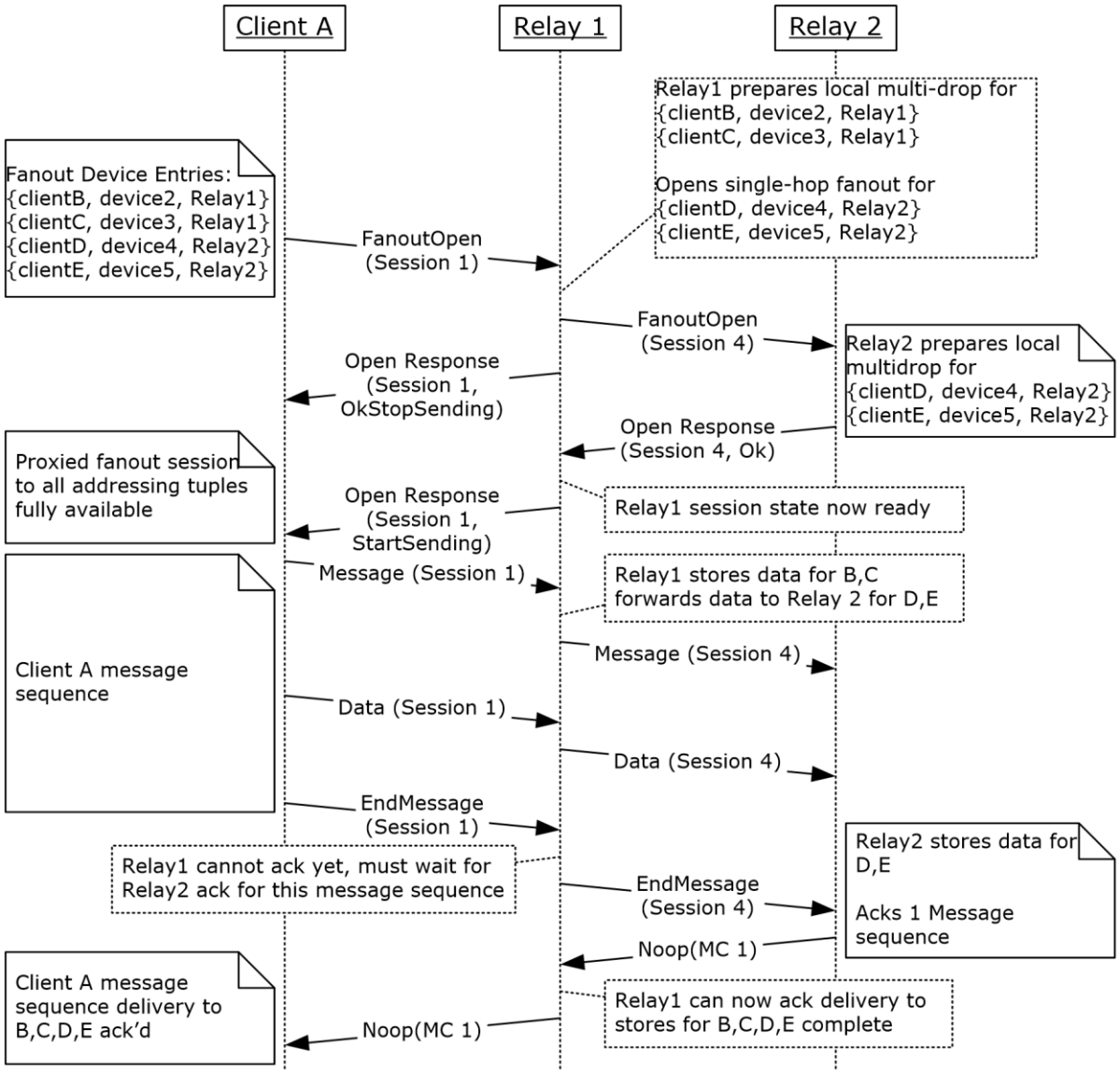


Figure 20: Single-hop fanout session messages

4.2.6.1 Single-Hop Fanout – Loss of Remote Resource Handler

In a fanout **session**, if any fanout target is initially unavailable or becomes unavailable during the session, the participating relay server reports the unavailable client to the originating client using a SessionStatus command. How an originating client recovers from a failed message delivery is application-specific.

As in the single-hop fanout scenario of section 4.2.6, client A has established a fanout session to Relay1, targeting clients B, C, D, and E; clients B and C are assigned to Relay1; clients D and E are assigned to Relay2.

If client D becomes unavailable, Relay2 informs Relay1 by sending it a SessionStatus command. Upon receiving the SessionStatus, Relay1 will remove client D from the single-hop OutboundFanoutAddressingList, and generate a fault event for the ResourceHandlerAvailability state for client D. The fault event handler then removes client D from the originating session

FanoutAddressingList, and informs the originating client A of the failure by sending a SessionStatus command to client A.

Client A then also removes the failed target from its OutboundFanoutAddressingList for the session with Relay1.

The following sequence depicts the message flow if resource D becomes permanently unavailable during the lifetime of the session.

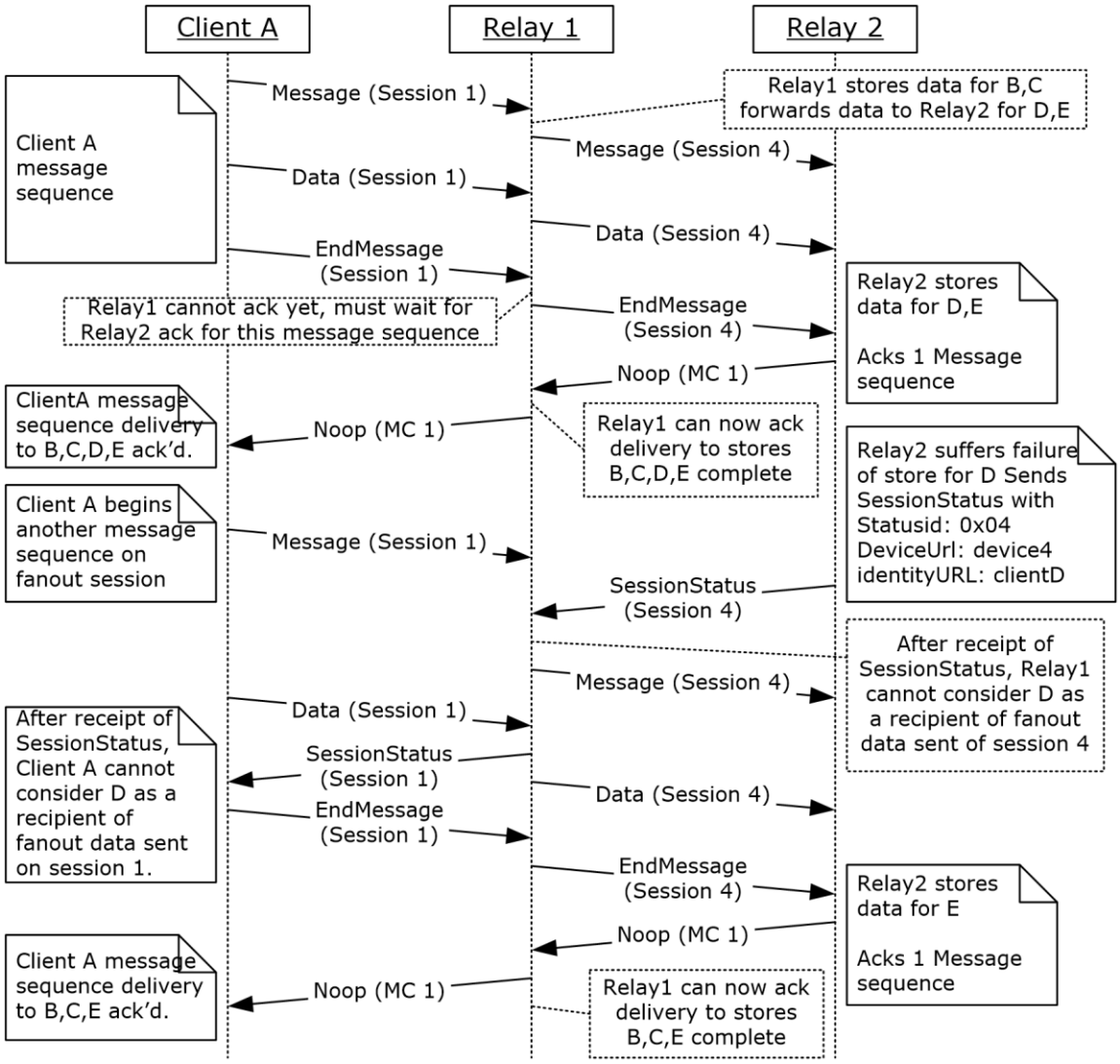


Figure 21: Single-hop fanout session with resource handler failure

4.2.6.2 Single-Hop Fanout Open – Loss of Remote Relay Server

If a single-hop **session** to a remote relay is lost, the relay server reports the loss to the originating client via a SessionStatus command.

As in the single-hop fanout scenario in section 4.2.6, client A has established a fanout session to Relay1, targeting clients B, C, D, and E; clients B and C are assigned to Relay1; clients D and E are assigned to Relay2.

If Relay1 loses connectivity with Relay2, Relay1 will remove state for the outbound single-hop session with Relay2, and generate a fault event for the ResourceHandlerAvailability state for clients D and E, as a ConnectionClosed StatusId for Relay2. The fault event handler then removes all clients with the Relay2 ResourceURL (clients D, E) from the originating session FanoutAddressingList, and informs the originating client A of the failure by sending the SessionStatus command to client A.

Client A then also removes the failed targets from its OutboundFanoutAddressingList for the session with Relay1.

The following sequence depicts the message flow if Relay2 becomes unavailable during the lifetime of the session.

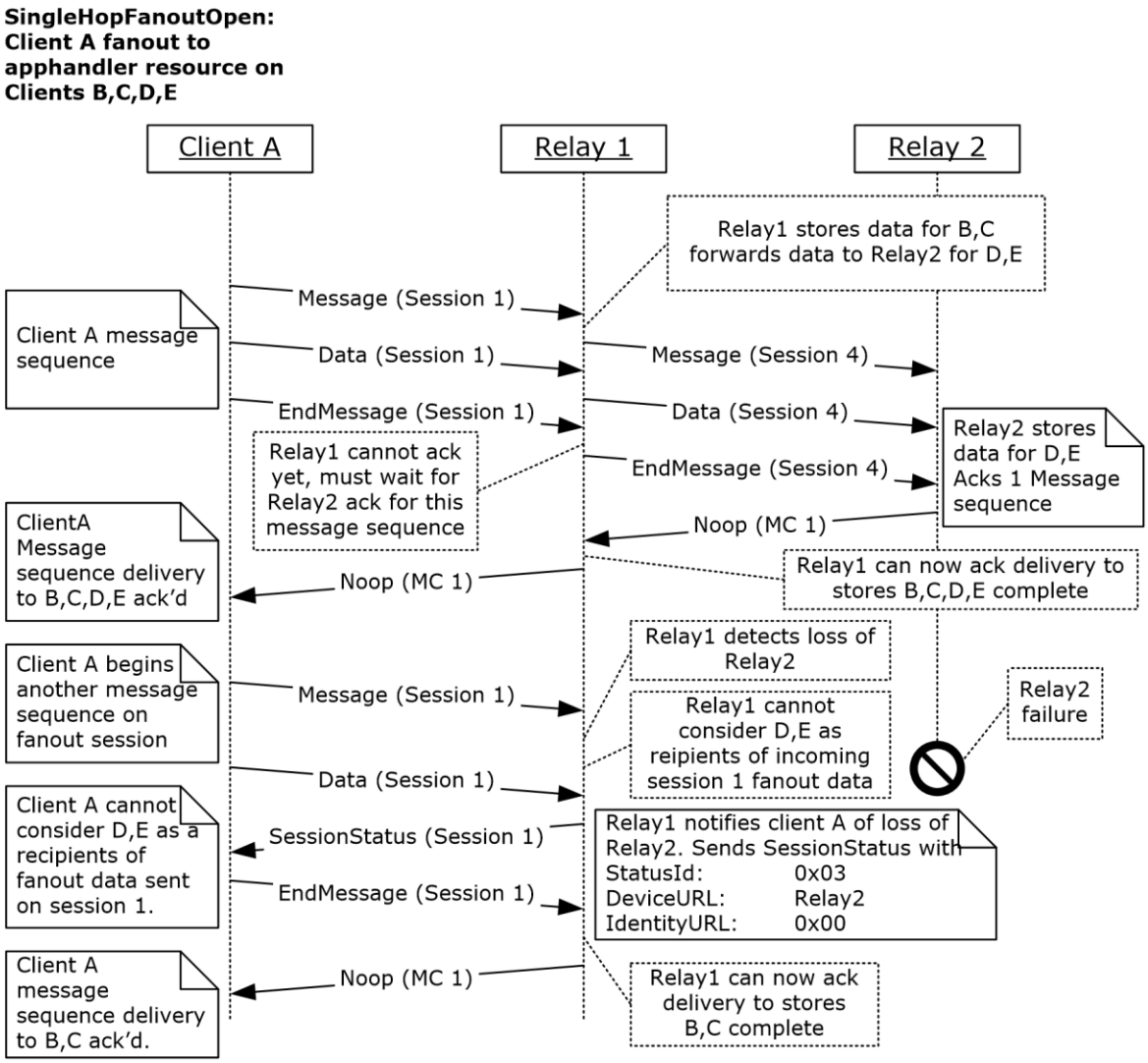


Figure 22: Single-hop fanout session with remote relay failure

4.3 Message Acknowledgment Examples

4.3.1 Acknowledgments for Multiple Sessions

If a device sends the message sequences A_1, A_2 on session 1, and message sequences B_1, B_2 on session 2, in the order A_1, B_1, A_2, B_2. The sender's OutboundMessageList is as follows (after processing specified in section 3.1.4.6 is finished):

Originators OutboundMessageList		
sequence	**session**	**message sequence**
1st	1	A_1
2nd	2	B_1
3rd	1	A_2
4th	2	B_2

Because the underlying stream transport preserves message ordering, after receiving all the inbound message sequences, the receiver's InboundMessageList is as follows (after processing specified in section 3.1.5.12 is finished):

Receivers InboundMessageList			
sequence	**session**	**message sequence**	**state**
1st	1	A_1	processing
2nd	2	B_1	processing
3rd	1	A_2	processing
4th	2	B_2	processing

If the processing required by the target resource handler for session 2 finishes before that for session 1, and all session 2 related resource handler completion notifications are processed, the receiver's InboundMessageList is as follows (after processing specified in section 3.1.4.7 is finished for session 2) :

Receivers InboundMessageList			
sequence	**session**	**message sequence**	**state**
1st	1	A_1	processing
2nd	2	B_1	complete

Receivers InboundMessageList			
3rd	1	A_2	processing
4th	2	B_2	complete

In this case, the receiver cannot yet send any acknowledgments to the sender, because the first received message sequence has not yet finished processing, and is still in the processing state. (because the processing specified in section 3.1.4.2.1 determines a MessageCount value of 0)

If the next message sequence to reach the processed state is A_1 (message sequence A_1 completes the processing specified in section 3.1.4.7), the receiver will update its InboundMessageList as follows:

Receivers InboundMessageList			
sequence	**session**	**message sequence**	**state**
1st	1	A_1	complete
2nd	2	B_1	complete
3rd	1	A_2	processing
4th	2	B_2	complete

The receiver now acknowledges both A_1 and B_1 simultaneously by sending a Noop or Message command containing a MessageCount field value of 2 (because the processing specified in section 3.1.4.2.1 determines a MessageCount value of 2), and remove those entries from its InboundMessageList, leaving the following:

Receivers InboundMessageList			
sequence	**session**	**message sequence**	**state**
1st	1	A_2	processing
2nd	2	B_2	complete

Upon receipt of the command containing MessageCount value 2 (as specified in section 3.1.5.15), the **session** originator has received acknowledgment of delivery and processing of the message sequences A_1 and B_1 and can update its OutboundMessageList correspondingly, as follows:

Originators OutboundMessageList		
sequence	**session**	**message sequence**
1st	1	A_2
2nd	2	B_2

On the receiving session, once the processing of A_2 completes (A_2 completes section 3.1.4.7), the receiver will update its InboundMessageList as follows

Receivers InboundMessageList			
sequence	**session**	**message sequence**	**state**
1st	1	A_2	complete
2nd	2	B_2	complete

The receiver can now acknowledge the final two message sequences, A_2 and B_2 simultaneously by sending a Noop or Message command containing a MessageCount field value of 2 (because the processing specified in section 3.1.4.2.1 again determines a MessageCount value of 2), and remove those entries from its InboundMessageList.

Upon receipt of the acknowledgment, the session originator can also remove the corresponding entries from its OutboundMessageList.

When viewed as a protocol exchange sequence, this exchange would be as in the following diagram (where the notation 'MS(1)-A_1' is used to signify the sequence of commands Message, Data, EndMessage, used to send the application supplied data block A_1 on session 1):

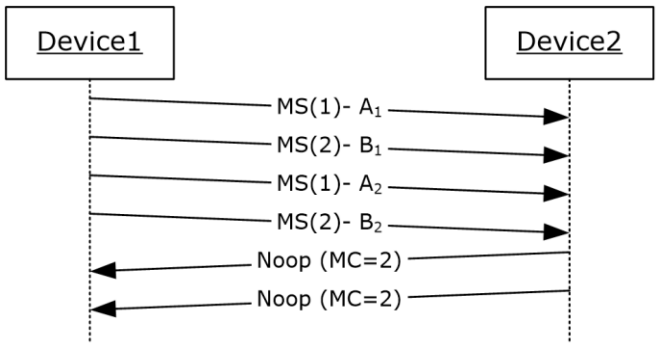


Figure 23: Multiple session acknowledgments - message sequences

4.3.2 Interleaved Message Sequences

In the preceding diagram, the depiction of a message sequence as a single transmission is a diagrammatic convenience only. The actual sequence of commands corresponding to any given message sequence is the specified Message, Data, [Data], …EndMessage commands. A sequence for one **session** can be interleaved with messages for another session.

The originator adds a message sequence to the OutboundMessageList upon sending the EndMessage command (as specified in section 3.1.4.6). The recipient adds the message sequence to the InboundMessageList on receipt of the EndMessage command.

The following sample sequence diagram illustrates this by explicitly showing the individual commands involved in transmission of message sequences from Device 1 to Device 2, with annotations indicating the appropriate acknowledgment list maintenance actions to be undertaken by the two devices:

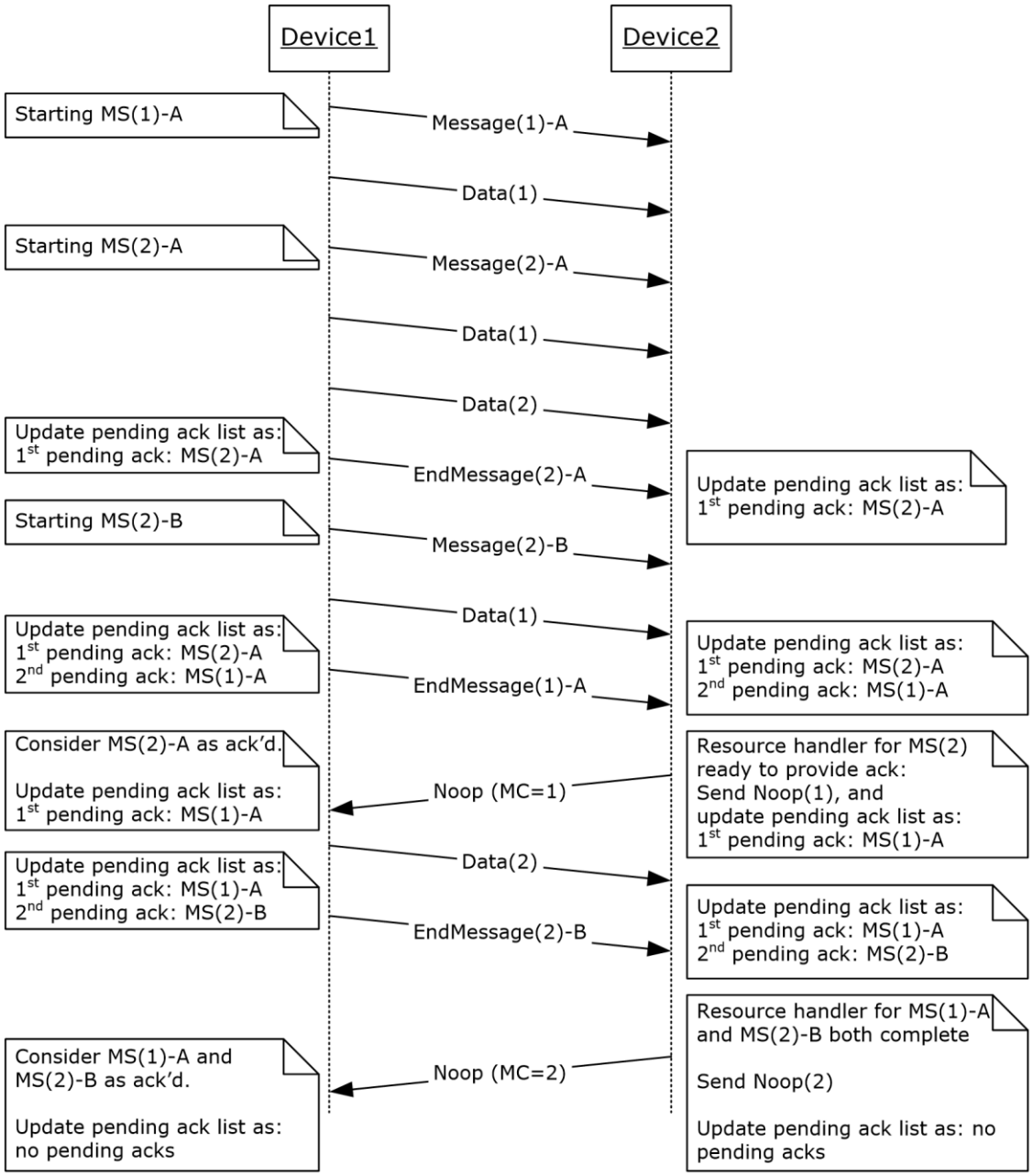


Figure 24: Interleaved message sequences

4.3.3 Acknowledgments for Single-Hop Fanout

In this example:

- Device1 session 1 addresses a resource handler resident on Relay2. The associated single-hop **session** from Relay1 to Relay2 is session 1 on the Relay1-Relay2 SSTP **connection (1)**.
- Device1 session 2 addresses a resource handler resident on Relay1
- Device1 session 3 addresses a resource handler resident on Relay3.The associated single-hop session from Relay1 to Relay3 is session 1 on the Relay1-Relay3 SSTP connection (1).

The following shows a sample of the acknowledgment handling for the case where the Device1 application sends one message on each session, in the following order:

- send A on session 1 (single hop to Relay2)
- send B on session 2 (handled locally on Relay1)
- send C on session 3 (single hop to Relay3)

First, the client, Device1 sends message A on session 1. Message A addresses a resource handler resident on Relay2. Device1 sends Message A to Relay1. Relay1 receives Message A and forwards it to Relay2 using outbound fanout session 1 on the Relay1-Relay2 SSTP connection (1):

Client Message Lists			**Relay1 Message Lists**			
Device1 to Relay1Originators OutboundMessageList			Device1 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	state
1st	1	A	1st	1	A	processing
			Outbound fanout forwards message A to Relay2.			
			Relay1 to Relay2 Originators OutboundMessageList			
			sequence	session	message	
			1st	1	A	
			Relay1 to Relay3 Originators OutboundMessageList			
			sequence	session	message	
			-	-	-	

After sending message A, Device1 sends message B on session 2. Message B addresses a resource handler resident on Relay1. Relay1 handles this message locally:

Client Message Lists	**Relay1 Message Lists**
Device1 to Relay1 Originators OutboundMessageList	Device1 to Relay1 Receivers InboundMessageList

Client Message Lists			Relay1 Message Lists			
sequence	session	message	sequence	session	message sequence	state
1st	1	A	1st	1	A	processing
2nd	2	B	2nd	2	B	complete
			Ack for A still pending from Relay2.			
			Relay1 to Relay2 Originators OutboundMessageList			
			sequence	session	message	
			1st	1	A	
			Relay1 to Relay3 Originators OutboundMessageList			
			sequence	session	message	
			-	-	-	

After sending messages A and B, Device1 sends message C on session 3. Message C addresses a resource handler resident on Relay3. Device1 sends Message C to Relay1. Relay1 receives Message C and forwards it to Relay3 using outbound fanout session 1 on the Relay1-Relay3 SSTP connection (1):

Client Message Lists			Relay1 Message Lists			
Device1 to Relay1 Originators OutboundMessageList			Device1 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	state
1st	1	A	1st	1	A	processing
2nd	2	B	2nd	2	B	complete
3rd	3	C	3rd	3	C	processing
			Ack for A still pending from Relay2.			
			Relay1 to Relay2 Originators OutboundMessageList			

Client Message Lists	Relay1 Message Lists		
	sequence	session	message
	1st	1	A
	Outbound fanout forwards C to Relay3		
	Relay1 to Relay3 Originators OutboundMessageList		
	sequence	session	message
	1st	2	C

Device1 acks for messages A, B, and C are still pending from Relay1. Relay1 receives Noop (MessageCount=1) from Relay3, as ack for message C:

Client Message Lists			Relay1 Message Lists			
Device1 to Relay1 Originators OutboundMessageList			Device1 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	state
1st	1	A	1st	1	A	processing
2nd	2	B	2nd	2	B	complete
3rd	3	C	3rd	3	C	complete
			Ack for A still pending from Relay2.			
			Relay1 to Relay2 Originators OutboundMessageList			
			sequence	session	message	
			1st	1	A	
			Ack has been received from Relay3, no pending acks.			
			Relay1 to Relay3 Originators OutboundMessageList			
			sequence	session	message	

Client Message Lists	Relay1 Message Lists		
	-	-	-

Device1 acks for messages A, B, and C are still pending from Relay1. Relay1 receives Noop (MessageCount=1) from Relay2, as ack for message A:

Client Message Lists			Relay1 Message Lists			
Device1 to Relay1 Originators OutboundMessageList			Device1 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	state
1st	1	A	1st	1	A	complete
2nd	2	B	2nd	2	B	complete
3rd	3	C	3rd	3	C	complete
			Ack has been received from Relay2, no pending acks.			
			Relay1 to Relay2 Originators OutboundMessageList			
			sequence	session	message	
			-	-	-	
			Ack has been received from Relay3, no pending acks.			
			Relay1 to Relay3 Originators OutboundMessageList			
			sequence	session	message	
			-	-	-	

Relay1 sends Noop (MessageCount=3) to Device1 and removes 3 entries from InboundMessagelist. Device1 receives Noop (MessageCount=3) from Relay1:

Client Message Lists			Relay1 Message Lists			
Device1 to Relay1 Originators OutboundMessageList			Device1 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	state
-	-	-	-	-	-	-
All sequences acknowledged.			Ack has been received from Relay2, no pending acks.			
			Relay1 to Relay2 Originators OutboundMessageList			
			sequence	session	message	
			-	-	-	
			Ack has been received from Relay3, no pending acks.			
			Relay1 to Relay3 Originators OutboundMessageList			
			sequence	session	message	
			-	-	-	
			All sequences acknowledged.			

When viewed as a protocol exchange sequence, this exchange would be as in the following diagram:

Ack Aggregation

-Consider MS(n) as notation for a full message sequence on session n.

-Assume Device1 has 3 outbound sessions to Relay1 as:
session 1 single-hop fanout to Relay2
session 2 direct enqueue to Relay1
session 3 single-hop fanout to Relay3

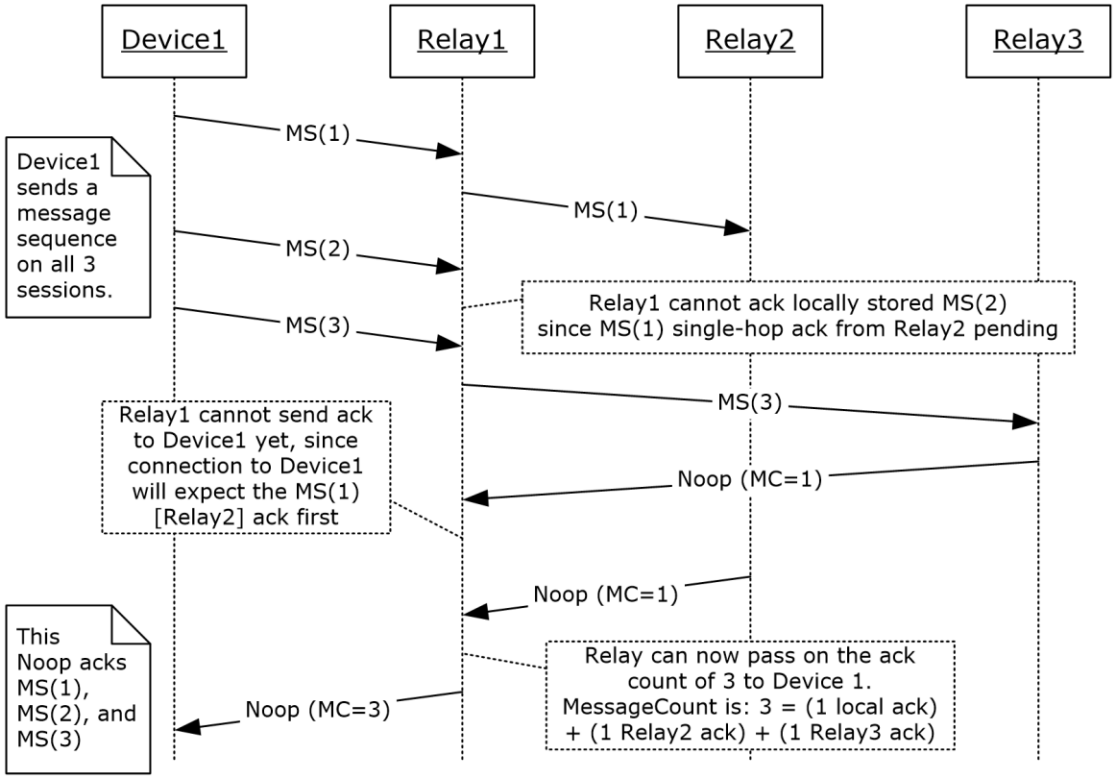


Figure 25: Single-hop fanout message acknowledgments

4.3.4 Acknowledgments for Multi-Client Single-Hop Fanout

Consider the scenario from section 4.3.3, with an additional client connected to Relay1, where that client is also sending data to a resource handler resident on Relay2. In this situation, Relay1 has two independent fanout **sessions** to Relay2.

To minimize network transport layer **connections (2)**, these fanout sessions between relays would normally be independent sessions on a single SSTP connection (1) between Relay1 and Relay2. This relay to relay SSTP connection (1) is also subject to the same rules regarding acknowledgment MessageCount ordering and accumulation. Such single-hop ack message counts can affect the ordering of eventual acks provided to the originating clients. It is important to note that because of connection (1) sharing by the relay amongst multiple single-hop fanout sessions, receipt of a MessageCount ack from a single-hop connection (1) can result in multiple distinct MessageCount acks being delivered to different client SSTP connections (1).

In this example:

- Device1 session 1 addresses a resource handler resident on Relay2. The associated single-hop session from Relay1 to Relay2 is session 1 on the Relay1-Relay2 SSTP connection (1).

- Device1 session 2 addresses a resource handler resident on Relay1
- Device1 session 3 addresses a resource handler resident on Relay3. The associated single-hop session from Relay1 to Relay3 is session 1 on the Relay1-Relay3 SSTP connection (1).
- Device2 session 1 addresses a resource handler resident on Relay2. The associated single-hop session from Relay1 to Relay2 is session 2 on the Relay1-Relay2 SSTP connection (1).

The following tables illustrate a sample of the acknowledgment handling for the case where:

- The Device1 application sends one message on each session, in the following order:
- send A on session 1 (single hop to Relay2)
- send B on session 2 (handled locally on Relay1)
- send C on session 3 (single hop to Relay3)
- The Device2 application sends one message on its session, as:
- send D on session 1 (single hop to Relay2)

Device1 sends message A on session 1. Message A addresses a resource handler resident on Relay2. Device1 sends Message A to Relay1. Relay1 receives Message A and forwards it to Relay2 using outbound fanout session 1 on the Relay1-Relay2 SSTP connection (1). Device2 has not yet sent anything:

Client Message Lists			**Relay1 Message Lists**			
Device1 to Relay1Originators OutboundMessageList			Device1 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	State
1st	1	A	1st	1	A	Processing
Device2 to Relay1Originators OutboundMessageList			Device2 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	State
-	-	-	-	-	-	-
			Outbound fanout forwards message A to Relay2.			
			Relay1 to Relay2 Originators OutboundMessageList			

Client Message Lists	Relay1 Message Lists		
	sequence	session	message
	1st	1	A
	Relay1 to Relay3 Originators OutboundMessageList		
	sequence	session	Message
	-	-	-

Device2 sends message D on session 1. Message D addresses a resource handler resident on Relay2. Device2 sends Message D to Relay1. Relay1 receives Message D and forwards it to Relay2 using outbound fanout session 2 on the Relay1-Relay2 SSTP connection (1). Device1 ack for message A is pending on session 1:

Client Message Lists			Relay1 Message Lists			
Device1 to Relay1Originators OutboundMessageList			Device1 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	State
1st	1	A	1st	1	A	Processing
Device2 to Relay1Originators OutboundMessageList			Device2 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	State
1st	1	D	1st	1	D	Processing
			Outbound fanout forwards message D to Relay2.			
			Relay1 to Relay2 Originators OutboundMessageList			
			sequence	session	message	
			1st	1	A	
			2nd	2	D	

Client Message Lists	Relay1 Message Lists		
	Relay1 to Relay3 Originators OutboundMessageList		
	sequence	session	Message
	-	-	-

After sending message A, Device1 sends message B on session 2. Message B addresses a resource handler resident on Relay1. Relay1 handles this message locally. Device1 ack for message A is pending on session 1:

Client Message Lists			Relay1 Message Lists			
Device1 to Relay1 Originators OutboundMessageList			Device1 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	State
1st	1	A	1st	1	A	Processing
2nd	2	B	2nd	2	B	Complete
Device2 to Relay1 Originators OutboundMessageList			Device2 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	State
1st	1	D	1st	1	D	Processing
			Relay1 to Relay2 Originators OutboundMessageList			
			sequence	session	message	
			1st	1	A	
			2nd	2	D	
			Relay1 to Relay3 Originators OutboundMessageList			
			sequence	session	message	
			-	-	-	

After sending messages A and B, Device1 sends message C on session 3. Message C addresses a resource handler resident on Relay3. Device1 sends Message C to Relay1. Relay1 receives Message C and forwards it to Relay3 using outbound fanout session 1 on the Relay1-Relay3 SSTP connection (1). Device1 acks for messages A and B and Device2 ack for message D are pending:

Client Message Lists			**Relay1 Message Lists**			
Device1 to Relay1 Originators OutboundMessageList			Device1 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	state
1st	1	A	1st	1	A	processing
2nd	2	B	2nd	2	B	complete
3rd	3	C	3rd	3	C	processing
Device2 to Relay1 Originators OutboundMessageList			Device2 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	state
1st	1	D	1st	1	D	processing
			Relay1 to Relay2 Originators OutboundMessageList			
			sequence	session	message	
			1st	1	A	
			2nd	2	D	
			Relay1 to Relay3 Originators OutboundMessageList			
			sequence	session	message	
			1st	2	C	

Device1 acks for messages A, B, and C are still pending from Relay1. Device2 ack for message D is still pending from Relay1. Relay1 receives Noop (MessageCount=1) from Relay3, as ack for message C:

Client Message Lists			Relay1 Message Lists			
Device1 to Relay1 Originators OutboundMessageList			Device1 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	state
1st	1	A	1st	1	A	processing
2nd	2	B	2nd	2	B	complete
3rd	3	C	3rd	3	C	complete
Device2 to Relay1 Originators OutboundMessageList			Device2 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	state
1st	1	D	1st	1	D	processing
			Relay1 to Relay2 Originators OutboundMessageList			
			sequence	session	message	
			1st	1	A	
			2nd	2	D	
			Relay1 to Relay3 Originators OutboundMessageList			
			sequence	session	message	
			-	-	-	

Device1 acks for messages A, B, and C are still pending from Relay1. Device2 ack for message D is still pending from Relay1. Relay1 receives Noop (MessageCount=2) from Relay2, as ack for messages A and D:

Client Message Lists			Relay1 Message Lists			
Device1 to Relay1 Originators OutboundMessageList			Device1 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message	state

Client Message Lists			Relay1 Message Lists			
					sequence	
1st	1	A	1st	1	A	complete
2nd	2	B	2nd	2	B	complete
3rd	3	C	3rd	3	C	complete
Device2 to Relay1 Originators OutboundMessageList			Device2 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	state
1st	1	D	1st	1	D	complete
			Relay1 to Relay2 Originators OutboundMessageList			
			sequence	session	message	
			-	-	-	
			Relay1 to Relay3 Originators OutboundMessageList			
			sequence	session	message	
			-	-	-	

Relay1 sends Noop (MessageCount=3) to Device1 and removes 3 entries from InboundMessagelist. Relay1 sends Noop (MessageCount=1) to Device2 and removes 1 entry from InboundMessagelist. Device1 receives Noop (MessageCount=3) and Device2 receives Noop (MessageCount=1) from Relay1:

Client Message Lists			Relay1 Message Lists			
Device1 to Relay1 Originators OutboundMessageList			Device1 to Relay1 Receivers InboundMessageList			
sequence	session	message	sequence	session	message sequence	state
-	-	-	-	-	-	-
Device2 to Relay1 Originators			Device2 to Relay1 Receivers			

<table>
<tr><th colspan="3">Client Message Lists</th><th colspan="4">Relay1 Message Lists</th></tr>
<tr><td colspan="3">OutboundMessageList</td><td colspan="4">InboundMessageList</td></tr>
<tr><td>sequence</td><td>session</td><td>message</td><td>sequence</td><td>session</td><td>message sequence</td><td>state</td></tr>
<tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td colspan="3">All sequences acknowledged.</td><td colspan="4">Relay1 to Relay2 Originators OutboundMessageList</td></tr>
<tr><td colspan="3"></td><td>sequence</td><td>session</td><td colspan="2">message</td></tr>
<tr><td colspan="3"></td><td>-</td><td>-</td><td colspan="2">-</td></tr>
<tr><td colspan="3"></td><td colspan="4">Relay1 to Relay3 Originators OutboundMessageList</td></tr>
<tr><td colspan="3"></td><td>sequence</td><td>session</td><td colspan="2">message</td></tr>
<tr><td colspan="3"></td><td>-</td><td>-</td><td colspan="2">-</td></tr>
<tr><td colspan="3"></td><td colspan="4">All sequences acknowledged.</td></tr>
</table>

When viewed as a protocol exchange sequence, this exchange would be as in the following diagram:

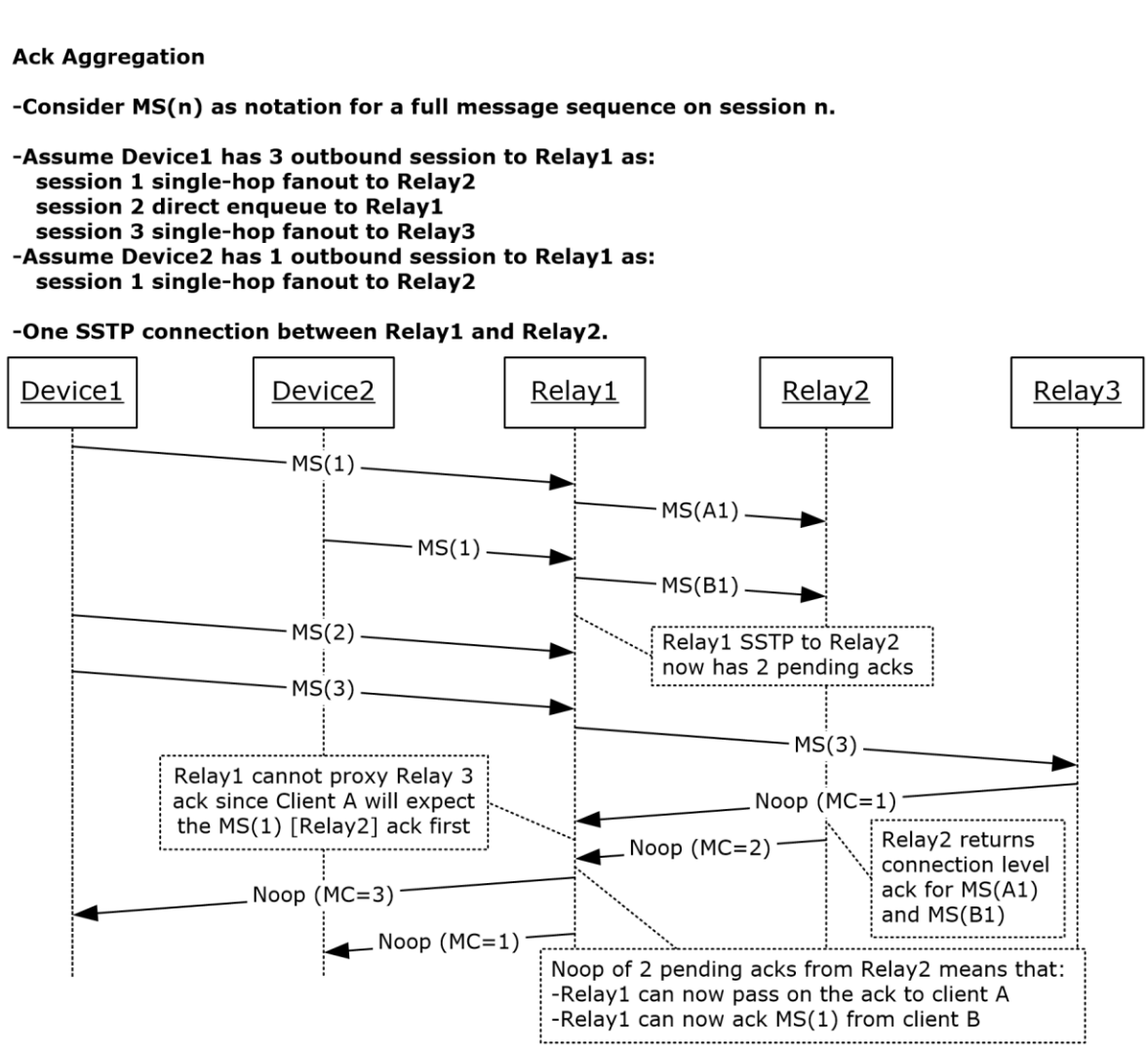


Figure 26: Multiple client single-hop fanout message sequence

4.4 Flow Control Example

Session flow control is managed by the receiving end of a session, by transmission of the OpenResponse message with ResponseIds StopSending and StartSending. Session flow control is triggered by changes in ResourceHandlerAvailability which can cause the session state to toggle between 'ready' and 'blocked'.

Consider again the example from section 4.3.1, where a device sends the message sequences A1, A2 on session 1, and message sequences B1, B2 on session 2. If the resource handler for session 1 causes that session to be flow controlled during the transmission, the order in which the message sequences are sent over the SSTP **connection (1)**, as well as the acknowledgment messages, could be different than was seen previously in the Figure titled Multiple Session Acknowledgments - Message Sequences in section 4.3.1. The method for managing MessageCount acknowledgments across all extant sessions remains unchanged.

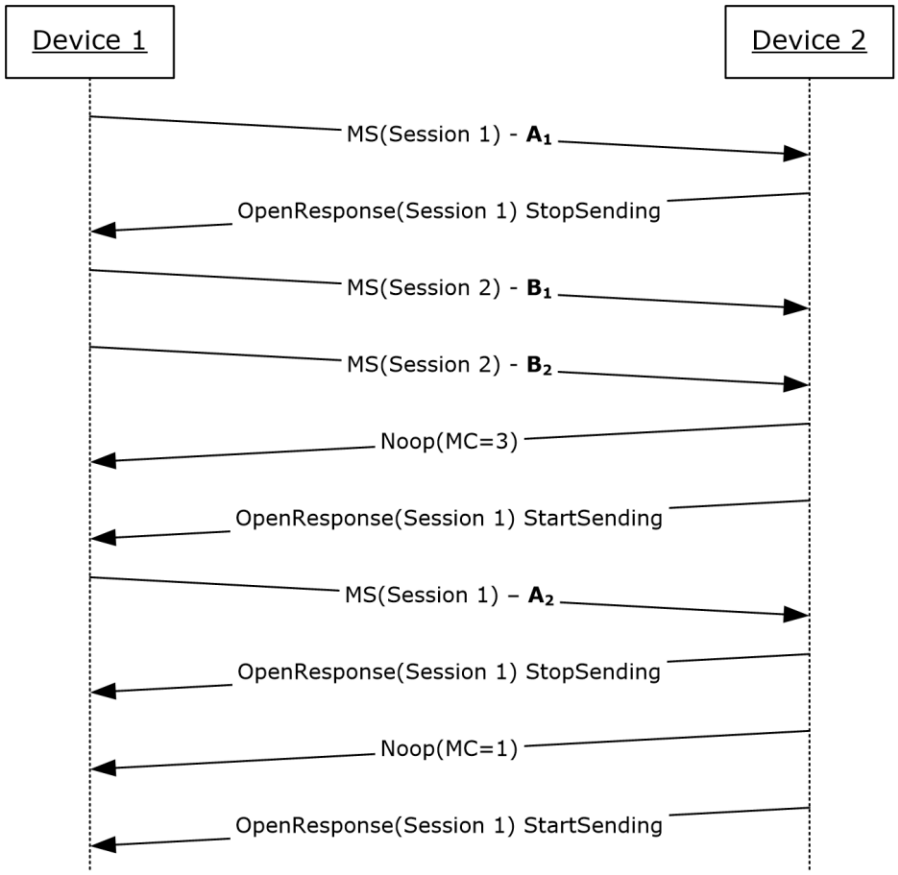


Figure 27: Flow control

4.4.1 Receiving Message Sequences while Blocked

Because of the latency of the underlying transport, situations can arise where the **session** originator has already transmitted message sequence MS(1)-B before it receives the session related flow control indication. The **connection (1)** level acknowledgment message counts are preserved even when such unexpected message sequences are received while the receiving resource handler is in a blocked state.

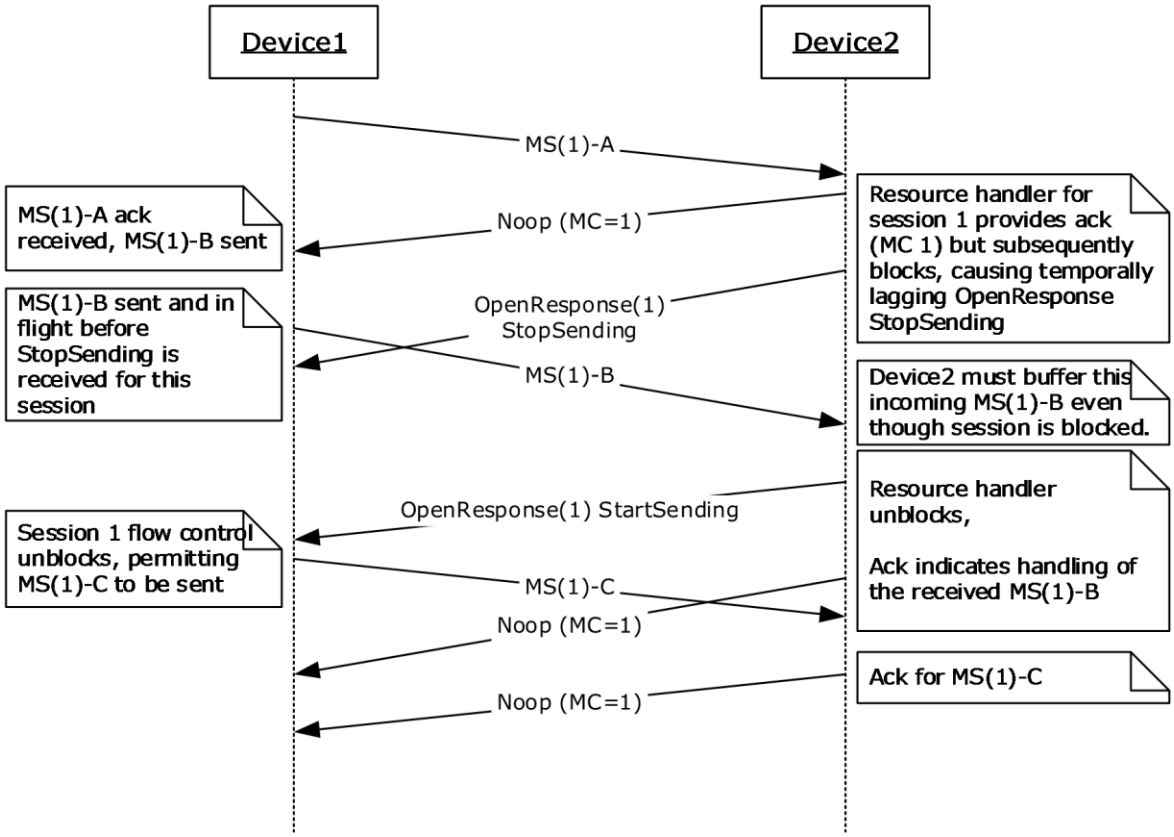


Figure 28: Message sequences in blocked state

5 Security

5.1 Security Considerations for Implementers

SSTP uses cryptographic algorithms for optional initial authentication of devices and users. SSTP provides no protection against connection take-over, eavesdropping, or message modification, insertion, or deletion. Nor does it provide built-in message encryption. These security features can be handled by higher-layers. For the purposes of specifying the SSTP protocol and message formats, the authentication and security registration payloads are assumed to be opaque binary fields.

SSTP supports application-level authentication of device users via the AuthenticationToken field in the Connect and ConnectResponse commands. The SSTP Security protocol Attach/Register command [MS-GRVSSTPS], hosted by an intermediary relay server, can then process the defined token to provide basic authentication of users connecting to a relay server.

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include updates to those products.

- Microsoft Office 2010 suites
- Microsoft Office Groove 2007
- Microsoft Office Groove Server 2007
- Microsoft Groove Server 2010
- Microsoft SharePoint Workspace 2010

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms "SHOULD" or "SHOULD NOT" implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term "MAY" implies that the product does not follow the prescription.

<1> Section 2.2.1.1: The Office Groove 2007 and SharePoint Workspace 2010 applications set this to space delimited set of tokens indicating "<*vendor*> <*product*> <*version*> <*build*>". Examples of this string for Office Groove 2007 and SharePoint Workspace 2010, respectively, are as follows:

- Groove Client 4.2 2623
- Groove Client 14.0 4006

The Office Groove Server 2007 Relay and Groove Server 2010 Relay applications set this to space delimited set of tokens indicating "<*vendor*> <*product*> <*version*> <*build*>". Examples of this string for Office Groove Server 2007 and Groove Server 2010, respectively, are as follows:

- Groove Relay 12.0 1336
- Groove Relay 14.0 4006

<2> Section 2.2.1.1: The Office Groove 2007 application, Office Groove Server 2007 Relay application, SharePoint Workspace 2010 application, and Groove Server 2010 Relay application all set this field to an empty string: 0x00.

<3> Section 2.2.5.1: [MS-GRVWDPP] specifies that a WAN DPP device presence session specifies a ResourceURL of "grooveWANDPP".

The Office Groove 2007 application and Microsoft SharePoint Workspace 2010 application use this ResourceURL value.

The Office Groove Server 2007 Relay Application and Microsoft Groove Server 2010 Relay application use this ResourceURL value.

<4> Section 2.2.5.1: [MS-GRVWDPP] specifies that the **Open** command **IdentityURL** field is set to 0x00 for a WAN DPP device presence session.

The Office Groove 2007 application and SharePoint Workspace 2010 application use this **IdentityURL** field value.

The Office Groove Server 2007 Relay Application and Groove Server 2010 Relay application use this **IdentityURL** field value.

<5> Section 2.2.5.1: If set to 1, this value indicates this is a session for transmitting identity targeted messages.

The Office Groove 2007 application and SharePoint Workspace 2010 application do not set this bit.

The Office Groove Server 2007 Relay application and Groove Server 2010 Relay application set this bit only for relay server originated sessions with an empty DeviceURL.

<6> Section 2.2.6.1: If set to 1, this value indicates this is a session for transmitting identity targeted messages.

The Office Groove 2007 application and SharePoint Workspace 2010 applications do not set this bit.

The Office Groove Server 2007 Relay application and Groove Server 2010 Relay application set this bit only for relay server originated sessions with an empty **DeviceURL**.

<7> Section 2.2.6.1: The Office Groove 2007 and SharePoint Workspace 2010 application sends a **FanoutOpen** command with an empty **DeviceURL** to indicate an identity-targeted session.

<8> Section 2.2.6.1: The Office Groove Server 2007 Relay application and Groove Server 2010 Relay application require that the **RelayURL** scheme [RFC3986] is of the form "grooveDNS://".

<9> Section 2.2.8.1: Office Groove 2007 and SharePoint Workspace 2010 applications can detect redundant failed endpoints, in case the same index is sent multiple times in multiple SessionStatus commands.

<10> Section 3.1.1.3: Identity URLs start with the "grooveIdentity://" prefix and the length of the URL after the prefix does not exceed 80 characters for the Office Groove 2007 application and SharePoint Workspace 2010 application and for the Office Groove Server 2007 Relay Application and the Groove Server 2010 Relay application.

<11> Section 3.1.2: Although there are no additional mandatory protocol-defined timeouts applicable to connection establishment or session open protocol exchanges, an application can implement specific timers to compensate for unacceptable response times.

Transport Connection Timer

A timer used to detect when a ConnectReponse is not received within 3 minutes during a connection of the underlying transport layer for a requested SSTP connection. If an application implements this timeout, the application can retry the connection attempt to correct the condition. The higher-layer application determines the duration for this timer.

The value of this timer MUST exceed the value of any inherent connection timer used by the underlying transport.

Upon expiry of this timer, the application retries the connection.

The Office Groove 2007 client application and SharePoint Workspace 2010 client application implement this as a 10 minute timer.

The Groove Server 2010 relay server implements this as a 3 minute timer. This timer is in effect for relay-to-relay single-hop fanout connections.

Connection Establishment Timer

A timer used to detect excessive delays in initial SSTP connection, following connection of the underlying transport layer and the transition to the SSTP established state. This timer is in effect only for the time span between a Connect command and receipt of the ConnectResponse command. If an

application implements this timeout, the only possible corrective action is to terminate the connection. The duration for this timer is application-dependent.

Upon expiry of this timer, the application closes the connection with ReasonId ResponseTimeout and closes the transport layer connection.

Authentication Timer

A timer used to detect excessive delays in initial connection authentication and registration. This timer can be restarted upon any change in SSTP Security protocol [MS-GRVSSTPS] state. If an application implements this timeout, the only possible corrective action is to terminate the connection. The duration for this timer is application-dependent.

Upon expiry of this timer, the application closes the connection with ReasonId ResponseTimeout and closes the transport layer connection.

Session Establishment Timer

A timer used to detect excessive delays in session creation and the transition between session states. This timer is started by the session originator upon transmission of the session Open or FanoutOpen command, and cancelled upon receipt of the corresponding OpenResponse command.

Upon expiry of this timer, the application issues a session Close command. The Groove Server 2010 relay server implements this as a 3 minute timer. This timer is in effect for relay-to-relay single-hop fanout connections.

Outbound Idle Timer

A timer used to detect periods of inactivity on an SSTP connection. This timer can be restarted upon transmission of any outbound SSTP command.

Upon expiry of this timer, the application can send a Noop command to the remote device, with a MessageCount field of zero. This is useful for detecting broken transport connections when SSTP is hosted on a transport that cannot detect loss of network connectivity.

The Office Groove 2007 client application, SharePoint Workspace 2010 client application, and Office Groove Server 2007 relay server implement this as a 10 minute timer. The Groove Server 2010 relay server implements this as a 4 minute timer on relay-to-relay single-hop fanout connections.

Inbound Idle Timer

A timer used to detect periods of inactivity on an SSTP connection. This timer can be restarted upon detection of any inbound SSTP command.

Upon expiry of this timer, the application can send a Noop command to the remote device, with a MessageCount field of zero. This is useful for detecting broken transport connections when SSTP is hosted on a transport that cannot detect loss of network connectivity.

The Office Groove 2007 and SharePoint Workspace 2010 applications implement this as a 10 minute timer.

<12> Section 3.1.2.1: The Office Groove 2007 and SharePoint Workspace 2010 applications implement this as a 5 second timer.

<13> Section 3.1.4.3.2: The Office Groove 2007, Office Groove Server 2007 Relay, SharePoint Workspace 2010, and Groove Server 2010 Relay support a strict-naming convention by default. When the strict-naming convention is in effect, the following session addressing entry schemes are enforced:

- The **IdentityURL** scheme is of the form "grooveIdentity://".
- If the **DeviceURL** field is non-empty, the **DeviceURL** scheme is of the form "dpp://".

<14> Section 3.1.4.3.3: The Office Groove 2007, Office Groove Server 2007 Relay, SharePoint Workspace 2010, and Groove Server 2010 Relay support a strict-naming convention by default. When the strict-naming convention is in effect, the following fanout session addressing schemes are enforced:

- Each **FanoutEntry IdentityURL** scheme is of the form "grooveIdentity://".
- Each **FanoutEntry DeviceURL** scheme is of the form "dpp://".
- For each **FanoutEntry RelayURL** field that is non-empty, the **RelayURL** scheme is of the form "grooveDNS://".

<15> Section 3.1.4.7: In the Office Groove 2007 and SharePoint Workspace 2010 products, if any session has a Message command to be sent as specified in section 3.1.4.6 at the time that a resource handler completion notification is received, the message sequence acknowledgment count as determined by the process specified in section 3.1.4.2.1 will be set in the MessageCount field value of that Message command.

<16> Section 3.2.5.8: The behavior of each version in this condition is as follows:

- SSTP 1.6 SharePoint Workspace 2010 clients will always send a Close.
- SSTP 1.5 Office Groove 2007 clients will never send a Close.
- SSTP 1.5 Office Groove Server 2007 relays will always send a Close following generation of a SessionStatus, which results in removal of all OutboundFanoutAddressingList members.
- SSTP 1.6 Groove Server 2010 relays will always defer the sending of a Close in this case by 60 seconds; it is expected that the receiving client will respond to such a SessionStatus with a Close prior to timeout of that session-close timer.

<17> Section 3.3.2.1: The Office Groove Server 2007 and Groove Server 2010 Relay Servers implement this as 5 minute timer.

<18> Section 3.3.4.1.2: If target destinations on a remote relay server fail during single-hop fanout, the remote relay server sends multiple SessionStatus commands, specifying the addressing entry of each failed target, to the originating home relay. SSTP 1.6 Groove Server 2010 relays can aggregate these SessionStatus commands and send a single SessionStatus command containing a list of indices of the failed targets back to the session originator.

<19> Section 3.3.5.6: The Groove Server 2010 Relay Application sends an OpenResponse command as specified. The Office Groove Server 2007 Relay Application sends a SessionStatus command and Close command.

<20> Section 4.2.5: This example uses SSTP 1.5 as used by the Office Groove 2007 and SharePoint Workspace 2010 applications and the Office Groove Server 2007 and Groove Server 2010 Relay applications. An SSTP 1.6 example would be similar except that the FanoutEntries inside the FanoutOpen commands would include additional FailoverDeviceURLs field.

7 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as Major, Minor, or None.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements.
- A document revision that captures changes to protocol functionality.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **None** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the relevant technical content is identical to the last released version.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Description	Revision class
3.1.1.3 SSTP Sessions	Updated description for product behavior note of IdentityURL.	Minor

8 Index

G

H

I

L

M

N

O

P

R

S

T

V