Internet-Draft March, 1995
ST Working Group
L. Delgrossi and L. Berger (Ed.)
File: draft-ietf-st2-spec-04.ps Expires: August 1995

Internet STream Protocol Version 2 (ST2)

- Protocol Specification -
Version ST2+

Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its Areas, and its Working Groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months. Internet-Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress".

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

Abstract

This memo contains a revised specification of the Internet STream Protocol Version 2 (ST2). ST2 is an experimental resource reservation protocol intended to provide end-to-end real-time guarantees over an internet. It allows applications to build multi-destination simplex data streams with a desired quality of service. The revised version of ST2 specified in this memo is called ST2+.

This specification is a product of the STream Protocol Working Group of the Internet Engineering Task Force.

1 Introduction

1.1 What is ST2?

The Internet Stream Protocol, Version 2 (ST2) is an experimental connection-oriented internetworking protocol that operates at the same layer as connectionless IP. It has been developed to support the efficient delivery of data streams to single or multiple destinations in applications that require guaranteed quality of service. ST2 is part of the IP protocol family and serves as an adjunct to, not a replacement for, IP. The main application areas of the protocol are the real-time transport of multimedia data, e.g. digital audio and video packet streams, and distributed simulation/gaming, across internets.

ST2 can be used to reserve bandwidth for real-time streams across network routes. This reservation, together with appropriate network access and packet scheduling mechanisms in all nodes running the protocol, guarantees a well-defined Quality of Service (QoS) to ST2 applications. It ensures that real-time packets are delivered within their deadlines, that is, at the time where they need to be presented. This facilitates a smooth delivery of data that is essential for time-critical applications, but can typically not be provided by best-effort IP communication.

Just like IP, ST2 actually consists of two protocols: ST for the data transport and SCMP, the Stream Control Message Protocol, for all control functions. ST is simple and contains only a single PDU format that is designed for fast and efficient data forwarding in order to achieve low communication delays. SCMP, however, is more complex than IP's ICMP. As with ICMP and IP, SCMP packets are transferred within ST packets as shown in Figure 1.

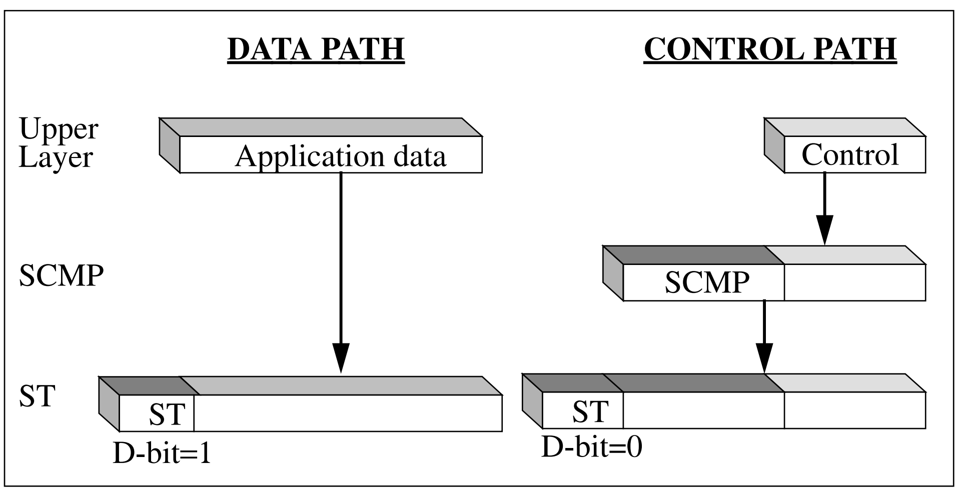

Figure 1: ST2 Data and Control Path

1.2 ST2 and IP

ST2 is designed to coexist with IP on each node. A typical distributed multimedia application would use both protocols: IP for the transfer of traditional data and control information, and ST2 for the transfer of real-time data. Whereas IP typically will be accessed from TCP or UDP, ST2 will be accessed via new end-to-end real-time protocols. The position of ST2 with respect to the other protocols of the Internet family is represented in Figure 2.

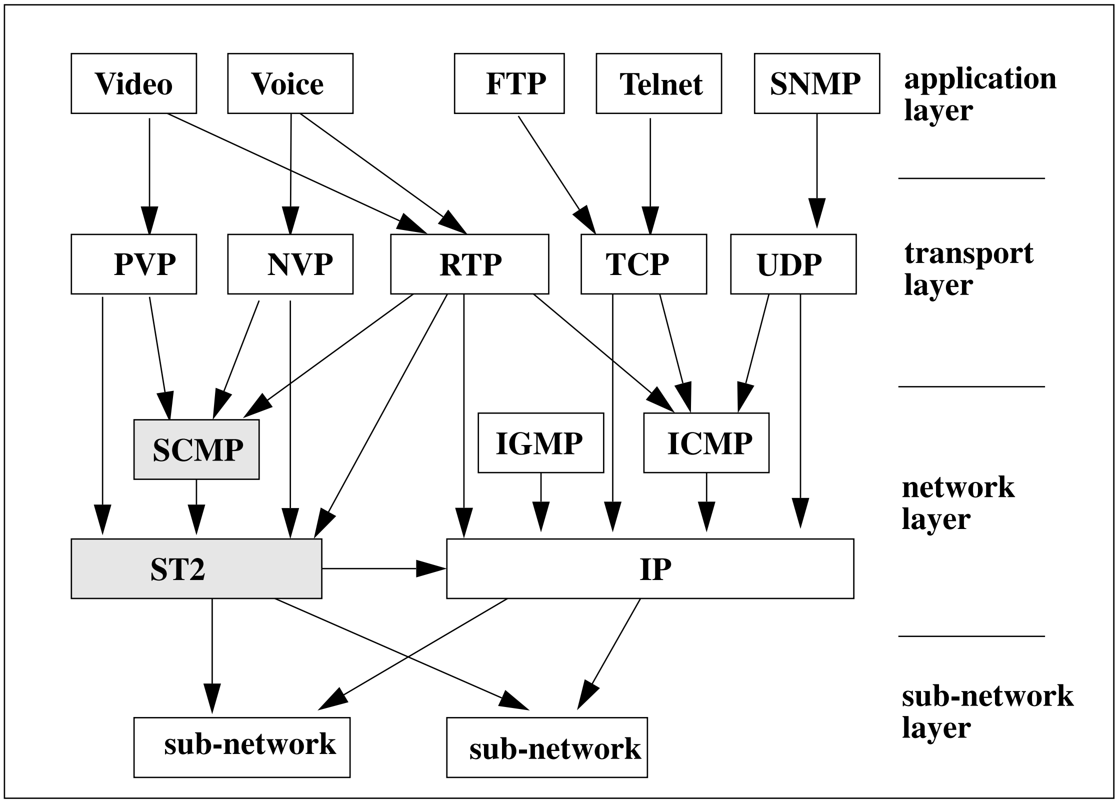

Figure 2: Internet Protocols Relationships

Both ST2 and IP apply the same addressing schemes to identify different hosts. ST2 and IP packets differ in the first four bits, which contain the internetwork protocol version number: number 5 is reserved for ST2 (IP itself has version number 4). As a network layer protocol, like IP, ST2 operates independently of its underlying subnets. Existing implementations use ARP for address resolution, and use the same Layer 2 SAPs as IP.

As a special function, ST2 messages can be encapsulated in IP packets. This is represented in Figure 2 as a link between ST2 and IP. This link allows ST2 messages to pass through routers which do not run ST2. Resource management is typically not available for these IP route segments. IP encapsulation is, therefore, suggested only for portions of the network which do not constitute a system bottleneck.

In Figure 2, the RTP protocol is shown as an example of transport layer on top of ST2. Others include the Packet Video Protocol (PVP) [Cole81], the Network Voice Protocol (NVP) [Cohe81], and others such as the Heidelberg Transport Protocol (HeiTP) [DHHS92].

1.3 Protocol History

The first version of ST was published in the late 1970's and was used throughout the 1980's for experimental transmission of voice, video, and distributed simulation. The experience gained in these applications led to the development of the revised protocol version ST2. The revision extends the original protocol to make it more complete and more applicable to emerging multimedia environments. The specification of this protocol version is contained in Internet RFC 1190 which was published in October 1990 [RFC1190].

With more and more developments of commercial distributed multimedia applications underway and with a growing dissatisfaction at the transmission quality for audio and video over IP in the MBONE, interest in ST2 has grown over the last years. Companies have products available incorporating the protocol. The BERKOM MMTS project of the German PTT [DeAl92] uses ST2 as its core protocol for the provision of multimedia teleservices such as conferencing and mailing. In addition, implementations of ST2 for Digital Equipment, IBM, NeXT, Macintosh, PC, Silicon Graphics, and Sun platforms are available.

In 1993, the IETF started a new working group on ST2 as part of ongoing efforts to develop protocols that address resource reservation issues. The group's mission was to clean up the existing protocol specification to ensure better interoperability between the existing and emerging implementations. It was also the goal to produce an updated experimental protocol specification that reflected the experiences gained with the existing ST2 implementations and applications. Which led to the specification of the ST2+ protocol contained in this document.

1.3.1 RFC1190 ST and ST2+ Major Differences

The protocol changes from RFC1190 were motivated by protocol simplification and clarification, and codification of extensions in existing implementations. This section provides a list of major differences, and is probably of interest only to those who have knowledge of RFC1190. The major differences between the versions are:

- Elimination of "Hop IDentifiers" or HIDs. HIDs added much complexity to the protocol and was found to be a major impediment to interoperability. HIDs have been replaced by globally unique identifiers called "Stream IDentifiers" or SIDs.
- Elimination of a number of stream options. A number of options were found to not be used by any implementation, or were thought to add more complexity than value. These options were removed. Removed options include: point-to-point, full-duplex, reverse charge, and source route.
- Elimination of the concept of "subset" implementations. RFC1190 permitted subset implementations, to allow for easy implementation and experimentation. This led to interoperability problems. Agents implementing the protocol specified in this document, MUST implement the full protocol. A number of the protocol functions are best-effort. It is expected that some implementations will make more effort than others in satisfying particular protocol requests.
- Clarification of the capability of targets to request to join a steam. RFC1190 can be interpreted to support target requests, but most implementors did not understand this and did not add support for this capability. The lack of this capability was found to be a significant limitation in the ability to scale the number of participants in a single ST stream. This clarification is based on work done by IBM Heidelberg.
- Separation of functions between ST and supporting modules. An effort was made to improve the separation of functions provided by ST and those provided by other modules. This is reflected in reorganization of some text and some PDU formats. ST was also made FlowSpec independent, although it does define a FlowSpec for testing and interoperability purposes.

- General reorganization and re-write of the specification. This document has been organized with the goal of improved readability and clarity. Some sections have been added, and an effort was made to improve the introduction of concepts.

1.4 Supporting Modules for ST2

ST2 is one piece of a larger mosaic. This section presents the overall communication architecture and clarifies the role of ST2 with respect to its supporting modules.

ST2 proposes a two-step communication model. In the first step, the real-time channels for the subsequent data transfer are built. This is called **stream setup**. It includes selecting the routes to the destinations and reserving the correspondent resources. In the second step, the data is transmitted over the previously established streams. This is called **data transfer**. While stream setup does not have to be completed in real-time, data transfer has stringent real-time requirements. The architecture used to describe the ST2 communication model includes:

- a **data transfer protocol** for the transmission of real-time data over the established streams,
- a **setup protocol** to establish real-time streams based on the flow specification,
- a **flow specification** to express user real-time requirements,
- a **routing function** to select routes in the Internet,
- a **local resource manager** to appropriately handle resources involved in the communication.

This document defines a data protocol (ST), a setup protocol (SCMP), and a flow specification (ST2+ FlowSpec). It does not define a routing function and a local resource manager. However, ST2 assumes their existence.

Alternative architectures are possible, see [RFC1633] for an example alternative architecture that could be used when implementing ST2.

1.4.1 Data Transfer Protocol

The data transfer protocol defines the format of the data packets belonging to the stream. Data packets are delivered to the targets along the stream paths previously established by the setup protocol. Data packets are delivered with the quality of service associated with the stream.

Data packets contain a globally unique stream identifier that indicates which stream they belong to. The stream identifier is also known by the setup protocol, which uses it during stream establishment. The data transfer protocol for ST2, known simply as ST, is completely defined by this document.

1.4.2 Setup Protocol

The setup protocol is responsible for establishing, maintaining, and releasing real-time streams. It relies on the routing function to select the paths from the source to the destinations. At each host/router on these paths, it presents the flow specification associated with the stream to the local resource manager. This causes the resource managers to reserve appropriate resources for the stream. The setup protocol for ST2 is called Stream Control Message Protocol, or SCMP, and is completely defined by this document.

1.4.3 Flow Specification

The flow specification is a data structure including the ST2 applications' QoS requirements. At each host/router, it is used by the local resource manager to appropriately handle resources so that such requirements are met. Distributing the flow specification to all resource managers along the communication paths is the task of the setup protocol. However, the contents of the flow specification are transparent to the setup protocol, which simply carries the flow specification. Any operations on the flow specification, including updating internal fields and comparing flow specifications are performed by the resource managers.

This document defines a specific flow specification format that allows for interoperability among existing ST2 implementations. Implementations may support more than one flow specification format and the means are provided to add new formats as they are defined in the future. However, the flow specification format has to be consistent throughout the stream, i.e. it is not possible to use different flow specification formats for different parts of the same stream.

1.4.4 Routing Function

The routing function is an external unicast route generation capability. It provides the setup protocol with the path to reach each of the desired destinations. The routing function is called on a hop-by-hop basis and provides next-hop information. Once a route is selected by the routing function, it persists for the whole stream lifetime. The routing function may try to optimize based on the number of targets, the requested resources, or use of local network multicast or bandwidth capabilities. Alternatively, the routing function may even be based on simple connectivity information.

The setup protocol is not necessarily aware of the criteria used by the routing function to select routes. It works with any routing function algorithm. The algorithm adopted is a local matter at each host/router and different hosts/routers may use different algorithms. The interface between setup protocol and routing function is also a local matter and therefore it is not specified by this document.

This version of ST does not support source routing. It does support route recording. It does include provisions that allow identification of ST capable neighbors. Identification of remote ST hosts/routers is not specifically addressed.

1.4.5 Local Resource Manager

At each host/router traversed by a stream, the Local Resource Manager (LRM) is responsible for handling local resources. The LRM knows which resources are on the system and what capacity they can provide. Resources include:

- CPUs on end systems and routers to execute the application and protocol software,
- main memory space for this software (as in all real-time systems, code should be pinned in main memory, as swapping it out would have detrimental effects on system performance),
- buffer space to store the data, e.g., communication packets, passing through the nodes,
- network adapters, and
- transmission networks between the nodes. Networks may be as simple as point-to-point links or as complex as switched networks such as Frame Relay and ATM networks.

During **stream setup** and **modification**, the LRM is presented by the setup protocol with the flow specification associated to the stream. For each resource it handles, the LRM is expected to perform the following functions:

- *Stream Admission Control:* it checks whether, given the flow specification, there are sufficient resources left to handle the new data stream. If the available resources are insufficient, the new data stream must be rejected.
- *QoS Computation:* it calculates the best possible performance the resource can provide for the new data stream under the current traffic conditions, e.g. throughput and delay values are computed.
- *Resource Reservation:* it reserves the resource capacities required to meet the desired QoS.

During **data transfer**, the LRM is responsible for:

- *QoS Enforcement:* it enforces the QoS requirements by appropriate *scheduling* of resource access. For example, data packets from an application with a short guaranteed delay must be served prior to data from an application with a less strict delay bound.

The LRM may also provide the following additional functions:

- *Data Regulation:* to smooth a stream's data traffic, e.g. as with the leaky bucket algorithm.
- *Policing:* to prevent applications exceed their negotiated QoS, e.g. to send data at a higher rate than indicated in the flow specification.
- *Stream Preemption*: to free up resources for other streams with higher priority or importance.

The strategies adopted by the LRMs to handle resources are resource-dependent and may vary at every host/router. However, it is necessary that all LRMs have the same understanding of the flow specification. The interface between setup protocol and LRM is a local matter at every host and therefore it is not specified by this document. An example of LRM is the Heidelberg Resource Administration Technique (HeiRAT) [VoHN93].

It is also assumed that the LRM provides functions to compare flow specifications, i.e. to decide whether a flow specification requires a greater, equal, or smaller amount of resource capacities to be reserved.

1.5 ST2 Basic Concepts

The following sections present at an introductory level some of the fundamental ST2 concepts including streams, data transfer, and flow specification.

1.5.1 Streams

Streams form the core concepts of ST2. They are established between a sending *origin* and one or more receiving *targets* in the form of a routing tree. Streams are *uni-directional* from the origin to the targets. Nodes in the tree represent so-called *ST agents*, entities executing the ST2 protocol; links in the tree are called *hops*. Any node that can sit in the middle of the tree is call an *intermediate* agent, or router. An agent may have any combination of origin, target, or intermediate capabilities.

Figure 3 illustrates a stream from an origin to four targets, where the ST agent on Target 2 also functions as an intermediate agent. Let us use this Target 2/Router node to explain some basic ST2 terminology: the direction of the stream from this node to Target 3 and 4 is called *downstream*, the direction towards the Origin node *upstream*. ST agents that are one hop away from a given node are called *previous-hops* in the upstream, and *next-hops* in the downstream direction.

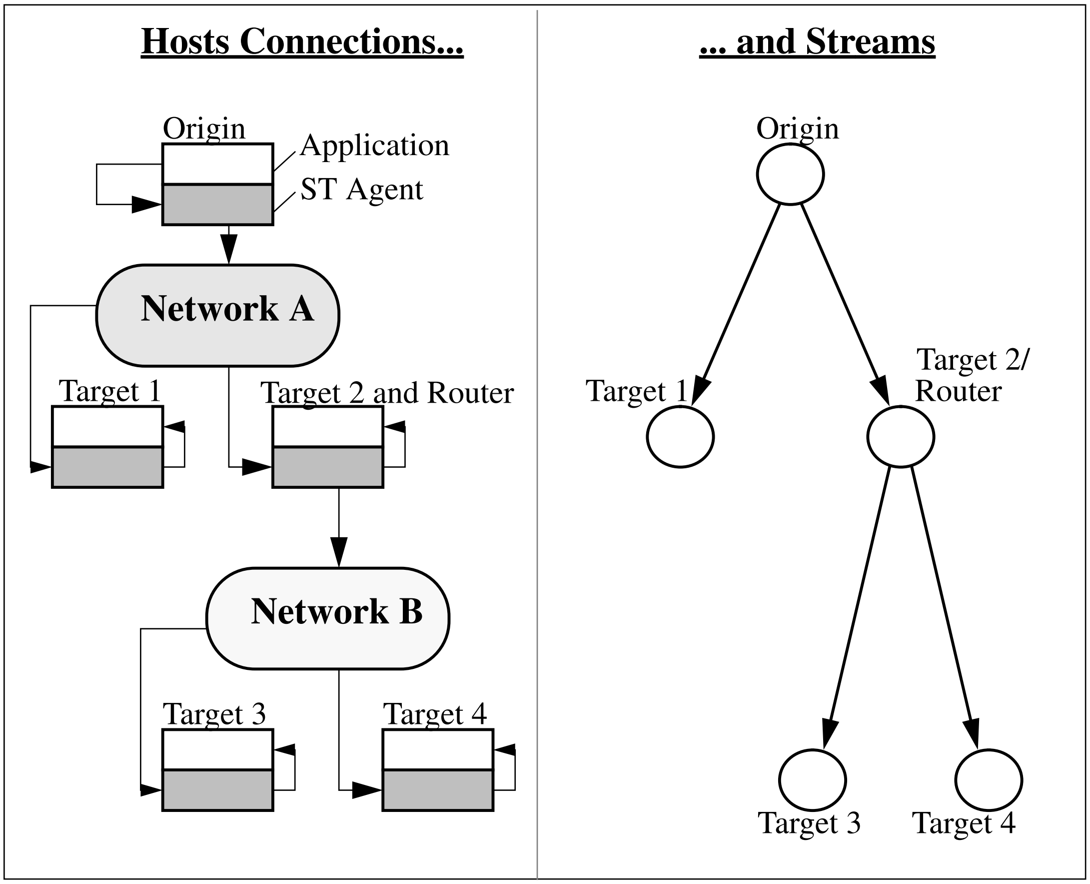

Figure 3: The Stream Concept

Streams are maintained using SCMP messages. Typical SCMP messages are CONNECT and ACCEPT to build a stream, DISCONNECT and REFUSE to close a stream, CHANGE to modify the quality of service associated with a stream, and JOIN to request to be added to a stream.

Each ST agent maintains state information describing the streams flowing through it. It can actively gather and distribute such information. It can recognize failed neighbor ST agents through the use of periodic HELLO message exchanges. It can ask other ST agents about a particular stream via a STATUS message. These ST agents then send back a STATUS-RESPONSE message. NOTIFY messages can be used to inform other ST agents of significant events.

ST2 offers a wealth of functionalities for stream management. Streams can be grouped together to minimize allocated resources or to process them in the same way in case of failures. During audio conferences, for example, only a limited set of participants may talk at once. Using the group mechanism, resources for only a portion of the audio streams of the group need to be reserved. Using the same concept, an entire group of related audio and video streams can be dropped if one of them is preempted.

1.5.2 Data Transmission

Data transfer in ST2 is simplex in the downstream direction. Data transport through streams is very simple. ST2 puts only a small header in front of the user data. The header contains a protocol identification that distinguishes ST2 from IP packets, an ST2 version number, a priority field (specifying a relative importance of streams in cases of conflict), a length counter, a stream identification, and a checksum. These elements form an 12-byte header.

Efficiency is also achieved by avoiding fragmentation and reassembly on all agents. Stream establishment yields a maximum message size for data packets on a stream. This maximum message size is communicated to the upper layers, so that they provide data packets of suitable size to ST2.

Communication with multiple next-hops can be made even more efficient using MAC Layer multicast when it is available. If a subnet supports multicast, a single multicast packet is sufficient to reach all next-hops connected to this subnet. This leads to a significant reduction of the bandwidth requirements of a stream. If multicast is not provided, separate packets need to be sent to each next-hop.

As ST2 relies on reservation, it does not contain error correction mechanisms features for data exchange such as those found in TCP. It is assumed that real-time data, such as digital audio and video, require partially correct delivery only. In many cases, retransmitted packets would arrive too late to meet their real-time delivery requirements. Also, depending on the data encoding and the particular application, a small number of errors in stream data are acceptable. In any case, reliability can be provided by layers on top of ST2 when needed.

1.5.3 Flow Specification

As part of establishing a connection, SCMP handles the negotiation of quality-of-service parameters for a stream. In ST2 terminology, these parameters form a flow specification (FlowSpec) which is associated with the stream. Different versions of FlowSpecs exist, see [RFC1190], [DHHS92] and [RFC1363], and can be distinguished by a version number. Typically, they contain parameters such as average and maximum throughput, end-to-end delay, and delay variance of a stream. SCMP itself only provides the mechanism for relaying the quality-of-service parameters.

Three kinds of entities participate in the quality-of-service negotiation: application entities on the origin and target sites as the service users, ST agents, and local resource managers (LRM). The origin application supplies the initial FlowSpec requesting a particular service quality. Each ST agent which obtains the FlowSpec as part of a connection establishment message, it presents the local resource manager with it. ST2 does not determine how resource managers make reservations and how resources are scheduled according to these reservations; ST2, however, assumes these mechanisms as its basis.

An example of the FlowSpec negotiation procedure is illustrated in Figure 4. Depending on the success of its local reservations, the LRM updates the FlowSpec fields and returns the FlowSpec to the ST agent, which passes it downstream as part of the connection message. Eventually, the FlowSpec is communicated to the application at the target which may base its accept/reject

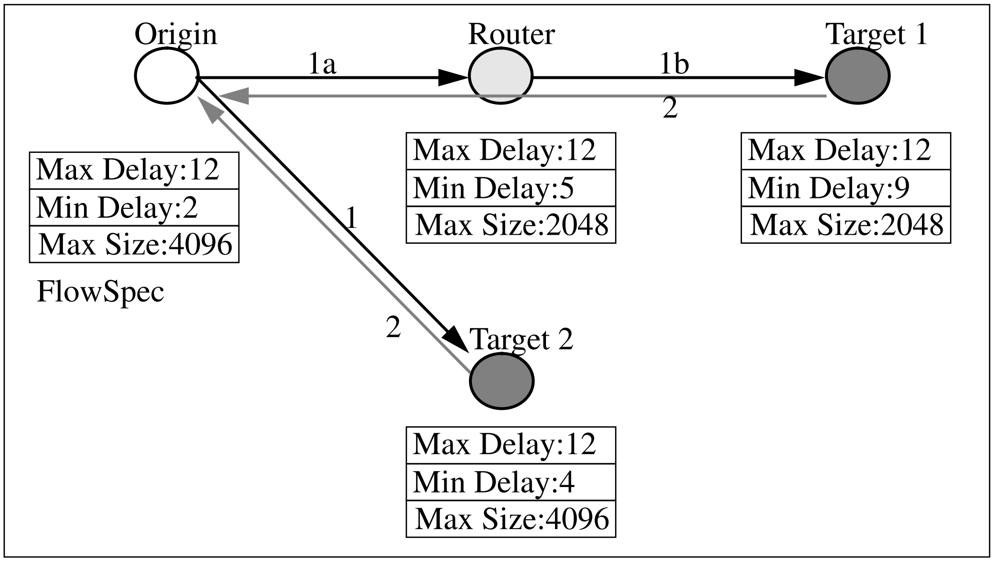

Figure 4: Quality-of-Service Negotiation with FlowSpecs

decision for establishing the connection on it and may finally also modify the FlowSpec. If a target accepts the connection, the (possibly modified) FlowSpec is propagated back to the origin which can then calculate an overall service quality for all targets. The application entity at the origin may later request a CHANGE to adjust reservations.

1.6 Outline of This Document

This document contains the specification of the ST2+ version of the ST2 protocol. In the rest of the document, whenever the terms “ST” or “ST2” are used, they refer to the ST2+ version of ST2.

The document is organized as follows:

- Section 2 describes the ST2 user service from an application point of view.
- Section 3 illustrates the ST2 data transfer protocol, ST.
- Section 4 through Section 8 specify the ST2 setup protocol, SCMP.
- the ST2 flow specification is presented in Section 9.
- the formats of protocol elements and PDUs are defined in Section 10.

2 ST2 User Service Description

This section describes the ST user service from the high-level point of view of an application. It defines the ST stream operations and primitive functions. It specifies which operations on streams can be invoked by the applications built on top of ST and when the ST primitive functions can be legally executed. Note that the presented ST primitives do not specify an API. They are used here with the only purpose of illustrating the service model for ST.

2.1 Stream Operations and Primitive Functions

An ST application at the origin may *create, expand, reduce, change, send data to,* and *delete a stream*. When a stream is expanded, new targets are added to the stream; when a stream is reduced, some of the current targets are dropped from it. When a stream is changed, the associated quality of service is modified.

An ST application at the target may *join, receive data from,* and *leave a stream*. This translates into the following **stream operations**:

- OPEN: create new stream [origin], CLOSE: delete stream [origin],
- ADD: expand stream, i.e. add new targets to it [origin],
- DROP: reduce stream, i.e. drop targets from it [origin],
- JOIN: join a stream [target], LEAVE: leave a stream [target],
- DATA: send data through stream [origin],
- CHG: change a stream's QoS [origin],

Each stream operation may require the execution of several **primitive functions** to be completed. For instance, to open a new stream, a *request* is first issued by the sender and an *indication* is generated at one or more receivers; then, the receivers may each *accept* or *refuse* the request and the correspondent *indications* are generated at the sender. A single receiver case is shown in Figure 5 below.

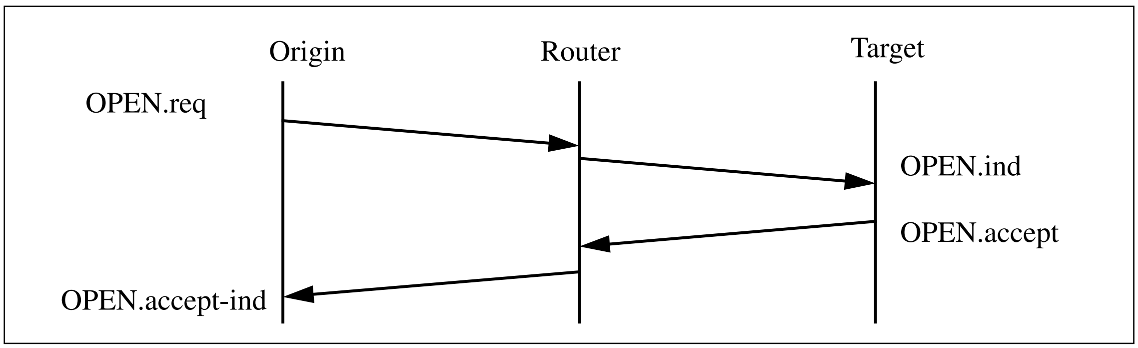

Figure 5: Primitives for the OPEN Stream Operation

Table 1 defines the ST service primitive functions associated to each stream operation. The column labelled "O/T" indicates whether the primitive is executed at the origin or at the target.

Primitive	Description	O/T	Primitive	Description	O/T
OPEN.req	open a stream	O	JOIN.reject-ind	join reject indication	T
OPEN.ind	connection request indication	T	DATA.req	send data	O
OPEN.accept	accept stream	T	DATA.ind	receive data indication	T
OPEN.refuse	refuse stream	T	CHG.req	change stream QoS	O
OPEN.accept-ind	connection accept indication	O	CHG.ind	change request indication	T
OPEN.refuse-ind	connection refuse indication	O	CHG.accept	accept change	T

Table 1: ST Primitives

Primitive	Description	O/T	Primitive	Description	O/T
ADD.req	add targets to stream	O	CHG.refuse	refuse change	T
ADD.ind	add request indication	T	CHG.accept-ind	change accept indication	O
ADD.accept	accept stream	T	CHG.refuse-ind	change refuse indication	O
ADD.refuse	refuse stream	T	DROP.req	drop targets	O
ADD.accept-ind	add accept indication	O	DROP.ind	disconnect indication	T
ADD.refuse-ind	add refuse indication	O	LEAVE.req	leave stream	T
JOIN.req	join a stream	T	LEAVE.ind	leave stream indication	O
JOIN.ind	join request indication	O	CLOSE.req	close stream	O
JOIN.reject	reject a join	O	CLOSE.ind	close stream indication	T

Table 1: ST Primitives

2.2 State Diagrams

It is not sufficient to define the set of ST stream operations. It is also necessary to specify when the operations can be legally executed. For this reason, a set of **states** is now introduced and the transitions from one state to the others are specified. States are defined with respect to a single stream. The previously defined stream operations can be legally executed only from an appropriate state.

An ST agent may, with respect to an ST stream, be in one of the following states:

- IDLE: the stream has not been created yet.
- PENDING: the stream is in the process of being established.
- ACTIVE: the stream is established and active.
- ADDING: the stream is established. A stream expansion is underway.
- CHGING: the stream is established. A stream change is underway.

Previous experience with ST has lead to limits on stream operations that can be executed simultaneously. These restrictions are:

1. *A single ADD or CHG operation can be processed at one time*. If an ADD or CHG is already underway, further requests are queued by the ST agent and handled only after the previous operation has been completed. This also applies to two subsequent requests of the same kind, e.g. two ADD or two CHG operations. The second operation is not executed until the first one has been completed.
2. Deleting a stream, leaving a stream, or dropping targets from a stream is possible *only after stream establishment has been completed*. A stream is considered to be established when all the next-hops of the origin have either accepted or refused the stream. Note that stream refuse is automatically forced after timeout if no reply comes from a next-hop.
3. An ST agent forwards data *only along already established paths to the targets*, see also Section 3.1. A path is considered to be established when the next-hop on the path has explicitly accepted the stream. This implies that the target and all other intermediate ST agents are ready to handle the incoming data packets. In no cases an ST agent will forward data to a

next-hop ST agent that has not explicitly accepted the stream. To be sure that all targets receive the data, an application should send the data only after all paths have been established, i.e. the stream is established.

4. It is allowed to send data from the CHGING and ADDING states. While sending data from the CHGING state, the quality of service to the targets affected by the change should be assumed to be the more restrictive quality of service. When sending data from the ADDING state, the targets that receive the data include at least all the targets that were already part of the stream at the time the ADD operation was invoked.

The rules introduced above require ST agents to queue incoming requests when the current state does not allow to process them immediately. In order to preserve the semantics, ST agents have to maintain the order of the requests, i.e. implement FIFO queuing. Exceptionally, the CLOSE request at the origin and the LEAVE request at the target may be immediately processed: in this cases, the queue is deleted and it is possible that requests in the queue are not processed.

The following state diagrams define the ST service. Separate diagrams are presented for the origin and the targets.

The symbol (a/r)* indicates that all targets in the target list have explicitly accepted or refused the stream, or refuse has been forced after timeout. If the target list is empty, i.e. it contains no targets, the (a/r)* condition is immediately satisfied, so the empty stream is created and state ESTBL is entered.

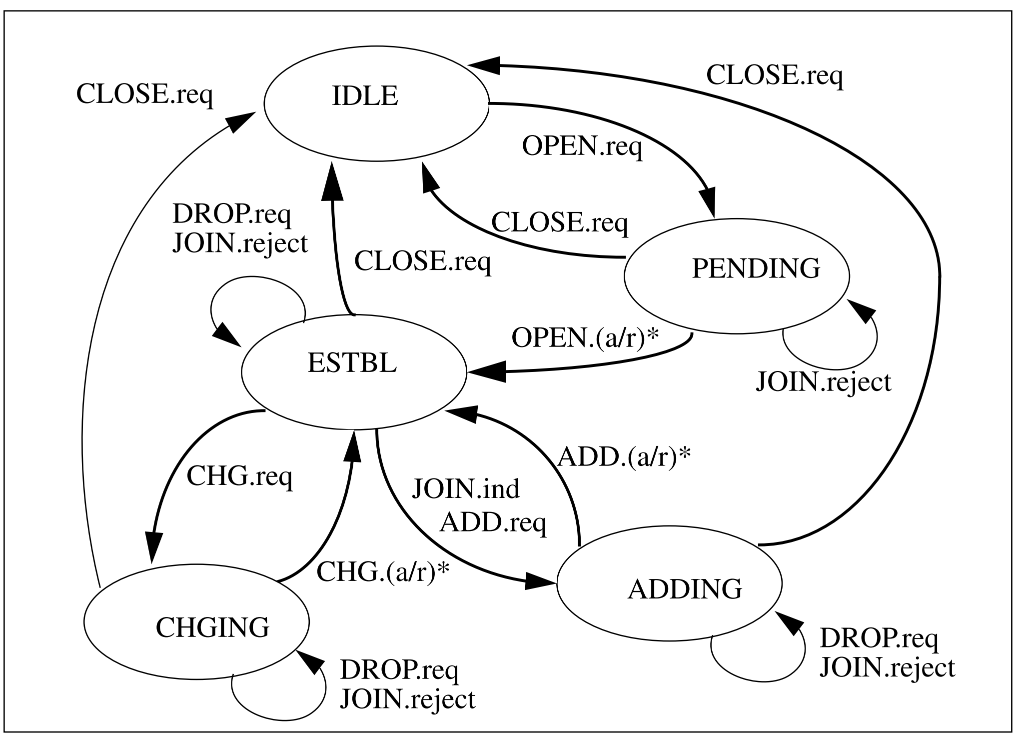

Figure 6: ST Service at the Origin

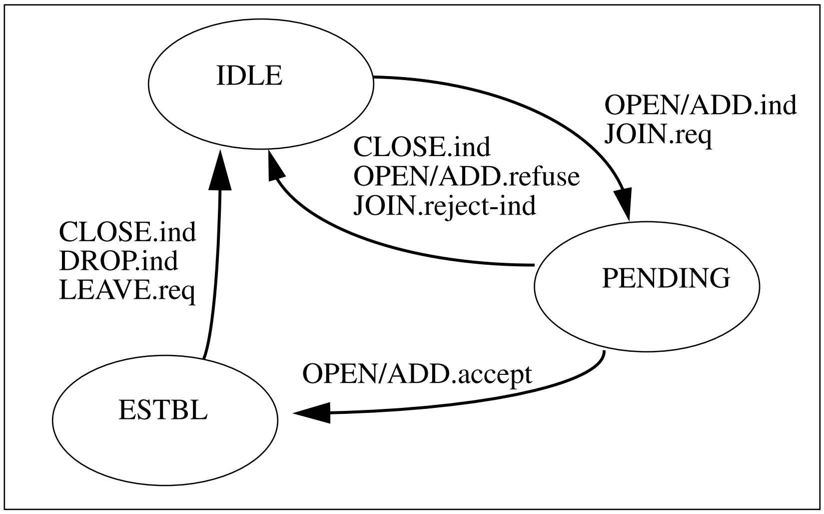

Figure 7: ST Service at the Target

The separate OPEN and ADD primitives at the target are for conceptual purposes only. The target is actually unable to distinguish between an OPEN and an ADD. This is reflected in Figure 7 and Table 3 through the notation OPEN/ADD.

2.3 State Transition Tables

Table 2 and Table 3 define which primitives can be processed from which states and the possible state transitions.

Primitive	IDLE	PENDING	ESTBL	CHGING	ADDING
OPEN.req	ok	-	-	-	-
OPEN.accept-ind	-	if (a,r)* -> ESTBL	-	-	-
OPEN.refuse-ind	-	if (a,r)* -> ESTBL	-	-	-
ADD.req	-	queued	-> ADDING	queued	queued
ADD.accept-ind	-	-	-	-	if (a,r)* -> ESTBL
ADD.refuse-ind	-	-	-	-	if (a,r)* -> ESTBL
JOIN.ind	-	queued	-> ADDING	queued	queued
JOIN.reject	-	ok	ok	ok	ok
DATA.req	-	-	ok	ok	ok
CHG.req	-	queued	-> CHGING	queued	queued
CHG.accept-ind	-	-	-	if (a,r)* -> ESTBL	-
CHG.refuse-ind	-	-	-	if (a,r)* -> ESTBL	-
DROP.req	-	-	ok	ok	ok
LEAVE.ind	-	ok	ok	ok	ok
CLOSE.req	-	ok	ok	ok	ok

Table 2: Primitives and States at the Origin

Primitive	IDLE	PENDING	ESTBL
OPEN/ADD.ind	-> PENDING	-	-
OPEN/ADD.accept	-	-> ESTBL	-
OPEN/ADD.refuse	-	-> IDLE	-
JOIN.req	-> PENDING	-	-
JOIN.reject-ind	-	-> IDLE	-
DATA.ind	-	-	ok
CHG.ind	-	-	ok
CHG.accept	-	-	ok
CHG.refuse	-	-	ok
DROP.ind	-	ok	ok
LEAVE.req	-	ok	ok
CLOSE.ind	-	ok	ok

Table 3: Primitives and States at the Target

3 The ST2 Data Transfer Protocol

This section presents the ST2 data transfer protocol, ST. First, data transfer is described in Section 3.1, then, the data transfer protocol functions are illustrated in Section 3.2.

3.1 Data Transfer with ST

Data transmission with ST is *unreliable*. An application is not guaranteed that the data reaches its destinations and ST makes no attempts to recover from packet loss, e.g. due to the underlying network. However, if the data reaches its destination, it should do so accordingly to the quality of service associated with the stream.

Additionally, ST may deliver data *corrupted* in transmission. Many types of real-time data, such as digital audio and video, require partially correct delivery only. In many cases, retransmitted packets would arrive too late to meet their real-time delivery requirements. On the other hand, depending on the data encoding and the particular application, a small number of errors in stream data are acceptable. In any case, reliability can be provided by layers on top of ST2 if needed.

Also, *no data fragmentation* is supported during the data transfer phase. The application is expected to segment its data PDUs according to the minimum MTU over all paths in the stream. The application receives information on the MTUs relative to the paths to the targets as part of the ACCEPT message, see Section 8.6. The minimum MTU over all paths can be calculated from the MTUs relative to the single paths. ST agents silently discard too long data packets, see also Section 5.1.1.

An ST agent forwards the data *only along already established paths* to targets. A path is considered to be established once the next-hop ST agent on the path sends an ACCEPT message, see Section 2.2. This implies that the target and all other intermediate ST agents on the path to the target are ready to handle the incoming data packets. In no cases will an ST agent forward data to a next-hop ST agent that has not explicitly accepted the stream.

To be reasonably sure that all targets receive the data with the desired quality of service, an application should send the data *only after the whole stream has been established*. Depending on the local API, an application may not be prevented from sending data before the completion of stream setup, but it should be aware that the data could be lost or not reach all intended targets. This behavior may actually be desirable to applications, such as those application that have multiple targets which can each process data as soon as it is available (e.g. a lecture or distributed gaming).

It is desirable for implementations to take advantage of networks that support *multicast*. If a network does not support multicast, or for the case where the next-hops are on different networks, multiple copies of the data packet must be sent.

3.2 ST Protocol Functions

The ST protocol provides two functions:

- stream identification
- data priority

3.2.1 Stream Identification

ST data packets are encapsulated by an ST header containing the Stream IDentifier (SID). This SID is selected at the origin so that it is globally unique over the Internet. The SID must be known by the setup protocol as well. At stream establishment time, the setup protocol builds, at each agent traversed by the stream, an entry into its local database containing stream information. The SID can be used as a reference into this database, to obtain quickly the necessary replication and forwarding information.

Stream IDentifiers are intended to be used to make the packet forwarding task most efficient. The time-critical operation is an intermediate ST agent receiving a packet from the previous-hop ST agent and forwarding it to the next-hop ST agents.

The format of data PDUs including the SID is defined in Section 10.1. Stream IDentifier generation is discussed in Section 8.1.

3.2.2 Packet Discarding based on Data Priority

ST provides a well defined quality of service to its applications. However, there may be cases where the network is temporarily congested and the ST agents have to discard certain packets to minimize the overall impact to other streams. The ST protocol provides a mechanism to discard data packets based on the Priority field in the data PDU, see Section 10.1. The application assigns each data packet with a discard-priority level, carried into the Priority field. ST agents will attempt to discard lower priority packets first during periods of network congestion. Applications may choose to send data at multiple priority levels so that less important data may be discarded first.

4 SCMP Functional Description

ST agents create and manage streams using the ST Control Message Protocol (SCMP). Conceptually, SCMP resides immediately above ST (as does ICMP above IP). SCMP follows a request-response model. SCMP messages are made reliable through the use of retransmission after timeout.

This section contains a functional description of stream management with SCMP. To help clarify the SCMP exchanges used to setup and maintain ST streams, we include an example of a simple network topology, represented in Figure 8. Using the SCMP messages described in this section it will be possible for an ST application to:

- Create a stream from A to the peers at B, C and D,
- Add a peer at E,
- Drop peers B and C, and
- Let F join the stream
- Delete the stream.

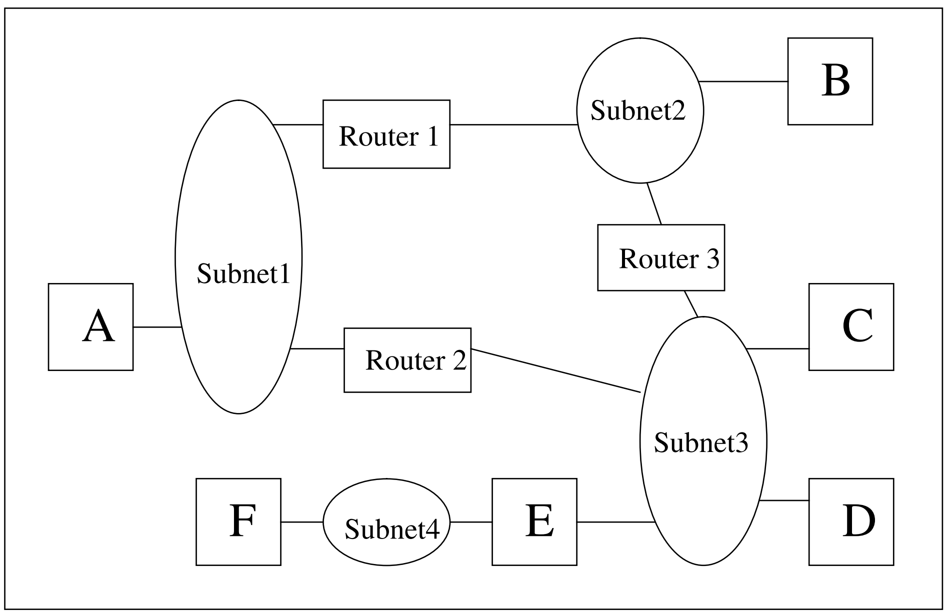

Figure 8: Sample Topology for an ST Stream

We first describe the possible types of stream in Section 4.1; Section 4.2 introduces SCMP control message types; SCMP reliability is discussed in Section 4.3; stream options are covered in Section 4.4; stream setup is presented in Section 4.5; Section 4.6 illustrates stream modification including stream expansion, reduction, changes of the quality of service associated to a stream. Finally, stream deletion is handled in Section 4.7.

4.1 Types of Streams

SCMP allows for the setup and management of different types of streams. Streams differ in the way they are built and the information maintained on connected targets.

4.1.1 Stream Building

Streams may be built in a sender-oriented fashion, receiver-oriented fashion, or with a mixed approach:

- in the **sender-oriented** fashion, the application at the origin provides the ST agent with the list of receivers for the stream. New targets, if any, are also added from the origin.
- in the **receiver-oriented** approach, the application at the origin creates an empty stream that contains no targets. Each target joins then the stream autonomously.
- in the **mixed** approach, the application at the origin creates a stream that contains some targets and other targets join then the stream autonomously.

ST2 provides stream options to support sender-oriented and mixed approach steams. Receiver-oriented streams can be emulated through the use of mixed streams. The fashion by which targets may be added to a particular stream is controlled via join authorization levels. Join authorization levels are described in Section 4.4.2.

4.1.2 Knowledge of Receivers

When streams are built in the sender-oriented fashion, all ST agents will have full information on all targets down stream of a particular agent. In this case, target information is relayed down stream from agent-to-agent during stream set-up.

When targets add themselves to mixed approach streams, upstream ST agents may or may not be informed. Propagation of information on targets that “join” a stream is also controlled via join authorization levels. As previously mentioned, join authorization levels are described in Section 4.4.2.

This leads to two types of streams:

- full target information is propagated in a **full-state** stream. For such streams, all agents are aware of all downstream targets connected to the stream. This results in target information being maintained at the origin and at intermediate agents. Operations on single targets are always possible, i.e. change a certain target, or, drop that target from the stream. It is also always possible for any ST agent to attempt recovery of all downsteam targets.

- in **light-weight** streams, it is possible that the origin and other upstream agents have no knowledge about some targets. This results in less maintained state and easier stream management, but it limits operations on specific targets. Special actions may be required to support change and drop operations on unknown targets, see Section 5.7. Also, stream recovery may not be possible. Of course, generic functions such as deleting the whole stream, are still possible. It is expected that applications that will have a large number of targets will use light-weight streams in order to limit state in agents and the number of targets per control message.

Full-state streams serve well applications as video conferencing or distributed gaming, where it is important to have knowledge on the connected receivers, e.g. to limit who participates. Light-weight streams may be exploited by applications such as remote lecturing or playback applications of radio and TV broadcast where the receivers do not need to be known by the sender. Section 4.4.2 defines join authorization levels, which support two types of full-state streams and one type of light-weight streams.

4.2 Control PDUs

SCMP defines the following PDUs (the main purpose of each PDU is also indicated):

1.	ACCEPT	to accept a new stream
2.	ACK	to acknowledge an incoming message
3.	CHANGE	to change the quality of service associated with a stream
4.	CONNECT	to establish a new stream or add new targets to an existing stream
5.	DISCONNECT	to remove some or all of the stream's targets
6.	ERROR	to indicate an error contained into an incoming message
7.	HELLO	to detect failures of neighbor ST agents
8.	JOIN	to request stream joining from a target
9.	JOIN-REJECT	to reject a stream joining request from a target
10.	NOTIFY	to inform an ST agent of a significant event
11.	REFUSE	to refuse the establishment of a new stream
12.	STATUS	to query an ST agent on a specific stream
13.	STATUS-RESPONSE	to reply queries on a specific stream

SCMP follows a request-response model with all requests expecting responses. Retransmission after timeout is used to allow for lost or ignored messages. Control messages do not extend across packet boundaries; if a control message is too large for the MTU of a hop, its information is partitioned and a control message per partition is sent, as described in Section 5.1.2.

The ACCEPT, CHANGE, CONNECT, DISCONNECT, JOIN, JOIN-REJECT, NOTIFY and REFUSE must always be explicitly acknowledged:

- with an **ACK** message, if the message was received correctly and it was possible to parse and correctly extract and interpret its header, fields and parameters,
- with an **ERROR** message, if a syntax error was detected in the header, fields, or parameters included in the message. The errored PDU may be optionally returned as part of the ERROR message. An ERROR message indicates a syntax error only. If any other errors are detected, it is necessary to first acknowledge with ACK and then take appropriate actions. For instance, suppose a CHANGE message contains an unknown SID: first, an ACK message has to be sent, then a REFUSE message with ReasonCode (SIDUnknown) follows.

If no ACK or ERROR message are received before the correspondent timer expires, a timeout failure occurs. The way an ST agent should handle timeout failures is described in Section 5.2.

ACK, ERROR, and STATUS-RESPONSE messages are never acknowledged.

HELLO messages are a special case. If they contain a syntax error, an ERROR message should be generated in response. Otherwise, no acknowledgment or response should be generated. Use of HELLO messages is discussed in Section 6.1.2.

STATUS messages containing a syntax error should be replied with an ERROR message. Otherwise, a STATUS-RESPONSE message should be sent back in response. Use of STATUS and STATUS-RESPONSE are discussed in Section 8.4.

4.3 SCMP Reliability

SCMP is made reliable through the use of retransmission when response is not received in a timely manner. The ACCEPT, CHANGE, CONNECT, DISCONNECT, JOIN, JOIN-REJECT, NOTIFY, and REFUSE messages all must be answered with an ACK message, see Section 4.2. In general, when sending a SCMP message which requires an ACK response, the sending ST agent needs to set the Toxxxx timer (where xxxx is the SCMP message type, e.g. ToConnect). If it does not receive an ACK before the Toxxxx timer expires, the ST agent should retransmit the SCMP message. If no ACK has been received within Nxxxx retransmissions, then a SCMP *timeout condition* occurs and the ST agent enters its SCMP timeout recovery state. The actions performed by the ST agent as the result of the SCMP timeout condition differ for different SCMP messages and are described in Section 5.2.

For some SCMP messages (CONNECT, CHANGE, JOIN, and STATUS) the sending ST agent also expects a response back (ACCEPT/REFUSE, CONNECT/JOIN-REJECT) after ACK has been received. For these cases, the ST agent needs to set the ToxxxxResp timer after it receives the ACK. (As before, xxxx is the initiating SCMP message type, e.g. ToConnectResp). If it does not receive the appropriate response back when ToxxxxResp expires, the ST agent updates its state and performs appropriate recovery action as described in Section 5.2.

The timeout and retransmission algorithm is implementation dependent and it is outside the scope of this document. Most existing algorithms are based on an estimation of the Round Trip Time (RTT) between two agents. Therefore, SCMP contains a mechanism, see Section 8.5, to estimate this RTT. Note that the timeout related variable names described above are for reference purposes only, implementors may choose to combine certain variables.

4.4 Stream Options

An application may select among some stream options. The desired options are indicated to the ST agent at the origin when a new stream is created. Options apply to single streams and are valid during the whole stream's lifetime. The options chosen by the application at the origin are included into the initial CONNECT message, see Section 4.5.3. When a CONNECT message reaches a target, the application at the target is notified of the stream options that have been selected, see Section 4.5.5.

4.4.1 No Recovery

When a stream failure is detected, an ST agent would normally attempt stream recovery, as described in Section 6.2. The NoRecovery option is used to indicate that ST agents should not attempt recovery for the stream. The protocol behaviour in case the NoRecovery option has been selected is illustrated in Section 6.2. The NoRecovery option is specified by setting the S-bit in the CONNECT message, see Section 10.4.4. The S-bit can be set only by the origin and it is never modified by intermediate and target ST agents.

4.4.2 Join Authorization Level

When a new stream is created, it is necessary to define the *join authorization level* associated with the stream. This level determines the protocol behavior in case of stream joining, see Section 4.1 and Section 4.6.3. The join authorization level for a stream is defined by the J-bit and N-bit in the CONNECT message header, see Section 10.4.4. One of the following authorization levels has to be selected:

- **Level 0 - Refuse Join** (JN = 00): No targets are allowed to join this stream.
- **Level 1 - OK, Notify Origin** (JN = 01): Targets are allowed to join the stream. The origin is notified that the target has joined.
- **Level 2 - OK** (JN = 10): Targets are allowed to join the stream. No notification is sent to the stream origin.

Some applications may choose to maintain tight control on their streams and will not permit any connections without the origin's permission. For such streams, target applications may request to be added by sending an out-of-band, i.e. via regular IP, request to the origin. The origin, if it so chooses, can then add the target following the process described in Section 4.6.1.

The selected authorization level impacts stream handling and the state that is maintained for the stream, as described in Section 4.1.

4.4.3 Record Route

The RecordRoute option can be used to request the route between the origin and a target be recorded and delivered to the application. This option may be used while connecting, accepting, changing, or refusing a stream. The results of a RecordRoute option requested by the origin, i.e. as part of the CONNECT or CHANGE messages, are delivered to the target. The results of a RecordRoute option requested by the target, i.e. as part of the ACCEPT or REFUSE messages, are delivered to the origin.

The RecordRoute option is specified by adding the RecordRoute parameter to the mentioned SCMP messages. The format of the RecordRoute parameter is shown in Section 10.3.5. When adding this parameter, the ST agent at the origin must determine the number of entries that may be recorded as explained in Section 10.3.5.

4.4.4 User Data

The UserData option can be used by applications to transport application specific data along with some SCMP control messages. This option can be included with ACCEPT, CHANGE, CONNECT, DISCONNECT, and REFUSE messages. The format of the UserData parameter is shown in Section 10.3.7. This option may be included by the origin, or the target, by adding the UserData parameter to the mentioned SCMP messages. This option may only be included once per SCMP message.

4.5 Stream Setup

This section presents a description of stream setup. For simplicity, we assume that everything succeeds, e.g. any required resources are available, messages are properly delivered, and the routing is correct. Possible failures in the setup phase are handled in Section 5.2.

4.5.1 Information from the Application

Before stream setup can be started, the application has to collect the necessary information to determine the characteristics for the connection. This includes identifying the participants and selecting the QoS parameters of the data flow. Information passed to the ST agent by the application includes:

- the *list of the stream's targets* (Section 10.3.6). The list may be *empty* (Section 4.5.3.1),
- the *flow specification* containing the desired quality of service for the stream (Section 9),
- information on the *groups* in which the stream is a member, if any (Section 7),
- information on the *options* selected for the stream (Section 4.4).

4.5.2 Initial Setup at the Origin

The ST agent at the origin then performs the following operations:

- allocates a stream ID (SID) for the stream (Section 8.1),
- invokes the routing function to determine the set of next-hops for the stream (Section 4.5.2.1),
- invokes the Local Resource Manager (LRM) to reserve resources (Section 4.5.2.2),
- creates local database entries to store information on the new stream,
- propagates the stream creation request to the next-hops determined by the routing function (Section 4.5.3).

4.5.2.1 Invoking the Routing Function

An ST agent that is setting up a stream invokes the routing function to find the next-hop to reach each of the targets specified by the target list provided by the application. This is similar to the routing decision in IP. However, in this case the route is to a multitude of targets with QoS requirements rather than to a single destination.

The result of the routing function is a set of next-hop ST agents. The set of next-hops selected by the routing function is not necessarily the same as the set of next-hops that IP would select given a number of independent IP datagrams to the same destinations. The routing algorithm may attempt to optimize parameters other than the number of hops that the packets will take, such as delay, local network bandwidth consumption, or total internet bandwidth consumption. Alternatively, the routing algorithm may use a simple route lookup for each target.

Once a next-hop is selected by the routing function, it persists for the whole stream lifetime, unless a network failure occurs.

4.5.2.2 Reserving Resources

The ST agent invokes the Local Resource Manager (LRM) to perform the appropriate reservations. The ST agent presents the LRM with information including:

- the *flow specification* with the desired quality of service for the stream (Section 9),
- the *version number* associated with the flow specification (Section 9).
- information on the *groups* the stream is member in, if any (Section 7),

The flow specification contains information needed by the LRM to allocate resources. The LRM updates the flow specification contents information before returning it to the ST agent. Section 9.2.3 defines the fields of the flow specification to be updated by the LRM.

The membership of a stream in a group may affect the amount of resources that have to be allocated by the LRM, see Section 7.

4.5.3 Sending CONNECT Messages

The ST agent sends a CONNECT message to each of the next-hop ST agents identified by the routing function. Each CONNECT message contains the SID, the selected stream options, the FlowSpec, and a TargetList. The format of the CONNECT message is defined by Section 10.4.4. In general, the FlowSpec and TargetList depend on both the next-hop and the intervening network. Each TargetList is a subset of the original TargetList, identifying the targets that are to be reached through the next-hop to which the CONNECT message is being sent.

The TargetList may be empty, see Section 4.5.3.1; if the TargetList causes a too long CONNECT message to be generated, the CONNECT message is partitioned as explained in Section 5.1.2. If multiple next-hops are to be reached through a network that supports network level multicast, a different CONNECT message must nevertheless be sent to each next-hop since each will have a different TargetList.

4.5.3.1 Empty Target List

An application at the origin may request the local ST agent to create an *empty stream*. It does so by passing an empty TargetList to the local ST agent during the initial stream setup. When the local ST agent receives request to create an empty stream, it allocates the stream ID (SID), updates its local database entries to store information on the new stream and notifies the application that stream setup is complete. The local ST agent does not generate any CONNECT message for streams with an empty TargetList. Targets may be later added by the origin, see Section 4.6.1, or they may autonomously join the stream, see Section 4.6.3.

4.5.4 CONNECT Processing by an Intermediate ST agent

An ST agent receiving a CONNECT message, assuming no errors, responds to the previous-hop with an ACK. The ACK message must identify the CONNECT message to which it corresponds by including the reference number indicated by the Reference field of the CONNECT message. The intermediate ST agent calls the routing function, invokes the LRM to reserve resources, and then propagates the CONNECT messages to its next-hops, as described in the previous sections.

4.5.5 CONNECT Processing at the Targets

An ST agent that is the target of a CONNECT message, assuming no errors, responds to the previous-hop with an ACK. The ST agent invokes the LRM to reserve local resources and then queries the specified application process whether or not it is willing to accept the connection.

The application is presented with parameters from the CONNECT message including the SID, the selected stream options, Origin, FlowSpec, TargetList, and Group, if any, to be used as a basis for its decision. The application is identified by a combination of the NextPcol field, from the Origin parameter, and the service access point, or SAP, field included in the correspondent (usually single remaining) Target of the TargetList. The contents of the SAP field may specify the port or other local identifier for use by the protocol layer above the host ST layer. Subsequently received data packets will carry the SID, that can be mapped into this information and be used for their delivery.

Finally, based on the application's decision, the ST agent sends to the previous-hop from which the CONNECT message was received either an ACCEPT or REFUSE message. Since the ACCEPT (or REFUSE) message has to be acknowledged by the previous-hop, it is assigned a new Reference number that will be returned in the ACK. The CONNECT message to which ACCEPT (or REFUSE) is a reply is identified by placing the CONNECT's Reference number in the LnkReference field of ACCEPT (or REFUSE). The ACCEPT message contains the FlowSpec as accepted by the application at the target.

4.5.6 ACCEPT Processing by an Intermediate ST agent

When an intermediate ST agent receives an ACCEPT, it first verifies that the message is a response to an earlier CONNECT. If not, it responds to the next-hop ST agent with an ERROR message, with ReasonCode (LnkRefUnknown). Otherwise, it responds to the next-hop ST agent with an ACK, and propagates the individual ACCEPT message to the previous-hop along the same path traced by the CONNECT but in the reverse direction toward the origin.

The FlowSpec is included in the ACCEPT message so that the origin and intermediate ST agents can gain access to the information that was accumulated as the CONNECT traversed the internet. Note that the resources, as specified in the FlowSpec in the ACCEPT message, may differ from the resources that were reserved when the CONNECT was originally processed. Therefore, the ST agent presents the LRM with the FlowSpec included in the ACCEPT message. It is expected that each LRM adjusts local reservations releasing any excess resources. The LRM may choose not to adjust local reservations when that adjustment may result in the loss of needed resources. It may also choose to wait to adjust allocated resources until all targets in transition have been accepted or refused.

In the case where the intermediate ST agent is acting as the origin with respect to this target, see Section 4.6.3.1, the ACCEPT message is not propagated upstream.

4.5.7 ACCEPT Processing by the Origin

The origin will eventually receive an ACCEPT (or REFUSE) message from each of the targets. As each ACCEPT is received, the application is notified of the target and the resources that were successfully allocated along the path to it, as specified in the FlowSpec contained in the ACCEPT message. The application may then use the information to either adopt or terminate the portion of the stream to each target. When ACCEPT (or REFUSE) from all targets have been received at the origin, the application is notified that stream setup is complete.

When an ACCEPT is received by the origin, the path to the target is considered to be established and the ST agent is allowed to forward the data along this path as explained in Section 2 and in Section 3.1.

4.5.8 REFUSE Processing by the Intermediate ST agent

If an application at a target does not wish to participate in the stream, it sends a REFUSE message back to the origin with ReasonCode (ApplDisconnect). An intermediate ST agent that receives a REFUSE message with ReasonCode (ApplDisconnect) acknowledges it by sending an ACK to the next-hop, invokes the LRM to adjusts reservations as appropriate, deletes the target entry from the internal database, and propagates the REFUSE message back to the previous-hop ST agent.

In the case where the intermediate ST agent is acting as the origin with respect to this target, see Section 4.6.3.1, the REFUSE message is only propagated upstream when there are no more downstream agents participating in the stream. In this case, the agent indicates that the agent is to be removed from the stream propagating the REFUSE message with the G-bit set (1).

4.5.9 REFUSE Processing by the Origin

When the REFUSE message reaches the origin, the ST agent at the origin sends an ACK and notifies the application that the target is no longer part of the stream and also if the stream has no remaining targets. If there are no remaining targets, the application may wish to terminate the stream or keep the stream active to allow addition of targets, or stream joining as described in Section 4.6.3.

4.5.10 Other Functions during Stream Setup

Some other functions have to be accomplished by an ST agent as CONNECT messages travel downstream and ACCEPT (or REFUSE) messages travel upstream during the stream setup phase. They were not mentioned in the previous sections to keep the discussion as simple as possible. These functions include:

- computing the smallest Maximum Transmission Unit size over the path to the targets, as part of the MTU discovery mechanism presented in Section 8.6. This is done by updating the *MaxMsgSize* field of the CONNECT message, see Section 10.4.4. This value is carried back to origin in the *MaxMsgSize* field of the ACCEPT message, see Section 10.4.1.
- counting the number of IP clouds that have to be traversed to reach the targets, as part of the IP encapsulation mechanism described in Section 8.7. This is done by updating the *IPHops* field of the CONNECT message, see Section 10.4.4. This value is carried back to origin in the *IPHops* field of the ACCEPT message, see Section 10.4.1.
- updating the RecoveryTimeout value for the stream based on what can the agent can support. This is part of the stream recovery mechanism, in Section 6.2. This is done by updating the *RecoveryTimeout* field of the CONNECT message, see Section 10.4.4. This value is carried back to origin in the *RecoveryTimeout* field of the ACCEPT message, see Section 10.4.1.

4.6 Modifying an Existing Stream

Some applications may wish to modify a stream after it has been created. Possible changes include expanding a stream, reducing it, and changing its FlowSpec. The origin may add or remove targets as described in Section 4.6.1 and Section 4.6.2. Targets may request to join the stream as described in Section 4.6.3 or, they may decide to leave a stream as described in Section 4.6.4. Section 4.6.5 explains how to change a stream's FlowSpec.

As defined by Section 2, an ST agent can handle only one stream modification at a time. If a stream modification operation is already underway, further requests are queued and handled when the previous operation has been completed. This also applies to two subsequent requests of the same kind, e.g. two subsequent changes to the FlowSpec.

4.6.1 The Origin Adding New Targets

It is possible for an application at the origin to add new targets to an existing stream any time after the stream has been established. Before new targets are added, the application has to collect the necessary information on the new targets. Such information is passed to the ST agent at the origin.

The ST agent at the origin issues a CONNECT message that contains the SID, the FlowSpec, and the TargetList specifying the new targets. This is similar to sending a CONNECT message during stream establishment, with the following exceptions: the origin checks that a) the SID is valid, b) the targets are not already members of the stream, c) that the LRM evaluates the FlowSpec of the new target to be the same as the FlowSpec of the existing stream, i.e it requires an equal or smaller amount of resources to be allocated. If the FlowSpec of the new target does not match the FlowSpec of the existing stream, an error is generated with ReasonCode (FlowSpecMismatch). Functions to compare flow specifications are provided by the LRM, see Section 1.4.5.

An intermediate ST agent that is already a participant in the stream looks at the SID and StreamCreationTime, and verifies that the stream is the same. It then checks if the intersection of the TargetList and the targets of the established stream is empty. If this is not the case, it responds with a REFUSE message with ReasonCode (TargetExists) that contains a TargetList of those targets that were duplicates. To indicate that the stream exists, and includes the listed targets, the ST agent sets to one (1) the E-bit of the REFUSE message, see Section 10.4.11.

The agent then proceeds processing each new target in the TargetList.

For each new target in the TargetList, processing is much the same as for the original CONNECT. The CONNECT is acknowledged, propagated, and network resources are reserved. Intermediate or target ST agents that are not already participants in the stream behave as in case of stream setup (see Section 4.5.4 and Section 4.5.5).

4.6.2 The Origin Removing a Target

It is possible for an application at the origin to remove existing targets of a stream any time after the targets have accepted the stream. The application at the origin specifies the set of targets that are to be removed and informs the local ST agent. Based on this information, the ST agent sends DISCONNECT messages with the ReasonCode (ApplDisconnect) to the next-hops relative to the targets.

An ST agent that receives a DISCONNECT message must acknowledge it by sending an ACK to the previous-hop. The ST agent updates its state and notifies the LRM of the target deletion so that the LRM can modify reservations as appropriate. When the DISCONNECT message reaches the target, the ST agent also notifies the application that the target is no longer part of the stream. When there are no remaining targets that can be reached through a particular next-hop, the ST agent informs the LRM and it deletes the next-hop from its next-hops set.

SCMP also provides a flooding mechanism to delete targets that joined the stream without notifying the origin. The special case of target deletion via flooding is described in Section 5.7.

4.6.3 A Target Joining a Stream

An application may request to join an existing stream. It has to collect information on the stream including the stream ID (SID) and the IP address of the stream's origin. This can be done out-of-band, e.g. via regular IP. The information is then passed to the local ST agent. The ST agent generates a JOIN message containing the application's request to join the stream and sends it toward the stream origin.

An ST agent receiving a JOIN message, assuming no errors, responds with an ACK. The ACK message must identify the JOIN message to which it corresponds by including the Reference number indicated by the Reference field of the JOIN message. If the ST agent is not traversed by the stream that has to be joined, it propagates the JOIN message toward the stream's origin. Once a JOIN message has been acknowledged, ST agents do not retain any state information related to the JOIN message.

Eventually, an ST agent traversed by the stream or the stream's origin itself is reached. This agent must respond to a received JOIN first with an ACK to the ST agent from which the message was received, then, it issues either a CONNECT or a JOIN-REJECT message and sends it toward the target. The response to the join request is based on the join authorization level associated with the stream, see Section 4.4.2:

- If the stream has authorization level #0 (refuse join):

  The ST agent sends a JOIN-REJECT message toward the target with ReasonCode (JoinAuthFailure).

- If the stream has authorization level #1 (ok, notify origin):

  The ST agent sends a CONNECT message toward the target with a TargetList including the target that requested to join the stream. This eventually results in adding the target to the stream. When the ST agent receives the ACCEPT message indicating that the new target has been added, it does not propagate the ACCEPT message backwards (Section 4.5.6). Instead, it issues a NOTIFY message with ReasonCode (TargetJoined) so that upstream agents, including the origin, may add the new target to maintained state information. The NOTIFY message includes all target specific information.

- If the stream has authorization level #2 (ok):

  The ST agent sends a CONNECT message toward the target with a TargetList including the target that requested to join the stream. This eventually results in adding the target to the stream. When the ST agent receives the ACCEPT message indicating that the new target has been added, it does not propagate the ACCEPT message backwards (Section 4.5.6), nor does it notify the origin. A NOTIFY message is generated with ReasonCode (TargetJoined) if the target specific information needs to be propagated back to the origin. An example of such information is change in MTU, see Section 8.6.

4.6.3.1 Intermediate Agent (Router) as Origin

When a stream has join authorization level #2, see Section 4.4.2, it is possible that the stream origin is unaware of some targets participating in the stream. In this case, the ST intermediate agent that first sent a CONNECT message to this target has to *act as the stream origin* for the given target. This includes:

- if the whole stream is deleted, the intermediate agent must disconnect the target.
- if the stream FlowSpec is changed, the intermediate agent must change the FlowSpec for the target as appropriate.
- proper handling of ACCEPT and REFUSE messages, without propagation to upstream ST agents.
- generation of NOTIFY messages when needed. (As described above.)

The intermediate agent behaves normally for all other targets added to the stream as a consequence of a CONNECT message issued by the origin.

4.6.4 A Target Deleting Itself

The application at the target may inform the local ST agent that it wants to be removed from the stream. The ST agent then forms a REFUSE message with the target itself as the only entry in the TargetList and with ReasonCode (ApplDisconnect). The REFUSE message is sent back to the origin via the previous-hop. If a stream has multiple targets and one target leaves the stream using this REFUSE mechanism, the stream to the other targets is not affected; the stream continues to exist.

An ST agent that receives a REFUSE message acknowledges it by sending an ACK to the next-hop. The target is deleted and the LRM is notified so that it adjusts reservations as appropriate. The REFUSE message is also propagated back to the previous-hop ST agent except in the case where the agent is acting as the origin. In this case a NOTIFY may be propagated instead, see Section 4.6.3.

When the REFUSE reaches the origin, the origin sends an ACK and notifies the application that the target is no longer part of the stream.

4.6.5 Changing a Stream's FlowSpec

The application at the origin may wish to change the FlowSpec of an established stream. Changing the FlowSpec is a critical operation and it may even lead in some cases to the deletion of the affected targets. Possible problems with FlowSpec changes are discussed in Section 5.6.

To change the stream's FlowSpec, the application informs the ST agent at the origin of the new FlowSpec and of the list of targets relative to the change. The ST agent at the origin then issues one CHANGE message per next-hop including the new FlowSpec and sends it to the relevant next-hop ST agents. If the G-bit field of the CHANGE message is set (1), the change affects all targets in the stream.

The CHANGE message contains a bit called I-bit, see Section 10.4.3. By default, the I-bit is set to zero (0) to indicate that the LRM is expected to try and perform the requested FlowSpec change without risking to tear down the stream. Applications that desire a higher probability of success and are willing to take the risk of breaking the stream can indicate this by setting the I-bit to one (1). Applications that require the requested modification in order to continue operating are expected to set this bit.

An intermediate ST agent that receives a CHANGE message first sends an ACK to the previous-hop and then provides the FlowSpec to the LRM. If the LRM can perform the change, the ST agent propagates the CHANGE messages along the established paths.

If the whole process succeeds, the CHANGE messages will eventually reach the targets. Targets respond with an ACCEPT (or REFUSE) message that is propagated back to the origin. In processing the ACCEPT message on the way back to the origin, excess resources may be released by the LRM as described in Section 4.5.6. The REFUSE message must have the ReasonCode (ApplRefused).

SCMP also provides a flooding mechanism to change targets that joined the stream without notifying the origin. The special case of target change via flooding is described in Section 5.7.

4.7 Stream Tear Down

A stream is usually terminated by the origin when it has no further data to send. A stream is also torn down if the application should terminate abnormally or if certain network failures are encountered. Processing in this case is identical to the previous descriptions except that the ReasonCode (ApplAbort, NetworkFailure, etc.) is different.

When all targets have left a stream, the origin notifies the application of that fact, and the application is then responsible for terminating the stream. Note, however, that the application may decide to add targets to the stream instead of terminating it, or may just leave the stream open with no targets in order to permit stream joins.

5 Exceptional Cases

The previous descriptions covered the simple cases where everything worked. We now discuss what happens when things do not succeed. Included are situations where messages exceed a network MTU, are lost, the requested resources are not available, the routing fails or is inconsistent.

5.1 Long ST Messages

It is possible that an ST agent, or an application, will need to send a message that exceeds a network's Maximum Transmission Unit (MTU). This case must be handled but not via generic fragmentation, since ST2 does not support generic fragmentation of either data or control messages.

5.1.1 Handling of Long Data Packets

ST agents discard data packets that exceed the MTU of the next-hop network. No error message is generated. Applications should avoid sending data packets larger than the minimum MTU supported by a given stream. The application, both at the origin and targets, can learn the stream minimum MTU through the MTU discovery mechanism described in Section 8.6.

5.1.2 Handling of Long Control Packets

Each ST agent knows the MTU of the networks to which it is connected, and those MTUs restrict the size of the SCMP message it can send. An SCMP message size can exceed the MTU of a given network for a number of reasons:

- the TargetList parameter (Section 10.3.6) may be too long;
- the RecordRoute parameter (Section 10.3.5) may be too long.
- the UserData parameter (Section 10.3.7) may be too long;
- the PDUInError field of the ERROR message (Section 10.4.6) may be too long;

An ST agent receiving or generating a too long SCMP message should:

- break the message into multiple messages, each carrying part of the TargetList. Any RecordRoute and UserData parameters are replicated in each message for delivery to all targets. Applications that support a large number of targets may avoid using long TargetList parameters, and are expected to do so, by exploiting the stream joining functions, see Section 4.6.3. One exception to this rule exists. In the case of a long TargetList parameter to be included in a STATUS-RESPONSE message, the TargetList parameter is just truncated to the point where the list can fit in a single message, see Section 8.4.

- for down stream agents: if the TargetList parameter contains a single Target element and the message size is still too long, the ST agent should issue a REFUSE message with ReasonCode (RecordRouteSize) if the size of the RecordRoute parameter causes the SCMP message size to exceed the network MTU, or with ReasonCode (UserDataSize) if the size of the UserData parameter causes the SCMP message size to exceed the network MTU. If both RecordRoute and UserData parameters are present the ReasonCode (UserDataSize) should be sent. For messages generated at the target: the target ST agent must check for SCMP messages that may exceed the MTU on the complete target-to-origin path, and inform the application that a too long SCMP messages has been generated. The format for the error reporting is a local implementation issue. The error codes are the same as previously stated.

ST agents generating too long ERROR messages, simply truncate the PDUInError field to the point where the message is smaller than the network MTU.

5.2 Timeout Failures

As described in Section 4.3, SCMP message delivery is made reliable through the use of acknowledgments, timeouts, and retransmission. The ACCEPT, CHANGE, CONNECT, DISCONNECT, JOIN, JOIN-REJECT, NOTIFY, and REFUSE messages must always be acknowledged, see Section 4.2. In addition, for some SCMP messages (CHANGE, CONNECT, JOIN) the sending ST agent also expects a response back (ACCEPT/REFUSE, CONNECT/JOIN-REJECT) after ACK has been received. Also, the STATUS message must be replied to with a STATUS-RESPONSE message.

The following sections describe the handling of each of the possible failure cases due to timeout situations while waiting for an acknowledgment or a response. The timeout related variables, and their names, used in the next sections are for reference purposes only. They may be implementation specific. Different implementations are not required to share variable names, or even the mechanism by which the timeout and retransmission behavior is implemented.

5.2.1 Failure due to ACCEPT Acknowledgment Timeout

An ST agent that sends an ACCEPT message upstream expects an ACK from the previous-hop ST agent. If no ACK is received before the *ToAccept* timeout expires, the ST agent should retry and send the ACCEPT message again. After *NAccept* unsuccessful retries, the ST agent sends a REFUSE message toward the origin, and a DISCONNECT message toward the targets. Both REFUSE and DISCONNECT must identify the affected targets and specify the ReasonCode (RetransTimeout).

5.2.2 Failure due to CHANGE Acknowledgment Timeout

An ST agent that sends a CHANGE message downstream expects an ACK from the next-hop ST agent. If no ACK is received before the *ToChange* timeout expires, the ST agent should retry and send the CHANGE message again. After *NChange* unsuccessful retries, the ST agent aborts the change attempt by sending a REFUSE message toward the origin, and a DISCONNECT message toward the targets. Both REFUSE and DISCONNECT must identify the affected targets and specify the ReasonCode (RetransTimeout).

5.2.3 Failure due to CHANGE Response Timeout

Only the origin ST agent implements this timeout. After correctly receiving the ACK to a CHANGE message, an ST agent expects to receive an ACCEPT, or REFUSE message in response. If one of these messages is not received before the *ToChangeResp* timer expires, the ST agent at the origin aborts the change attempt, and behaves as if a REFUSE message with the E-bit set and with ReasonCode (ResponseTimeout) is received.

5.2.4 Failure due to CONNECT Acknowledgment Timeout

An ST agent that sends a CONNECT message downstream expects an ACK from the next-hop ST agent. If no ACK is received before the *ToConnect* timeout expires, the ST agent should retry and send the CONNECT message again. After *NConnect* unsuccessful retries, the ST agent sends a REFUSE message toward the origin, and a DISCONNECT message toward the targets. Both REFUSE and DISCONNECT must identify the affected targets and specify the ReasonCode (RetransTimeout).

5.2.5 Failure due to CONNECT Response Timeout

Only the origin ST agent implements this timeout. After correctly receiving the ACK to a CONNECT message, an ST agent expects to receive an ACCEPT or REFUSE message in response. If one of these messages is not received before the *ToConnectResp* timer expires, the origin ST agent aborts the connection setup attempt, acts as if a REFUSE message is received, and it sends a DISCONNECT message toward the targets. Both REFUSE and DISCONNECT must identify the affected targets and specify the ReasonCode (ResponseTimeout).

5.2.6 Failure due to DISCONNECT Acknowledgment Timeout

An ST agent that sends a DISCONNECT message downstream expects an ACK from the next-hop ST agent. If no ACK is received before the *ToDisconnect* timeout expires, the ST agent should retry and send the DISCONNECT message again. After *NDisconnect* unsuccessful retries, the ST agent simply gives up and it assumes the next-hop ST agent is not part in the stream any more.

5.2.7 Failure due to JOIN Acknowledgment Timeout

An ST agent that sends a JOIN message toward the origin expects an ACK from a neighbor ST agent. If no ACK is received before the *ToJoin* timeout expires, the ST agent should retry and send the JOIN message again. After *NJoin* unsuccessful retries, the ST agent sends a JOIN-REJECT message back in the direction of the target with ReasonCode (RetransTimeout).

5.2.8 Failure due to JOIN Response Timeout

Only the target agent implements this timeout. After correctly receiving the ACK to a JOIN message, the ST agent at the target expects to receive a CONNECT or JOIN-REJECT message in response. If one of these message is not received before the *ToJoinResp* timer expires, the ST agent aborts the stream join attempt and returns an error corresponding with ReasonCode (RetransTimeout) to the application.

Note that, after correctly receiving the ACK to a JOIN message, intermediate ST agents do not maintain any state on the stream joining attempt. As a consequence, they do not set the *ToJoinResp* timer and do not wait for a CONNECT or JOIN-REJECT message. This is described in Section 4.6.3.

5.2.9 Failure due to JOIN-REJECT Acknowledgment Timeout

An ST agent that sends a JOIN-REJECT message toward the target expects an ACK from a neighbor ST agent. If no ACK is received before the *ToJoinReject* timeout expires, the ST agent should retry and send the JOIN-REJECT message again. After *NJoinReject* unsuccessful retries, the ST agent simply gives up.

5.2.10 Failure due to NOTIFY Acknowledgment Timeout

An ST agent that sends a NOTIFY message to a neighbor ST agent expects an ACK from that neighbor ST agent. If no ACK is received before the *ToNotify* timeout expires, the ST agent should retry and send the NOTIFY message again. After *NNotify* unsuccessful retries, the ST agent simply gives up and behaves as if the ACK message was received.

5.2.11 Failure due to REFUSE Acknowledgment Timeout

An ST agent that sends a REFUSE message upstream expects an ACK from the previous-hop ST agent. If no ACK is received before the *ToRefuse* timeout expires, the ST agent should retry and send the REFUSE message again. After *NRefuse* unsuccessful retries, the ST agent gives up and it assumes it is not part in the stream any more.

5.2.12 Failure due to STATUS Response Timeout

After sending a STATUS message to a neighbor ST agent, an ST agent expects to receive a STATUS-RESPONSE message in response. If this message is not received before the *ToStatusResp* timer expires, the ST agent sends the STATUS message again. After *NStatus* unsuccessful retries, the ST agent gives up and assumes that the neighbor ST agent is not active.

5.3 Setup Failures due to Routing Failures

It is possible for an ST agent to receive a CONNECT message that contains a known SID, but from an ST agent other than the previous-hop ST agent of the stream with that SID. This may be:

1. that two branches of the tree forming the stream have joined back together,
2. the result of an attempted recovery of a partially failed stream, or
3. a routing loop.

The TargetList contained in the CONNECT is used to distinguish the different cases by comparing each newly received target with those of the previously existing stream:

- if the IP address of the target(s) differ, it is case #1;
- if the target matches a target in the existing stream, it may be case #2 or #3.

Case #1 is handled in Section 5.3.1, while the other cases are handled in Section 5.3.2.

5.3.1 Path Convergence

It is possible for an ST agent to receive a CONNECT message that contains a known SID, but from an ST agent other than the previous-hop ST agent of the stream with that SID. This might be the result of two branches of the tree forming the stream have joined back together. Detection of this case and other possible sources was discussed in Section 5.2.

SCMP does not allow for streams which have converged path, i.e streams are always tree-shaped and not graph-like. At the point of convergence, the ST agent which detects the condition generates a REFUSE message with ReasonCode (PathConvergence). Also, as a help to the upstream ST agent, the detecting agent places the IP address of one of the stream's connected targets in the ValidTargetIPAddress field of the REFUSE message. This IP address will be used by upstream ST agents to avoid splitting the stream.

An upstream ST agent that receives the REFUSE with ReasonCode (PathConvergence) will check to see if the listed IP address is one of the known stream targets. If it is not, the REFUSE is propagated to the previous-hop agent. If the listed IP address is known by the upstream ST agent, this ST agent is the ST agent that caused the split in the stream. (This agent may even be the origin.) This agent then avoids splitting the stream by using the next-hop of that known target as the next-hop for the refused targets. It sends a CONNECT with the affected targets to the existing valid next-hop.

The above process will proceed, hop by hop, until the ValidTargetIPAddress matches the IP address of a known target. The only case where this process will fail is when the known target is deleted prior to the REFUSE propagating to the origin. In this case the origin can just reissue the CONNECT and start the whole process over again.

5.3.2 Other Cases

The remaining cases including a partially failed stream and a routing loop, are not easily distinguishable. In attempting recovery of a failed stream, an ST agent may issue new CONNECT messages to the affected targets. Such a CONNECT may reach an ST agent downstream of the failure before that ST agent has received a DISCONNECT from the neighborhood of the failure. Until that ST agent receives the DISCONNECT, it cannot distinguish between a failure recovery and an erroneous routing loop. That ST agent must therefore respond to the CONNECT with a REFUSE message with the affected targets specified in the TargetList and an appropriate ReasonCode (StreamExists).

The ST agent immediately preceding that point, i.e., the latest ST agent to send the CONNECT message, will receive the REFUSE message. It must release any resources reserved exclusively for traffic to the listed targets. If this ST agent was not the one attempting the stream recovery, then it cannot distinguish between a failure recovery and an erroneous routing loop. It should repeat the CONNECT after a *ToConnect* timeout, see Section 5.2.4. If after *NConnect* retransmissions it continues to receive REFUSE messages, it should propagate the REFUSE message toward the origin, with the TargetList that specifies the affected targets, but with a different ReasonCode (RouteLoop).

The REFUSE message with this ReasonCode (RouteLoop) is propagated by each ST agent without retransmitting any CONNECT messages. At each ST agent, it causes any resources reserved exclusively for the listed targets to be released. The REFUSE will be propagated to the origin in the case of an erroneous routing loop. In the case of stream recovery, it will be propagated to the ST agent that is attempting the recovery, which may be an intermediate ST agent or the origin itself. In the case of a stream recovery, the ST agent attempting the recovery may issue new CONNECT messages to the same or to different next-hops.

If an ST agent receives both a REFUSE message and a DISCONNECT message with a target in common then it can, for the each target in common, release the relevant resources and propagate neither the REFUSE nor the DISCONNECT.

If the origin receives such a REFUSE message, it should attempt to send a new CONNECT to all the affected targets. Since routing errors in an internet are assumed to be temporary, the new CONNECTs will eventually find acceptable routes to the targets, if one exists. If no further routes exist after *NRetryRoute* tries, the application should be informed so that it may take whatever action it seems necessary.

5.4 Problems due to Routing Inconsistency

When an intermediate ST agent receives a CONNECT, it invokes the routing algorithm to select the next-hop ST agents based on the TargetList and the networks to which it is connected. If the resulting next-hop to any of the targets is across the same network from which it received the CONNECT (but not the previous-hop itself), there may be a routing problem. However, the routing algorithm at the previous-hop may be optimizing differently than the local algorithm would in the same situation. Since the local ST agent cannot distinguish the two cases, it should permit the setup but send back to the previous-hop ST agent an informative NOTIFY message with the appropriate ReasonCode (RouteBack), pertinent TargetList, and in the NextHopIPAddress element the address of the next-hop ST agent returned by its routing algorithm.

The ST agent that receives such a NOTIFY should ACK it. If the ST agent is using an algorithm that would produce such behaviour, no further action is taken; if not, the ST agent should send a DISCONNECT to the next-hop ST agent to correct the problem.

Alternatively, if the next-hop returned by the routing function is in fact the previous-hop, a routing inconsistency has been detected. In this case, a REFUSE is sent back to the previous-hop ST agent containing an appropriate ReasonCode (RouteInconsist), pertinent TargetList, and in the NextHopIPAddress element the address of the previous-hop. When the previous-hop receives the REFUSE, it will recompute the next-hop for the affected targets. If there is a difference in the routing databases in the two ST agents, they may exchange CONNECT and REFUSE messages again. Since such routing errors in the internet are assumed to be temporary, the situation should eventually stabilize.

5.5 Problems in Reserving Resources

As mentioned in Section 1.4.5, resource reservation is handled by the LRM. The LRM may not be able to satisfy a particular request during stream setup or modification for a number of reasons, including a mismatched FlowSpec, an unknown FlowSpec version, an error in processing a FlowSpec, and an inability to allocate the requested resource. This section discusses these cases and specifies the ReasonCodes that should be used when these error cases are encountered.

5.5.1 Mismatched FlowSpecs

In some cases the LRM may require a requested FlowSpec to match an existing FlowSpec, e.g. when adding new targets to an existing stream, see Section 4.6.1. In case of FlowSpec mismatch the LRM notifies the processing ST agent which should respond with ReasonCode (FlowSpecMismatch).

5.5.2 Unknown FlowSpec Version

When the LRM is invoked, it is passed information including the version of the FlowSpec, see Section 4.5.2.2. If this version is not known by the LRM, the LRM notifies the ST agent. The ST agent should respond with a REFUSE message with ReasonCode (FlowVerUnknown).

5.5.3 LRM Unable to Process FlowSpec

The LRM may encounter an LRM or FlowSpec specific error while attempting to satisfy a request. An example of such an error is given in Section 9.2.1. These errors are implementation specific and will not be enumerated with ST ReasonCodes. They are covered by a single, generic ReasonCode. When an LRM encounters such an error, it should notify the ST agent which should respond with the generic ReasonCode (FlowSpecError).

5.5.4 Insufficient Resources

If the LRM cannot make the necessary reservations because sufficient resources are not available, an ST agent may:

- *try alternative paths to the targets:* the ST agent calls the routing function to find a different path to the targets. If an alternative path is found, stream connection setup continues in the usual way, as described in Section 4.5.
- *refuse to establish the stream along this path:* the origin ST agent informs the application of the stream setup failure; intermediate and target ST agents issue a REFUSE message (as described in Section 4.5.8) with ReasonCode (CantGetResrc).

It depends on the local implementations whether an ST agent tries alternative paths or refuses to establish the stream. In any case, if enough resources cannot be found over different paths, the ST agent has to explicitly refuse to establish the stream.

5.6 Problems Caused by CHANGE Messages

A CHANGE might fail for several reasons, including:

- *insufficient resources*: the request may be for a larger amount of network resources when those resources are not available, ReasonCode (CantGetResrc);
- a *target application not agreeing* to the change, ReasonCode (ApplRefused);

The affected stream can be left in one of two states as a result of change failures: a) the stream can revert back to the state it was in prior to the CHANGE message being processed, or b) the stream may be torn down.

The expected common case of failure will be when the requested change cannot be satisfied, but the pre-change resources remain allocated and available for use by the stream. In this case, the ST agent at the point where the failure occurred must inform upstream ST agents of the failure. (In the case where this ST agent is the target, there may not actually be a failure, the application may merely have not agreed to the change). The ST agent informs upstream ST agents by sending a REFUSE message with ReasonCode (CantGetResrc or ApplRefused). To indicate that the pre-change FlowSpec is still available and that the stream still exists, the ST agent sets the E-bit of the REFUSE message to one (1), see Section 10.4.11. Upstream ST agents receiving the REFUSE message inform the LRM so that it can attempt to revert back to the pre-change FlowSpec. It is permissible, but not desirable, for excess resources to remain allocated.

For the case when the attempt to change the stream results in the loss of previously reserved resources, the stream is torn down. This can happen, for instance, when the I-bit is set (Section 4.6.5) and the LRM releases pre-change stream resources before the new ones are reserved, and neither new nor former resources are available. In this case, the ST agent where the failure occurs must inform other ST agents of the break in the affected portion of the stream. This is done by the ST agent by sending a REFUSE message upstream and a DISCONNECT message downstream, both with the ReasonCode (CantGetResrc). To indicate that pre-change stream resources have been lost, the E-bit of the REFUSE message is set to zero (0).

Note that a failure to change the resources requested for specific targets should not cause other targets in the stream to be deleted.

5.7 Unknown Targets in DISCONNECT and CHANGE

The handling of unknown targets listed in a DISCONNECT or CHANGE message is dependent on a stream's join authorization level, see Section 4.4.2. For streams with join authorization levels #0 and #1, see Section 4.4.2, all targets must be known. In this case, when processing a CHANGE message, the agent should generate a REFUSE message with ReasonCode (TargetUnknown). When processing a DISCONNECT message, it is possible that the DISCONNECT is a duplicate of an old request so the agent should respond as if it has successfully disconnected the target. That is, it should respond with an ACK message.

For streams with join authorization level #2, it is possible that the origin is not aware of some targets that participate in the stream. The origin may delete or change these targets via the following flooding mechanism.

If no next-hop ST agent can be associated with a target, the CHANGE/DISCONNECT message including the target is replicated to all known next-hop ST agents. This has the effect of propagating the CHANGE/DISCONNECT message to all downstream ST agents. Eventually, the ST agent that acts as the origin for the target (Section 4.6.3.1) is reached and the target is deleted.

Target deletion/change via flooding is not expected to be the normal case. It is included to present the applications with uniform capabilities for all stream types. Flooding only applies to streams with join authorization level #2.

6 Failure Detection and Recovery

6.1 Failure Detection

The SCMP failure detection mechanism is based on two assumptions:

1. If a neighbor of an ST agent is up, and has been up without a disruption, and has not notified the ST agent of a problem with streams that pass through both, then the ST agent can assume that there has not been any problem with those streams.
2. A network through which an ST agent has routed a stream will notify the ST agent if there is a problem that affects the stream data packets but does not affect the control packets.

The purpose of the robustness protocol defined here is for ST agents to determine that the streams through a neighbor have been broken by the failure of the neighbor or the intervening network. This protocol should detect the overwhelming majority of failures that can occur. Once a failure is detected, the recovery procedures described in Section 6.2 are initiated by the ST agents.

6.1.1 Network Failures

An ST agent can detect network failures by two mechanisms:

- the network can report a failure, or
- the ST agent can discover a failure by itself.

They differ in the amount of information that an ST agent has available to it in order to make a recovery decision. For example, a network may be able to report that reserved bandwidth has been lost and the reason for the loss and may also report that connectivity to the neighboring ST agent remains intact. On the other hand, an ST agent may discover that communication with a neighboring ST agent has ceased because it has not received any traffic from that neighbor in some time period. If an ST agent detects a failure, it may not be able to determine if the failure was in the network while the neighbor remains available, or the neighbor has failed while the network remains intact.

6.1.2 Detecting ST Agents Failures

Each ST agent periodically sends each neighbor *with which it shares one or more streams* a HELLO message. This message exchange is between ST agents, not entities representing streams or applications. That is, an ST agent need only send a single HELLO message to a neighbor regardless of the number of streams that flow between them. All ST agents (host as well as intermediate) must participate in this exchange. However, only ST agents that share active streams can participate in this exchange and it is an error to send a HELLO message to a neighbor ST agent with no streams in common, e.g. to check whether it is active. STATUS messages can be used to poll status of neighbor ST agents, see Section 8.4.

The HELLO message has two fields:

- a *HelloTimer* field that is in units of milliseconds modulo the maximum for the field size, and
- a *Restarted-bit* specifying that the ST agent has been restarted recently.

The *HelloTimer* must appear to be incremented every millisecond whether a HELLO message is sent or not. The *HelloTimer* wraps around to zero after reaching the maximum value. Whenever an ST agent suffers a catastrophic event that may result in it losing ST state information, it must reset its *HelloTimer* to zero and must set the *Restarted-bit* in all HELLO messages sent in the following *HelloTimerHoldDown* seconds.

If an ST agent receives a HELLO message that contains the *Restarted-bit* set, it must assume that the sending ST agent has lost its state. If it shares streams with that neighbor, it must initiate stream recovery activity, see Section 6.2. If it does not share streams with that neighbor, it should not attempt to create one until that bit is no longer set. If an ST agent receives a CONNECT message from a neighbor whose *Restarted-bit* is still set, it must respond with ERROR with the appropriate ReasonCode (RestartRemote). If it receives a CONNECT message while its own *Restarted-bit* is set, it must respond with ERROR with the appropriate ReasonCode (RestartLocal).

Each ST stream has an associated *RecoveryTimeout* value. This value is assigned by the origin and carried in the CONNECT message, see Section 4.5.10. Each agent checks to see if it can support the requested value. If it can not, it updates the value to the smallest timeout interval it can support. The *RecoveryTimeout* used by a particular stream is obtained from the ACCEPT message, see Section 4.5.10, and is the smallest value seen across all ACCEPT messages from participating targets.

An ST agent must send HELLO messages to its neighbor with a period shorter than the smallest *RecoveryTimeout* of all the active streams that pass between the two ST agents, regardless of direction. This period must be smaller by a factor, called *HelloLossFactor*, which is at least as large as the greatest number of consecutive HELLO messages that could credibly be lost while the communication between the two ST agents is still viable.

An ST agent may send simultaneous HELLO messages to all its neighbors at the rate necessary to support the smallest *RecoveryTimeout* of any active stream. Alternately, it may send HELLO messages to different neighbors independently at different rates corresponding to *RecoveryTimeouts* of individual streams.

The ST agent that receives a HELLO message expects to receive at least one new HELLO message from a neighbor during the *RecoveryTimeout* of every active stream through that neighbor. It can detect duplicate or delayed HELLO messages by saving the *HelloTimer* field of the most recent valid HELLO message from that neighbor and comparing it with the *HelloTimer* field of incoming HELLO messages. It will only accept an incoming HELLO message from that neighbor if it has a *HelloTimer* field that is greater than the most recent valid HELLO message by the time elapsed since that message was received plus twice the maximum likely delay variance from that neighbor.

If the ST agent does not receive a valid HELLO message within the *RecoveryTimeout* of a stream, it must assume that the neighboring ST agent or the communication link between the two has failed and it must initiate stream recovery activity, as described below in Section 6.2.

6.2 Failure Recovery

If an intermediate ST agent fails or a network or part of a network fails, the previous-hop ST agent and the various next-hop ST agents will discover the fact by the failure detection mechanism described in Section 6.1.

The recovery of an ST stream is a relatively complex and time consuming effort because it is designed in a general manner to operate across a large number of networks with diverse characteristics. Therefore, it may require information to be distributed widely, and may require relatively long timers. On the other hand, since a network is typically a homogeneous system, failure recovery in the network may be a relatively faster and simpler operation. Therefore an ST agent that detects a failure should attempt to fix the network failure before attempting recovery of the ST stream. If the stream that existed between two ST agents before the failure cannot be reconstructed by network recovery mechanisms alone, then the ST stream recovery mechanism must be invoked.

If stream recovery is necessary, the different ST agents will need to perform different functions, depending on their relation to the failure:

- An **ST agent that is a next-hop of a failure** should first verify that there was a failure. It can do this using STATUS messages to query its upstream neighbor. If it cannot communicate with that neighbor, then for each active stream from that neighbor it should first send a REFUSE message upstream with the appropriate ReasonCode (STAgentFailure). This is done to the neighbor to speed up the failure recovery in case the hop is unidirectional, i.e., the neighbor can hear the ST agent but the ST agent cannot hear the neighbor. The ST agent detecting the failure must then, for each active stream from that neighbor, send DISCONNECT messages with the same ReasonCode toward the targets. All downstream ST agents process this DISCONNECT message just like the DISCONNECT that tears down the stream. If recovery is successful, targets will receive new CONNECT messages.
- An **ST agent that is the previous-hop before the failed component** first verifies that there was a failure by querying the downstream neighbor using STATUS messages. If the neighbor has lost its state but is available, then the ST agent may try and reconstruct (explained below) the affected streams, for those streams that do not have the NoRecovery option selected. If it cannot communicate with the next-hop, then the ST agent detecting the failure sends a DISCONNECT message, for each affected stream, with the appropriate ReasonCode (STAgentFailure) toward the affected targets. It does so to speed up failure recovery in case the communication may be unidirectional and this message might be delivered successfully.

Based on the NoRecovery option, the ST agent that is the previous-hop before the failed component takes the following actions:

- If the **NoRecovery option is selected**, then the ST agent sends, per affected stream, a REFUSE message with the appropriate ReasonCode (STAgentFailure) to the previous-hop. The TargetList in these messages contains all the targets that were reached through the broken branch. As discussed in Section 5.1.2, multiple REFUSE messages may be required if the PDU is too long for the MTU of the intervening network. The REFUSE message is propagated all the way to the origin. The application at the origin can attempt recovery of the stream by sending a new CONNECT to the affected targets. For established streams, the new CONNECT will be treated by intermediate ST agents as an addition of new targets into the established stream.
- If the **NoRecovery option is not selected**, the ST agent can attempt recovery of the affected streams. It does so one a stream by stream basis by issuing a new CONNECT message to the affected targets. If the ST agent cannot find new routes to some targets, or if the only route to

some targets is through the previous-hop, then it sends one or more REFUSE messages to the previous-hop with the appropriate ReasonCode (CantRecover) specifying the affected targets in the TargetList. The previous-hop can then attempt recovery of the stream by issuing a CONNECT to those targets. If it cannot find an appropriate route, it will propagate the REFUSE message toward the origin.

Regardless of which ST agent attempts recovery of a damaged stream, it will issue one or more CONNECT messages to the affected targets. These CONNECT messages are treated by intermediate ST agents as additions of new targets into the established stream. The FlowSpecs of the new CONNECT messages are the same as the ones contained in the most recent CONNECT or CHANGE messages that the ST agent had sent toward the affected targets when the stream was operational.

Upon receiving an ACCEPT during the a stream recovery, the agent reconstructing the stream must ensure that the FlowSpec and other stream attributes (e.g. MaxMsgSize and RecoveryTimeout) of the re-established stream are equal to, or are less restrictive, than the pre-failure stream. If they are more restrictive, the recovery attempt must be aborted. If they are equal, or are less restrictive, then the recovery attempt is successful. When the attempt is a success, failure recovery related ACCEPTs are not forwarded upstream by the recovering agent.

Any ST agent that decides that enough recovery attempts have been made, or that recovery attempts have no chance of succeeding, may indicate that no further attempts at recovery should be made. This is done by setting the N-bit in the REFUSE message, see Section 10.4.11. This bit must be set by agents, including the target, that know that there is no chance of recovery succeeding. An ST agent that receives a REFUSE message with the N-bit set (1) will not attempt recovery, regardless of the NoRecovery option, and it will set the N-bit when propagating the REFUSE message upstream.

6.2.1 Problems in Stream Recovery

The reconstruction of a broken stream may not proceed smoothly. Since there may be some delay while the information concerning the failure is propagated throughout an internet, routing errors may occur for some time after a failure. As a result, the ST agent attempting the recovery may receive ERROR messages for the new CONNECTs that are caused by internet routing errors. The ST agent attempting the recovery should be prepared to resend CONNECTs before it succeeds in reconstructing the stream. If the failure partitions the internet and a new set of routes cannot be found to the targets, the REFUSE messages will eventually be propagated to the origin, which can then inform the application so it can decide whether to terminate or to continue to attempt recovery of the stream.

The new CONNECT may at some point reach an ST agent downstream of the failure before the DISCONNECT does. In this case, the ST agent that receives the CONNECT is not yet aware that the stream has suffered a failure, and will interpret the new CONNECT as resulting from a routing failure. It will respond with an ERROR message with the appropriate ReasonCode (StreamExists). Since the timeout that the ST agents immediately preceding the failure and immediately following the failure are approximately the same, it is very likely that the remnants of the broken stream will soon be torn down by a DISCONNECT message. Therefore, the ST agent that receives the ERROR message with ReasonCode (StreamExists) should retransmit the CONNECT message after the *ToConnect* timeout expires. If this fails again, the request will be retried for *NConnect* times. Only if it still fails will the ST agent send a REFUSE message with the appropriate ReasonCode (RouteLoop) to its previous-hop. This message will be propagated back to the ST agent that is attempting recovery of the damaged stream. That ST agent can issue a new CONNECT message if it so chooses. The REFUSE is matched to a CONNECT message created by a recovery operation through the LnkReference field in the CONNECT.

ST agents that have propagated a CONNECT message and have received a REFUSE message should maintain this information for some period of time. If an ST agent receives a second CONNECT message for a target that recently resulted in a REFUSE, that ST agent may respond with a REFUSE immediately rather than attempting to propagate the CONNECT. This has the effect of pruning the tree that is formed by the propagation of CONNECT messages to a target that is not reachable by the routes that are selected first. The tree will pass through any given ST agent only once, and the stream setup phase will be completed faster.

If a CONNECT message reaches a target, the target should as efficiently as possible use the state that it has saved from before the stream failed during recovery of the stream. It will then issue an ACCEPT message toward the origin. The ACCEPT message will be intercepted by the ST agent that is attempting recovery of the damaged stream, if not the origin. If the FlowSpec contained in the ACCEPT specifies the same selection of parameters as were in effect before the failure, then the ST agent that is attempting recovery will not propagate the ACCEPT. FlowSpec comparison is done by the LRM. If the selections of the parameters are different, then the ST agent that is attempting recovery will send the origin a NOTIFY message with the appropriate ReasonCode (FailureRecovery) that contains a FlowSpec that specifies the new parameter values. The origin may then have to change its data generation characteristics and the stream's parameters with a CHANGE message to use the newly recovered subtree.

6.3 Stream Preemption

As mentioned in Section 1.4.5, it is possible that the LRM decides to break a stream intentionally. This is called stream preemption. Streams are expected to be preempted in order to free resources for a new stream which has a higher priority.

If the LRM decides that it is necessary to preempt one or more of the stream traversing it, the decision on which streams have to be preempted has to be made. There are two ways for an application to influence such decision:

1. *based on FlowSpec information*. For instance, with the ST2+ FlowSpec, streams can be assigned a *precedence* value from 0 (least important) to 256 (most important). This value is carried in the FlowSpec when the stream is setup, see Section 9.2, so that the LRM is informed about it.
2. *with the group mechanism*. An application may specify that a set of streams are related to each other and that they are all candidate for preemption if one of them gets preempted. It can be done by using the *fate-sharing* relationship defined in Section 7.1.2. This helps the LRM making a good choice when more than one stream have to be preempted, because it leads to breaking a single application as opposed to as many applications as the number of preempted streams.

If the LRM preempts a stream, it must notify the local ST agent. The following actions are performed by the ST agent:

- The **ST agent at the host where the stream was preempted** sends DISCONNECT messages with the appropriate ReasonCode (StreamPreempted) toward the affected targets. It sends a REFUSE message with the appropriate ReasonCode (StreamPreempted) to the previous-hop.
- A **previous-hop ST agent of the preempted stream** acts as in case of failure recovery, see Section 6.2.
- A **next-hop ST agent of the preempted stream** acts as in case of failure recovery, see Section 6.2.

Note that, as opposite to failure recovery, there is no need to verify that the failure actually occurred, because this is explicitly indicated by the ReasonCode (StreamPreempted).

7 A Group of Streams

There may be need to associate related streams. The *group* mechanism is simply an association technique that allows ST agents to identify the different streams that are to be associated.

A group consists of a *set of streams* and a *relationship*. The set of streams may be empty. The relationship applies to all group members. Each group is identified by a *group name*. The group name must be globally unique.

Streams belong to the same group if they have the same *GroupName* in the *GroupName* field of the *Group* parameter, see Section 10.3.2. The relationship is defined by the *Relationship* field. Group membership must be specified at stream creation time and persists for the whole stream lifetime. A single stream may belong to multiple groups.

The ST agent that creates a new group is called *group initiator*. Any ST agent can be a group initiator. The initiator allocates the *GroupName* and the *Relationship* among group members. The initiator may or may not be the origin of a stream belonging to the group. GroupName generation is described in Section 8.2.

7.1 Basic Group Relationships

This version of ST defines four basic group relationships. An ST2+ implementation must support all four basic relationships. Adherence to specified relationships are usually best effort. The basic relationships are described in detail below in Section 7.1.1 - Section 7.1.4.

7.1.1 Bandwidth Sharing

Streams associated with the same group share the same network bandwidth. The intent is to support applications such as audio conferences where, of all participants, only some are allowed to speak at one time. In such a scenario, global bandwidth utilization can be lowered by allocating only those resources that can be used at once, e.g. it is sufficient to reserve bandwidth for a small set of audio streams.

The basic concept of a shared bandwidth group is that the LRM will allocate up to some specified multiplier of the most demanding stream that it knows about in the group. The LRM will allocate resources incrementally, as stream setup requests are received, until the total group requirements are satisfied. Subsequent setup requests will share the group's resources and will not need any additional resources allocated. The procedure will result in standard allocation where only one stream in a group traverses an agent, and shared allocations where multiple streams traverse an agent.

To illustrate, let's call the multiplier mentioned above "N", and the most demanding stream that an agent knows about in a group B_{max}. For an application that intends to allow three participants to speak at the same time, N has a value of three and each LRM will allocate for the group an amount of bandwidth up to $3*B_{max}$ even when there are many more steams in the group. The LRM will reserve resources incrementally, per stream request, until $N*B_{max}$ resources are allocated. Each agent may be traversed by a different set and number of streams all belonging to the same group.

An ST agent receiving a stream request presents the LRM with all necessary group information, see Section 4.5.2.2. If maximum bandwidth, $N*B_{max}$, for the group has already been allocated and a new stream with a bandwidth demand less than B_{max} is being established, the LRM won't allocate any further bandwidth.

If there is less than $N*B_{max}$ resources allocated, the LRM will expand the resources allocated to the group by the amount requested in the new FlowSpec, up to N*Bmax resources. The LRM will update the FlowSpec based on what resources are available to the stream, but not the total resources allocated for the group.

It should be noted that ST agents and LRMs become aware of a group's requirements only when the streams belonging to the group are created. In case of the bandwidth sharing relationship, an application should attempt to establish the most demanding streams first to minimize stream setup efforts. If on the contrary the less demanding streams are built first, it will be always necessary to allocate additional bandwidth in consecutive steps as the most demanding streams are built. It is also up to the applications to coordinate their different FlowSpecs and decide upon an appropriate value for N.

7.1.2 Fate Sharing

Streams belonging to this group share the same fate. If a stream is deleted, the other members of the group are also deleted. This is intended to support stream preemption by indicating which streams are mutually related. If preemption of multiple streams is necessary, this information can be used by the LRM to delete a set of related streams, e.g. with impact on a single application, instead of making a random choice with the possible effect of interrupting several different applications. This attribute does not apply to normal stream shut down, i.e. ReasonCode (ApplDisconnect). On normal disconnect, other streams belonging to such groups remain active.

This relationship provides a hint on which streams should be preempted. Still, the LRM responsible for the preemption is not forced to behave accordingly, and other streams could be preempted first based on different criteria.

7.1.3 Route Sharing

Streams belonging to this group share the same paths as much as is possible. This can be desirable for several reasons, e.g. to exploit the same allocated resources or in the attempt to maintain the transmission order. An ST agent attempts to select the same path although the way this is implemented depends heavily on the routing algorithm which is used.

If the routing algorithm is sophisticated enough, an ST agent can suggest that a stream is routed over an already established path. Otherwise, it can ask the routing algorithm for a set of legal routes to the destination and check whether the desired path is included in those feasible.

Route sharing is a hint to the routing algorithm used by ST. Failing to route a stream through a shared path should not prevent the creation of a new stream or result in the deletion of an existing stream.

7.1.4 Subnet Resources Sharing

This relationship provides a hint to the data link layer functions. Streams belonging to this group may share the same MAC layer resources. As an example, the same MAC layer multicast address may be used for all the streams in a given group. This mechanism allows for a better utilization of MAC layer multicast addresses and it is especially useful when used with network adapters that offer a very small number of MAC layer multicast addresses.

7.2 Relationships Orthogonality

The four basic relationships, as they have been defined, are *orthogonal*. This means, any combinations of the basic relationships are allowed. For instance, let's consider an application that requires *full-duplex service* for a stream with multiple targets. Also, let's suppose that only N targets are allowed to send data back to the origin at the same time. In this scenario, all the reverse streams could belong to the same group. They could be sharing both the paths and the bandwidth attributes. The Path&Bandwidth sharing relationship is obtained from the basic set of relationships. This example is important because it shows how full-duplex service can be efficiently obtained in ST.

8 Ancillary Functions

Certain functions are required by ST host and intermediate agent implementations. Such functions are described in this section.

8.1 Stream ID Generation

The stream ID, or SID, is composed of 16-bit unique identifier and the stream origin's 32-bit IP address. Stream IDs must be globally unique. The specific definition and format of the 16 -bit field is left to the implementor. This field is expected to have only local significance.

An ST implementation has to provide a stream ID generator facility, so that an application or higher layer protocol can obtain a unique IDs from the ST layer. This is a mechanism for the application to request the allocation of stream ID that is independent of the request to create a stream. The Stream ID is used by the application or higher layer protocol when creating the streams.

For instance, the following two functions could be made available:

- AllocateStreamID() -> result, *StreamID*
- ReleaseStreamID(*StreamID*) -> result

An implementation may also provide a StreamID deletion function.

8.2 Group Name Generator

GroupName generation is similar to Stream ID generation. The *GroupName* includes a 16-bit unique identifier, a 32-bit creation timestamp, and a 32-bit IP address. Group names are globally unique. A GroupName includes the creator's IP address, so this reduces a global uniqueness problem to a simple local problem. The specific definitions and formats of the 16-bit field and the 32-bit creation timestamp are left to the implementor. These fields must be locally unique, and only have local significance.

An ST implementation has to provide a group name generator facility, so that an application or higher layer protocol can obtain a unique *GroupName* from the ST layer. This is a mechanism for the application to request the allocation of a *GroupName* that is independent of the request to create a stream. The *GroupName* is used by the application or higher layer protocol when creating the streams that are to be part of the group.

For instance, the following two functions could be made available:

- AllocateGroupName() -> result, *GroupName*
- ReleaseGroupName(*GroupName*) -> result

An implementation may also provide a GroupName deletion function.

8.3 Checksum Computation

The standard Internet checksum algorithm is used for ST: "The checksum field is the 16-bit one's complement of the one's complement sum of all 16-bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero (0)." See [RFC1071], [RFC1141], and [RFC791] for suggestions for efficient checksum algorithms.

8.4 Neighbor ST Agent Identification and Information Collection

The STATUS message can be used to collect information about neighbor ST agents, streams the neighbor supports, and specific targets of streams the neighbor supports. An agent receiving a STATUS message provides the requested information via a STATUS-RESPONSE message.

The STATUS message can be used to collect different information from a neighbor. It can be used to:

- **identifying ST capable neighbors**. If an ST agent wishes to check if a neighbor is ST capable, it should generate a STATUS message with an SID which has all its fields set to zero. An agent receiving a STATUS message with such SID should answer with a STATUS-RESPONSE containing the same SID, and no other stream information. The receiving ST agent must answer as soon as possible to aid in Round Trip Time estimation, see Section 8.5;
- **obtain information on a particular stream.** If an ST agent wishes to check a neighbor's general information related to a specific stream, it should generate a STATUS message containing the stream's SID. An ST agent receiving such a message, will first check to see if the stream is known. If not known, the receiving ST agent sends a STATUS-RESPONSE containing the same SID, and no other stream information. If the stream is known, the receiving ST agent sends a STATUS-RESPONSE containing the stream's SID, IPHops, FlowSpec, group membership (if any), and as many targets as can be included in a single message as

limited by MTU, see Section 5.1.2. Note that all targets may not be included in a response to a request for general stream information. If information on a specific target in a stream is desired, the mechanism described next should be used.

- **obtain information on particular targets in a stream.** If an ST agent wishes to check a neighbor's information related to on or more specific targets of a specific stream, it should generate a STATUS message containing the stream's SID and a TargetList parameter listing the relevant targets. An ST agent receiving such a message, will first check to see if the stream and target are known. If the stream is not known, the agent follows the process described above. If both the stream and targets are known, the agent responds with STATUS-RESPONSE containing the stream's SID, IPHops, FlowSpec, group membership (if any), and the requested targets that are known. If the stream is known but the target is not, the agent responds with a STATUS-RESPONSE containing the stream's SID, IPHops, FlowSpec, group membership (if any), but no targets.

The specific formats for STATUS and STATUS-RESPONSE messages are defined in Section 10.4.12 and Section 10.4.13.

8.5 Round Trip Time Estimation

SCMP is made reliable through use of retransmission when an expected acknowledgment is not received in a timely manner. Timeout and retransmission algorithm is implementation dependent and it is outside the scope of this document. However, it must be reasonable enough not to cause excessive retransmission of SCMP message while maintaining the robustness of the protocol. Algorithms on this subject are described in [WoHD95], [Jaco88], [KaPa87].

Most existing algorithms are based on an estimation of the Round Trip Time (RTT) between two hosts. With SCMP, if an ST agent wishes to have an estimate of the RTT to and from a neighbor, it should generate a STATUS message with an SID which has all its fields set to zero. An ST agent receiving a STATUS message with such SID should answer as rapidly as possible with a STATUS-RESPONSE message containing the same SID, and no other stream information. The time interval between the send and receive operations can be used as an estimate of the RTT to and from the neighbor.

8.6 Network MTU Discovery

At connection setup, the application at the origin asks the local ST agent to create streams with certain QoS requirements. The local ST agent fills out its network MTU value in the MaxMsgSize parameter in the CONNECT message and forwards it to the next-hop ST agents. Each ST agent in the path checks to see if it's network MTU is smaller than the one specified in the CONNECT message and, if it is, the ST agent updates the MaxMsgSize in the CONNECT message to it's network MTU. If the target application decides to accept the stream, the ST agent at the target copies the MTU value in the CONNECT message to appropriate field in the ACCEPT message and sends it back to the application at the origin. The MaxMsgSize field in the ACCEPT message is the minimum MTU of network to that target. If the application has multiple targets then the minimum MTU of the stream is the smallest MaxMsgSize received from all the ACCEPT messages. It is the responsibility of the application to segment its PDUs according to the minimum MaxMsgSize of the stream since no data fragmentation is supported during the data transfer phase. If a particular target's MaxMsgSize is unacceptable to an application, it may disconnect the target from the stream and assume that the target cannot be supported. When evaluating a particular target's MaxMsgSize, the application or the application interface will need to take into account the size of the ST data header.

8.7 IP Encapsulation of ST

ST packets may be encapsulated in IP to allow them to pass through routers that don't support the ST Protocol. Of course, ST resource management is precluded over such a path, and packet overhead is increased by encapsulation, but if the performance is reasonably predictable this may be better than not communicating at all.

IP-encapsulated ST packets begin with a normal IP header. Most fields of the IP header should be filled in according to the same rules that apply to any other IP packet. Three fields of special interest are:

- Protocol is 5, see [RFC1700], to indicate an ST packet is enclosed , as opposed to TCP or UDP, for example.
- Destination Address is that of the next-hop ST agent. This may or may not be the target of the ST stream. There may be an intermediate ST agent to which the packet should be routed to take advantage of service guarantees on the path past that agent. Such an intermediate agent would not be on a directly-connected network (or else IP encapsulation wouldn't be needed), so it would probably not be listed in the normal routing table. Additional routing mechanisms, not defined here, will be required to learn about such agents.
- Type-of-Service may be set to an appropriate value for the service being requested, see [RFC1700]. This feature is not implemented uniformly in the Internet, so its use can't be precisely defined here.

IP encapsulation adds little difficulty for the ST agent that receives the packet. However, when IP encapsulation is performed it must be done in both directions. To process the encapsulated IP message, the ST agents simply remove the IP header and proceed with ST header as usual.

The more difficult part is during setup, when the ST agent must decide whether or not to encapsulate. If the next-hop ST agent is on a remote network and the route to that network is through a router that supports IP but not ST, then encapsulation is required. The routing function provides ST agents with the route and capability information needed to support encapsulation.

On forwarding, the (mostly constant) IP Header must be inserted and the IP checksum appropriately updated.

Applications are informed about the number of IP hops traversed on the way to the targets. The IPHops field value of the CONNECT message, see Section 10.4.4, is incremented at this purpose in case an ST agent uses IP encapsulation to reach its next-hop. The value is then returned to the origin in the IPHops field of the ACCEPT message, Section 10.4.1.

When using IP Encapsulation, the MaxMsgSize field will not reflect the MTU of the IP encapsulated segments. This means that IP fragmentation and reassembly may be needed in the IP cloud to support a message of MaxMsgSize. IP fragmentation can only occur when the MTU of the IP cloud, less IP header length, is the smallest MTU in a stream's network path.

8.8 IP Multicasting

If an ST agent must use IP encapsulation to reach multiple next-hops toward different targets, then either the packet must be replicated for transmission to each next-hop, or IP multicasting may be used if it is implemented in the next-hop ST agents and in the intervening IP routers.

When the stream is established, the collection of next-hop ST agents must be set up as an IP multicast group. The ST agent must allocate appropriate IP multicast address (see Section 10.3.3) and fill that address in the IPMulticastAddress field of the CONNECT message. The IP multicast address in the CONNECT message is used to inform the next-hop ST agents that they should join the multicast group to receive subsequent PDUs. Obviously, the CONNECT message itself must be sent using unicast. The next-hop ST agents must be able to receive on the specified multicast address in order to accept the connection.

If the next-hop ST agent can not receive on the specified multicast address, it sends a REFUSE message with ReasonCode (BadMcastAddress). Upon receiving the REFUSE, the upstream agent can choose to retry with a different multicast address. Alternatively, it can choose to loose the efficiency of multicast and use unicast delivery.

The following permanent IP multicast addresses have been assigned to ST:

224.0.0.7 All ST routers (intermediate agents)

224.0.0.8 All ST hosts (agents)

In addition, a block of transient IP multicast addresses, 224.1.0.0 - 224.1.255.255, has been allocated for ST multicast groups. For instance, the following two functions could be made available:

- AllocateMcastAddr() -> result, *McastAddr*
- ListenMcastAddr(McastAddr) -> result
- ReleaseMcastAddr(*McastAddr*) -> result

9 The ST2+ Flow Specification

This section defines the ST2+ flow specification. The flow specification contains the user application requirements in terms of quality of service. Its contents are LRM dependent and are transparent to the ST2 setup protocol. ST2 carries the flow specification as part of the FlowSpec parameter, which is described in Section 10.3.1. The required ST2+ flow specification is included in the protocol only to support interoperability. ST2+ also defines a “null” flow specification to be used only to support testing.

ST2 is not dependent on a particular flow specification format and it is expected that other versions of the flow specification will be needed in the future. Different flow specification formats are distinguished by the value of the Version field of the FlowSpec parameter, see Section 10.3.1. A single stream is always associated with a single flow specification format, i.e. the Version field is consistent throughout the whole stream. The following Version field values are defined:

0 - Null FlowSpec **/* must be supported */**
1 - ST Version 1
2 - ST Version 1.5
3 - RFC 1190 FlowSpec
4 - HeiTS FlowSpec
5 - BerKom FlowSpec
6 - RFC 1363 FlowSpec
7 - ST2+ FlowSpec **/* must be supported */**

FlowSpecs version #0 and #7 must be supported by ST2+ implementations. Version numbers in the range 1-6 indicate flow specifications are currently used in existing ST2 implementations. Values in the 128-255 range are reserved for private and experimental use.

9.1 FlowSpec Version #0 - (Null FlowSpec)

The flow specification identified by a #0 value of the Version field is called the **Null FlowSpec**. This flow specification causes no resources to be allocated. It is ignored by the LRMs. Its contents are never updated. Stream setup takes place in the usual way leading to successful stream establishment, but no resources are actually reserved.

The purpose of the Null FlowSpec is that of facilitating interoperability tests by allowing streams to be built without actually allocating the correspondent amount of resources. The Null FlowSpec may also be used for testing and debugging purposes.

The Null FlowSpec comprises the 4-byte FlowSpec parameter only, see Section 10.3.1. The third byte (Version field) must be set to 0.

9.2 FlowSpec Version #7 - ST2+ FlowSpec

The flow specification identified by a #7 value of the Version field is the ST2+ FlowSpec, to be used by all ST2+ implementations. It allows the user applications to express their real-time requirements in the form of a *QoS class*, *precedence*, and three basic *QoS parameters*:

- message size,
- message rate,
- end-to-end delay.

The QoS class indicates what kind of QoS guarantees are expected by the application, e.g. strict guarantees or predictive, see Section 9.2.1. QoS parameters are expressed via a set of values:

- the “*desired*” values indicate the QoS desired by the application. These values are assigned by the application and never modified by the LRM.
- the “*limit*” values indicate the lowest QoS the application is willing to accept. These values are also assigned by the application and never modified by the LRM.
- the “*actual*” values indicate the QoS that the system is able to provide. They are updated by the LRM at each node. The “actual” values are always bounded by the “limit” and “desired” values.

9.2.1 QoS Classes

Two QoS classes are defined:

1 - QOS_PREDICTIVE **/* QoSClass field value = 0x01, must be supported*/**
2 - QOS_GUARANTEED /* QoSClass field value = 0x10, optional */

- The QOS_PREDICTIVE class implies that the negotiated QoS may be violated for short time intervals during the data transfer. An application has to provide values that take into account the average case, e.g. the "desired" message rate is the average rate for the transmission. Reservations are done for the average case as opposite to the peak case required by the QOS_GUARANTEED service class. This QoS class must be supported by all implementations.
- The QOS_GUARANTEED class implies that the negotiated QoS for the stream is never violated during the data transfer. An application has to provide values that take into account the worst possible case, e.g. the "desired" message rate is the peak rate for the transmission. As a result, sufficient resources to handle the peak rate are reserved. This strategy may lead to overbooking of resources, but it provides strict real-time guarantees. Support of this QoS class is optional.

If a LRM that doesn't support class QOS_GUARANTEED receives a FlowSpec containing QOS_GUARANTEED class, it informs the local ST agent. The ST agent may try different paths or delete the correspondent portion of the stream as described in Section 5.5.3, i.e. ReasonCode (FlowSpecError).

9.2.2 Precedence

Precedence is the importance of the connection being established. Zero represents the lowest precedence. The lowest level is expected to be used by default. In general, the distinction between precedence and priority is that precedence specifies streams that are permitted to take previously committed resources from another stream, while priority identifies those PDUs that a stream is most willing to have dropped.

9.2.3 Maximum Data Size

This parameter is expressed in bytes. It represents the maximum amount of data, excluding ST and other headers, allowed to be sent in a messages as part of the stream. The LRM first checks whether it is possible to get the value desired by the application (DesMaxSize). If not, it updates the actual value (ActMaxSize) with the available size unless this value is inferior to the minimum allowed by the application (LimitMaxSize), in which case it informs the local ST agent that it is not possible to build the stream along this path.

9.2.4 Message Rate

This parameter is expressed in messages/second. It represents the transmission rate for the stream. The LRM first checks whether it is possible to get the value desired by the application (DesRate). If not, it updates the actual value (ActRate) with the available rate unless this value is inferior to the minimum allowed by the application (LimitRate), in which case it informs the local ST agent that it is not possible to build the stream along this path.

9.2.5 Delay and Delay Jitter

The delay parameter is expressed in milliseconds. It represents the maximum end-to-end delay for the stream. The LRM first checks whether it is possible to get the value desired by the application (DesMaxDelay). If not, it updates the actual value (ActMaxDelay) with the available delay unless this value is greater than the maximum delay allowed by the application (LimitMaxDelay), in which case it informs the local ST agent that it is not possible to build the stream along this path.

The LRM also updates at each node the MinDelay field by incrementing it by the minimum possible delay to the next-hop. Information on the minimum possible delay allows to calculate the maximum end-to-end delay range, i.e. the time interval in which a data packet can be received. This interval should not exceed the DesMaxDelayRange value indicated by the application. The maximum end-to-end delay range is an upper bound to the delay jitter.

9.2.6 ST2+ FlowSpec Format

The ST2+ FlowSpec has the following format:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    QosClass   |   Precedence  |           0(unused)           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             DesRate                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            LimitRate                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             ActRate                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           DesMaxSize          |          LimitMaxSize         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           ActMaxSize          |          DesMaxDelay          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           LimitMaxDelay       |          ActMaxDelay          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           DesMaxDelayRange    |          ActMinDelay          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 9: The ST2+ FlowSpec.

The LRM modifies only “actual” fields, i.e. those beginning with “Act”. The user application assigns values to all other fields.

- **QoSClass** indicates which of the two defined classes of service applies. The two classes are: QOS_PREDICTIVE (QoSClass = 1) and QOS_GUARANTEED (QoSClass = 2).
- **Precedence** indicates the stream’s precedence. Zero represents the lowest precedence, and should be the default value.
- **DesRate** is the desired transmission rate for the stream in messages/second. This field is set by the origin and is not modified by intermediate agents.
- **LimitRate** is the minimum acceptable transmission rate in messages/second. This field is set by the origin and is not modified by intermediate agents.

- **ActRate** is the actual transmission rate allocated for the stream in messages/second. Each agent updates this field with the available rate unless this value is less than LimitRate, in which case a REFUSE is generated.
- **DesMaxSize** is the desired maximum data size in bytes that will be sent in a message in the stream. This field is set by the origin.
- **LimitMaxSize** is the minimum acceptable data size in bytes. This field is set by the origin
- **ActMaxSize** is the actual maximum data size that may be sent in a message in the stream. This field is updated by each agent based on MTU and available resources. If available maximum size is less than LimitMaxSize, the connection must be refused with ReasonCode (CantGetResrc).
- **DesMaxDelay** is the desired maximum end-to-end delay for the stream in milliseconds. This field is set by the origin.
- **LimitMaxDelay** is the upper-bound of acceptable end-to-end delay for the stream in milliseconds. This field is set by the origin.
- **ActMaxDelay** is the maximum end-to-end delay that will be seen by data in the stream. Each ST agent adds to this field the maximum delay that will be introduced by the agent, including transmission time to the next-hop ST agent. If the actual maximum exceeds LimitMaxDelay, then the connection is refused with ReasonCode (CantGetResrc).
- **DesMaxDelayRange** is the desired maximum delay range that may be encountered end-to-end by stream data in milliseconds. This value is set by the application at the origin.
- **ActMinDelay** is the actual minimum end-to-end delay that will be encountered by stream data in milliseconds. Each ST agent adds to this field the minimum delay that will be introduced by the agent, including transmission time to the next-hop ST agent. Each agent must add at least 1 millisecond. The delay range for the stream can be calculated from the actual maximum and minimum delay fields. It is expected that the range will be important to some applications.

10 ST2 Protocol Data Units Specification

10.1 Data PDU

IP and ST packets can be distinguished by the IP Version Number field, i.e. the first four (4) bits of the packet; ST has been assigned the value 5 (see [RFC1700]). There is no requirement for compatibility between IP and ST packet headers beyond the first four bits. (IP uses value 4.)

The ST PDUs sent between ST agents consist of an ST Header encapsulating either a higher layer PDU or an ST Control Message. Data packets are distinguished from control messages via the D-bit (bit 8) in the ST header.

The ST Header also includes an ST Version Number, a total length field, a header checksum, a unique id, and the stream origin 32-bit IP address. The unique id and the stream origin 32-bit IP address form the stream id (SID). This is shown in Figure 10. Please refer to Section 10.6 for an explanation of the notation.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  ST=5 | Ver=3 |D| Pri |   0   |           TotalBytes          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         HeaderChecksum        |            UniqueID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        OriginIPAddress                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 10: ST Header

- **ST** is the IP Version Number assigned to identify ST packets. The value for ST is 5.
- **Ver** is the ST Version Number. The value for the current ST2+ version is 3.
- **D** (bit 8) is set to 1 in all ST data packets and to 0 in all SCMP control messages.
- **Pri** (bits 9-11) is the packet-drop priority field with zero (0) being lowest priority and seven the highest. The field is to be used as described in Section 3.2.2.
- **TotalBytes** is the length, in bytes, of the entire ST packet, it includes the ST Header but does not include any local network headers or trailers. In general, all length fields in the ST Protocol are in units of bytes.
- **HeaderChecksum** covers only the ST Header (12 bytes). The ST Protocol uses 16-bit checksums here in the ST Header and in each Control Message. For checksum computation, see Section 8.3.
- **UniqueID** is the first element of the stream ID (SID). It is locally unique at the stream origin, see Section 8.1.
- **OriginIPAddress** is the second element of the SID. It is the 32-bit IP address of the stream origin, see Section 8.1.

Bits 12-15 must be set to zero (0) in the current ST version and are reserved for future use, e.g., as described in [WoHD95].

10.1.1 ST Data Packets

ST packets whose D-bit is non-zero are data packets. Their interpretation is a matter for the higher layer protocols and consequently is not specified here. The data packets are not protected by an ST checksum and will be delivered to the higher layer protocol even with errors. ST agents will not pass data packets over a new hop whose setup is not complete.

10.2 Control PDUs

SCMP control messages are exchanged between neighbor ST agents using a D-bit of zero (0). The control protocol follows a request-response model with all requests expecting responses. Retransmission after timeout (see Section 4.3) is used to allow for lost or ignored messages. Control messages do not extend across packet boundaries; if a control message is too large for the MTU of a hop, its information is partitioned and a control message per partition is sent (see Section 5.1.2). All control messages have the following format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  OpCode       |    Options    |           TotalBytes          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Reference            |          LnkReference         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        SenderIPAddress                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |           ReasonCode          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                     OpCodeSpecificData                        :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 11: ST Control Message Format

- **OpCode** identifies the type of control message.
- **Options** is used to convey OpCode-specific variations for a control message.
- **TotalBytes** is the length of the control message, in bytes, including all OpCode specific fields and optional parameters. The value is always divisible by four (4).
- **Reference** is a transaction number. Each sender of a request control message assigns a Reference number to the message that is unique with respect to the stream. The Reference number is used by the receiver to detect and discard duplicates. Each acknowledgment carries the Reference number of the request being acknowledged. Reference zero (0) is never used, and Reference numbers are assumed to be monotonically increasing with wraparound so that the older-than and more-recent-than relations are well defined.
- **LnkReference** contains the Reference field of the request control message that caused this request control message to be created. It is used in situations where a single request leads to multiple responses from the same ST agent. Examples are CONNECT and CHANGE messages that are first acknowledged hop-by-hop and then lead to an ACCEPT or REFUSE response from each target.
- **SenderIPAddress** is the 32-bit IP address of the network interface that the ST agent used to send the control message. This value changes each time the packet is forwarded by an ST agent (hop-by-hop).

- **Checksum** is the checksum of the control message. Because the control messages are sent in packets that may be delivered with bits in error, each control message must be checked to be error free before it is acted upon.
- **ReasonCode** is set to zero (0 = NoError) in most SCMP messages. Otherwise, it can be set to an appropriate value to indicate an error situation as defined in Section 10.5.3.
- **OpCodeSpecificData** contains any additional information that is associated with the control message. It depends on the specific control message and is explained further below. In some response control messages, fields of zero (0) are included to allow the format to match that of the corresponding request message. The OpCodeSpecificData may also contain optional parameters. The specifics of OpCodeSpecificData are defined in Section 10.3.

10.3 Common SCMP Elements

Several fields and parameters (referred to generically as *elements*) are common to two or more PDUs. They are described in detail here instead of repeating their description several times. In many cases, the presence of a parameter is optional. To permit the parameters to be easily defined and parsed, each is identified with a PCode byte that is followed by a PBytes byte indicating the length of the parameter in bytes (including the PCode, PByte, and any padding bytes). If the length of the information is not a multiple of four (4) bytes, the parameter is padded with one to three zero (0) bytes. PBytes is thus always a multiple of four (4). Parameters can be present in any order.

10.3.1 FlowSpec

The FlowSpec parameter (PCode = 1) is used in several SCMP messages to convey the ST2 flow specification. The FlowSpec parameter has the following format:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   PCode = 1   |    PBytes     |   Version     |       0       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                         FlowSpec detail                       :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 12: FlowSpec Parameter

- the **Version** field contains the FlowSpec version.
- the **FlowSpec** detail field contains the flow specification and is transparent to the ST agent. It is the data structure to be passed to the LRM. It must be 4-byte aligned.

The Null FlowSpec, see Section 9.1, has no FlowSpec detail field. PBytes is set to four (4), and Version is set to zero (0). The ST2+ FlowSpec, see Section 9.2, is a 32-byte data structure. PBytes is set to 36, and Version is set to seven (7).

10.3.2 Group

The Group parameter (PCode = 2) is an optional argument used to indicate that the stream is a member in the specified group.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  PCode = 2    |  PBytes = 16  |           GroupUniqueID       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        GroupCreationTime                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     GroupInitiatorIPAddress                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Relationship       |                N              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 13: Group Parameter

- **GroupUniqueID**, **GroupInitiatorIPAddress**, and **GroupCreationTime** together form the *GroupName* field. They are allocated by the group name generator function, see Section 8.2. GroupUniqueID and GroupCreationTime are implementation specific and have only local definitions.
- **Relationship** has the following format:

```
                     0
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    0 (unused)         |S|P|F|B|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 14: Relationship Field

The B, F, P, S bits correspond to Bandwidth, Fate, Path, and Subnet resources sharing, see Section 7. A value of 1 indicates that the relationship exists for this group. All combinations of the four bits are allowed. Bits 0-11 of the Relationship field are reserved for future use and must be set to 0.

- **N** contains a legal value only if the B-bit is set. It is the value of the N parameter to be used as explained in Section 7.1.1.

10.3.3 MulticastAddress

The MulticastAddress parameter (PCode = 3) is an optional parameter that is used when using IP encapsulation and setting up an IP multicast group. This parameter is used to communicate the desired IP multicast address to next-hop ST agents that should become members of the group, see Section 8.8.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  PCode = 3    |  PBytes = 8   |               0               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        IPMulticastAddress                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 15: MulticastAddress

- **IPMulticastAddress** is the 32-bit IP multicast address to be used to receive data packets for the stream.

10.3.4 Origin

The Origin parameter (PCode = 4) is used to identify the next higher protocol, and the SAP being used in conjunction with that protocol.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |  PCode = 5    |    PBytes     | NextPcol      |OriginSAPBytes |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 :                OriginSAP                      :     Padding   |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 16: Origin

- **NextPcol** is an 8-bit field used in demultiplexing operations to identify the protocol to be used above ST. The values of NextPcol are in the same number space as the IP header's Protocol field and are consequently defined in the Assigned Numbers RFC [RFC1700].
- **OriginSAPBytes** specifies the length of the OriginSAP, exclusive of any padding required to maintain 32-bit alignment.
- **OriginSAP** identifies the origin's SAP associated with the NextPcol protocol.

Note that the 32-bit IP address of the stream origin is not included in this parameter because it is always available as part of the ST header.

10.3.5 RecordRoute

The RecordRoute parameter (PCode = 5) is used to request that the route between the origin and a target be recorded and delivered to the user application. The ST agent at the origin (or target) including this parameter, has to determine the parameter's length, indicated by the PBytes field. ST agents processing messages containing this parameter add their receiving IP address in the position indicated by the FreeOffset field, space permitting. If no space is available, the parameter is passed unchanged. When included by the origin, all agents between the origin and the target add their IP addresses and this information is made available to the application at the target. When included by the target, all agents between the target and the origin, inclusive, add their IP addresses and this information is made available to the application at the origin.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   PCode = 5   |     PBytes    |       0       |   FreeOffset  |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                           IP Address 1                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 :                               ...                             :
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                           IP Address N                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 17: RecordRoute

- **PBytes** is the length of the parameter in bytes. Length is determined by the agent (target or origin) that first introduces the parameter. Once set, the length of the parameter remains unchanged.
- **FreeOffset** indicates the offset, relative to the start of the parameter, for the next IP address to be recorded. When the FreeOffset is greater than, or equal to, PBytes the RecordRoute parameter is full.
- **IP Address** is filled in, space permitting, by each ST agent processing this parameter.

10.3.6 Target and TargetList

Several control messages use a parameter called TargetList (PCode = 6), which contains information about the targets to which the message pertains. For each Target in the TargetList, the information includes the 32-bit IP address of the target, the SAP applicable to the next higher layer protocol, and the length of the SAP (SAPBytes). Consequently, a Target structure can be of variable length. Each entry has the format shown in Figure 18.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Target IP Address                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  TargetBytes  |  SAPBytes     |       SAP       :    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 18: Target

- **TargetIPAddress** is the 32-bit IP Address of the Target.
- **TargetBytes** is the length of the Target structure, beginning with the TargetIPAddress.
- **SAPBytes** is the length of the SAP, excluding any padding required to maintain 32-bit alignment.
- **SAP** may be longer than 2 bytes and it includes a padding when required. There would be no padding required for SAPs with lengths of 2, 6, 10, etc., bytes.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  PCode = 6    |   PBytes      |           TargetCount = N     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Target 1                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                               :                               :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Target N                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 19: TargetList

10.3.7 UserData

The UserData parameter (PCode = 7) is an optional parameter that may be used by the next higher protocol or an application to convey arbitrary information to its peers. This parameter is propagated in some control messages and its contents have no significance to ST agents. Note that since the size of control messages is limited by the smallest MTU in the path to the targets, the maximum size of this parameter cannot be specified a priori. If the size of this parameter causes a message to exceed the network MTU, an ST agent behaves as described in Section 5.1.2. The parameter must be padded to a multiple of 32 bits.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  PCode = 7    |    PBytes     |           UserBytes           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                       UserInfo                :   Padding     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 20: UserData

- **UserBytes** specifies the number of valid UserInfo bytes.
- **UserInfo** is arbitrary data meaningful to the next higher protocol layer or application.

10.3.8 Handling of Undefined Parameters

An ST agent must be able to handle all parameters listed above. To support possible future uses, parameters with unknown PCodes must also be supported. If an agent receives a message containing a parameter with an unknown Pcode value, the agent should handle the parameter as if it was a UserData parameter. That is, the contents of the parameter should be ignored, and the message should be propagated, as appropriate, along with the related control message.

10.4 ST Control Message PDUs

ST Control messages are described in the following section. Please refer to Section 10.6 for an explanation of the notation.

10.4.1 ACCEPT

ACCEPT (OpCode = 1) is issued by a target as a positive response to a CONNECT message. It implies that the target is prepared to accept data from the origin along the stream that was established by the CONNECT. ACCEPT is also issued as a positive response to a CHANGE message. It implies that the target accepts the proposed stream modification.

ACCEPT is relayed by the ST agents from the target to the origin along the path established by CONNECT (or CHANGE) but in the reverse direction. ACCEPT must be acknowledged with ACK at each hop.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  OpCode = 1   |       0       |           TotalBytes          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Reference               |         LnkReference          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         SenderIPAddress                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Checksum          |          ReasonCode = 0       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           MaxMsgSize          |          RecoveryTimeout      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      StreamCreationTime                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   IPHops      |                        0                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                            FlowSpec                           :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                            TargetList                         :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                            RecordRoute                        :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                            UserData                           :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 21: ACCEPT Control Message

- **Reference** contains a number assigned by the ST agent sending ACCEPT for use in the acknowledging ACK.
- **LnkReference** is the Reference number from the corresponding CONNECT (or CHANGE)
- **MaxMsgSize** indicates the smallest MTU along the path traversed by the stream. This field is only set when responding to a CONNECT request.
- **RecoveryTimeout** reflects the nominal number of milliseconds that the application is willing to wait for a failed system component to be detected and any corrective action to be taken. This field represents what can actually be supported by each participating agent, and is only set when responding to a CONNECT request.
- **StreamCreationTime** is the 32- bits system dependent timestamp copied from the corresponding CONNECT request.
- **IPHops** is the number of IP encapsulated hops traversed by the stream. This field is set to zero by the origin, and is incremented at each IP encapsulating agent.

10.4.2 ACK

ACK (OpCode = 2) is used to acknowledge a request. The ACK message is not propagated beyond the previous-hop or next-hop ST agent.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |  OpCode = 2   |       0       |           TotalBytes          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |       Reference               |           LnkReference = 0    |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                         SenderIPAddress                       |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |       Checksum                |           ReasonCode          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 22: ACK Control Message

- **Reference** is the Reference number of the control message being acknowledged.
- **ReasonCode** is usually NoError, but other possibilities exist, e.g., DuplicateIgn.

10.4.3 CHANGE

CHANGE (OpCode = 3) is used to change the FlowSpec of an established stream. The CHANGE message is processed similarly to CONNECT, except that it travels along the path of an established stream. CHANGE must be propagated until it reaches the related stream's targets. CHANGE must be acknowledged with ACK at each hop.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |  OpCode = 3   |G|I|     0     |           TotalBytes          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |           Reference           |          LnkReference = 0     |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                        SenderIPAddress                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |            Checksum           |          ReasonCode = 0       |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 :                            FlowSpec                           :
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 :                           TargetList                          :
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 :                           RecordRoute                         :
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 :                            UserData                           :
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 23: CHANGE Control Message

- **G** (bit 8) is used to request a global, stream-wide change; the TargetList parameter may be omitted when the G bit is specified.
- **I** (bit 7) is used to indicate that the LRM is permitted to interrupt and, if needed, break the stream in the process of trying to satisfy the requested change.

- **Reference** contains a number assigned by the ST agent sending CHANGE for use in the acknowledging ACK.

10.4.4 CONNECT

CONNECT (OpCode = 4) requests the setup of a new stream or an addition to or recovery of an existing stream. Only the origin can issue the initial set of CONNECTs to setup a stream, and the first CONNECT to each next-hop is used to convey the SID.

The next-hop initially responds with an ACK, which implies that the CONNECT was valid and is being processed. The next-hop will later relay back either an ACCEPT or REFUSE from each target. An intermediate ST agent that receives a CONNECT behaves as explained in Section 4.5.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  OpCode = 4   |J N|S|     0     |           TotalBytes          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Reference           |       LnkReference = 0        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       SenderIPAddress                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |        ReasonCode = 0         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           MaxMsgSize          |        RecoveryTimeout        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      StreamCreationTime                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   IPHops      |                      0                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                            Origin                             :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                           FlowSpec                            :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                          TargetList                           :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                          RecordRoute                          :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                             Group                             :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                       MulticastAddress                        :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                           UserData                            :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 24: CONNECT Control Message

- **JN** (bits 8 and 9) indicate the join authorization level for the stream, see Section 4.4.2.

- **S** (bit 10) indicates the NoRecovery option (Section 4.4.1). When the S-bit is set (1), the NoRecovery option is specified for the stream.
- **Reference** contains a number assigned by the ST agent sending CONNECT for use in the acknowledging ACK.
- **MaxMsgSize** indicates the smallest MTU along the path traversed by the stream. This field is initially set to the network MTU of the agent issues the CONNECT.
- **RecoveryTimeout** is the nominal number of milliseconds that the application is willing to wait for failed system component to be detected and any corrective action to be taken.
- **StreamCreationTime** is the 32- bits system dependent timestamp generated by the ST agent issuing the CONNECT.
- **IPHops** is the number of IP encapsulated hops traversed by the stream. This field is set to zero by the origin, and is incremented at each IP encapsulating agent.

10.4.5 DISCONNECT

DISCONNECT (OpCode = 5) is used by an origin to tear down an established stream or part of a stream, or by an intermediate ST agent that detects a failure between itself and its previous-hop, as distinguished by the ReasonCode. The DISCONNECT message specifies the list of targets that are to be disconnected. An ACK is required in response to a DISCONNECT message. The DISCONNECT message is propagated all the way to the specified targets. The targets are expected to terminate their participation in the stream.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  OpCode = 5   |G|     0       |           TotalBytes          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Reference                |      LnkReference = 0         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         SenderIPAddress                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Checksum           |          ReasonCode           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      GeneratorIPAddress                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                          TargetList                           :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                           UserData                            :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 25: DISCONNECT Control Message

- **G** (bit 8) is used to request a DISCONNECT of all the stream's targets. TargetList should be omitted when the G-bit is set (1). If TargetList is present, it is ignored.
- **Reference** contains a number assigned by the ST agent sending DISCONNECT for use in the acknowledging ACK.
- **ReasonCode** reflects the event that initiated the message.

- **GeneratorIPAddress** is the 32-bit IP address of the host that first generated the DISCONNECT message.

10.4.6 ERROR

ERROR (OpCode = 6) is sent in acknowledgment to a request in which an error is detected. No action is taken on the erroneous request. No ACK is expected. The ERROR message is not propagated beyond the previous-hop or next-hop ST agent. An ERROR is never sent in response to another ERROR. The receiver of an ERROR is encouraged to try again without waiting for a retransmission timeout.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  OpCode = 6   |       0       |           TotalBytes          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Reference                |     LnkReference = 0          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         SenderIPAddress                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Checksum           |          ReasonCode           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                           PDUInError                          :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 26: ERROR Control Message

- **Reference** is the Reference number of the erroneous request.
- **ReasonCode** indicates the error that triggered the message.
- **PDUInError** is the PDU in error, beginning with the ST Header. This parameter is optional. Its length is limited by network MTU, and may be truncated when too long.

10.4.7 HELLO

HELLO (OpCode = 7) is used as part of the ST failure detection mechanism, see Section 6.1.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  OpCode = 7   |R|     0       |           TotalBytes          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Reference = 0           |        LnkReference = 0       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         SenderIPAddress                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Checksum              |         ReasonCode = 0        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           HelloTimer                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 27: HELLO Control Message

- **R** (bit 8) is used for the Restarted-bit.

- **HelloTimer** represents the time in millisecond since the agent was restarted, modulo the precision of the field. It is used to detect duplicate or delayed HELLO messages.

10.4.8 JOIN

JOIN (OpCode = 8) is used as part of the ST steam joining mechanism, see Section 4.6.3.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  OpCode = 8   |       0       |           TotalBytes          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Reference                |        LnkReference = 0       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         SenderIPAddress                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Checksum           |         ReasonCode = 0        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      GeneratorIPAddress                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                           TargetList                          :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 28: JOIN Control Message

- **Reference** contains a number assigned by the ST agent sending JOIN for use in the acknowledging ACK.
- **GeneratorIPAddress** is the 32-bit IP address of the host that generated the JOIN message.
- **TargetList** is the information of the target to be added to the stream.

10.4.9 JOIN-REJECT

JOIN-REJECT (OpCode = 9) is used as part of the ST steam joining mechanism, see Section 4.6.3.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  OpCode = 9   |       0       |           TotalBytes          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Reference                |          LnkReference         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         SenderIPAddress                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Checksum           |           ReasonCode          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      GeneratorIPAddress                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 29: JOIN-REJECT Control Message

- **Reference** contains a number assigned by the ST agent sending the REFUSE for use in the acknowledging ACK.
- **LnkReference** is the Reference number from the corresponding JOIN message.

- **ReasonCode** reflects the reason why the JOIN request was rejected.
- **GeneratorIPAddress** is the 32-bit IP address of the host that first generated the JOIN-REJECT message.

10.4.10 NOTIFY

NOTIFY (OpCode = 10) is issued by an ST agent to inform other ST agents of events that may be significant. NOTIFY may be propagated beyond the previous-hop or next-hop ST agent depending on the ReasonCode, see Section 10.5.3; NOTIFY must be acknowledged with an ACK.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  OpCode = 10  |       0       |           TotalBytes          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Reference                |        LnkReference = 0       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        SenderIPAddress                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Checksum           |          ReasonCode           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     DetectorIPAddress                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          MaxMsgSize           |          RecoveryTimeout      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                          FlowSpec                             :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                          TargetList                           :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                          UserData                             :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 30: NOTIFY Control Message

- **Reference** contains a number assigned by the ST agent sending the NOTIFY for use in the acknowledging ACK.
- **ReasonCode** identifies the reason for the notification.
- **DetectorIPAddress** is the 32-bit IP address of the ST agent that detects the event.
- **MaxMsgSize** is set when the MTU of the listed targets has changed (e.g. due to recovery), or when the notification is generated after a successful JOIN. Otherwise it is set to zero (0).
- **RecoveryTimeout** is set when the notification is generated after a successful JOIN. Otherwise it is set to zero (0).
- **FlowSpec** is present when the notification is generated after a successful JOIN.
- **TargetList** is present when the notification is related to one or more targets, or when MaxMsgSize is set

- **UserData** is present if the notification is generated after a successful JOIN and the UserData parameter was set in the ACCEPT message.

10.4.11 REFUSE

REFUSE (OpCode = 11) is issued by a target that either does not wish to accept a CONNECT message or wishes to remove itself from an established stream. It might also be issued by an intermediate ST agent in response to a CONNECT or CHANGE either to terminate a routing loop, or when a satisfactory next-hop to a target cannot be found. It may also be a separate command when an existing stream has been preempted by a higher precedence stream or an ST agent detects the failure of a previous-hop, next-hop, or the network between them. In all cases, the TargetList specifies the targets that are affected by the condition. Each REFUSE must be acknowledged by an ACK.

The REFUSE is relayed back by the ST agents to the origin (or intermediate ST agent that created the CONNECT or CHANGE) along the path traced by the CONNECT. The ST agent receiving the REFUSE will process it differently depending on the condition that caused it, as specified in the ReasonCode field. No special effort is made to combine multiple REFUSE messages since it is considered most unlikely that separate REFUSEs will happen to both pass through an ST agent at the same time and be easily combined, e.g., have identical ReasonCodes and parameters.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  OpCode = 11  |G|E|N|    0    |           TotalBytes          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Reference                |         LnkReference          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          SenderIPAddress                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Checksum           |          ReasonCode           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        DetectorIPAddress                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        ValidTargetIPAddress                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                           TargetList                          :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                          RecordRoute                          :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                            UserData                           :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 31: REFUSE Control Message

- **G** (bit 8) is used to indicate that all targets down stream from the sender are refusing. It is expected that this will be set most commonly due to network failures. The TargetList parameter is ignored or not present when this bit is set, and must be included when not set.

- **E** (bit 9) is set by an ST agent to indicate that the request failed and that the pre-change stream attributes, including resources, and the stream itself still exist.
- **N** (bit 10) is used to indicate that no further attempts to recover the stream should be made. This bit must be set when stream recovery should not be attempted, even in the case where the target application has shut down normally (ApplDisconnect).
- **Reference** contains a number assigned by the ST agent sending the REFUSE for use in the acknowledging ACK.
- **LnkReference** is either the Reference number from the corresponding CONNECT or CHANGE, if it is the result of such a message, or zero when the REFUSE was originated as a separate command.
- **DetectorIPAddress** is the 32-bit IP address of the host that first generated the REFUSE message.
- **ValidTargetIPAddress** is the 32-bit IP address of a host that is properly connected as part of the stream. This parameter is only used when recovering from stream convergence, otherwise it is set to zero (0).

10.4.12 STATUS

STATUS (OpCode = 12) is used to inquire about the existence of a particular stream identified by the SID. Use of STATUS is intended for collecting information from an neighbor ST agent, including general and specific stream information, and round trip time estimation. The use of this message type is described in Section 8.4.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| OpCode = 12   |       0       |           TotalBytes          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Reference                |       LnkReference = 0        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         SenderIPAddress                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Checksum           |         ReasonCode = 0        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                            TargetList                         :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 32: STATUS Control Message

- **Reference** contains a number assigned by the ST agent sending STATUS for use in the replying STATUS-RESPONSE.
- **TargetList** is an optional parameter that when present indicates that only information related to the specific targets should be relayed in the STATUS-RESPONSE.

10.4.13 STATUS-RESPONSE

STATUS-RESPONSE (OpCode = 13) is the reply to a STATUS message. If the stream specified in the STATUS message is not known, the STATUS-RESPONSE will contain the specified SID but no other parameters. It will otherwise contain the current SID, FlowSpec, TargetList, and possibly Groups of the stream. It the full target list can not fit in a single message, only those targets that can be included in one message will be included. As mentioned in Section 10.4.12, it is possible to request information on a specific target.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  OpCode = 13  |       0       |           TotalBytes          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Reference                |       LnkReference = 0        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         SenderIPAddress                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |        ReasonCode = 0         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                          FlowSpec                             :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                          Groups                               :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                         TargetList                            :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 33: STATUS-RESPONSE Control Message

- **Reference** contains a number assigned by the ST agent sending the STATUS.

10.5 Suggested Protocol Constants

The ST Protocol uses several fields that must have specific values for the protocol to work, and also several values that an implementation must select. This section specifies the required values and suggests initial values for others. It is recommended that the latter be implemented as variables so that they may be easily changed when experience indicates better values. Eventually, they should be managed via the normal network management facilities.

ST uses IP Version Number 5.

When encapsulated in IP, ST uses IP Protocol Number 5.

10.5.1 SCMP Messages

1) ACCEPT
2) ACK
3) CHANGE
4) CONNECT
5) DISCONNECT
6) ERROR
7) HELLO
8) JOIN
9) JOIN-REJECT
10) NOTIFY
11) REFUSE

12) STATUS
13) STATUS-RESPONSE

10.5.2 SCMP Parameters

1) FlowSpec
2) Group
3) MulticastAddress
4) Origin
5) RecordRoute
6) TargetList
7) UserData

10.5.3 ReasonCode

Several errors may occur during protocol processing. All ST error codes are taken from a single number space. The currently defined values and their meaning is presented in the list below. Note that new error codes may be defined from time to time. All implementations are expected to handle new codes in a graceful manner. If an unknown ReasonCode is encountered, it should be assumed to be fatal. The ReasonCode is an 8-bit field. Following values are defined:

1	NoError	No error has occurred.
2	ErrorUnknown	An error not contained in this list has been detected.
3	AccessDenied	Access denied.
4	AckUnexpected	An unexpected ACK was received.
5	ApplAbort	The application aborted the stream abnormally.
6	ApplDisconnect	The application closed the stream normally.
7	ApplRefused	Applications refused requested connection or change.
8	AuthentFailed	The authentication function failed.
9	BadMcastAddress	IP Multicast address is unacceptable in CONNECT.
10	CantGetResrc	Unable to acquire (additional) resources.
11	CantRelResrc	Unable to release excess resources.
12	CantRecover	Unable to recover failed stream.
13	CksumBadCtl	Control PDU has a bad message checksum.
14	CksumBadST	PDU has a bad ST Header checksum.
15	DuplicateIgn	Control PDU is a duplicate and is being acknowledged.
16	DuplicateTarget	Control PDU contains a duplicate target, or an attempt to add an existing target.
17	FlowSpecMismatch	FlowSpec in request does not match existing FlowSpec.
18	FlowSpecError	An error occurred while processing the FlowSpec
19	FlowVerUnknown	Control PDU has a FlowSpec Version Number that is not supported.
20	GroupUnknown	Control PDU contains an unknown Group Name.
21	InconsistGroup	An inconsistency has been detected with the streams forming a group.
22	IntfcFailure	A network interface failure has been detected.
23	InvalidSender	Control PDU has an invalid SenderIPAddress field.
24	InvalidTotByt	Control PDU has an invalid TotalBytes field.
25	JoinAuthFailure	Join failed due to stream authorization level.
26	LnkRefUnknown	Control PDU contains an unknown LnkReference.
27	NetworkFailure	A network failure has been detected.
28	NoRouteToAgent	Cannot find a route to an ST agent.
29	NoRouteToHost	Cannot find a route to a host.
30	NoRouteToNet	Cannot find a route to a network.
31	OpCodeUnknown	Control PDU has an invalid OpCode field.
32	PCodeUnknown	Control PDU has a parameter with an invalid PCode.
33	ParmValueBad	Control PDU contains an invalid parameter value.

34	PathConvergence	Two branches of the stream join during the CONNECT setup.
35	ProtocolUnknown	Control PDU contains an unknown next-higher layer protocol identifier.
36	RecordRouteSize	RecordRoute parameter is too long to permit message to fit a network's MTU.
37	RefUnknown	Control PDU contains an unknown Reference.
38	ResponseTimeout	Control message has been acknowledged but not answered by an appropriate control message.
39	RestartLocal	The local ST agent has recently restarted.
40	RestartRemote	The remote ST agent has recently restarted.
41	RetransTimeout	An acknowledgment has not been received after several retransmissions.
42	RouteBack	Route to next-hop through same interface as previous-hop and is not previous-hop.
43	RouteInconsist	A routing inconsistency has been detected.
44	RouteLoop	A routing loop has been detected.
45	SAPUnknown	Control PDU contains an unknown next-higher layer SAP (port).
46	SIDUnknown	Control PDU contains an unknown SID.
47	STAgentFailure	An ST agent failure has been detected.
48	STVer3Bad	A received PDU is not ST Version 3.
49	StreamExists	A stream with the given SID already exists.
50	StreamPreempted	The stream has been preempted by one with a higher precedence.
51	TargetExists	A CONNECT was received that specified an existing target.
52	TargetUnknown	A target is not a member of the specified stream.
53	TargetMissing	A target parameter was expected and is not included, or is empty.
54	TruncatedCtl	Control PDU is shorter than expected.
55	TruncatedPDU	A received ST PDU is shorter than the ST Header indicates.
56	UserDataSize	UserData parameter too large to permit a message to fit into a network's MTU.

10.5.4 Timeouts and Other Constants

SCMP uses retransmission to effect reliability and thus has several "retransmission timers". Each "timer" is modeled by an initial time interval (ToXxx), which may get updated dynamically through measurement of control traffic, and a number of times (NXxx) to retransmit a message before declaring a failure. All time intervals are in units of milliseconds. Note that the variables are described for reference purposes only, different implementations may not include the identical variables.

```
Value  Timeout Name            Meaning
----------------------------------------------------------------------------
 500   ToAccept                Initial hop-by-hop timeout for acknowledgment
                               of ACCEPT
   3   NAccept                 ACCEPT retries before failure
 500   ToChange                Initial hop-by-hop timeout for acknowledgment
                               of CHANGE
   3   NChange                 CHANGE retries before failure
 5000  ToChangeResp            End-to-End CHANGE timeout for receipt of
                               ACCEPT or REFUSE
 500   ToConnect               Initial hop-by-hop timeout for acknowledgment
                               of CONNECT
   5   NConnect                CONNECT retries before failure
 5000  ToConnectResp           End-to-End CONNECT timeout for receipt of
                               ACCEPT or REFUSE from targets by origin
 500   ToDisconnect            Initial hop-by-hop timeout for acknowledgment
```

		of DISCONNECT
3	NDisconnect	DISCONNECT retries before failure
500	ToJoin	Initial hop-by-hop timeout for acknowledgment of JOIN
3	NJoin	JOIN retries before failure
500	ToJoinReject	Initial hop-by-hop timeout for acknowledgment of JOIN-REJECT
3	NJoinReject	JOIN-REJECT retries before failure
5000	ToJoinResp	Timeout for receipt of CONNECT or JOIN-REJECT from origin or intermediate hop
500	ToNotify	Initial hop-by-hop timeout for acknowledgment of NOTIFY
3	NNotify	NOTIFY retries before failure
500	ToRefuse	Initial hop-by-hop timeout for acknowledgment of REFUSE
3	NRefuse	REFUSE retries before failure
500	ToRetryRoute	Timeout for receipt of ACCEPT or REFUSE from targets during failure recovery
5	NRetryRoute	CONNECT retries before failure
1000	ToStatusResp	Timeout for receipt of STATUS-RESPONSE
3	NStatus	STATUS retries before failure
10000	HelloTimerHoldDown	Interval that Restarted bit must be set after ST restart
5	HelloLossFactor	Number of consecutively missed HELLO messages before declaring link failure
2000	DefaultRecoveryTimeout	Interval between successive HELLOs to/from active neighbors

10.6 Data Notations

The convention in the documentation of Internet Protocols is to express numbers in decimal and to picture data with the most significant octet on the left and the least significant octet on the right.

The order of transmission of the header and data described in this document is resolved to the octet level. Whenever a diagram shows a group of octets, the order of transmission of those octets is the normal order in which they are read in English. For example, in the following diagram the octets are transmitted in the order they are numbered.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       1       |       2       |       3       |       4       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       5       |       6       |       7       |       8       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       9       |      10       |      11       |      12       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 34: Transmission Order of Bytes

Whenever an octet represents a numeric quantity the left most bit in the diagram is the high order or most significant bit. That is, the bit labeled 0 is the most significant bit. For example, the following diagram represents the value 170 (decimal).

```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|1 0 1 0 1 0 1 0|
+-+-+-+-+-+-+-+-+
```

Figure 35: Significance of Bits

Similarly, whenever a multi-octet field represents a numeric quantity the left most bit of the whole field is the most significant bit. When a multi-octet quantity is transmitted the most significant octet is transmitted first.

Fields whose length is fixed and fully illustrated are shown with a vertical bar (|) at the end; fixed fields whose contents are abbreviated are shown with an exclamation point (!); variable fields are shown with colons (:). Optional parameters are separated from control messages with a blank line. The order of parameters is not meaningful.

11 Security Considerations

This memo does not address security issues.

12 Acknowledgments and Author's Addresses

Many individuals have contributed to the work described in this memo. We thank the participants in the ST Working Group for their input, review, and constructive comments. George Mason University C3I Center for hosting an interim meeting. Murali Rajagopal for his efforts on ST2+ state machines. We also thank Lynne Kendall Beltran for translating our drawings into ASCII. Special thanks are due to Steve DeJarnett, who served as working group co-chair until summer 1993.

We would also like to acknowledge the authors of [RFC1190]. All authors of [RFC1190] should be considered authors of this document since this document contains much of their text and ideas.

Louis Berger
BBN Systems and Technologies
1300 North 17th Street, Suite 1200
Arlington, VA 22209
Phone: 703-284-4651
EMail: lberger@bbn.com

Luca Delgrossi
IBM ENC
Multimedia Technology Center

Vangerowstr. 18
D69020 Heidelberg, Germany
Phone: +49-6221-594330
EMail: luca@heidelbg.ibm.com

Dat Duong
BBN Systems and Technologies
1300 North 17th Street, Suite 1200
Arlington, VA 22209
Phone: 703-284-4760
EMail: dat@bbn.com

Steve Jackowski
Syzygy Communications Incorporated
269 Mt. Hermon Road
Scotts Valley, CA 95066
Phone: 408-439-6834
EMail: stevej@syzygycomm.com

Sibylle Schaller
IBM ENC
Broadband Multimedia Communications
Vangerowstr. 18
D69020 Heidelberg, Germany
Phone: +49-6221-5944553
EMail: schaller@heidelbg.ibm.com

13 References

[RFC1071]
Braden, Borman, Partridge: *Computing the Internet Checksum*, RFC 1071, USC/Information Sciences Institute, Cray Research, BBN Laboratories, September 1988.

[RFC1112]
Deering, S.: Host Extensions for IP multicasting, RFC 1112, Stanford University, August 1989.

[WoHD95]
L. Wolf, R. G. Herrtwich, L. Delgrossi: *Filtering Multimedia Data in Reservation-based Networks*, Kommunikation in Verteilten Systemen 1995 (KiVS), Chemnitz-Zwickau, Germany, February 1995.

[RFC1122]
Braden, R.: *Requirements for Internet Hosts -- Communication Layers*, RFC 1122, USC/Information Sciences Institute, October 1989.

[Jaco88]

Jacobson, V.: *Congestion Avoidance and Control*, ACM SIGCOMM-88, August 1988.

[KaPa87]

Karn, P. and C. Partridge: *Round Trip Time Estimation*, ACM SIGCOMM-87, August 1987.

[RFC1141]

Mallory, T. and A. Kullberg: Incremental Updating of the Internet Checksum, RFC 1141, BBN, January 1990.

[RFC1363]

C. Partridge: *A Proposal Flow Specification*, RFC 1363.

[RFC791]

Postel: Internet Protocol, RFC 791, DARPA, September 1981.

[RFC1700]

Reynolds, Postel: Assigned Numbers, RFC 1700, ISI, October 1994.

[RFC1190]

Topolcic C.: Internet Stream Protocol Version 2 (ST-II), RFC1190, October 1990.

[RFC1633]

R. Braden, D. Clark, S. Shenker: Integrated Services in the Internet Architecture: an Overview, RFC1633, June 1994.

[VoHN93]

C. Vogt, R. G. Herrtwich, R. Nagarajan: *HeiRAT: the Heidelberg Resource Administration Technique - Design Philosophy and Goals*, Kommunikation In Verteilten Systemen, Munich, Informatik Aktuell, Springer-Verlag, Heidelberg, 1993.

[Cohe81]

D. Cohen: *A Network Voice Protocol NVP-II*, University of Southern California, Los Angeles, 1981.

[Cole81]

R. Cole: *PVP – A Packet Video Protocol*, University of Southern California, Los Angeles, 1981.

[DeAl92]

L. Delgrossi (Ed.) *The BERKOM-II Multimedia Transport System*, Version 1, BERKOM Working Document, October, 1992.

[DHHS92]

L. Delgrossi, C. Halstrick, R. G. Herrtwich, H. Stuettgen: *HeiTP: a Transport Protocol for ST-II*, GLOBECOM'92, Orlando (Florida), December 1992.

[Schu94]

H. Schulzrinne: *RTP: A Transport Protocol for Real-Time Applications.* Internet Draft, work in progress, 1994.