Internet-Draft November 1994
ST Working Group
L. Delgrossi (Ed.)
File: draft-ietf-st2-spec-01.ps Expires: April 1995

Internet STream Protocol Version 2 (ST2)

- Protocol Specification -
Version ST2+

Status of this Memo

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 its 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+.

Editor’s Note

This memo is available both in ASCII format (file: draft-ietf-st2-spec-01.txt) and in PostScript (file: draft-ietf-st2-spec-01.ps). The PostScript version contains some additional pictures that help to clarify the text, and it is therefore recommended.

This draft is not complete. It will serve as the basis for discussions at the December IETF ST2 sessions. The work remaining is largely nontechnical. No major technical details are expected to change, some minor details do need to be updated or are not yet documented.

1 Introduction

1.1 What is ST2?

The Internet Stream Protocol, Version 2 (ST2) is a 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 data throughput and controlled delay characteristics. The main application area of the protocol is the real-time transport of digital audio and video packet streams across internets.

ST2 can be used to reserve bandwidth for multimedia 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 to ST2 applications. It ensures that each multimedia packet is delivered within its deadline, that is, at the time where it needs to be presented. This facilitates a smooth playout of digital audio and video that is essential for this time-critical data, 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, mainly those for resource reservation. ST is simple and contains only one PDU that is designed for fast and efficient data forwarding in order to achieve low communication delays. SCMP, however, is quite complex. As with ICMP and IP, SCMP packets are transferred within ST packets as shown in Figure 1.

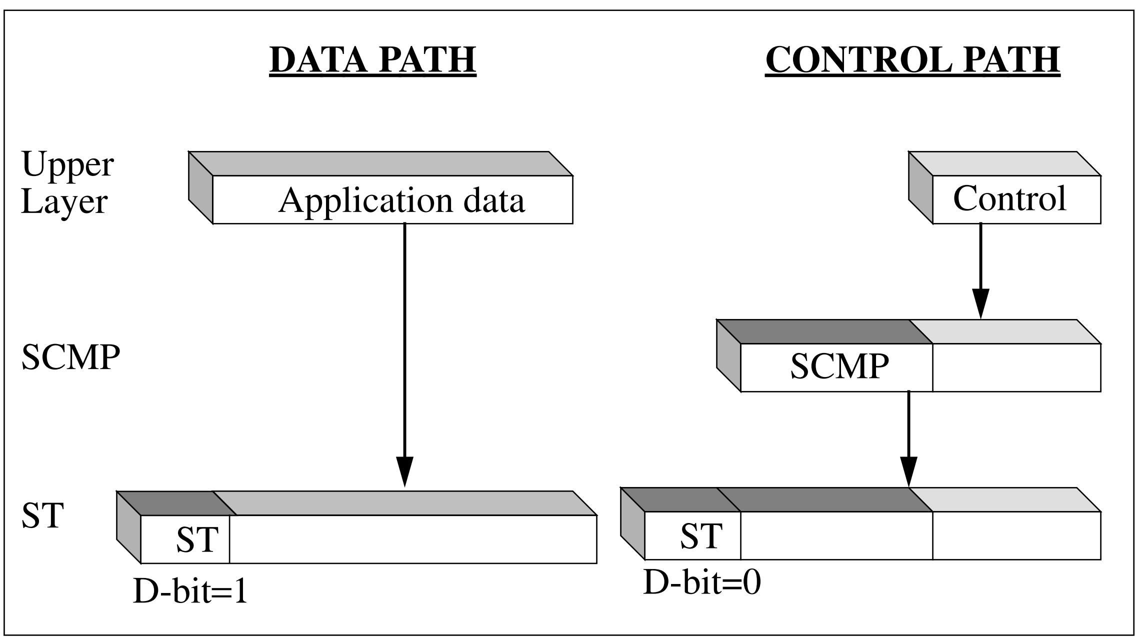

Figure 1: ST2 Data and Control Path

1.2 Protocol History

The first version of ST was published in the late 1970's and was used throughout the 1980's for experimental voice and video transmission. 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 such as BBN have products available incorporating the protocol. The BERKOM project of the German PTT uses ST2 as its core protocol for the provision of multimedia teleservices such as conferencing and mailing. Among others, Digital, HP, IBM, and Siemens-Nixdorf participate in this project. In addition, implementations of ST2 for Sun, Silicon Graphics, Macintosh, NeXT, and PC platforms are available.

In 1993, the IETF has started a new working group on ST2. Its mission is to clean up the current protocol specification to ensure better interoperability between the existing and emerging implementations. It shall also reflect the experiences gained with the current ST2 implementations and applications. This has led to the specification of the ST2+ version contained in this document.

1.3 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. Nodes in the tree represent so-called *ST agents*, entities executing the ST2 protocol; links in the tree are called *hops*.

Figure 2 illustrates a stream from an origin to four targets, where the ST agent on Target 2 also functions as a router. 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.

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

Each ST agent maintains state information describing the streams flowing through it. It can actively gather and distribute such information. If, for example, an intermediate ST agent fails, the neighboring ST agents can recognize this via HELLO messages that are periodically exchanged between ST agents that share streams. STATUS packets can be used to ask other ST agents about a particular stream. These ST agents then send back a STATUS-RESPONSE message. NOTIFY messages serve to inform ST agents of changes such as a route change.

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 one person should speak at a time. Using the group mechanism, resources for only one audio stream 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 fails.

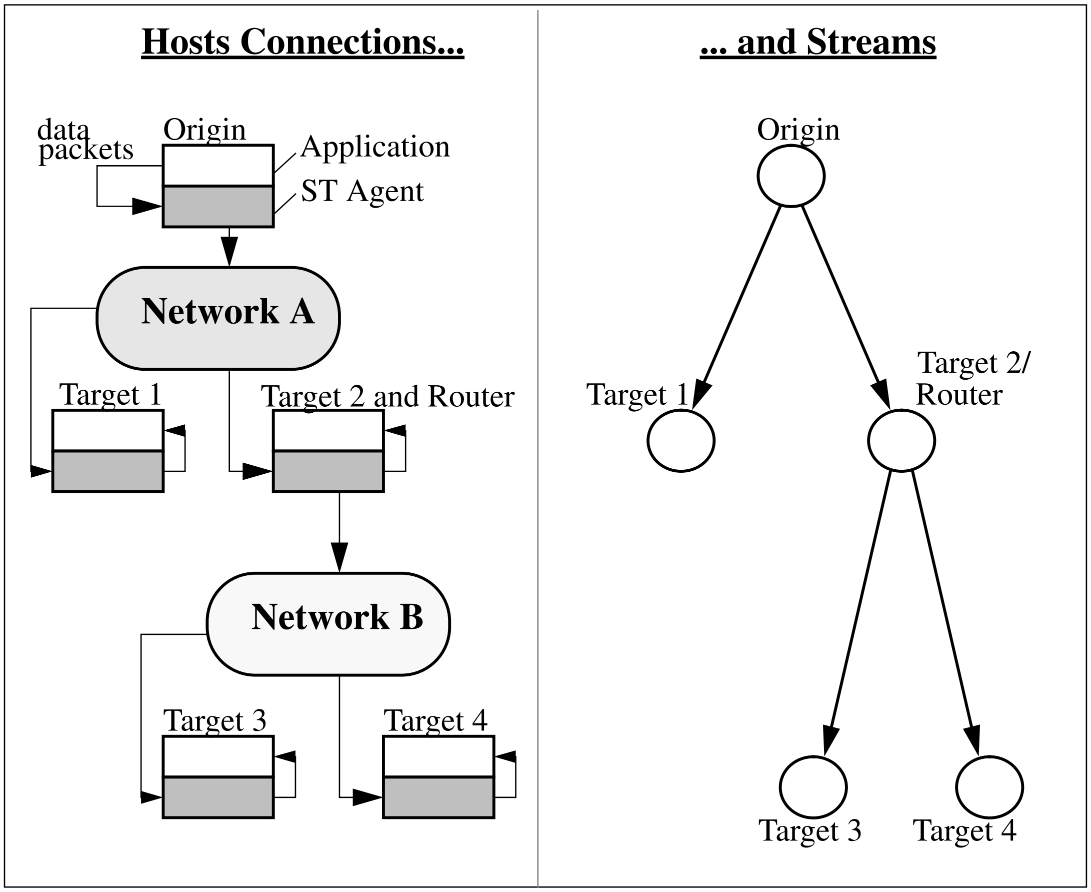

Figure 2: The Stream Concept

1.4 Data Transmission

Data transfer in ST2 is simplex in the downstream direction. Data transport through streams is very efficient. 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 8-byte header which can be extended by an optional 8-byte timestamp.

Efficiency is also achieved by avoiding fragmentation and reassembly on router nodes. Negotiations at stream establishment time yield a maximum transmission unit (MTU) for data packets on a stream. This MTU 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. 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 retransmission known from TCP. It is assumed that 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 audio and video streams are acceptable. In any case, reliability can be provided by layers on top of ST2 if needed.

1.5 Flow Specifications

As part of establishing a connection, SCMP negotiates quality-of-service parameters for a stream. In ST2 terminology, these parameters form a flow specification (FlowSpec, for short) which is associated with the stream. Different versions of FlowSpecs exist 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.

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 specification as part of a connection establishment message initiates the reservation of local resources by the corresponding resource manager. These resource managers control the usage of CPU capacity for protocol processing, buffer space for storing messages, and bandwidth in the outgoing network. 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.

The FlowSpec negotiation procedure is illustrated in Figure 3. Depending on the success of its local reservations, an ST agent updates the FlowSpec while the connection establishment message passes downstream (for example, keeping track of accumulated delay). The final FlowSpec is communicated to the target application which may base its accept/reject 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. If all targets in a particular ST2 connection need to adhere to the same FlowSpec, the origin may – during a second phase of connection establishment – issue a CHANGE request to adjust reservations.

1.6 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 digital audio and video. Whereas IP typically will be accessed from TCP or UDP, ST2 will have new multimedia end-to-end protocols on top of it.

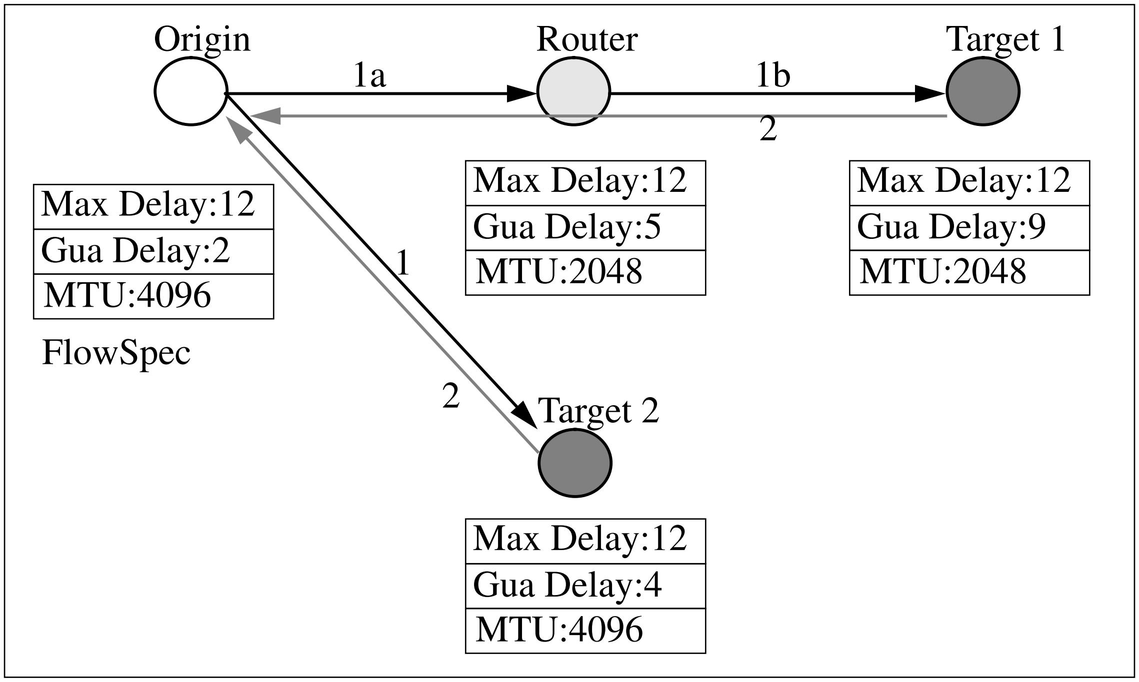

Figure 3: Quality-of-Service Negotiation with FlowSpecs

Both ST2 and IP apply the same addressing schemes to identify different hosts and use ARP for address resolution. ST2 can easily be modified to include the longer host addresses of the next generation IP. ST2 uses the same Layer 2 SAPs as IP. ST2 and IP packets differ in the first four bits, containing the internetwork protocol version number: number 5 is reserved for ST2 (IP itself has version number 4). An ST agent receives a packet over the IP SAP using the first 4 bits of the frame to select ST2 packets.

As a special function, ST2 messages can be encapsulated in IP packets. This allows them 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.

1.7 Outline of This Document

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

The document is organized as follows: Section 2 describes the ST user service; Section 3.1 through Section 3.5 describe stream setup, modification, and tear down; exceptional cases are handled in Section 4; failure detection and groups of streams respectively in Section 5 and Section 6; the FlowSpec is presented in Section ; finally, the formats of the different protocol elements and PDUs are defined by Section 10 through Section 15.

2 ST 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 ST primitives do not form 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 the receiver; then, the receiver may *accept* or *refuse* the request and the correspondent *indication* is generated at the sender. This is shown in Figure 4 below.

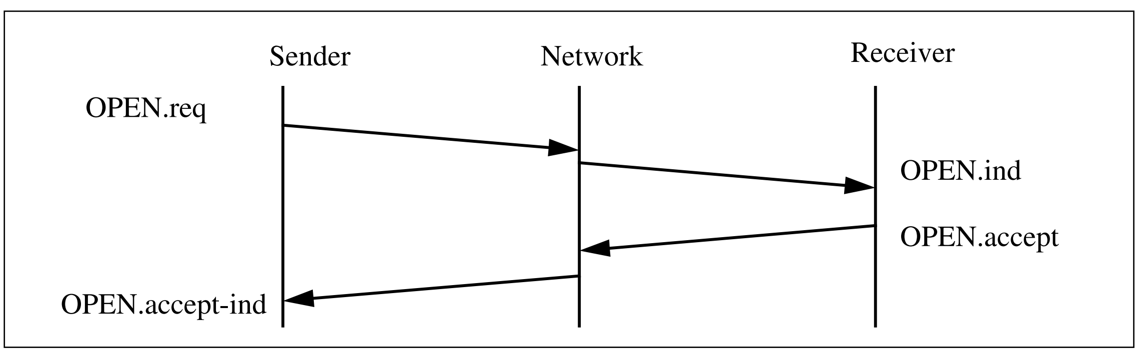

Figure 4: Primitives for the OPEN Stream Operation

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	DATA.req	send data	O
OPEN.ind	indication	T	DATA.ind	indication	T
OPEN.accept	accept stream	T	CHG.req	change stream qos	O
OPEN.refuse	refuse stream	T	CHG.ind	indication	T
OPEN.accept-ind	indication to application	O	CHG.accept	accept change	T
OPEN.refuse-ind	indication to application	O	CHG.refuse	refuse change	T
ADD.req	add targets to stream	O	CHG.accept-ind	indication to application	O
ADD.ind	indication	T	CHG.refuse-ind	indication to application	O
ADD.accept	accept stream	T	DROP.req	drop targets	O
ADD.refuse	refuse stream	T	DROP.ind	indication	T
ADD.accept-ind	indication to application	O	LEAVE.req	leave stream	T
ADD.refuse-ind	indication to application	O	LEAVE.ind	indication	O
JOIN.req	join a stream	T	CLOSE.req	close stream	O
JOIN.ind	indication	O	CLOSE.ind	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 suggested to impose limits on the stream operations that can be executed at the same time. These restrictions are:

1. A single ADD or CHG operation can be processed at one time. If another ADD or CHG is already underway, further requests are queued by the ST agent and handled only after the previous operation has been completed. It 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. 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. When sending data from the CHGING state the quality of service to the targets affected by the change is undefined. 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. To keep the figure simple, only the primitives that cause state transitions are represented.

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.

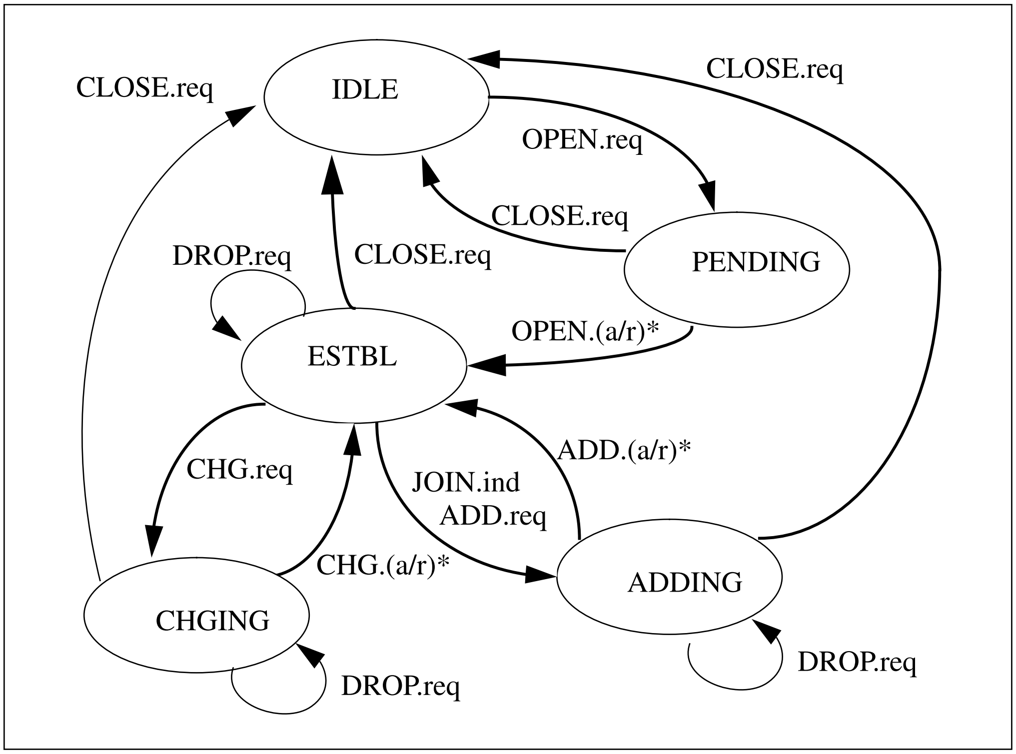

Figure 5: ST Service at the Origin

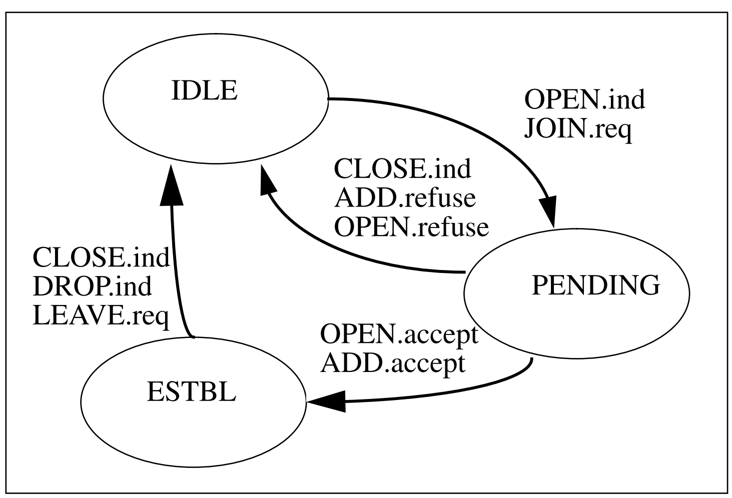

Figure 6: ST Service at the Target

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

Table 3: Primitives and States at the Target

3 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, cf. Section 7.3.

This section contains a functional description of 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 7. The topology is used to illustrate the protocol interactions during the execution of stream operations. For instance, an ST application may:

- 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.

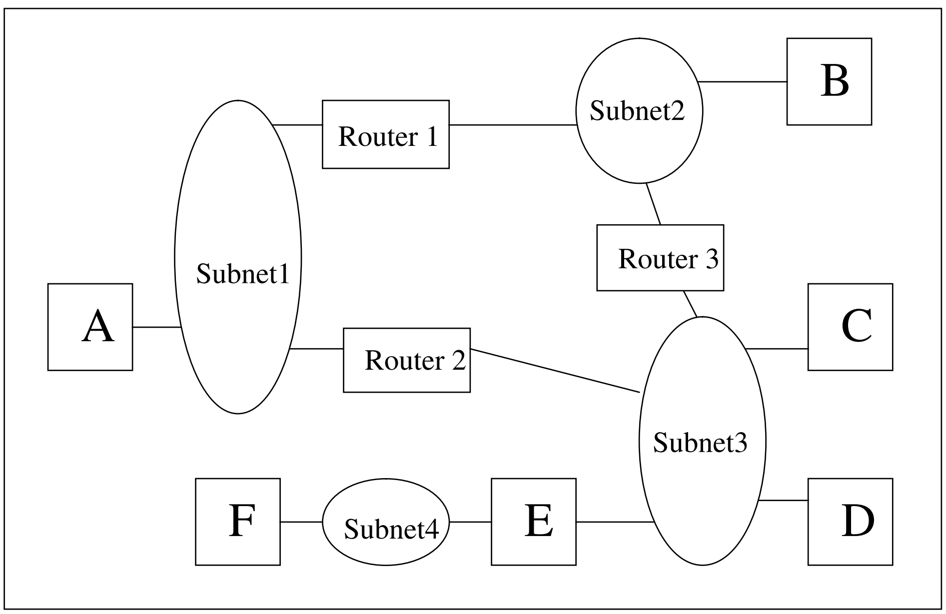

Figure 7: Sample Topology for an ST Stream

We begin with a description of stream setup, see Section 3.1; stream option are presented in Section 3.2; data transfer in Section 3.3; Section 3.4 illustrates stream modification including stream expansion, reduction, changes of the quality of service associated to a stream. Finally, stream deletion is handled in Section 3.5.

3.1 Stream Setup

This section presents a description of stream setup. For simplicity, we assume that everything succeeds, e.g. any required resources are available, and the routing is correct. Possible failures in the setup phase are handled in Section 4.1.

3.1.1 Initial Setup at the Origin

Before stream setup can be started, the application has to collect the necessary information to determine the structure of the communication. This includes identifying the participants and selecting the characteristics of the data flow. Such information is passed to the ST agent at the stream's origin. The ST agent performs the following operations:

- allocates a stream ID (SID) for the stream, cf. Section 7.1,
- invokes the routing function to determine the set of next-hops for the stream, cf. Section 3.1.1.1,
- invokes the Local Resource Manager (LRM), cf. Section 3.1.1.2, to reserve local and network resources
- 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, see Section 3.1.2.

3.1.1.1 Invoking the Routing Function

An ST agent that is setting up a stream invokes the routing function to find a path 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 rather than to a single destination. The routing function is not part of the ST protocol and therefore it is not specified by this document.

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.

3.1.1.2 Reserving Resources

An ST agent helps reserving both *local* and *network* resources. Local resources may include CPU processing time and buffer space at the local host. Network resources may comprise bandwidth over the outgoing links to the next-hops determined by the routing function. Resource reservation is not part of the ST protocol and therefore it is not specified by this document. ST invokes at every host the **Local Resource Manager (LRM)** to perform the appropriate reservations. Functions as resource scheduling and reservation enforcement are part of the LRM's tasks and not of an ST agent's.

The ST **FlowSpec** contains all the information needed to allocate the necessary resources. The information contained in the FlowSpec is passed to the LRM as parameter of the reservation functions. The LRM updates the FlowSpec information before it passes it back to the ST agent. Further information on the ST FlowSpec can be found in Section .

Note that if the data has to be sent across a network to a single next-hop, then only the point-to-point bandwidth needs to be reserved. If the data has to be sent to multiple next-hop ST agents across a single network and network layer multicasting is not available, the ST agent replicates the data to each next-hop ST agent and therefore bandwidth has to be reserved by the LRM for all the next-hops. If network layer multicast is supported, its use reduces the bandwidth required since one single copy of the data is received by all next-hop ST agents. The membership of a stream in a Group may also affect the amount of resources that have to be allocated by the LRM, cf. Section 6.

Effects similar to reservation of the necessary resources may be obtained even when the network cannot provide direct support for the reservation. Certainly if total reservations are a small fraction of the overall resources, such as packet switch processing bandwidth, buffer space, or network bandwidth, then the desired performance can be honoured if the degree of confidence is consistent with the requirements as stated in the FlowSpec. Other solutions can be designed for specific networks.

3.1.2 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, an updated FlowSpec, and a TargetList. 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 3.1.2.1.; if the TargetList causes a too long CONNECT message to be generated, the CONNECT message is partitioned as explained in Section 3.1.2.2. 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.

Let us consider the network topology in Figure 7 on page 16. Suppose that the original TargetList contains targets B, C, and D. The routing function invoked at A returns that B is reachable via Router 1 and C and D are reachable via Router 2. Thus, A generates two CONNECT messages, one for Router 1 and one for Router 2. The CONNECT message for Router 1 contains a TargetList including target B only; the CONNECT message for Router 2 contains a TargetList including targets C and D.

3.1.2.1 Empty Target List

An application at the origin may request the local ST agent to create *empty streams*. 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.

3.1.2.2 Long Target Lists

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. SCMP messages with long TargetList can cause the size of the SCMP message to exceed the network MTU. The ST agent which receives an SCMP message bigger than its MTU must break the original message into multiple fragments, each carrying part of the TargetList. The effect of this partition is to compromise the performance but still carry out the function of the SCMP message. If the original SCMP message contains any Userdata parameters, these parameters are replicated in each fragment for delivery to all targets. Applications that support a large number of receivers may avoid using long target lists by exploiting the stream joining functions, cf. Section 3.4.2.

3.1.3 Processing CONNECT Messages

3.1.3.1 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 to which it corresponds by including the reference number indicated by the Reference field of the CONNECT message. The intermediate ST agent invokes the routing function, reserves resources via the LRM, and then propagates the CONNECT messages to its next-hops, as described in the previous section.

3.1.3.2 Setup 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 reserves local resources and inquires from the specified application process whether or not it is willing to accept the connection.

In particular, the application must be presented with parameters from the CONNECT, such as the SID, FlowSpec, Options, and Group, to be used as a basis for its decision. The application is identified by a combination of the NextPcol field and the 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 was received 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 to which the ACCEPT (or REFUSE) is a reply is identified by placing the CONNECT’s Reference number in the LnkReference field of the ACCEPT (or REFUSE). The ACCEPT message contains the FlowSpec as accepted by the application at the target.

3.1.4 Processing ACCEPT Messages

3.1.4.1 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 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 by the ST agent when the CONNECT was originally processed. However, the ST agent does not adjust the reservation in response to the ACCEPT. It is expected that any excess resource allocation will be released for use by other stream or datagram traffic through an explicit CHANGE message initiated by the application at the origin if it does not wish to be charged for any excess resource allocations.

3.1.4.2 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. For problems due to CONNECT timeout, please refer to Section 4.1.1.

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 3.3 and in the ST user service description in Section 2.

3.1.5 Processing REFUSE Messages

3.1.5.1 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, considers which resources are to be released, deletes the target entry from the internal database, and propagates the REFUSE message back to the previous-hop ST agent.

If, after deleting the specified target, the next-hop has no remaining targets, then those resources associated with that next-hop ST agent may be released. Note that network resources may not actually be released if network multicasting is being used since they may still be required for traffic to other next-hops in the multicast group.

3.1.5.2 REFUSE Processing by the Origin

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 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 stream joining as described in Section 3.4.2.

3.2 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(s). When a CONNECT message reaches a target, the application at the target is notified of the stream options that have been selected.

3.2.1 No Recovery

The NoRecovery option is used to indicate that ST agents should not attempt recovery in case of network or component failure. If a failure occurs, the origin will be notified via a REFUSE message and the targets via a DISCONNECT, with an appropriate ReasonCode indicating the reason of the failure. The application at the origin may decide whether to rebuild the deleted portion of the stream by sending a CONNECT message. The NoRecovery option is specified by setting the S-bit in the CONNECT message, see Section 11.4.

3.2.2 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 3.4.2. The join authorization level for a stream is defined by the J-bit and N-bit in the CONNECT message header, see Section 11.4. 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 - Ask Origin** (JN = 01): The application at the stream origin is asked whether the new target is allowed to join the stream.
- **Level 2 - OK, Notify Origin** (JN = 10): The targets joins the stream. The origin is notified that the target has joined.
- **Level 3 - OK** (JN = 11): The targets joins the stream. No notifications are sent to the stream origin.

3.3 Data Transfer

An application is not guaranteed that the data reaches its destinations: ST is unreliable and it does not make any attempt to recover from packet loss, e.g. due to the underlying network. In case the data reaches its destination, it does it accordingly to the negotiated quality of service.

An ST agent forwards the data only along already established paths to targets. A path is considered to be established when the next-hop ST agent on the path sends an ACCEPT message. 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 an ST agent will forward data to a next-hop ST agent that has not explicitly accepted the stream.

To be fairly 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 to send data before the completion of stream setup, but it should be aware that the data could be lost or not reach all the intended targets.

At the end of the connection setup phase, the origin, each target, and each intermediate ST agent has a database entry that allows it to forward the data packets from the origin to the targets and to recover from failures of the intermediate ST agents or networks. The database should be optimized 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 database entry must also contain the FlowSpec, utilization information, the address of the origin and previous-hop, and the addresses of the targets and next-hops, so it can perform enforcement and recover from failures.

An ST agent receives data packets encapsulated by an ST header. A data packet received by an ST agent contains the SID. This SID was selected at the origin so that it is globally unique and thus can be used as an index into the database, to obtain quickly the necessary replication and forwarding information.

The forwarding information will be network and implementation specific, but must identify the next-hop ST agents. It is suggested that the cached information for a next-hop agent include the local network address of the next- hop. If the data packet must be forwarded to multiple next-hops across a single network that supports multicast, the database may specify the next-hops by a (local network) multicast address. If the network does not support multicast, or the next-hops are on different networks, multiple copies of the data packet must be sent.

No data fragmentation is supported during the data transfer phase. The application is expected to segment its 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 also Section . The minimum MTU over all paths has to be calculated from the MTUs relative to the single paths. If the application at the origin sends a too large data packet, the ST agent at the origin generates an error and it does not forward the data.

3.4 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. In ST, changes to a stream may be initiated both by the origin and the targets. Targets may be added by the origin as described in Section 3.4.1 or they may request to join the stream as described in Section 3.4.2. The origin can reduce a stream by dropping some or all of its targets. This is described in Section 3.4.3. Targets may spontaneously decide to leave a stream as described in Section 3.4.4. Section 3.4.5 explains how to change a stream's FlowSpec.

As defined by the ST service model, see 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.

3.4.1 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) the FlowSpec of the new target, if present, matches 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, it is simply ignored.

An intermediate ST agent that is already a node in the stream looks at the SID 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 an ERROR message with the appropriate ReasonCode (RouteLoop) that contains a TargetList of those targets that were duplicates.

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. However, it may be possible to route to the new targets using previously allocated paths or an existing multicast group. In that case, additional resources do not need to be reserved but more next-hops might have to be added to an existing multicast group.

Intermediate or target ST agents that are not already nodes in the stream behave as in case of stream setup (see Section 3.1.3.1 and Section 3.1.3.2).

3.4.2 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 together with the FlowSpec. 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. Eventually, an ST agent traversed by the stream or the stream's origin itself is reached. This ST agent responds to the join request based on the join authorization level associated with the stream, cf. Section 3.2.2.:

- level 0 (refuse join)

  It is not allowed to join the stream. No further actions are taken.

- level 1 (ask origin)

  The JOIN message is propagated back until the origin is reached. At the origin, the application is requested to either grant or deny the permission to join the stream. If the permission is denied, no further actions are taken. Otherwise, the origin issues a CONNECT message with a TargetList including the target that requested to join the stream. The target is then added as in normal stream setup.

- level 2 (ok, notify origin)

  The ST agent sends a CONNECT message with a TargetList including the target that requested to join the stream. This results in adding the target to the stream. When the ST agent which is already part in the stream receives the ACCEPT message indicating that the new target has been added, it does not propagate the ACCEPT message backwards. Instead, it issues a NOTIFY message with ReasonCode(TargetJoined) to inform the origin of the new target.

- level 3 (ok)

  The ST agent sends a CONNECT message with a TargetList including the target that requested to join the stream. This results in adding the target to the stream. When the ST agent which is already part in the stream receives the ACCEPT message indicating that the new target has been added, it does not propagate the ACCEPT message backwards, nor it notifies the origin.

3.4.2.1 ST FlowSpec

Some rules on the FlowSpec are defined for targets that join an existing stream. Targets are allowed to express their wishes in terms of FlowSpec as part of the JOIN-REQUEST message, see Section 11.8. Let us call this FlowSpec F_T and the current FlowSpec for the stream F. There are three cases:

- $F > F_T$, i.e., F requires a larger amount of resources to be allocated than F_T. If authorized to join the stream, the target will receive a CONNECT message including FlowSpec F_T.
- $F = F_T$, i.e., F requires the same amount of resources to be allocated as F_T. If authorized to join the stream, the target will receive a CONNECT message including FlowSpec $F_T = F$.

- $F < F_T$, i.e., F requires a smaller amount of resources to be allocated than F_T. If authorized to join the stream, the target will receive a CONNECT message including FlowSpec F. The target may always REFUSE the stream if FlowSpec F is believed to be insufficient, or, it may join the stream and perhaps later request the stream origin to modify the whole stream's FlowSpec. This request has to be sent out-of-band.

If the target does not specify any FlowSpec, i.e., no FlowSpec parameter is included in the JOIN-REQUEST message, see Section 11.8, it will receive FlowSpec F as part of the CONNECT message.

3.4.2.2 Router as Origin

When join authorization level 3 is chosen, see Section 3.2.2 and Section 3.4.2, it is possible that the stream origin is unaware that a target participates in the stream. In this case, the router 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 router must disconnect the target.
- if the stream FlowSpec is changed, the router must change the FlowSpec for the target as appropriate.

Of course, the router behaves normally for all other targets added to the stream as a consequence of a CONNECT message issued by the origin. Note that, for targets that joined a stream without notifying the origin, some operations are not possible. For instance, the origin cannot explicitly drop these targets from the stream.

3.4.3 The Origin Removing Targets

The application at the origin specifies a set of targets that are to be removed from the stream and an appropriate ReasonCode (ApplDisconnect). The targets are partitioned into multiple DISCONNECT messages based on the next-hop to the individual targets. If the TargetList is too long to fit into one DISCONNECT message, it is partitioned as described in Section 3.1.2.2.

An ST agent that receives a DISCONNECT message acknowledges it by sending an ACK back to the previous-hop. The DISCONNECT is also propagated to the relevant next-hop ST agents. Before propagating the message, the TargetList is partitioned based on next-hop ST agents.

If, after deleting the specified targets, any next-hop has no remaining targets, then those resources associated with that next-hop ST agent may be released. Note that network resources may not actually be released if network multicasting is being used since they may still be required for traffic to other next-hops in the multicast group.

When the DISCONNECT reaches a target, the target sends an ACK and notifies the application that it is no longer part of the stream and for which reason. The ST agent at the target deletes the stream from its database after performing any necessary management and accounting functions. Note that the stream is not deleted if the ST agent is also a router for the stream and there are remaining downstream targets.

3.4.4 A Target Deleting Itself

The application at the target may inform ST that it wants to be removed from the stream and the appropriate ReasonCode (ApplDisconnect). The ST agent then forms a REFUSE message with itself as the only entry in the TargetList. The REFUSE 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 such a REFUSE message acknowledges it by sending an ACK to the next-hop. The target is deleted and, if the next-hop has no remaining targets, then the resources associated with that next-hop ST agent may be released. Note that network resources may not actually be released if network multicasting is being used since they may still be required for traffic to other next-hops in the multicast group. The REFUSE is also propagated back to the previous-hop ST agent.

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.

3.4.5 Changing a Stream's FlowSpec

The application at the sender may wish to change the FlowSpec of an established stream. To do so, it informs the ST agent at the origin of the new FlowSpec and of the list of targets relative to the change. The origin then issues one CHANGE message with the new FlowSpec per next-hop and sends it to the relevant next-hop ST agents. CHANGE messages are structured and processed similarly to CONNECT messages.

A next-hop ST agent that is an intermediate ST agent and receives a CHANGE message similarly determines if it can implement the new FlowSpec along the hop to each of its next-hop ST agents, and if so, it propagates the CHANGE messages along the established paths. If this process succeeds, the CHANGE messages will eventually reach the targets, which will each respond with an ACCEPT (or REFUSE) message that is propagated back to the origin.

If the change to the FlowSpec is in a direction that makes fewer demands of the involved networks, then the change has a high probability of success along the path of the established stream. Each ST agent receiving the CHANGE message makes the necessary requested changes to the network resource allocations, and if successful, propagates the CHANGE message along the established paths. If the change cannot be made then the ST agent must recover using DISCONNECT and REFUSE messages as in the case of a network failure, see Section 5.2. Note that a failure to change the resources requested for specific targets should not cause other targets in the stream to be deleted.

3.5 Stream Tear Down

A stream is usually terminated by the origin when it has no further data to send, but may also be partially torn down by the individual targets. These cases will not be further discussed since they have already been described above.

A stream is also torn down if the application should terminate abnormally. Processing in this case is identical to the previous descriptions except that the ReasonCode (ApplAbort) is different.

When all targets have left a stream, the origin notifies the application of that fact, and the application then is responsible for terminating the stream. Note, however, that the application may decide to add targets to the stream instead of terminating it.

4 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 are lost, the requested resources are not available, the routing fails or is inconsistent.

4.1 Setup Failures

4.1.1 Setup Failure due to CONNECT Timeout

When sending a CONNECT message, an ST agent expects an ACK from the next hop ST agent. If the CONNECT fails due to timeout (see Section 7.3), the ST agent sends a REFUSE message back in the direction of the origin with the appropriate ReasonCode (ConnectTimeout).

4.1.2 Setup Failure due to ACCEPT Timeout

An ST agent that propagates an ACCEPT message backward toward the origin expects an ACK from the previous hop ST agent. If the ACCEPT fails due to timeout (see Section 7.3), the ST agent replaces the ACCEPT with a REFUSE and sends a DISCONNECT in the direction toward the target. Both REFUSE and DISCONNECT must identify the affected targets and specify the appropriate ReasonCode (AcceptTimeout).

4.1.3 Setup Failure 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. an erroneous 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 targets 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 4.2.2 on path convergence. The remaining cases requiring recovery, a partially failed stream and an erroneous 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 neighbourhood 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, cf. Section 7.3 and Section 4.1.1. 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 error code (RouteLoop).

The REFUSE message with this error code (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 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.

4.2 Further Issues

4.2.1 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.

4.2.2 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. Other cases are discussed in Section 4.1.3.

This version of ST does not allow streams which have converged path, i.e streams are always tree-shaped and not graph-like. The ST agent which detects this condition informs the previous hop ST agent (the latest ST agent to send the CONNECT message) by sending a NOTIFY message with ReasonCode(PathConverge). Upon receipt of the NOTIFY message, the previous hop ST agent will find alternate route to the listed targets with a different next hop ST agent. If there is no next hop ST agent other than the one it receives the NOTIFY message from, the ST agent must release any resources reserved for the listed targets and send a REFUSE message with ReasonCode(PathConverge) to its previous hop ST agent. In the same way, the REFUSE message is possibly propagated back by each ST agent. At each ST agent, it causes any resources reserved exclusively for the listed targets to be released. When the REFUSE reaches the origin, the ST agent at the origin should attempt to send a CONNECT with the listed targets to a different route. If no route exists, or after NRetryRoute tries, the application should be informed so that it may take whatever actions it seems necessary.

4.2.3 Problems in Reserving Resources

If the local or network 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 3.1.
- *preempt one or more of the already established streams:* this way, the ST agent attempts to free enough resources to allow for the new stream to be established. Stream preemption is discussed in Section 5.3.
- *refuse to establish the stream along this path:* the origin ST agent informs the application of the stream setup failure; an ST agent at a router or target issues a REFUSE message (as described in Section 3.1.5) with ReasonCode (CantGetResrc).

It depends on the local implementations whether an ST agent tries alternative paths or preempts other streams. Also, the order of the actions taken is not defined here. In any case, if enough resources cannot be found over different paths or as a consequence of stream preemption, the ST agent has to explicitly refuse to establish the stream.

4.2.4 Problems Caused by CHANGE Messages

A CHANGE might fail for several reasons, including:

- the request may be for a larger amount of network resources when those resources are not available;
- it might be required that all the former resources are released before the new ones are requested and, due to unlucky timing, an unrelated request for network resources might be processed between the time the resources are released and the time the new resources are requested, so that the former resources are no longer available.

If the attempt to change the FlowSpec fails then the ST agent where the failure occurs must intentionally break the affected portion of the stream. This is done by sending REFUSE and DISCONNECT messages with ReasonCode (ChgFailed).

5 Failure Detection and Recovery

5.1 Failure Detection

The ST 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 5.2 are initiated by the ST agents.

5.1.1 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. In this case, the ST agent may request the network to allocate bandwidth anew. 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.

5.1.2 Detecting ST Agents Failures

Each ST agent periodically sends each neighbour 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 neighbour 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 need to participate in this exchange and it is an error to send a HELLO message to a neighbour ST agent with no streams in common, e.g. to check whether it is active. Note that STATUS messages can be used to poll neighbour ST agents.

A HELLO message is ACKed if the Reference field is non-zero. As well as identifying the sender, 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, but it is allowable for an ST agent to create a new *HelloTimer* only when it sends a HELLO message. 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* for the following *HelloTimerHoldDown* seconds.

Each ST stream has a *RecoveryTimeout* value associated with it. This value is assigned by the origin and carried into the CONNECT message, see Section 11.4.

An ST agent must send HELLO messages to its neighbour 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.

Furthermore, 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 ST state. If it shares streams with that neighbor, it must initiate stream recovery activity. 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 (RemoteRestart). If it receives a CONNECT message while its own *Restarted-bit* is set, it must respond with ERROR with the appropriate ReasonCode (RestartLocal).

5.2 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 5.1.

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 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 may 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 it should first send a REFUSE message with the appropriate ReasonCode ("failure") 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 send DISCONNECT messages with the same ReasonCode toward the targets.

  The intermediate ST agents process this DISCONNECT message just like the DISCONNECT that tears down the stream. However, a target ST agent that receives a DISCONNECT message with the appropriate ReasonCode ("failure") will maintain the stream state and notify the next higher protocol of the failure. In effect, these DISCONNECT messages tear down the stream from the point of the failure to the targets, but inform the targets that the stream may be fixed shortly.

- 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 reconstruct the stream if the NoRecovery option is not selected. If it cannot communicate with the next-hop, then the ST agent detecting the failure releases any resources that are dedicated exclusively to sending data on the broken branch and sends a DISCONNECT message with the appropriate ReasonCode ("failure") 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.

The ST agent that is the previous-hop before the failed component can attempt to recover the streams for which the NoRecovery option is not selected:

- **If the NoRecovery option is selected,** then the ST agent sends a REFUSE message with the appropriate ReasonCode ("failure") to the previous-hop. The TargetList in these messages contains all the targets that were reached through the broken branch. 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, which can attempt recovery of the stream by sending a new CONNECT to the affected targets. 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 stream. It does so 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 (“failure”) 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.

5.2.1 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 with the appropriate ReasonCode (“failure”). 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. 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.

5.3 Stream Preemption

An intermediate ST agent may decide to break a stream intentionally. This is called stream preemption. Usually streams are preempted in order to free resources for a new stream which has a higher priority. ST does not define when stream preemption should be used but it provides the means to implement it.

If an ST agent 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. ST provides two ways of optimizing such decision:

1. Streams can be assigned an *StreamImportance* value from 0 (most important) to 7 (least important). This value is carried in the CONNECT message when the stream is setup, see Section 11.4.

2. 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 6. This helps making a good choice when more than one stream have to be preempted, because it leads to breaking a single application as opposite to as many applications as the number of preempted streams.

Stream preemption requires the following actions from the ST agents:

- An **intermediate ST agent that breaks the stream intentionally** 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, cf. Section 5.2. If the NoRecovery option is set, is propagates the REFUSE message back to the origin. If the NoRecovery option is not set, it attempts to rebuild the deleted paths and, in case this does not work, it propagates the REFUSE message to the previous-hop.
- A **target or next-hop ST agent of the preempted stream** acts as in case of failure recovery, cf. Section 5.2. It releases resources that are allocated to the stream, but it maintains the internal state information describing the stream for some time in case the stream is quickly fixed.

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).

6 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 is unique across the Internet.

Streams belong to the same group if they have the same *GroupName* in the *GroupName* field of the *Group* parameter. 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. The group name has to be generated as described in Section 6.1. Relationships defined by this version of the protocol are listed in Section 6.2.

6.1 Group Name Generator

The *GroupName* includes a 16-bit unique identifier, a 32-bit IP address, and a 32-bit creation timestamp. It is defined in Section 6.3. 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() -> result

6.2 Basic ST Relationships

This version of ST defines four basic relationships. An ST2+ implementation must support all four basic relationships. The basic relationships are described in detail below in Section 6.2.1 - Section 6.2.4.

ST provides the means to define new relationships as the need for them becomes clear in the future. This can be done by assigning one of the unused bits of the *Relationship* field of the *Group* parameter.

The next sections describe the four basic relationships.

6.2.1 Bandwidth Sharing

Streams belonging to this group share the same network bandwidth. This is intended to support applications as audio conferences where, of all participants, only some at a time are allowed to speak. In such a scenario, global bandwidth utilization can be optimized, e.g. it is sufficient to reserve bandwidth for a small set of audio streams.

The N parameter indicates for how many streams at the same time bandwidth should be allocated. An ST agent allocates N times the bandwidth required by the most demanding stream in the group, say B_{max}. If the application intends for instance to allow three participants to speak at the same time, N has a value of three and the ST agent will allocate for the group an amount of bandwidth equal to $3*B_{max}$.

This mechanism does not always allocate an optimal amount of bandwidth (as when a stream requires 4 Mbits/s and all the other streams in the same group require 1 Mbits/s only: N=3 causes the allocation of 12 Mbits/s). However, it is simple to implement and it works well with streams that have homogeneous requirements. An alternative would be to keep track of the single streams requirements and allocate the exact amount of bandwidth.

An ST agent always attempts to reserve $N*B_{max}$ bandwidth. If less bandwidth than $N*B_{max}$ is available, the new stream is not built. If bandwidth for the group has already been allocated and a new stream with a bandwidth demand inferior to B_{max} is being established, the ST agent, depending on the local implementation, may not need to contact the local resource manager and it can proceed directly with the stream setup.

Note that ST agents 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.

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

6.2.3 Route Sharing

Streams belonging to this group share the same paths. 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 the shared path does not normally cause the deletion of the stream: the stream is built over an alternative path whenever possible.

6.2.4 Subnet Resources Sharing

Streams belonging to this group share the same MAC layer subnetwork addresses. As an example, the same MAC layer multicast address can 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.

This relationship provides a hint to the data link layer functions.

6.3 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. 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 obtained in ST.

As new relationships are defined, it should be indicated whether they are or not orthogonal with respect to the previously defined ones. This will be reflected by illegal values for the *Relationship* field of the *Group* parameter (see Section 10.3.3).

7 Ancillary Functions

7.1 Stream IDs Generation

To Be Written

7.2 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.

7.3 SCMP Reliability

The ST Control Message Protocol is made reliable through the use of retransmission when response is not received in a timely manner. In general, when sending a SCMP messages which requires an ACK back, 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 back 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 message. In some cases (CONNECT,ACCEPT) the ST agent handles the timeout by sending additional SCMP message (REFUSE/DISCONNECT) to its neighbour ST agents (see Section 4.1.1 & Section 4.1.2), while in other cases (REFUSE, DISCONNECT) it simply gives up sine there is nothing else it can do.

For some SCMP messages (CONNECT,CHANGE) the sending ST agent also expects a response back (ACCEPT/REFUSE) after ACK has been received. For these cases, the ST agent needs to set the Rtoxxxx timer after it receives the ACK. If it does not receive the appropriate response back when Rtoxxxx expires, the ST agent updates its state data and perform appropriate recovery action as described in other sections.

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 maintain the robustness of the protocol. Algorithms on this subject are described in [WoHD95], [Jaco88], [KaPa87].

7.4 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 as part of the parameter in the CONNECT message and forwards it to the next hop ST agents. Each ST agent in the path checks to see if its network MTU is smaller than the one specified in the CONNECT message and, if it is, the ST agent updates the MTU in the CONNECT message accordingly. 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 MTU 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 MTU received from all the ACCEPT messages. It is the responsibility of the application to segment its PDUs according to the minimum MTU of the stream since no data fragmentation is supported during the data transfer phase.

7.5 Packet Discarding on Network Congestion

TBD

7.6 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 to indicate an ST packet is enclosed, as opposed to TCP or UDP, for example. The assignment of protocol 5 to ST is an arranged coincidence with the assignment of IP Version 5 to ST [RFC1190].
- 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 (usually low delay, high throughput, normal reliability). 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 ST agents make encapsulation decision based on information provided by routing function to indicate whether the routers in the path support ST. It is outside the scope of this document to address routing function and therefore neither its algorithm nor implementation is specified here. ST assumes that appropriate routing algorithm exists to which ST has access.

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

7.7 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 10.3.4) 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.

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

224.0.0.7 All ST routers

224.0.0.8 All ST hosts

In addition, a block of transient IP multicast addresses, 224.1.0.0 - 224.1.255.255, has been allocated for ST multicast groups. Note that in the case of Ethernet, an ST multicast address of “224.1.cc.dd” maps to an Ethernet multicast address of “01:00:5E:01:cc:dd”, see [RFC1112].

7.8 Routing

ST requires access to routing information in order to select a path from an origin to the destination(s). However, routing is considered to be a separate issue and neither the routing algorithm nor its implementation is specified here. ST should operate equally well with any reasonable routing algorithm.

While ST may be capable of using several types of information that are not currently available, the minimal information required is that provided by IP, namely the ability to find an interface and next hop router for a specified IP destination address and Type of Service. Methods to make more information available and to use it are left for further study. For initial ST implementations, any routing information that is required but not automatically provided will be assumed to be manually configured into the ST agents.

8 FlowSpec

The FlowSpec is used to convey stream service requirements end-to-end. The contents of the FlowSpec are transparent to the ST agents. An ST agent extracts the FlowSpec from the correspondent incoming SCMP message and passes it to the LRM as required. The LRM updates the FlowSpec values based on the amount of resources that it has allocated to the stream.

8.1 FlowSpec Versions

ST is not dependent on a particular FlowSpec format and it is expected that other versions of the FlowSpec than those introduced below in this section will be needed in the future. Different FlowSpec formats are distinguished by the value of the Version field. The following values are reserved:

0 - Null FlowSpec /* mandatory */

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 /* mandatory */

A single stream is always associated to a single FlowSpec format. Changes to the FlowSpec are also relative to the same FlowSpec format, i.e. the value of the Version field cannot be changed during the lifetime of the stream.

8.2 The Null FlowSpec (#0)

The FlowSpec identified by a value of 0 for its Version field is called the “Null FlowSpec”. An ST agent that receives the Null FlowSpec always assumes that sufficient resources for the stream are available. The Null FlowSpec fields values are never updated. Stream setup takes place in the usual way, but no resources are actually reserved.

The main 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 complete format is specified in Section 10.3.2.

8.3 The ST Current FlowSpec (#7)

FlowSpec #7 is the FlowSpec to be used by the current version of ST. It contains values for 3 basic QoS parameters: message size, throughput, and delay. Also, it is possible to specify a QoS class, e.g. guaranteed. Each parameter has to be expressed via a set of values:

- the “desired” values are assigned by the application and never changed by the LRM
- the “limit” values are assigned by the application and never changed by the LRM

- the “actual” values indicate the guarantees 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” values.

8.3.1 Qos Classes

We also define two QoS classes:

1. QOS_GUARANTEED
2. QOS_PREDICTIVE

- The QOS_GUARANTEED service class implies that the negotiated QoS for the stream is never violated during the data transfer. For instance, the desired rate is the peak rate for the transmission. This may sometimes lead to overbooking of resources, but it provides strict real-time guarantees.
- The QOS_PREDICTIVE service class implies that the negotiated QoS may be violated for short time intervals. Reservations are done for the average case as opposite to the peak case required by the QOS_GUARANTEED service class.

If a LRM that doesn’t support class QOS_PREDICTIVE (QOS_GUARANTEED) receives a FlowSpec containing a QOS_PREDICTIVE (QOS_GUARANTEED) class, it informs the local ST agent. The ST agent may try different paths or delete the correspondent portion of the stream with ReasonCode (QoSClassUnknown).

8.3.2 Maximum Message Size

This parameter is expressed in bytes. It represents the maximum size allowed for messages sent 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.

8.3.3 Rate or Throughput

This parameter is expressed in bytes/seconds. 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.

8.3.4 Maximum Delay and Delay Jitter

This parameter is expressed in milliseconds. It represents the maximum end-to-end 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 rate 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 the MinDelay field by adding the minimum possible delay to the next-hop. Information on the minimum possible delay allows to calculate another important QoS parameter, the delay jitter.

The complete format is specified in Section 10.3.2.

9 ST State Machines

To be separately published.

10 ST Protocol Data Units

All ST packets arrive at the same Network Service Access Point (NSAP) that IP uses to receive IP datagrams, e.g., ST would use the same ethertype (0x800) as does IP. The two types of packets are distinguished by the IP Version Number field, i.e. the first four (4) bits of the packet; IP currently uses a value of 4, while ST has been assigned the value 5 (see [RFC791]). There is no requirement for compatibility between IP and ST packet headers beyond the first four bits.

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 8. Please refer to Section 13 for an explanation of the notation.

<table>
<tr><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th>1</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th>2</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th>3</th><th></th></tr>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>0</th><th>1</th></tr>
<tr><td colspan="4">ST = 5</td><td colspan="4">Ver = 3</td><td colspan="1">D</td><td colspan="3">Pri</td><td colspan="4">MBZ</td><td colspan="16">TotalBytes</td></tr>
<tr><td colspan="16">UniqueID</td><td colspan="16">HeaderChecksum</td></tr>
<tr><td colspan="32">OriginIPAddress</td></tr>
</table>

Figure 8: 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, to be used as described in Section 7.5.
- **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 7.2.
- **UniqueID** is the first element of the stream id (SID). It is locally unique at the stream origin, see Section 7.1.
- **OriginIPAddress** is the second element of the SID. It is the 32-bit IP address of the stream origin, see Section 7.1.

Bits 12-15 must be set to zero (0) in the current ST version. In the future, it is possible that this 4-bit field will be used for substream filtering, e.g., as described in [WoHD95].

10.1 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.

10.1.1 Stream ID

The UniqueID and OriginIPAddress fields form the Stream ID (SID), which is used by the ST agents to identify which stream the data packets belong to. Within the same stream, the same SID is used in data packets and control messages.

NOTE: In some exceptional situations, e.g. usually due to a crash and subsequent reboot, it is possible that an ST agent receives a data packet belonging to a stream of which the ST agent has lost state information. In this case, the ST agent is not able to forward the packet and has to discard it. Since SIDs include the 32-bit IP address of the stream origin, it is possible for the ST agent to disconnect from the stream.

10.2 ST Control Messages

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 7.3) 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 3.1.2.2). All control messages have the following format:

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">OpCode</td><td colspan="8">Options</td><td colspan="16">TotalBytes</td></tr>
<tr><td colspan="16">Reference</td><td colspan="16">LnkReference</td></tr>
<tr><td colspan="32">SenderIPAddress</td></tr>
<tr><td colspan="16">Checksum</td><td colspan="16">ReasonCode</td></tr>
</table>

<table>
<tr><td>:
OpCodeSpecificData
:</td></tr>
</table>

Figure 9: 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 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 10.3.7.
- **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 any of the optional parameters 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 ErroredPDU

The ErroredPDU parameter (PCode = 1) is used for diagnostic purposes to encapsulate a received ST PDU that contained an error. It may be optionally included in the ERROR message. Its use is primarily diagnostic.

0–7	8–15	16–23	24–31
PCode = 1	PBytes	PDUBytes	0
: PDUInError :			

Figure 10: ErroredPDU

- **PDUBytes** indicates how many bytes of the PDUInError are actually present.
- **PDUInError** is the PDU in error, beginning with the ST Header.

10.3.2 FlowSpec

The FlowSpec parameter (PCode = 2) is used in several messages to convey the FlowSpec. The format of the FlowSpec field depends on the Version field. Two FlowSpec versions must be implemented in the current ST version, FlowSpec #0 and FlowSpec #7, see Section 8. Their format is described below.

0–7	8–15	16–23	24–31
PCode = 2	PBytes = 4	Version = 0	0

Figure 11: FlowSpec #0

0–7	8–15	16–23	24–31
PCode = 2	PBytes	Version = 7	QoSClass
DesRate			
LimitRate			
ActRate			
DesMaxSize		LimitMaxSize	
ActMaxSize		DesMaxDelay	
LimitMaxDelay		ActMaxDelay	
DesMaxDelayRange		ActMinDelay	

Figure 12: FlowSpec #7

- **QoSClass** indicates which of the two defined classes of service applies. The two classes are: QOS_GUARANTEED (QoSClass = 1) and QOS_PREDICTIVE (QoSClass = 2).
- **DesRate** is the desired transmission rate for the stream in bytes/second. This field is set by the origin and is not modified by intermediate agents.
- **LimitRate** is the minimum acceptable transmission rate in bytes/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 bytes/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 message size in bytes that will be sent in the stream. This field is set by the origin.
- **LimitMaxSize** is the minimum acceptable message size in bytes. This field is set by the origin
- **ActMaxSize** is the actual maximum message size that may be sent 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
- **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 agent. If the actual maximum exceeds LimitMaxDelay, then the connection is refused.
- **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 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 agent. 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.

10.3.3 Group

The Group parameter (PCode = 3) is an optional argument used to indicate that the stream is a member in the specified group.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="8">PCode = 3</td><td colspan="8">PBytes = 16</td><td colspan="16">GroupUniqueID</td></tr>
<tr><td colspan="32">GroupInitiatorIPAddress</td></tr>
<tr><td colspan="32">GroupCreationTime</td></tr>
<tr><td colspan="16">Relationship</td><td colspan="16">N</td></tr>
</table>

Figure 13: Group Parameter

- **GroupUniqueID**, **GroupInitiatorIPAddress**, and **GroupCreationTime** together form the *GroupName* field. They are allocated by the group name generator function, see Section 6.1.
- **Relationship** has the following format:

<table>
<tr><td>0 (unused)</td><td>S</td><td>P</td><td>F</td><td>B</td></tr>
</table>

Figure 14: Relationship Field

The B, F, P, S bits correspond to Bandwidth, Fate, Path, and Subnet resources sharing, see Section 6.2. 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 6.2.1.

10.3.4 MulticastAddress

The MulticastAddress parameter (PCode = 4) is an optional parameter that is used when setting up a network level multicast group, to communicate an IP and/or local network multicast address to the next-hop ST agents that should become members of the group, see Section 7.7.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="8">PCode = 4</td><td colspan="8">PBytes = 8</td><td colspan="16">0</td></tr>
<tr><td colspan="32">IPMulticastAddress</td></tr>
</table>

Figure 15: MulticastAddress

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

10.3.5 NextHopIPAddress

The NextHopIPAddress parameter (PCode = 5) is an optional parameter of NOTIFY and contains the 32-bit IP address of a suggested next-hop ST agent.

0–7	8–15	16–23	24–31
PCode = 5	PBytes = 8	0	
Next-Hop IP Address			

Figure 16: NextHopIPAddress

- **NextHopIPAddress** is the 32-bit IP address of the suggested next-hop ST agent.

10.3.6 Origin

The Origin parameter (PCode = 6) is used to identify the next higher protocol, and the SAP being used in conjunction with that protocol.

0–7	8–15	16–23	24–31
PCode = 6	PBytes	NextPcol	OriginSAPBytes
: OriginSAP (# Padding #) :			

Figure 17: 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 [RFC791].
- **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.

10.3.7 ReasonCode

TBD: review ReasonCodes

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:

NoError	0	No error has occurred.
ErrorUnknown	1	An error not contained in this list has been detected
AcceptTimeout	2	An Accept has not been acknowledged.
AccessDenied	3	Access denied.
AckUnexpected	4	An unexpected ACK was received.
ApplAbort	5	The application aborted the stream abnormally.
ApplDisconnect	6	The application closed the stream normally.
AuthentFailed	7	The authentication function failed.
CantGetResrc	8	Unable to acquire (additional) resources.
CantRelResrc	9	Unable to release excess resources.
CksumBadCtl	10	Control PDU has a bad message checksum.
CksumBadST	11	PDU has a bad ST Header checksum.
DuplicateIgn	22	Control PDU is a duplicate and is being acknowledged.
DuplicateTarget	23	Control PDU contains a duplicate target, or an attempt to add an existing target.
FailureRecovery	24	A notification that recovery is being attempted.
FlowVerBad	25	Control PDU has a FlowSpec Version Number that is not supported.
GroupUnknown	26	Control PDU contains an unknown Group Name.
SIDUnknown	29	Control PDU contains an unknown SID.
InconsistGroup	31	An inconsistency has been detected with the streams forming a group.
IntfcFailure	32	A network interface failure has been detected.
InvalidSender	34	Control PDU has an invalid SenderIPAddress field.
InvalidTotByt	35	Control PDU has an invalid TotalBytes field.
LnkRefUnknown	36	Control PDU contains an unknown LnkReference.
NameUnknown	37	Control PDU contains an unknown stream Name.
NetworkFailure	38	A network failure has been detected.
NoRouteToAgent	39	Cannot find a route to an ST agent.
NoRouteToDest	40	Cannot find a route to the destination.
NoRouteToHost	41	Cannot find a route to a host.
NoRouteToNet	42	Cannot find a route to a network.
OpCodeUnknown	43	Control PDU has an invalid OpCode field.
PCodeUnknown	44	Control PDU has a parameter with an invalid PCode.
ParmValueBad	45	Control PDU contains an invalid parameter value.
ProtocolUnknown	46	Control PDU contains an unknown next-higher layer protocol identifier.
ProtocolError	47	A protocol error was detected.
RefUnknown	49	Control PDU contains an unknown Reference.
RestartLocal	50	The local ST agent has recently restarted.
RemoteRestart	51	The remote ST agent has recently restarted.
RetransTimeout	52	An acknowledgment to a control message has not been received after several retransmissions.
RouteBack	53	The routing function indicates that the route to the next-hop is through the same interface as the previous-hop and is not the previous-hop.
RouteInconsist	54	A routing inconsistency has been detected, e.g., a route loop.
RouteLoop	55	A CONNECT was received that specified an existing target.
SAPUnknown	56	Control PDU contains an unknown next-higher layer SAP (port).
STAgentFailure	57	An ST agent failure has been detected.
StreamExists	58	A stream with the given Name or SID already exists.
StreamPreempted	59	The stream has been preempted by one with a higher precedence.
STVerBad	60	A received PDU is not ST Version 3.
TooManySIDs	61	Attempt to add more SIDs to a stream than the implementation supports.

TruncatedCtl	62	Control PDU is shorter than expected.
TruncatedPDU	63	A received ST PDU is shorter than the ST Header indicates.
UserDataSize	64	UserData parameter too large to permit a control message to fit into a network's MTU.
ConnectTimeOut	65	A CONNECT has not been acknowledged
ChgFailed	66	An attempt to change the FlowSpec of existing stream failed
QosClassUnknown	67	QoS class is not supported
PathConverge	68	Two branches of the stream join during the CONNECT setup.

TBD: failure N/A An abbreviation used in the text for any of the more specific errors: DropFailAgt, DropFailHst, DropFailIfc, DropFailNet, IntfcFailure, NetworkFailure, STAgentFailure, FailureRecovery.

TBD: indicate which ReasonCodes can be used in NOTIFY messages, which cause NOTIFY to be propagated back to the Origin.

10.3.8 Target and TargetList

Several control messages use a parameter called TargetList (PCode = 7), 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.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">Target IP Address</td></tr>
<tr><td colspan="8">TargetBytes</td><td colspan="8">SAPBytes</td><td colspan="16">SAP (# Padding #)</td></tr>
</table>

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	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Pcode = 7								PBytes								TargetCount = N															
Target 1																															
:																															
Target N																															

Figure 19: TargetList

10.3.9 UserData

The UserData parameter (PCode = 8) is an optional parameter that may be used by the next higher protocol or an application to convey arbitrary information to its peers. 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 parameter is too large for some network's MTU, an error with ReasonCode(UserDataSize) will occur. The parameter must be padded to a multiple of 32 bits.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Pcode = 8								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.

TBD: GeneratorIPAddress & ReasonCodes

11 ST Control Message PDUs

Each control message is described in a following section. Please refer to Section 13 for an explanation of the notation.

11.1 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.

The ACCEPT message includes the FlowSpec that contains the cumulative information that was calculated by the intervening ST agents as CONNECT (or CHANGE) made its way from the origin to the target, as well as any modifications made by the application at the target. The FlowSpec is not modified on this trip from the target back to the origin.

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.

Since the cumulative FlowSpec information can be different for different targets, no attempt is made to combine the ACCEPTs from the various targets. The TargetList included in each ACCEPT contains the 32-bit IP address of a single target, i.e. the one that issued the ACCEPT.

TBD: include network MTU field, MaxRecoveryTimeout.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="8">OpCode = 1</td><td colspan="8">0</td><td colspan="16">TotalBytes</td></tr>
<tr><td colspan="16">Reference</td><td colspan="16">LnkReference</td></tr>
<tr><td colspan="32">SenderIPAddress</td></tr>
<tr><td colspan="16">Checksum</td><td colspan="16">ReasonCode = 0</td></tr>
<tr><td colspan="32">:
TargetList
:</td></tr>
<tr><td colspan="32">:
FlowSpec
:</td></tr>
</table>

<table>
<tr><td>:
UserData
:</td></tr>
</table>

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).

11.2 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.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="8">OpCode = 2</td><td colspan="8">0</td><td colspan="16">TotalBytes</td></tr>
<tr><td colspan="16">Reference</td><td colspan="16">LnkReference = 0</td></tr>
<tr><td colspan="32">SenderIPAddress</td></tr>
<tr><td colspan="16">Checksum</td><td colspan="16">ReasonCode</td></tr>
</table>

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.

11.3 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 all the stream's targets. CHANGE must be acknowledged with ACK at each hop.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="8">OpCode = 3</td><td>G</td><td colspan="7">0</td><td colspan="16">TotalBytes</td></tr>
<tr><td colspan="16">Reference</td><td colspan="16">LnkReference = 0</td></tr>
<tr><td colspan="32">SenderIPAddress</td></tr>
<tr><td colspan="16">Checksum</td><td colspan="16">ReasonCode = 0</td></tr>
<tr><td colspan="32">:
FlowSpec
:</td></tr>
</table>

<table>
<tr><td>:
TargetList
:</td></tr>
</table>

<table>
<tr><td>:
UserData
:</td></tr>
</table>

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

11.4 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 3.1.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="8">OpCode = 4</td><td colspan="2">JN</td><td colspan="1">S</td><td colspan="5">0</td><td colspan="16">TotalBytes</td></tr>
<tr><td colspan="16">Reference</td><td colspan="16">LnkReference = 0</td></tr>
<tr><td colspan="32">SenderIPAddress</td></tr>
<tr><td colspan="16">Checksum</td><td colspan="16">ReasonCode = 0</td></tr>
<tr><td colspan="32">:
Origin
:</td></tr>
<tr><td colspan="32">:
MulticastAddress
:</td></tr>
<tr><td colspan="32">:
FlowSpec
:</td></tr>
<tr><td colspan="32">:
TargetList
:</td></tr>
<tr><td colspan="32">:
Group
:</td></tr>
<tr><td colspan="32">:
UserData
:</td></tr>
</table>

Figure 24: CONNECT Control Message

- **JN** (bits 8 and 9) indicate the join authorization level for the stream, see Section 3.2.2.
- **S** (bit 10) indicates the NoRecovery option, described in Section 3.2.1.
- **Reference** contains a number assigned by the ST agent sending CONNECT for use in the acknowledging ACK.
- TBD: add MTU field to CONNECT, MaxRecoveryTimeout
- TBD: add stream importance filed for preemption to CONNECT

11.5 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.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="8">OpCode = 5</td><td>G</td><td colspan="7">0</td><td colspan="16">TotalBytes</td></tr>
<tr><td colspan="16">Reference</td><td colspan="16">LnkReference = 0</td></tr>
<tr><td colspan="32">SenderIPAddress</td></tr>
<tr><td colspan="16">Checksum</td><td colspan="16">ReasonCode</td></tr>
<tr><td colspan="32">GeneratorIPAddress</td></tr>
</table>

<table>
<tr><td>:
TargetList
:</td></tr>
</table>

<table>
<tr><td>:
UserData
:</td></tr>
</table>

Figure 25: DISCONNECT Control Message

- **G** (bit 8) is used to request a DISCONNECT of all the stream's targets; the TargetList parameter may be omitted when the G bit is set (1).
- **Reference** contains a number assigned by the ST agent sending DISCONNECT for use in the acknowledging ACK.
- **GeneratorIPAddress** is the 32-bit IP address of the host that first generated the DISCONNECT message.

11.6 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	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
OpCode = 6								0								TotalBytes															
Reference																LnkReference = 0															
SenderIPAddress																															
Checksum																ReasonCode = 0															

:
ErroredPDU
:

Figure 26: ERROR Control Message

- **Reference** is the Reference number of the erroneous request.

11.7 HELLO

HELLO (OpCode = 7) is used as part of the ST failure detection mechanism, see Section 5.1.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
OpCode = 7								R	0							TotalBytes															
Reference																LnkReference															
SenderIPAddress																															
Checksum																ReasonCode = 0															
HelloTimer																															

Figure 27: HELLO Control Message

- **R** (bit 8) is used for the Restarted-bit.
- **Reference** is non-zero to inform the receiver that an ACK should be promptly sent so that the sender can update its round-trip time estimates. If Reference is zero, no ACK should be sent.

TBD: HelloTimer

11.8 JOIN-REQUEST

JOIN-REQUEST (OpCode = 8) is used as part of the ST steam joining mechanism, see Section 3.4.2.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="8">OpCode = 8</td><td colspan="8">0</td><td colspan="16">TotalBytes</td></tr>
<tr><td colspan="16">Reference</td><td colspan="16">LnkReference = 0</td></tr>
<tr><td colspan="32">SenderIPAddress</td></tr>
<tr><td colspan="16">Checksum</td><td colspan="16">ReasonCode = 0</td></tr>
<tr><td colspan="32">GeneratorIpAddress</td></tr>
<tr><td colspan="32">:
TargetList
:</td></tr>
<tr><td colspan="32">:
FlowSpec
:</td></tr>
</table>

Figure 28: JOIN-REQUEST Control Message

- **Reference** contains a number assigned by the ST agent sending JOIN-REQUEST for use in the acknowledging ACK.
- **GeneratorIPAddress** is the 32-bit IP address of the host that first generated the JOIN-REQUEST message.

11.9 NOTIFY

NOTIFY (OpCode = 9) 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 10.3.7; NOTIFY must be acknowledged with an ACK.

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">OpCode = 9</td><td colspan="8">0</td><td colspan="16">TotalBytes</td></tr>
<tr><td colspan="16">Reference</td><td colspan="16">LnkReference = 0</td></tr>
<tr><td colspan="32">SenderIPAddress</td></tr>
<tr><td colspan="16">Checksum</td><td colspan="16">ReasonCode</td></tr>
<tr><td colspan="32">DetectorIpAddress</td></tr>
<tr><td colspan="32">NextHopIPAddress</td></tr>
<tr><td colspan="32">:
TargetList
:</td></tr>
</table>

Figure 29: 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.
- **NextHopIPAddress** is the 32-bit IP address of a suggested next-hop ST agent.
- **TargetList** is present when the notification is related to one or more targets.

11.10 REFUSE

REFUSE (OpCode = 10) 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.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="8">OpCode = 10</td><td colspan="8">0</td><td colspan="16">TotalBytes</td></tr>
<tr><td colspan="16">Reference</td><td colspan="16">LnkReference</td></tr>
<tr><td colspan="32">SenderIPAddress</td></tr>
<tr><td colspan="16">Checksum</td><td colspan="16">ReasonCode</td></tr>
<tr><td colspan="32">GeneratorIpAddress</td></tr>
</table>

Figure 30: REFUSE Control Message

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

11.11 STATUS

STATUS (OpCode = 11) is used to inquire about the existence of a particular stream identified by the SID. Use of STATUS is intended for diagnostic purposes and to assist in stream cleanup operations.

TBD: write section on use of STATUS, STATUS-RESPONSE

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="8">OpCode = 11</td><td>Q</td><td colspan="7">0</td><td colspan="16">TotalBytes</td></tr>
<tr><td colspan="16">Reference</td><td colspan="16">LnkReference = 0</td></tr>
<tr><td colspan="32">SenderIPAddress</td></tr>
<tr><td colspan="16">Checksum</td><td colspan="16">ReasonCode = 0</td></tr>
<tr><td colspan="32">GeneratorIpAddress</td></tr>
</table>

Figure 31: STATUS Control Message

- **Q** (bit 9) is set to one (1) for remote diagnostic purposes when the receiving ST agent should return a stream's parameters, whether or not the source of the message is believed to be a previous-hop or next-hop in the specified stream. Note that this use has potential for disclosure of sensitive information.

11.12 STATUS-RESPONSE

STATUS-RESPONSE (OpCode = 12) 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.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="8">OpCode = 12</td><td colspan="8">0</td><td colspan="16">TotalBytes</td></tr>
<tr><td colspan="16">Reference</td><td colspan="16">LnkReference = 0</td></tr>
<tr><td colspan="32">SenderIPAddress</td></tr>
<tr><td colspan="16">Checksum</td><td colspan="16">ReasonCode = 0</td></tr>
</table>

<table>
<tr><td>:
FlowSpec
:</td></tr>
</table>

<table>
<tr><td>:
Groups
:</td></tr>
</table>

<table>
<tr><td>:
TargetList
:</td></tr>
</table>

Figure 32: STATUS-RESPONSE Control Message

12 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.

12.1 SCMP Messages

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

12.2 SCMP Parameters

1) ErroredPDU
2) FlowSpec
3) Group
4) MulticastAddress
5) NextHopIPAddress
6) Origin
7) TargetList
8) UserData

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.

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

[RFC791]
Postel: Internet Protocol, RFC 791, DARPA, September 1981.

[RFC1060]
Reynolds, Postel: Assigned Numbers, RFC 1060, USC/ISI, March 1990.

[RFC1190]
Topolcic C.: Internet Stream Protocol Version 2 (ST2), October 1990.