Internet-Draft November, 1995
ST Working Group
M. Rajagopal and Sharon Sergeant
File: draft-ietf-st2-state-01.ps Expires: May, 1996

Internet Stream Protocol Version 2 (ST2)
Protocol State Machines - Version ST2+

Status of this Memo

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

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

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

Abstract:

This memo contains a description of state machines for the revised specification of the Internet STream Protocol Version 2 (ST2). The revised version of ST2 specified in this memo is called ST2+ and described in RFC 1819. The state machines in this document are descriptions of the ST2+ protocol states and message sequences specifications for normal behavior. Exception processing issues are defined and discussed for protocol compliance and implementation options.

Editor's Note:

This memo is available both in ASCII format (file: draft-ietf-st2-state-01.txt) and in PostScript (file: draft-ietf-st2-state-01.ps). The PostScript version contains the essential state diagrams and is absolutely required.

TABLE OF CONTENTS

SECTION 1

Introduction

This section gives a brief overview of the ST2 protocol and the protocol FiniteState Machine (FSM) issues addressed in this document. It is assumed that the reader is familiar with the ST2+ Protocol Specification document listed in [1]. Unless otherwise stated ST2 in this document refers to the enhanced ST2 protocol (ST2+).

The Internet Stream Protocol, Version 2 (ST2) is a connection-oriented internetworking protocol that operates at the same layer as connectionless IP. An ST2 stream is established as a connection between an Origin sending data to one or more Targets requiring reliable delivery of that data. Each network node, in the path between the Origin and Targets, participates in resource reservation negotiations during stream setup. The resource reservation request is based on a Flow Specification sent by the Origin. The FlowSpec provides a definition of the delivery requirements as either predictive or guaranteed in character.

If the reservation negotiations are successful, paths of reserved resources result in a Quality of Service (QOS) reservation for the stream data. This QOS is only established, monitored and maintained by nodes with ST2 Agent capabilities. Each ST2 Agent is called a hop in the ST2 stream routing tree. ST2 Agents that are one hop away from a given node are called Previous-Hops in the upstream direction and Next-Hops in the down stream direction.

Data transfer in the ST2 stream is simplex in the downstream direction. As such, a single Origin sending data to many Targets is similar to a media broadcast model. However, each ST2 Agent may simultaneously need to perform Origin, Previous-Hop, Next-Hop and Target functions for a number of different streams. These streams may be part of a conference (as in the telephone model) or Group of related streams, such that resource reservation and routing issues may be interrelated. The streams may also be unrelated to each other, but ranked by Precedence within an internetwork in the event that limited or changing resources are reallocated. Some streams may request an automatic Recovery option in the event of network failure, Change to the QOS after the original setup or allow Targets to Join the stream, with or without Notifying the Origin.

Thus, every ST2 Agent may be required to support a complex web of intersecting streams with competing QOS requirements, changing resource allocations or members. In addition to standard network management, IP routing and packet-switching duties, an ST2 Agent also supports the ST2 protocol and ST2 QOS features for routing, resource management and packet-switching. An ST2 Agent is a packet-switching router, supporting ST2 components in the context of what is known as the ST2 Service Model.

Individual implementations of this ST2 Service Model will have various QOS algorithms.These algorithms may be based on the architecture of the underlying router model and the resource management goals of the internetwork topology, as well as other issues concerning interactions between supporting services. Such issues are discussed throughout the ST2+ Specification document. This Protocol State Machine document addresses the ST2 protocol in any ST2 Agent, regardless of the QOS algorithms chosen to support the ST2 Service Model.

Stream Control Message Protocol (SCMP) messages form a request-response protocol where the particulars of the Flow Specification, as well as other Protocol Data Unit (PDU) parameters, are interpreted by the chosen QOS Service Model algorithms for routing, Local Resource Management (LRM) and packet-switching. The ST2+ Specification explicitly defines all required and allowable, functions and sequences of SCMP message operations, such that stream FSM behavior can also be explicitly described.

Stream setup and tear down has four fundemental states: IDLE, ESTABLISHED, ADD and CHANGE. The basic transition from IDLE to ESTABLISHED is through a CONNECT request with an ACCEPT response. Additional CONNECT and JOIN requests may result in an ADD stream state, while a CHANGE request would result in a CHANGE stream state. DISCONNECT and REFUSE messages may remove one or more Targets while the stream is in any state.

Each ST2 Agent maintains state information describing the streams flowing through it. It can actively gather and distribute such information. If, for example, an Intermediate ST2 Agent fails, the neighboring Agents can recognize this via HELLO messages that are periodically exchanged between the ST2 Agents that share streams. STATUS packets can be used to ask other ST2 Agents about a particular stream. These agents then send back a STATUS-RESPONSE message. NOTIFY messages serve to inform ST2 Agents of additional mangement information. The failure of an ST2 neighbor will result in stream DISCONNECTs for all Targets that had an Origin or an upstream Previous-Hop and REFUSEs for all Origins that had downstream Targets or a Next-hop.

Each SCMP request-response sequence is defined with next hop ACKnowledgement , ERROR, timeout or retry circumstances. Some SCMP requests also have end-to-end response requirements. This exception processing is designed to resolve incomplete functions during times of network or ST2 Agent failure. QOS Service Model algorithms for packet-switching, routing and LRM services may also initiate exception processing functions in the form of API responses to the ST2 protocol request for services.

ST2 Reason Codes are used to inform other ST2 Agents of the source and type of problem such that the correct response sequences will be followed. These Reason Codes are inserted in the appropriate SCMP PDUs and available to the ST2 Agent management functions. Thus an ST2 Agent not only manages the normal request- response protocol between the Origin and Targets of each stream, but also is actively involved in the detection and distribution of error and QOS implications. The fundemental stream state transitions can become somewhat complicated by these ST2 neighbor , Target management and exception processing issues. The ST2 Agent should anticipate, detect and filter conflicting SCMP messages.

It is assumed that a sophisticated internetwork would be subject to frequent stream membership changes, network failures, multiple routes and resource allocation activities, requiring interpretation and mangement by each ST2 Agent. Simultaneous, or possibly conflicting, request conditions suggests that SCMP message filtering and FIFO queues be used for all SCMP messages destined for an individual stream. In addition, ST2 Agent management functions and exception processing features can provide a hierarchy to protect the individual stream FSMs from unnecessary complications.

The ST2 Agent architecture and FSM models contained in this document have been chosen to illustrate a method for an ST2+ protocol implementation. There are many alternative techniques. Every effort has been made t note the relevant tradeoffs between protocol requirements and implementation choices, suchthat the atomic components of this model may be rearranged to accomodate platform and implementation issues. Every effort has been made to ensure that the described state tables accurately reflect state transitions of the ST2+ version of SCMP, and that the described state diagrams accurately reflect the state transition tables. If there are discrepancies, the tables take precedence over the diagrams and the protocol specification takes precedence over the tables.

Outline of rest of this Document:

Section 2 : ST2 Agent Architecture

Section 3: Stream Finite State Machines

Section 4: Agent Finite State Machines

Section 5: Exception Processing Issues

SECTION 2

ST2 Agent Architecture

The architecture of an ST2 Agent for all of the ST2 Finite State Machines (FSMs) is described in this section. The architectural descriptions are necessarily at a high level and are meant to serve as a guide to the protocol implementer. The state machine models are expected to provide the implementer with useful information such as valid message sequences. The ST2+ Specification provides all detailed supporting documentation.

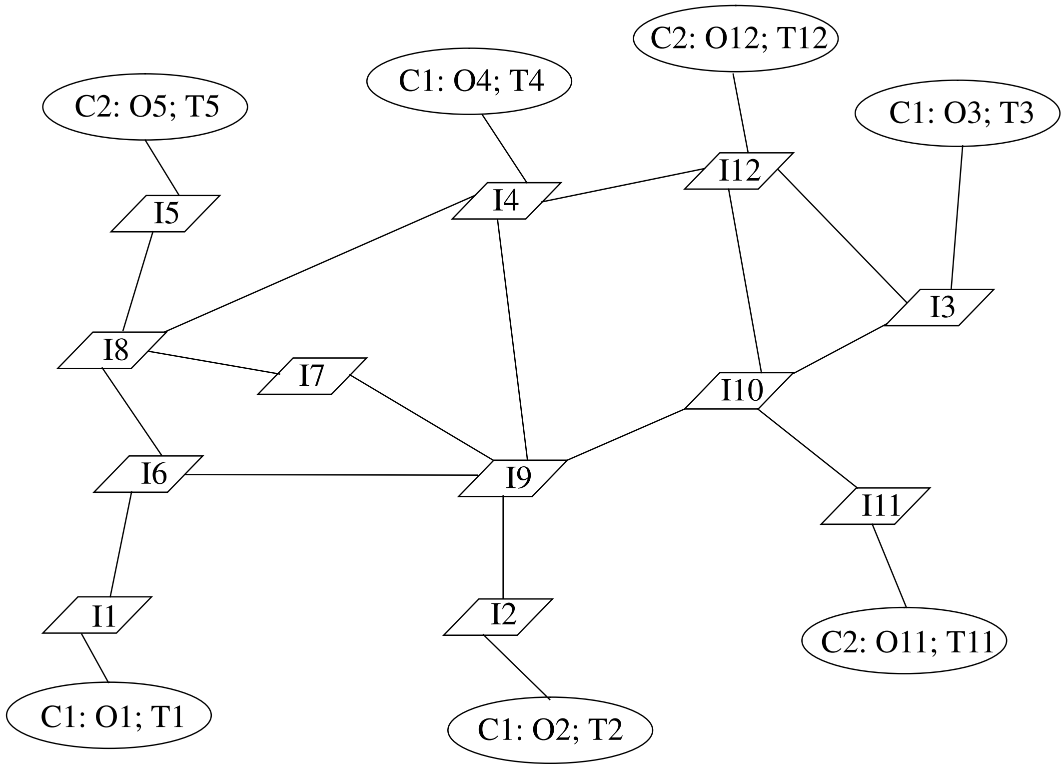
2.1 ST2 Origin, Intermediate and Target Agent Roles

The following internetwork diagram of ST2 Agents (Figure 1) indicates the Origin (O), Intermediate (I) and Target (T) roles of each ST2 Agent in relation to two ST2 conferences. Conference 1 (C1) has 4 participants, all of which are sending and receiving data from each other. Intermediate Agents I1, I2, I3 and I4 each have an ST2 application with an Origin sending data to all other members, and simultaneously receiving data as a Target for each of the other members.

Each ST2 Agent participating in C1, as illustrated, has all four streams to manage, representing a fully meshed stream conference with Targets and Origins communicating along the same paths. Conference 2 (C2) has 3 participants and is also a fully meshed set of streams for each member of the conference. All ST2 Agents in the C2 illustration also have multiple ST2 neighbors, streams and interfaces to manage. In addition, since both conferences are being conducted simultaneously, several Agents are managing streams from both conferences, which may be Grouped for Resource or Fatesharing characteristics. Appendix B includes additional discussions about the possible scenarios that may occur in such an internetwork of conferences.

The ST2 FSMs of any individual stream for the Origin, Intermediate and Target ST2 Agents will be separately modeled. However, since each ST2 Agent may be required to manage multiple FSMs for any one stream, as well as all competitive, intersecting streams in the internetwork topology, the interactions of the individual FSMs is very important. Fortunately, ST2 neighbor communications and SCMP exception processing can be used to create a first line of defense for the ST2 Agent individual stream FSMs. Normal operations for individual streams may be protected and simplified. Network errors and conflicting message sequences can be filtered out of the fundemental stream state transitions. These complexities will be addressed in incremental stages as the ST2 Agent architecture and FSMs are discussed.

Figure 1. Internetwork of ST2 Conferences



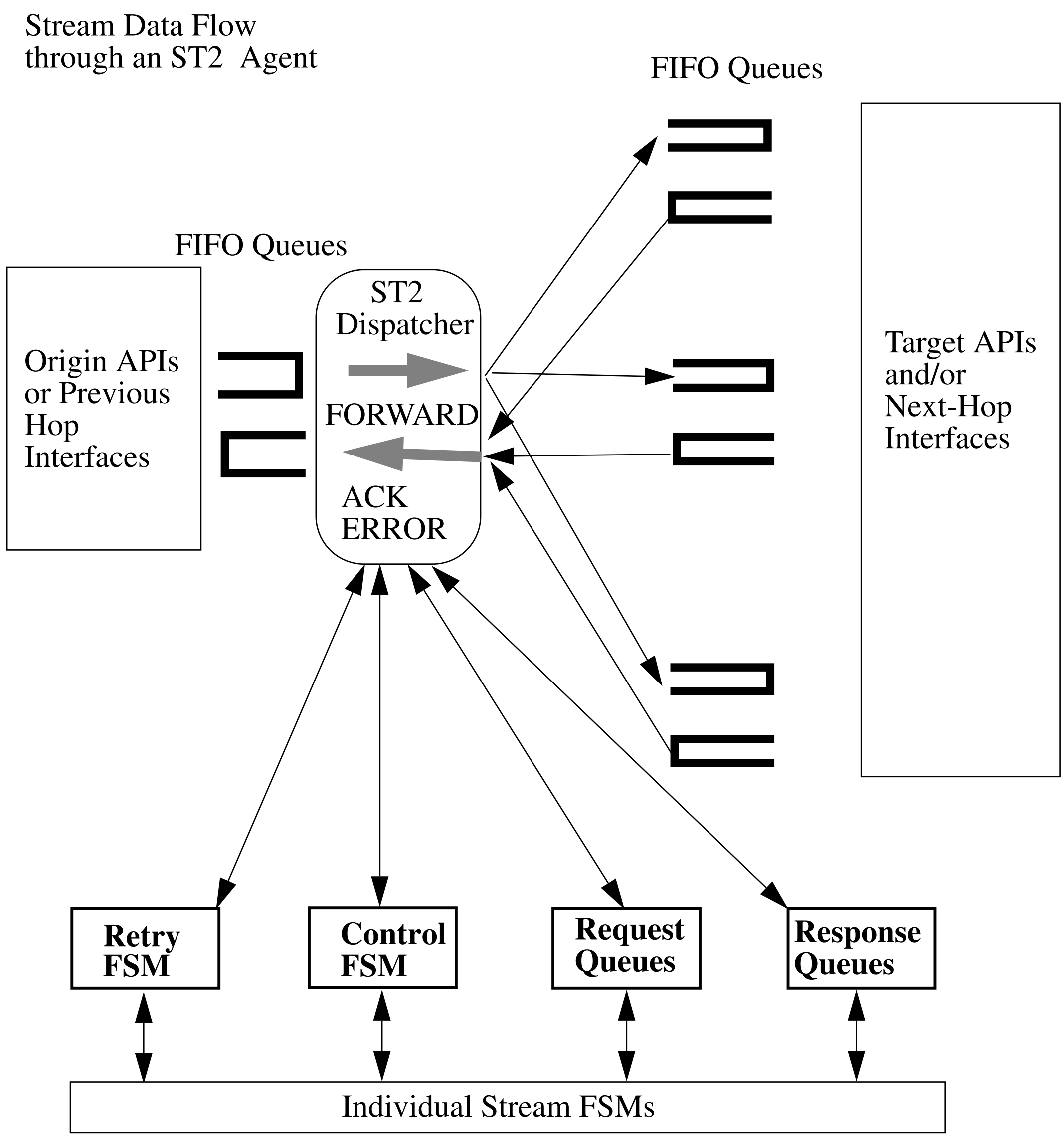
2.2 Stream Data Flow through an ST2 Agent

In the following diagram(Figure 2) , an ST2 Agent is depicted with an ST2 Dispatcher sending and receiving ST2 PDUs from interface queues (and PDU surrogates from application interfaces), representing a high order ST2 message management scheme. This dispatcher unpacks or forwards incoming PDUs, creates and forwards outgoing PDUs.

The forwarding of data or the forwarding of certain command sequences that are not following a negotiatied QOS path (i.e., JOIN and/or JOIN flooding messages) requires a packet-forwarding mechanism, separate from the stream operations that unpack, interpret or create PDUs.

The ST2 PDU validation and delivery functions manage information about the the messaging success or failure, i.e. the Retry, timeout, ERROR , or ACK status of messages. This information concerns a datalink reliability that is separate, but not unrelated to, the state of an individual stream. State transitions that occur as a result of time-outs and retransmissions are first taken into consideration as an ST2 neighbor communication and datalink reliability issue.

Figure 2. ST2 Agent Architecture



Some SCMP FSM message transitions may occur while any individual stream state exists. The ST2 Agent management function is the Control FSM that interprets and sends messages with information relevant to many streams , i.e., HELLO, STATUS, STATUS-RESPONSE , NOTIFY.

The filtering of these messages is relevant to the architectural assumptions for the individual stream FSM transitions. Such message transitions and implications are described in detail in Section 4.

The first level of filtering is designed to determine whether the incoming PDU (or PDU surrogate) is data or one of the JOIN sequences where the destination is, in fact, another ST2 Agent or an application. Such PDUs are then forwarded to the destination. Data delivery to a resident Target or replication for multiple Next-hop Targets are other issues to consider in a packet forwarding function. The efficient packet switching of ST2 PDUs through Intermediate hops is the main emphasis for this filtering priority.

Second-level filtering occurs when an ST2 Agent validates incoming SCMP PDUs and sends the required ACKnowledge (or ERROR PDU, if there are semantic errors) to the originator of the incoming PDU. Conversely, incoming ACKnowledgement and ERROR PDUs trigger the Retry FSM, where all the timeout and retry values are updated or a signal is generated to the appropriate stream FSM for specialized exception processing.

All SCMP PDUs, except ACK, ERROR, HELLO, STATUS and STATUS-RESPONSE, require an ACK from the next hop Agent upon receipt. Signaling the implicit occurrence of a datalink failure to an FSM or queueing the requests and responses to the resident stream FSMs occurs only after such protocol message formalities have been accounted for.

The Control FSM is primarily concerned with ST2 neighbor status. The implications of these issues with an implementations's QOS services and ST2 auxiliary functions will be discussed in detail in Section 4.

Once an ST2 Dispatcher has validated and filtered the PDUs, the individual stream SCMP messages are separately queued into Requests (CONNECT, CHANGE, JOIN) and Responses (ACCEPT, DISCONNECT, JOIN-REJECT, REFUSE). Requests must wait for the completion of any preceding Requests for the same stream, while Responses must be handled immediately without regard to Request state transitions or queues.

The convention of discussing issues in terms of individual stream state transistions will be used throughout Section 3 (Stream FSMs) as a way of simplifying the discussion. Section 4 (ST2 Agent FSMs) and Section 5 (Exception Processing) will provide more detail about the architecture, filtering and queues used with the individual stream FSMs, such that network failures can be managed with competing and intersecting streams. Appendix A contains tables providing specific categories of the ST2 message flow characteristics.

2.3 Origin, Next Hop, Previous Hop and Target Finite State Machines

Discussion surrounding the individual stream state machines most often refers to normal (typical) behavior rather than all pathological cases. Error control and recovery in the architecture is expected to alleviate many problems in the individual stream FSMs, but will be discussed in detail in Section 5.

However, the architecture of a particular Agent may be such that ST2 intra-Agent communications are actually between multiple processors, where Next-hop and Previous-hop FSM communications require the concept of multiple ST2 Agents within what might otherwise appear to be one ST2 Agent. As such, the rationale for the approach to filtering and queueing of the SCMP messages , not just the particular method of illustration, is very important to the Finite State Machine Model architecture.

Communicating Finite State Machine (CFSM) models have been extensively used in the trade to formally describe protocol behavior [2]. Many variations of the basic CFSM model exist and our model is also a variation of the basic model. Our model uses the basic CFSM model with FIFO queues combined with predicates. The model describes the ST2 protocol behavior and consists of ST2 SCMP messages along with a number of predicates. These predicates are not part of the formal ST2 Protocol specifications but are useful mechanisms that simplify the state machine specifications

ST2 Agents - Origin, Intermediate and Target - are all modeled separately using state machines. Because a stream diverges in a tree-like graph, every Intermediate ST2 Agent has to communicate with one upstream ST2 Agent and one or more downstream Agents. Therefore an ST2 Intermediate Agent (Fig. 4) is modeled with 2 or more state machines, one for communicating with an upstream neighbor and a separate state machine for communicating with each downstream neighbor. These machines are called the Previous Hop State Machine (PHSM) and Next Hop State Machine (NHSM) respectively. An Intermediate Agent will therefore have exactly one PHSM and one or more NHSMs for each stream. Note that, it is possible to have more than one NHSM per physical interface, when that interface has more than one Agent on the associated communications link.

The state machine model architecture at the Origin is similar to an Intermediate (Fig. 4). The Origin may have one or more NHSMs. There is no PHSM in this case. However, in the place of the PHSM we have a Origin State Machine (OSM) which interfaces with the application layer above it.

The Target is modeled with one PHSM (Fig. 4). There are no NHSMs here. However, in the place of a NHSM we have a Target State Machine (TSM) that interfaces with the upper layer application protocol.

Because the role of each ST Agent (Origin, Intermediate, or Target) is different, the finite state machine models are not identical. However, the model for communication between FSMs inside or outside an Agent is uniform.

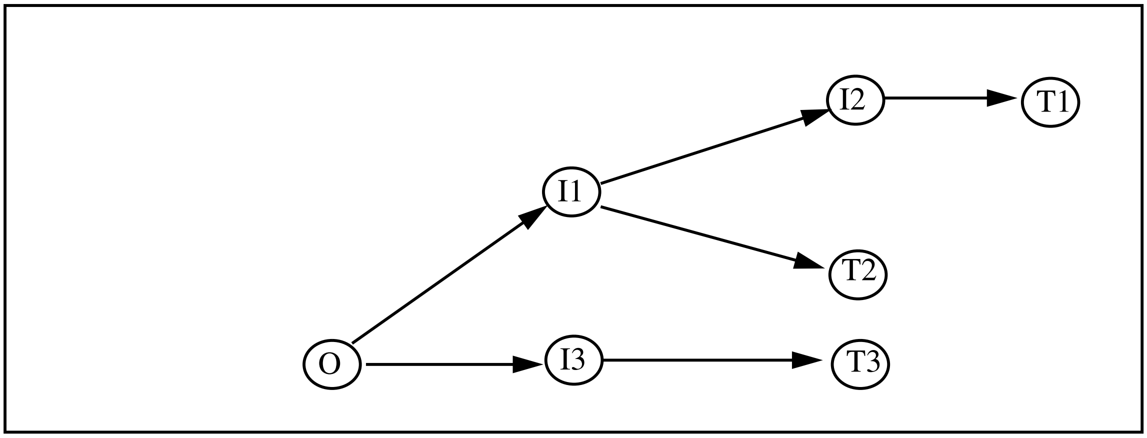
Details of the individual stream machine states and transitions are given in Section 3.

The ST2 individual stream state model architecture is further described here with the help of an example.

Consider a stream topology shown in Figure 3. The figure shows an ST2 Origin (O) connected to 2 Intermediate Agents (I1 and I3). I1 is also connected to I2 and a target T2. I2 is connected to Target T1 and I3 is connected to Target T3. With reference to this example configuration we define a number of state machines. The arrangement of the state machines for this example configuration is shown in Fig. 4.

The Origin is modeled with one OSM and 2 NHSMs (one per next hop). Each Target is modeled with one PHSM and one TSM. I1 and I3 are both modeled with one PHSM and 2 NHSMs; I2 is modeled with one PHSM and 2 NHSMs.

Figure 3. Example Stream Configuration



2.4 Stream Finite State Machines

2.4.1 Externally Communicating FSMs

Communication between two ST2 agents is External Communication and always happens between a NHSM and a PHSM pair. We note that, in the case of Origin and Target Agent as direct neighbors, it is possible for a Target to be directly connected to an Origin . It is also possible that one Target Agent is an Intermediate Agent for another Target. in which case an Agent will have a PHSM communicating with a TSM and one or more NHSMs.

2.4.2 Internally Communicating FSMs

Communicating entities inside an ST2 Agent is different for each Agent type, i.e., Origin, Intermediate or Target. However, all FSMs inside an Agent communicate via a Message Separator/ Combiner box (MS/C). The function of the MS/C box is described later in this section.. Internal Communication within the Origin occurs:

- between the OSM and an Upper Layer Protocol and
- between the OSM and one or more NHSMs via a MS/C box (Note that the NHSMs themselves do not communicate with each other)

Internal Communication within a Target occurs:

- between the TSM and an Upper Layer Protocol and
- between the TSM and a PHSM via a MS/C box

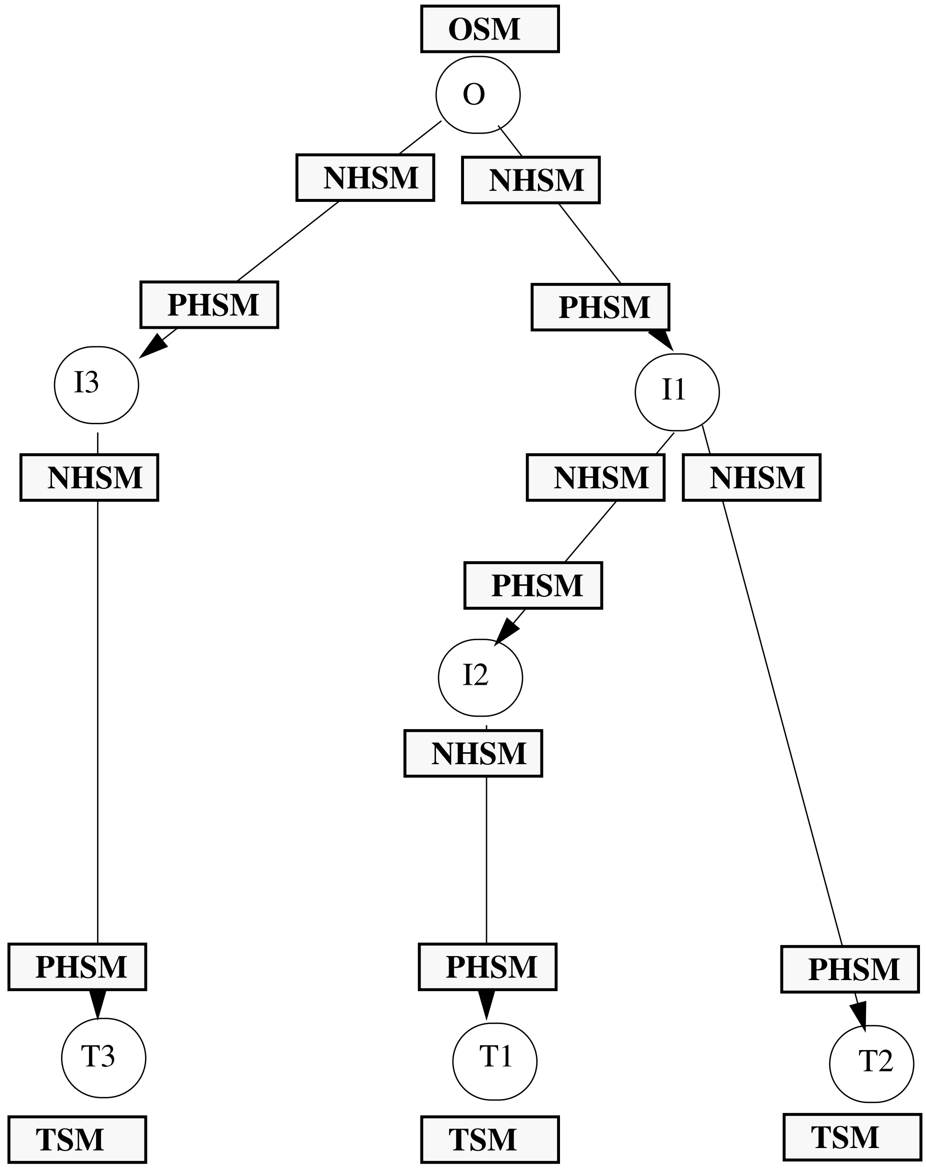
Internal Communication within an Intermediate Agent occurs:

- between a PHSM & and one or more NHSMs via a MS/C box (Note that the NHSMs themselves do not communicate with each other)

For the example configuration below we have:

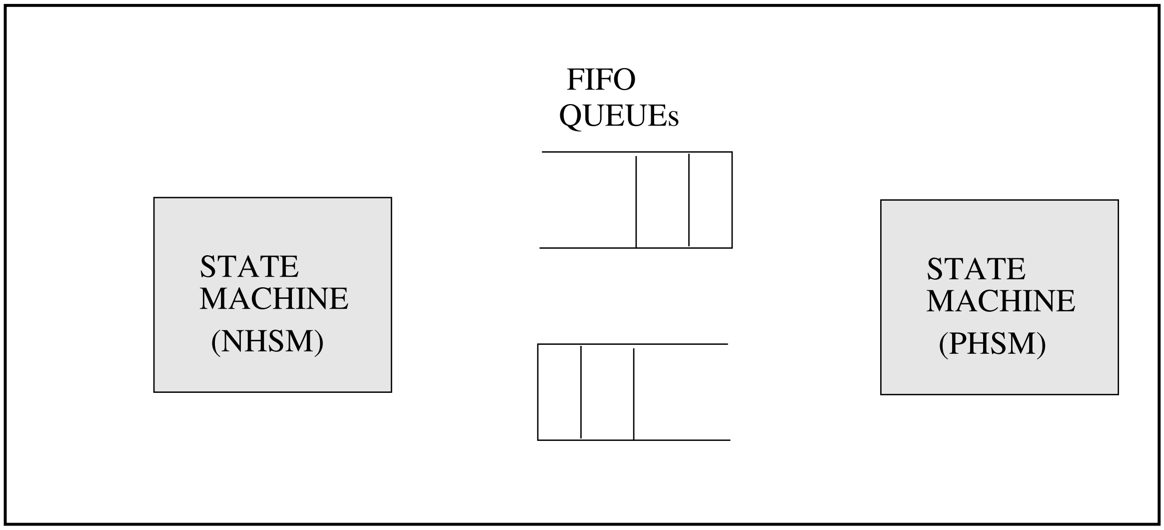
- Origin:
  - an OSM communicating with 2 NHSMs
  - an OSM communicating with the Upper Layer Protocol
- All Targets:
  - a TSM communicating with a PHSM
  - a TSM communicating with the Upper Layer Protocol
- Intermediate Agents:
  - a PHSM communicating with a NHSM inside both I3 and I2
  - a PHSM communicating with 2 NHSMs inside I1

Figure 4. State Machines at the Origin, Intermediate and Target Agents



Acronyms:
OSM = Origin State Machine
NHSM = Next Hop State Machine
PHSM = Previous Hop State Machine
TSM = Target State Machine

Figure 5. Implicit Queues between an External Communicating FSM pair



2.5 Queues between External Communicating FSMs

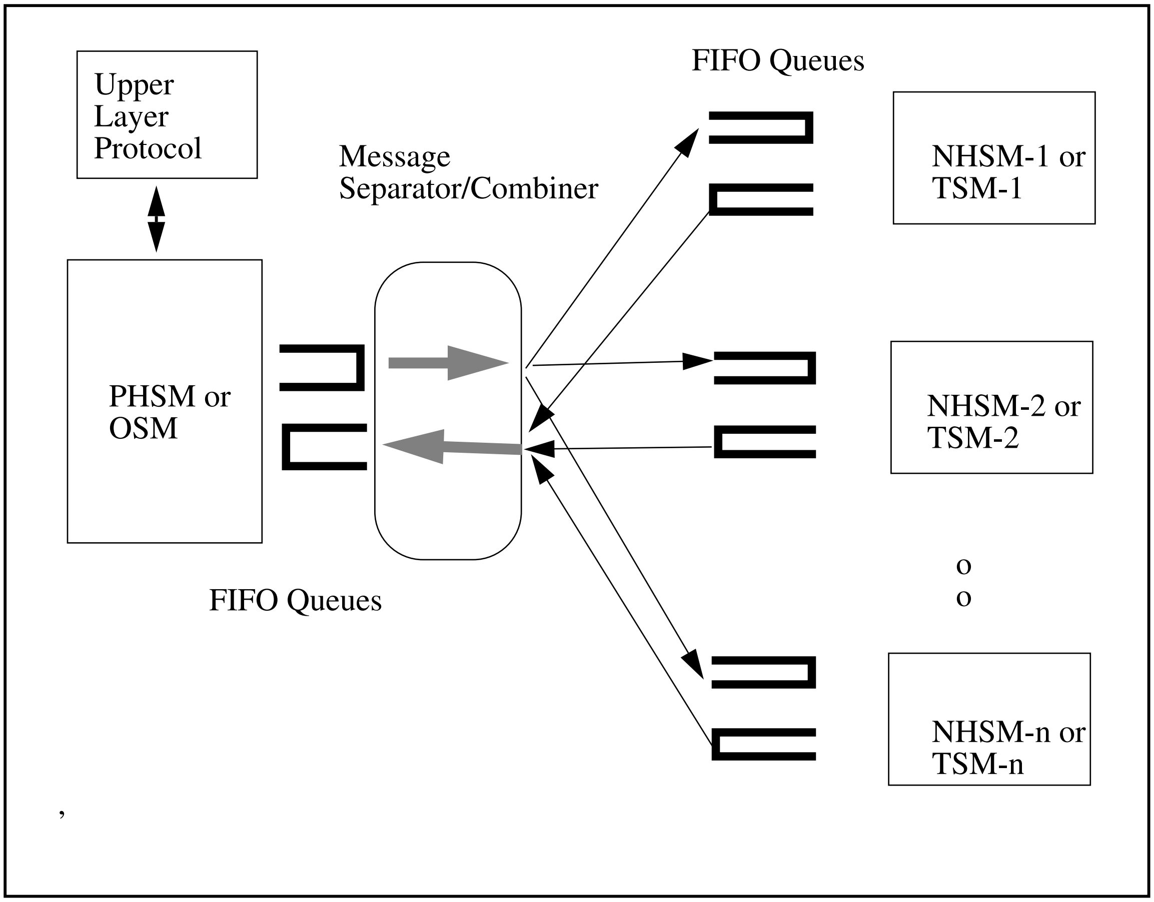
For the purposes of modelling, we assume that messages are filtered and queued in FIFO queues for the case of external Communicating FSM pairs, i.e. between any two ST2 Agents. However, as indicated in the previous discussion and diagrams of the ST2 dispatcher and filtering hierarchy, it is somewhat more complex in reality. The concept shown in Figure 5 allows the discussion of the inter-Agent and intra-Agent state machines to focus on the basic individual stream issues without regard to message and neighbor management issues.

2.6 Queues Inside an Agent

Recall that each Agent is modeled with at least 2 state machines for each stream. These state machines also need to communicate just like the external communicating FSM pairs described above. The queue model in this case also requires filtering mechanisms. This model requires a message Separating and Combining function shown as Message Separator/Combiner (MS/C) box in Fig. 6.

Figure 6 pictorially describes the multi-stage FIFO queue model for an Agent. Implicit FIFO queues are assumed between the PHSM and the MS/C, and also between the MS/C and one or more NHSMs. Use of such FIFO queues eliminates the need for a separate synchronizing state machine that would normally be required to synchronize the flows .

Figure 6. Queues between Internal Communicating FSMs inside an Agent



The function of this Message Separator/Combiner box is many:

- Performing a multicasting function by replicating an OSM or PHSM message and sending them to different NHSMs
- Combining messages coming from different TSMs or NHSMs and sending them to the appropriate OSM or PHSM

Designing the Agent to contain separate upstream and downstream state machines (PHSM and NHSMs respectively) with FIFO queues as shown in Fig. x, offers several benefits:

- It simplifies the Agent design considerably by separating the neighbor upstream and downstream communications
- Use of FIFO queues simplifies the Agent management since no other synchronization mechanisms need to be used to streamline messages flowing through the Agent.

The Agent architecture model is described in more detail in Section 4.

SECTION 3

Stream Finite State Machines

Each ST2 Agent must maintain state for each stream supported by that Agent. There are many ways to represent the state that must be maintained by Agents. This section presents a reference set of state machines described in diagram and table form.

Implementations may support machines based on this section or may even support a completely different set of state machines. In particular, the full implications of the Section 4 (ST2 Agent FSMs) and Section 5 (Exception Processing) have not been detailed in these diagrams and tables. These individual stream FSM represent normal operations of the atomic request-response scenarios. The later sections will discuss the tradeoffs involved in adding the specialized exception processing issues into these atomic FSMs.

This section represents stream state through four state machines. The defined machines are:

- The Origin State Machine, or OSM. It represents the state of a stream at the Origin Agent.
- The Next-Hop State Machine, or NHSM. It represents the state of the stream for Targets reached via a particular next-hop.
- The Previous-Hop State Machine, or PHSM. It represents the state of a stream at an Intermediate Agent or a Target Agent. The OSM is essentially a special case of the PHSM, where the delivery of SCMP to the Origin is via an API.
- The Target State Machine, or TSM. It represents the state of a stream for a particular target application at the Target Agent. This state machine is essentially a special case of the NHSM, where there is only a ever a single Target and delivery of SCMP to the Target is via an API.

A number of NHSMs related to the same stream, could conceivably all be running in parallel -one for each next hop. In some cases, where there is a network-level multipoint link (e.g., ethernet), it is also possible we may have more than one NHSM associated with the same physical interface.

The model assumes that a data engine separate from the control engine exists.

A Message Separator/Combiner (MS/C) box separates all downstream messages modifying the Targetlist and placing them in the respective NHSM FIFO queues. The MS/C box also functions as a combiner of messages flowing up stream. In this role it multiplexes all local messages and places them in the PHSM FIFO queues. Note that the MS/C relies on separate routing and LRM functions to determine the appropriate separation since route and resource computation is not part of ST2 protocol. Full-duplex FIFO queues are assumed between the MS/C box and PHSM, and also between the MS/C box and the NHSMs.

The multi-machine Agent model breaks the complexity that results with only one gigantic model with the aid of the FIFO queue buffers and a MS/C box. The FIFO queues eliminate the need for a separate synchronizing state machine while reducing the complexity. The MS/C reduces the explicit next-hop identification modelling that would otherwise be required..

The Intermediate Agent PHSM always communicates with a NHSM on the upstream side and the NHSM always communicates with a PHSM on the downstream side.

3.1 Assumptions

Some basic assumptions were made as part of the development of the enclosed state machines. These included:

- All state machines exist as part of an ST2 Agent and that the Agent will instantiate state machines as needed to represent state on a per stream basis.
- The ST2 Agent implements logic that unpacks incoming SCMP packets, validates the contents, updates the Agent databases and routes the message signal to the appropriate stream and it's associated state machine.
- Detection and handling of messages that are broken, duplicates, or not valid for a particular stream state does not affect stream state and is not represented in the state machines. The mechanisms to prevent such misleading signals to individual state machines will be discussed in the later sections.
- All reliable delivery of intra- and inter-Agent SCMP messages is handled by the ST2 Agent independent of the described state machines except in the case where stream state is dependent on the outcome of the message delivery.
- All communication within the same Agent should follow the same Request Response paradigm as inter-Agent messages in order to be as reliable as SCMP communications. This assumes that all API communications and intra-agent communications, whether with FSM signals or MS/C messages, creates the reliability available with the ACK, timeout and retry paradigms. Options for accomplishing this in the architecture are discussed in the later sections.
- The described state tables accurately reflect state transitions of the ST2+ version of SCMP, and that the described state diagrams accurately reflect the state transition tables. If there are discrepancies, the tables take precedence over the diagrams and the protocol specification takes precedence over the tables.

3.2 State Machine Model Conventions

3.2.1 Notations

Tables show states, events, output, and transitions.

Diagrams show states, events, and transitions. Initial states are indicated by an asterisk "*".

Messages that trigger events are proceeded by a plus sign "+".

Outputs are proceeded by a minus sign "-".

Transitions are represented by arrows in the diagrams and by “>>” in the tables.

3.2.2 Transmissions and Receptions

In all the state machine models, we follow the standard convention of prefixing message transition labels, with a + or - symbol, to explicitly indicate a transmission and reception respectively. The prefixes are not part of the message syntax. In addition, the tables will show both transmitted and received messages, but the diagrams show only recieived messages. This simplifies the diagrams and alleviates some of the complexity of error handling, as discussed in Section 4 (Agent FSMs) and Section 5 (Exception Processing).

3.2.3 Application-Initiated Transitions

State transitions are sometimes dictated by conditions outside the scope of the protocol specification. Predicates are mechanisms that allow such transitions to occur. For example, terminating a protocol session (a result of many conditions) should allow the Agent to transition to either the initial state or some idle state. This decision is of course Application-initiated but a means should nevertheless exist to allow transitioning to the correct state. In the ST2 protocol there is no message which accomplishes this. Thus, a close predicate is introduced in the FSMs to carry out this function.

3.2.4 Predicates

Predicates allow a state machine to express conditions and control not explicitly possible with the protocol messages. Generally speaking, they add clarity to the state diagram while reducing the complexity in terms of states. The addition of control predicates allows user defined change of states. These predicates are meant to give hints to the protocol implementer and are not part of the ST2 protocol. For example, a **close** predicate is implicit in every state machine as a means to transfer control to the **Init** state. Some triggers and events are combinations of implicit and explicit message conditions. This is particularly true for the timeout and retry mechanisms, as well as the requirement that responses from all Targets in a Request be complete before the Request state can complete.

3.2.5 Naming conventions

All state names are in bold and start out with a capital letter followed by the lower case letters. All message names are in capitals usually prefixed with a + or - sign. Predicates are in bold and lower case string.

3.3 Normal Behavior versus Exception Processing

These state models describe the protocol under normal conditions. Detailed error handling will be discussed in both Section 4 and Section 5. Also, control messages STATUS, STATUS-RESPONSE, NOTIFY and ERROR are not shown. Otherwise, if a core message transition is not specified from a state, it implicitly means that this message is not allowed from that state.

3.3.1 Individual Stream FSM Issues versus the complete context of the ST2 Agent Stream Databases and the ST2 Agent FSM

The OSM, NHSM, PHSM and TSM diagrams and tables in this section do not represent state as the complete context of the stream as it would be defined by all possible combinations of conditions. The G-bit (all targets), the S-bit (stream recovery), the I-bit and E-bit (change failure-stream teardown) involve combinations of FSM and stream database managment issues. There are multiple dimensions inherent in the partitioning and management of the TargetList in relation to the particula NHSM and ST2 Agent neighbor associated with each Target or set of Targets. Multiple SCMPPDUs for the same transaction or SCMP propagation failures may be due to MTU size limitations. Such issues involve additional data planes of information that define the context of the individual stream FSMs.

The diagrams and tables in this section reflect an atomic grouping of the core ST2 stream setup and teardown procedures. This perspective delineates three classes of Responses. The first class is a Response that does not have any signifigance for state change, where these Responses are not specifically either the last one required to complete the TargetList of the current Request, nor a deletion of the last of all Targets associated with that stream's FSM. Completion of the Responses for the current TargetList is the second class. Removal of the last of all Targets for that stream's FSM is the third class.All api references are illustrative and are not intended to fully define the application interface.

Class 1 Responses: api_accept, api_disconnect, api_refuse, ACCEPT, DISCONNECT, REFUSE, and RetryTimeout indicate that an individual TargetList member has signaled a Response. **api_disconnect, api_refuse, DISCONNECT and REFUSE** may also be a Request to delete a Target(whether or not it is in the current TargetList of an Add or Change state transaction). Individual and/or Global Target deletion may occur at any time, but any Global (G-bit set) Target Response or deletion Request falls into one of the second two classes.

Class 2 Responses:api_accept_last, api_disconnect_last, api_refuse_last, ACCEPT_LAST, DISCONNECT_LAST and REFUSE_LAST, RetryTimeout_last, E2E_Timeout_last are only relevant to the current stream Request and refer to the completion of the Request state by occurring as the Response that incidently completes the TargetList .

Class 3 responses:api_disconnect_all, api_refuse_all, DISCONNECT_ALL and REFUSE_ALL refer to the Requests or Responses that remove the last active Target from that FSM for that stream.

These classes represent the delineation of the asynchronous Request/Response activity that may occur. Network changes or conditions may result in interruptions of any individual stream stream FSM operations. The ST2 Agent Retry FSM, Control FSM, Routing function or LRM may initiate stream teardown for any Target or TargetList at any point. The OSM, NHSM, PHSM and TSM diagrams and tables in this section define stream state as it relates to atomic setup and teardown functions. Every attempt has been made to delineate the atomic SCMP request-response specifications such that implementors may reorganize the Agent architecture to address implementation-specific issues.

3.3.2 Recovery, Retry and Timer Failures

The following table provides a quick reference for ST2 recovery and retry implications across ST2 Agent FSMs as atomic conditions of the request-response specifications. These conditions are explicit in that an ST2 Agent neighbor ACK will terminate the neighbor ACK timer and retry count for that transaction. Any allowable End to End response will also terminate the neighbor ACK timer and retry count for that transactiont, as well as the End to End response timer. Implicit conditions occur when any of the timer or the retry count values have been exhausted. The general paradigm is an implicit REFUSE for unsatisfied downstream requests and an implicit DISCONNECT for unsatisfied upstream requests. The secondary consequence of a timeout is that explicit REFUSE and DISCONNECT messages may also be issued.

However, each table entry below has its own variation of the basic paradigm. In addition, the ST2 specification indicates many secondary and tertiary implications for SCMP message failures. As a particular example, once any ST2 Agent has completed a downstream request-response scenario, any upstream propagation problem may or may not cause the stream to be torn down. The I-bit in CHANGE processing and the S-bit in all transactions are examples of causes for the secondary and tertiary implications.

Table 1:

ST2 Agent FSM	SCMP message	Nbr ACK timer	Nbr Retry count	End to End explicit Responses	End to End Response timer
Retry	ACCEPT	ToAccept	NAccept		
	ACK				
Retry	CHANGE	ToChange	NChange	ACCEPT/ REFUSE	ToChange-Resp
Retry	CONNECT	ToConnect	NConnect	ACCEPT/ REFUSE	ToConnect-Resp
Retry	DISCON-NECT	ToDiscon-nect	Ndisconnect		
	ERROR				
Retry	JOIN	ToJoin	NJoin	CONNECT/ JOIN-REFUSE	ToJoinResp
Retry	JOIN-REJECT	ToJoinRe-ject	NJoinReject		
Retry	NOTIFY	ToNotify	NNotify		

Table 1:

ST2 Agent FSM	SCMP message	Nbr ACK timer	Nbr Retry count	End to End explicit Responses	End to End Response timer
Retry	REFUSE	ToRefuse	NRefuse		
Control - NRetry-Route	Recovery-CONNECT	ToConnect	NConnect	ACCEPT/ REFUSE	ToRetry-Route
Control - NStatus	STATUS			STATUS-RESPONS E	ToStatus-Resp
Control - Hellotimer-Holddown	HELLO	implicit not explicit ACK - DefaultRe-coveryTim-eout	implicit not explicit retry - Helloloss-Factor		

3.3.3 Normal Behavior of the Atomic Individual Stream FSMs

The exception processing issues that are most represented in this section include Reason Codes:

1 NoError No error has occurred.

5 ApplAbort The application aborted the stream abnormally.

6 ApplDisconnect The application closed the stream normally.

7 ApplRefused Applications refused requested connection or change.

25 JoinAuthFailure Join failed due to stream authorization level.

The following Reason Codes are illustrated for Requests (CONNECT, CHANGE), but not Responses propagations that would result in special exception processing (i.e, ACCEPT-ACK, DISCONNECT-ACK, REFUSE-ACK or JOIN-REJECT-ACK failures):,

38 ResponseTimeout Control message has been acknowledged but not answered by an appropriate control message.

41 RetransTimeout An acknowledgment has not been received after several retransmissions.

3.3.4 Exception Processing not Fully Covered in Atomic Individual Stream FSMs

The following management and exception processing issues are to be addressed in detail in Sections 4 and 5:

Retry and Timeout Failures for Responses (e.g., an ACCEPT-ACK, DISCONNECT-ACK, JOIN-REJECT-ACK, REFUSE-ACK) :

38 ResponseTimeout Control message has been acknowledged but not answered by an appropriate control message.

41 RetransTimeout An acknowledgment has not been received after several retransmissions.

ST2 Dispatcher, ACKknowledgement and ERROR functions with Reason Codes:

4 AckUnexpected An unexpected ACK was received.

13 CksumBadCtl Control PDU has a bad message checksum.

14 CksumBadST PDU has a bad ST Header checksum.

15 DuplicateIgn Control PDU is a duplicate and is being acknowledged.

16 DuplicateTarget Control PDU contains a duplicate target, or an attempt to add an existing target.

23 InvalidSender Control PDU has an invalid SenderIPAddress field.

24 InvalidTotByt Control PDU has an invalid TotalBytes field.

26 LnkRefUnknown Control PDU contains an unknown LnkReference.

31 OpCodeUnknown Control PDU has an invalid OpCode field.

32 PCodeUnknown Control PDU has a parameter with an invalid PCode.

33 ParmValueBad Control PDU contains an invalid parameter value.

35 ProtocolUnknown Control PDU contains an unknown next-higher layer protocol identifier.

36 RecordRouteSize RecordRoute parameter is too long to permit message to fit a network's MTU.

37 RefUnknown Control PDU contains an unknown Reference.

45 SAPUnknown Control PDU contains an unknown next-higher layer SAP (port).

46 SIDUnknown Control PDU contains an unknown SID.

48 STVer3Bad A received PDU is not ST Version 3.

49 StreamExists A stream with the given SID already exists.

51 TargetExists A CONNECT was received that specified an existing target.

52 TargetUnknown A target is not a member of the specified stream.

53 TargetMissing A target parameter was expected and is not

included, or is empty.

54 TruncatedCtl Control PDU is shorter than expected.

55 TruncatedPDU A received ST PDU is shorter than the ST Header indicates.

56 UserDataSize UserData parameter too large to permit a message to fit into a network's MTU.

Control FSM issues with neighbor failure and stream recovery with Reason Codes:

12 CantRecover Unable to recover failed stream.

22 IntfcFailure A network interface failure has been detected.

27 NetworkFailure A network failure has been detected.

39 RestartLocal The local ST agent has recently restarted.

40 RestartRemote The remote ST agent has recently restarted.

47 STAgentFailure An ST agent failure has been detected.

Routing issues with Reason Codes:

9 BadMcastAddress IP Multicast address is unacceptable in CONNECT

28 NoRouteToAgent Cannot find a route to an ST agent.

29 NoRouteToHost Cannot find a route to a host.

30 NoRouteToNet Cannot find a route to a network.

34 PathConvergence Two branches of the stream join during the CONNECT setup.

42 RouteBack Route to next-hop through same interface as previous-hop and is not previous-hop.

43 RouteInconsist A routing inconsistency has been detected.

44 RouteLoop A routing loop has been detected.

LRM issues with Reason Codes:

10 CantGetResrc Unable to acquire (additional) resources.

11 CantRelResrc Unable to release excess resources.

17 FlowSpecMismatch FlowSpec in request does not match existing FlowSpec.

18 FlowSpecError An error occurred while processing the FlowSpec.

19 FlowVerUnknown Control PDU has a FlowSpec Version Number that is not supported.

20 GroupUnknown Control PDU contains an unknown Group Name.

21 InconsistGroup An inconsistency has been detected with the streams forming a group.

50 StreamPreempted The stream has been preempted by one with a higher precedence.

Miscellaneous errors with Reason Codes:

2 ErrorUnknown An error not contained in this list has been detected.

3 AccessDenied Access denied.

8 AuthentFailed The authentication function failed.

3.4 Individual Stream State Machines

3.4.1 Glossary

All individual stream FSMs have the following 4 states in common:

Init: The stream is not active.

Establd: The stream is established and may or may not have Target members.

Add: The stream is currently adding Targets as the result of a Connect of Join initiated Connect.

Change: The stream is currently attempting to Change according to the new FlowSpec.

A list of predicates, api interactions and combination conditions include the following:

api_close - the Origin api explicitly terminates a stream, since a stream with no Targets at the Origin may remain Established

api_open - the Origin api explicitly establishes a stream to initiate all database setup functions whether or not any Targets are initially specified.

api_connect- the Origin api adds Targets.

api_change - the Origin api initiates a CHANGE to the FlowSpec.

api_disconnect - the Origin api initiates a DISCONNECT to Targets.

accept_api - the OSM propagates an ACCEPT received from either a TSM or a NHSM to the Origin api.

notify_api - the OSM propagates a NOTIFY received from either a TSM or a NHSM to the Origin api.

refuse_api - the OSM propagates a REFUSE received from either a TSM or a NHSM to the Origin api.

nexthop_open - the first time each unique NHSM is invoked for each unique stream in an Agent, the Agent explicitly establishes a NHSM database and Establd state.

prevhop_open - the first time the PHSM is invoked for each unique stream in an Agent, the Agent explicitly establishes a PHSM database and Establd state.

api_join - the Target api initiates a JOIN request**.**

api_refuse - the Target api initiates a REFUSE to the TSM.

api_refuse_change - the Target api initiates a REFUSE of a CHANGE request to the TSM.

connect_api - the TSM propagates a CONNECT to the Target api.

change_api - the TSM propagates a CHANGE to the Target api.

join_reject_api - the TSM propagates a JOIN_REJECT to the Target api.

disconnect_api - the TSM propagates a DISCONNECT to the Target api.

JOIN_AUTH - a PHSM or OSM JOIN is authorized.

JOIN_NOT_AUTH - a PHSM or OSM JOIN is not authorized.

CH_DISC - a CHANGE failure when the I-bit is set results in stream teardown as an enforced DISCONNECT downstream.

CH_REF - a CHANGE failure when the I-bit is set results in stream teardown as an enforced REFUSE upstream.

RetryTimeout - an FSM recieves an implicit REFUSE response to a CONNECT or CHANGE request to one Target in the TargetList by exceeding the ACK retry and timeout values (i.e., ToChange/NChange, ToConnect/NConnect timers and retry counts) for that particular transaction.

The Add and Change states cannot transition back to the Establd state until all Targets have given implicit or explicit responses.

ACCEPT_LAST - the last Target in the TargetList for a CONNECT or CHANGE has responded with an ACCEPT.

DISC_LAST - the last Target in the TargetList for a CONNECT or CHANGE has responded with a DISCONNECT.

REFUSE_LAST - the last Target in the TargetList for a CONNECT or CHANGE has responded with a REFUSE.

RetryTimeout_last - the last Target in the TargetList for a CONNECT or CHANGE has responded with an implicit REFUSE by exceeding the ACK retry and timeout values (i.e., ToChange/NChange, ToConnect/NConnect timers and retry counts).

E2E_Timeout_last - the last Target in the TargetList for a CONNECT or CHANGE has responded with an implicit REFUSE by exceeding the End-to-End timeout value (i.e., ToChange-Resp, ToConnectResp timers).

Except in the case of the OSM (which must be explicitly closed by the Origin api), the Establd, Add and Change states cannot transition back to the Init state until all Targets in the unique FSM TargetList have given implicit or explicit stream teardown instructions.

DISC_ALL -a DISCONNECT has been received for the last Target in the entire TargetList for a stream FSM (as opposed to the TargetList for a particular CONNECT or CHANGE request).

REFUSE_ALL - a REFUSE has been received for the last Target in the entire TargetList for a stream FSM (as opposed to the TargetList for a particular CONNECT or CHANGE request).

3.4.2 Origin State Machine (OSM)

The Origin State Machine (OSM) is a high level state machine which communicates with one or more NHSMs. The OSM also talks to the Upper Layer Protocol via primitives. This OSM to Upper Layer Interface is outside the scope of this document and the draft assumes that such an Interface exists but does not specify the Interface. All ST2 Dispatcher and MS/C Box diagrams have indicated that API messages could be included. The actual mechanism used for API communications should be decided by implementation factors.

The OSM consists of a small number of states: **Init, Establd, Add and Change.**

Init: The initial state is called **Init**. An **api_open** predicate moves the control to the **Establd** state. An **api_close** is required to return the stream to the **Init** state.

Establd: The **Establd** state is the stable state from which all **api_connect**, JOIN and **api_change** requests may cause a transition to the **Add** or **Change** states. All CONNECT, JOIN and CHANGE requests that occur while a stream is in either an **Add** or **Change** state will be queued up until the stream returns to the **Establd** state. Data transfer may occur to established Targets. The removal of Targets from previous operations or current operations may occur in the **Establd, Add or Change** states with an **api_disconnect** or a REFUSE.

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. A JOIN message received by an OSM indicates that the Origin Agent happens to be the first Agent for that stream in the path between the JOIN originator and the Origin. JOIN messages from potential Targets require the authorization process to determine if the JOIN will be allowed. The OSM (or PHSM in the case of an Intermediate Agent) then issues either a -JOIN-REJECT message or a -CONNECT message. Issuing a JOIN-REJECT brings it back to the **Establd** state. Issuing a -CONNECT message moves the control to the **Add** state awaiting an ACCEPT or REFUSE response.

Figure 7. **Origin State Machine (OSM)**

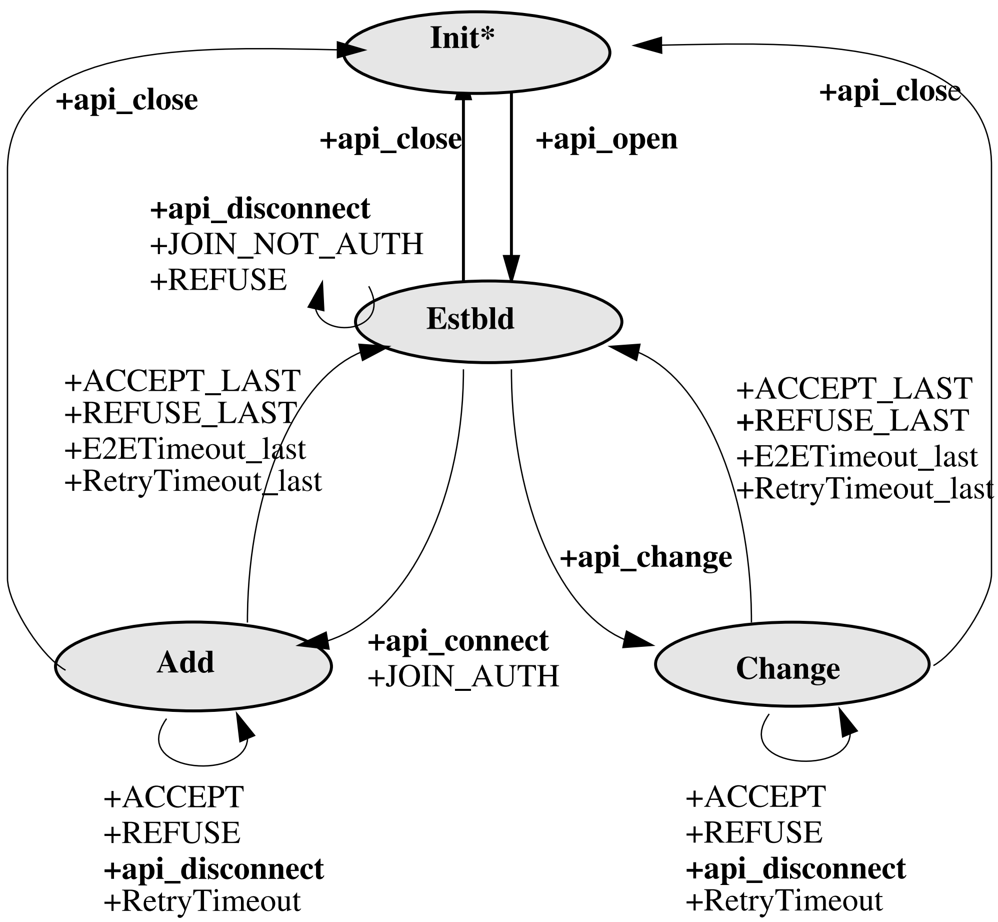


Table 2: OSM

OSM	Init	Estbld	Add	Change
+ACCEPT	-	-	>> Self -accept_api	>> Self -accept_api
+ACCEPT_LAST	-	-	>> Estbld -accept_api	>> Estbld -accept_api

Table 2: OSM

OSM	Init	Estbld	Add	Change
+JOIN_AUTHORIZED	-	>> Add -CONNECT (-notify_api)	>> Self -queue	>> Self -queue
+JOIN_NOT_AUTHORIZED	-	>> Self -JOIN_REJ -notify_api	>> Self -queue	>> Self -queue
+REFUSE	-	>> Self -refuse_api	>> Self -refuse_api	>> Self -refuse_api (-CH_DISC)
+REFUSE_LAST	-	-	>> Estbld -refuse_api	>> Estbld -refuse_api (-CH_DISC)
+api_open	>> Estbld	-	-	-
+api_change	-	>> Change -CHANGE	>> Self -queue	>> Self -queue
+api_connect	-	>> Add -CONNECT	>> Self -queue	>> Self -queue
+api_disconnect	-	>> Self -DISCON	>> Self -DISCON	>> Self -DISCON
+api_close	-	>> Init -DISCON	>> Init -DISCON	>> Init -DISCON
+RetryTimeout	-	-	>>self -DISCON -refuse_api	>>self -DISCON -refuse_api (-CH_DISC)
+RetryTimeout_last			>> Estbld -DISCON -refuse_api	>> Estbld -DISCON -refuse_api (-CH_DISC)
+E2ETimeout_last	-	-	>> Estbld -DISCON -refuse_api	>> Estbld -DISCON -refuse_api (-CH_DISC)

An ST2 Agent processes a JOIN request when that Agent has the designated stream active in its databases and FSMs. The authorization level associated with the stream is examined to determine whether to transition from the **Establd** state to the **Add** state.

- level 0 (JOIN_NOT_AUTHORIZED) JN bits=00

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

- level 1 (JOIN_AUTHORIZED, NOTIFY Origin)JN bits=01

The ST2 Agent sends a CONNECT message with a TargetList containing the Target that requested to join the stream. This results in adding the Target to the stream. When the ST2 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 2(JOIN_AUTHORIZED)JN bits=10

The ST2 Agent sends a CONNECT message with a TargetList containing the Target that requested to join the stream. When the ST2 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 (in the OSM an accept_api), nor does it notify the Origin (notify_api).

The Origin also checks that

1. the SID is valid

2. the Targets are not already members of the stream

3. 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.

If this validation is complete and the stream JOIN option allows authorization to be completed,the ST2 Agent at the Origin transitions to the **Add** state and then issues a CONNECT message that contains the SID, the FlowSpec, and the TargetList specifying the new Targets.

If this is not the case, a JOIN-REJECT message is sent to the Target with the appropriate ReasonCode (e.g., JoinAuthFailure, DuplicateTarget or RouteLoop).

Add: Once in the **Establd** state the api may issue an **api_connect**. A transition to **Add** will create a CONNECT message that is placed in the FIFO queue between the OSM and the MS/C box. The CONNECT message contains the SID, an updated FlowSpec, and a TargetList. The MS/C box will then make a copy of the CONNECT message, partition the Targetlist parameter and place it the NHSMs queues.The spliting (or separating) information is derived from the implementation's routing and LRM functions.

This model has placed the routing and LRM functions in the MS/C box, such that OSM (or PHSM in the case of an Intermediate Agent) may find that the Response has been initiatedf by either the local functions or a remote Agent.

If multiple next-hops are to be reached through a network that supports network level multipoint (e.g., an ethernet link), a different CONNECT message must nevertheless be sent to each next-hop since each will have a different TargetList. In this case, we have one physical interface but more than one or more logical NHSMs associated with this interface.

Once in the **Add** state the OSM waits to get ACCEPT or REFUSE responses. The stream will not transition back to the **Establd** state until all Targets have responded. The expiration of the retry timer and count (if the next hop is not ACKing the request) or the expiration of the end-to-end timer will be interpreted as an implicit REFUSE.

The OSM will record the status of each response from each Target. As each ACCEPT is received, the OSM updates its database and records the status of each 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 either an ACCEPT or REFUSE (explicit or implicit by timeout failure) from all Targets has been received at the Origin, the stream state returns to **Establd** and any additional queued up requests may then be processed**.**

When an ACCEPT is received by the OSM, the path to the Target is considered to be established and the ST2 Agent is allowed to forward the data along this path. When a REFUSE reaches the OSM, the OSM notifies the Application that the Target is no longer part of the stream. If there are no remaining Targets, the Application may wish to terminate the stream or keep the stream active to allow stream joining.

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 all stream Targets.

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 reservedHowever, 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 ST2 Agents that are not already nodes in the stream behave as in the case of stream setup.

The OSM may issue a -DISCONNECT as a result of an **api_disconnect**. This message may be processed in any state. The OSM then records this fact and appropriately updates its database.

A +REFUSE message may arrive at the OSM asynchronously at any time.This message is sent as a result of an intermediate Agent failure or a Target leaving a stream. The database update processing that occurs after receiving a +REFUSE is the same as that which occurs after sending a -DISCONNECT message. Therefore, sending a -DISCONNECT or receiving a +REFUSE leads to the same state transitions.

Change:The Application at the Origin may wish to change the FlowSpec of an established stream. To do so, it informs the ST2 Agent at the Origin of the new FlowSpec and of the list of Targets relative to the change with an **api_change**. The Origin then issues one CHANGE message with the new FlowSpec per next-hop and sends it to the relevant next-hop Agents. The control flow to the **Change** state is very similar to the control to the **Add** state from the **Establd** state. Depending on the CHANGE options selected and the resources avalailable in each of the stream paths, the CHANGE may result in either a simple refusal of any change or the disconnect of the entire stream. A REFUSE response to a CHANGE request may simply mean that the stream for that Target is unchanged, depending on the I-bit setting. In all other cases, a REFUSE results in the teardown of the stream for that TargetList.

3.4.3 Next Hop State Machine (NHSM)

The NHSM is pictorially shown in Figure 8. This model is common to the Origin as well as an Intermediate Agent .The NHSM consists of the same fundamental states as the OSM: **Init, Establd, Add and Change.**

Init: The state machine for each next hop enters its **Init** state at Agent start-up time. A dot indicates that this is the initial state. A **nexthop_open** predicate moves control to the **Establd** state when the next hop associated with an NHSM is required by Targets in a stream.

Establd: Once in the **Establd** state a number of things can happen. Targets may be added by the Origin or Targets may request to join the stream. However, the processing of a JOIN request is always handled by either an OSM or a PHSM. Within each ST2 Agent, the ST2 Dispatcher examines incoming JOIN requests and determines whether the stream referenced is a stream that that Agent supports. If not, the JOIN is forwarded on towards the Origin. Once a JOIN request reaches an Agent that can process the JOIN, the ST2 Dispatcher ACKs the JOIN and queues it up to the resident OSM or PHSM. The NHSM only sees the resultant CONNECT when stream authorization has completed successfully and the OSM or PHSM has issued a CONNECT through the MS/C Box.

As previously described in the OSM, an ST2 Agent can handle only one stream **Add** or **Change** at a time. If such a stream operation is already underway, further requests are queued and handled when the previous operation has been completed. Either a DISCONNECT or REFUSE for all Targets transfers control from the **Establd** state to the **Init** state.

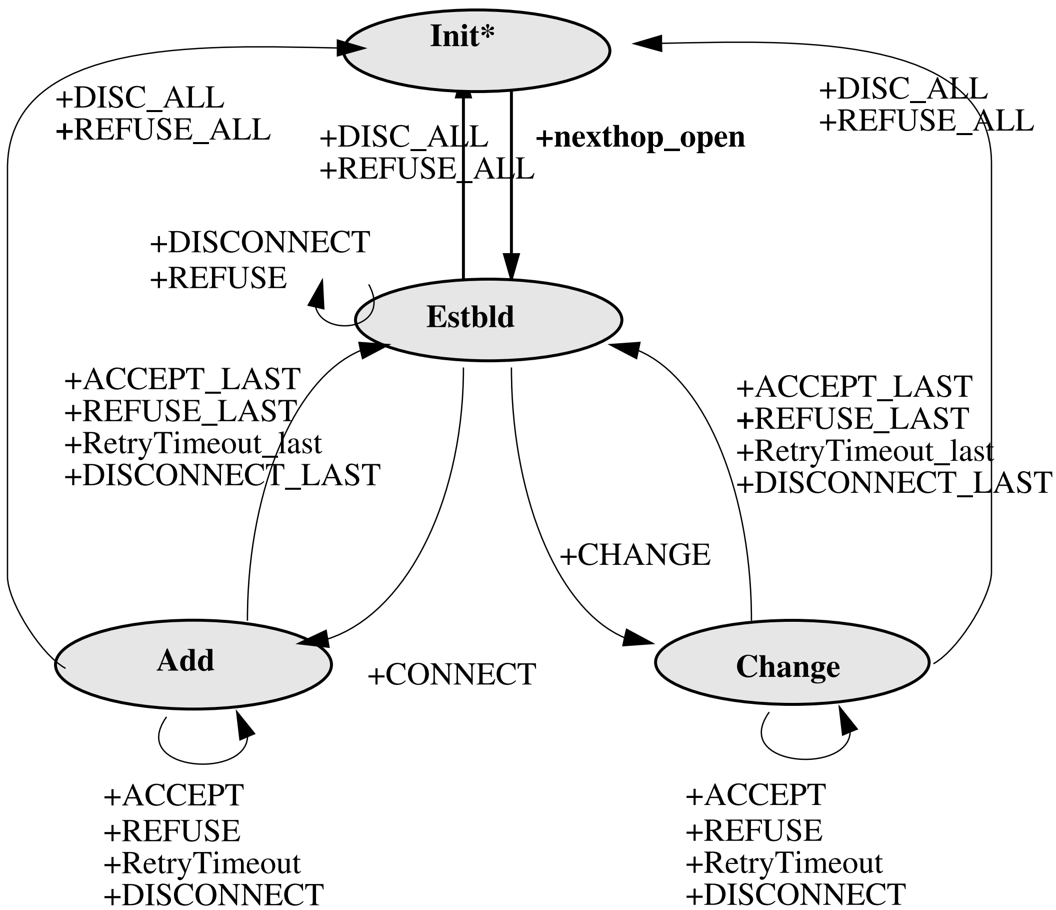
Add: A CONNECT that has been propagated from the NHSM **Add** state to the next hop Agent PHSM and will require a response in the form an ACK . If not, the timeout and retry mechanisms of the Retry FSM will invoke an implicit refuse, a RetryTimeout signal. Every PDU has a unique reference number, so that all ACKs may be matched to the appropriate Request or Response.

The ACK, if it is received, does not need to be reported to the NHSM. However, if the ACK is not received and the retries are exhausted, a RetryTimeout signal will be reported to the NHSM and interpreted as a REFUSE. Otherwise, the NHSM will record all Target explicit responses until the last Target in the TargetList has sent an ACCEPT or REFUSE (or an implicit REFUSE due to Retry exhaustion or a DISCONNECT from the Origin or a Control FSM).

An Origin DISCONNECT may be due to the exhaustion of the end-to-end timer. A DISCONNECT signal from a Control FSM may be due to the failure of an upstream or a downstream nieghbor.

The Application at the Origin may specify a set of Targets that are to be removed from the stream with an appropriate ReasonCode (ApplDisconnect). The Targets are partitioned by the MS/C box 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.

Figure 8. Next Hop State Machine (NHSM)



If, after deleting the specified Targets, any next-hop has no remaining Targets, then those resources associated with that next-hop agent may be released. Note that the 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 to the upstream NHSM and notifies the Application (at target) that it is no longer part of the stream and for which reason. The ST2 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 ST2 Target Agent is also an Intermediate Agent for the stream and there are remaining downstream Targets.

The ST2 Dispatcher and MS/C Box process the PDU, PDU surrogates and api message contents into a common database within the Agent, such that the Agent OSM/PHSM and NHSM/TSM pairs may be updated with simultaneous signals, rather than waiting for the signal to be propagated by the individual stream FSMs through the MS\C Box. The tradeoff of such implementation choices revolves around exception processing conditions. The basic states in each of the FSM models are very similar. Thus in the following table, all messages sent (-) by the NHSM are explicitly noted, even though there is an opportunity for such signalling to be managed more simply or simultaneously through the architecture. The filtering and exception processing mechanisms will be discussed in more detail in a later section.

NHSM	**Init**	**Estbld**	**Add**	**Change**
+nexthop_open	>>Estbld	-	-	-
+ACCEPT	-	-	>> Self -ACCEPT	>> Self -ACCEPT
+ACCEPT_LAST	-	-	>> Estbld -ACCEPT	>> Estbld -ACCEPT
+CHANGE	-	>> Change -CHANGE	>> Self -queue	>> Self -queue
+CONNECT	-	>> Add -CONNECT	>> Self -queue	>> Self -queue
+DISCONNECT	-	>> Self -DISCON	>> Self -DISCON	>> Self -DISCON
+DISCONNECT_LAST	-	-	>>Establd -DISCON	>>Establd -DISCON
+DISCONNECT_ALL	-	>>Init -DISCON	>>Init -DISCON	>>Init -DISCON
+REFUSE	-	>> Self -REFUSE	>> Self -REFUSE	>> Self -REFUSE (-CH_DISC) (-CH_REF)
+REFUSE_LAST	-	-	>> Estbld -REFUSE	>> Estbld -REFUSE (-CH_DISC) (-CH_REF)
+REFUSE_ALL	-	>>Init -REFUSE	>>Init -REFUSE	>>Init -REFUSE (-CH_DISC) (-CH_REF)

Table 3: NHSM

NHSM	Init	Estbld	Add	Change
+RetryTimeout	-	-	>> Self -REFUSE	>> Self -REFUSE (-CH_DISC) (-CH_REF)
+RetryTimeout_Last	-	-	>> Estbld -REFUSE	>> Estbld -REFUSE (-CH_DISC) (-CH_REF)

Table 3: NHSM

The 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. If the TargetList causes a too long CONNECT message to be generated, the CONNECT message is partitioned.

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. 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.

If an Application at a Target does not wish to participate in the stream, it sends a REFUSE message back to the Origin with a ReasonCode (ApplDisconnect). When an NHSM receives a REFUSE message with ReasonCode (ApplDisconnect), the acknowledgement has already been sent by the ST2 Dispatcher as an ACK to the next-hop. The Agent considers which resources are to be released, deletes the Target entry from the internal database, and propagates the REFUSE message back to the OSM or PHSM.

If, after deleting the specified Target, the next-hop has no remaining Targets, then those resources associated with that next-hop 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.

Change: The Application at the Origin may wish to change the FlowSpec of an established stream. To do so, it informs the OSM of the new FlowSpec and of the list of Targets relative to the change. The OSM then issues one CHANGE message with the new FlowSpec per next-hop and sends it with the correct Targetlist. The MS/C box then places copies (as required) of this in the NHSM queues.This takes the control to the **Change** state from the **Establd** state. CHANGE messages are structured and processed similar to CONNECT messages.

A next-hop agent that is an Intermediate Agent that receives a CHANGE message similarly determines if it can implement the new FlowSpec along the path to each of its next-hop 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 OSM.

At this point the Application decides whether all replies have been received. 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 ST2 agent receiving the CHANGE message makes the necessary request changes to the network resource allocations, and if successful, propagates the CHANGE message along the established paths. If the change cannot be made, but the I-bit indicates that stream should be torn down, then the ST2 Agent must recover using DISCONNECT and REFUSE messages as in the case of a network failure. Note that a failure to change the resources requested for specific Targets should not cause other targets in the stream to be deleted.

Data Forwarding: Once the Application or OSM determines that the stream is established Data may be transferred to the targets. An Application is not guaranteed that the data reaches its destinations: ST2 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 ST2 Agent forwards the data only along already established paths to Targets.

Since a path is considered to be established when the ST2 next-hop agent on the path sends an ACCEPT message, it implies that the target and all other intermediate ST2 Agents on the path to the Target are ready to handle the incoming data packets. In no case will an ST2 Agent forward data to a next-hop Agent that has not explicitly accepted the stream.

At the end of the connection setup phase, the Origin, each Target, and each Intermediate ST2 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 Agents or networks. The database should be optimized to make the packet forwarding task most efficient. The time critical operation is an Intermediate Agent receiving a packet from the previous-hop Agent and forwarding it to the next- hop 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 ST2 Agent receives data packets encapsulated by an ST header. A data packet received by an ST2 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 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 FlowSpec contained in the ACCEPT message. 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 ST2 Agent at the Origin generates an error and it does not forward the data.

3.4.4 Previous Hop State Machine (PHSM)

The Previous Hop State Machine Model is common to a Target or Intermediate Agent. A PHSM communicates with an upstream NHSM and downstream with one or more NHSMs and/or a TSM via a MS/C box. When a CONNECT message is received, the Intermediate ST2 Agent invokes the routing function, reserves resources via the Local Resource Manager, and then propagates the CONNECT messages to its next-hops. For the most part the Intermediate Agent behaves like a relay. In the cases when the Intermediate Agent is not able to successfully send out a -CONNECT message to a downstream PHSM, a -REFUSE message from the PHSM is sent to the upstream NHSM..

The PHSM consists of a small number of states: **Init, Establd, Add and Change.**

Init: The Application initially takes control from the **Init** state to the **Establd** state via the **phsm_open** predicate. A DISCONNECT or REFUSE of all Targets in a stream will take the stream from the **Establd** to a terminating state which is also the **Init** state.

Establd:Once in the Establd state, Targets may be added or changed by the Origin or Targets may request to join the stream. The processing of a JOIN request is always handled by either an OSM or a PHSM. Within each ST2 Agent, the ST2 Dispatcher examines incoming JOIN requests and determines whether the stream referenced is a stream that that Agent supports. If not, the JOIN is forwarded on towards the Origin. Once a JOIN request reaches an Agent that can process the JOIN, the ST2 Dispatcher ACKs the JOIN and queues it up to the resident OSM or PHSM. When stream authorization has completed successfully, the PHSM issues a CONNECT through the MS/C Box to either a NHSM or a TSM.

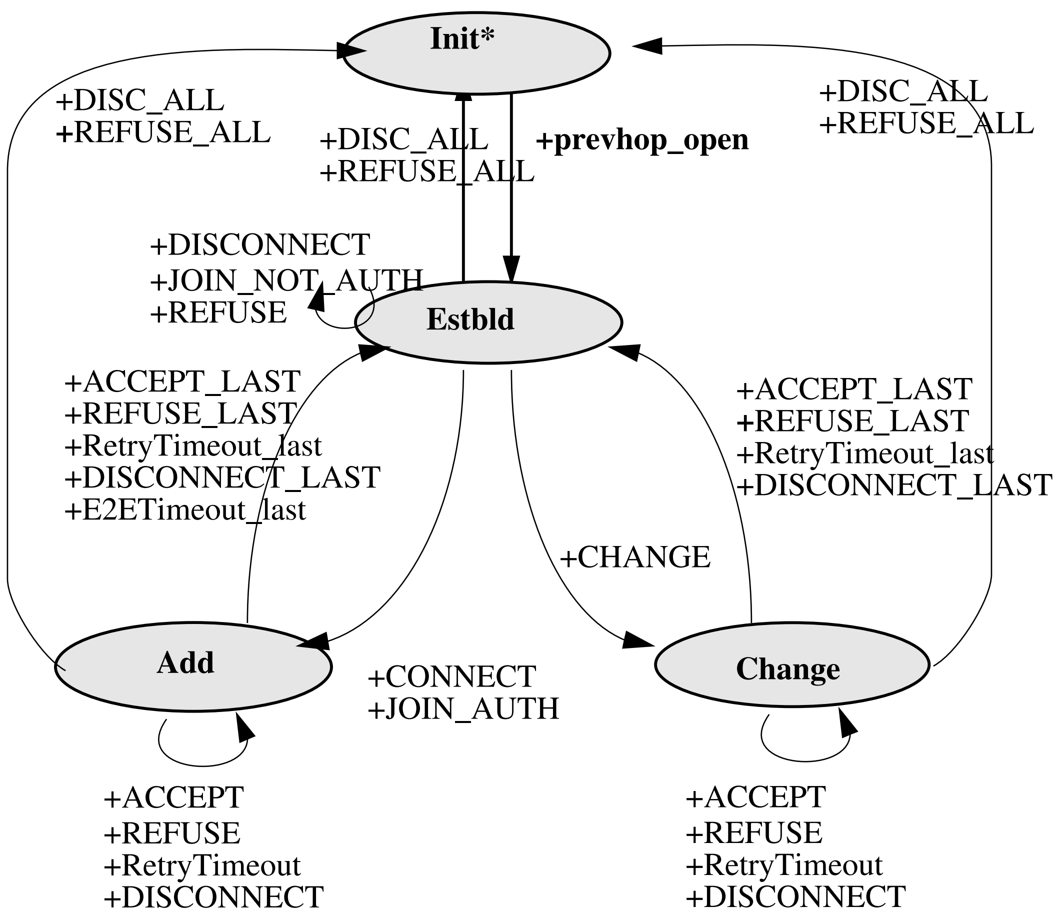
As previously described in the OSM, an ST2 Agent can handle only one stream Add or Change at a time. If such a stream operation is already underway, further requests are queued and handled when the previous operation has been completed. Either a DISCONNECT or REFUSE for all Targets transfers control from the Establd state to the Init state.

Add: Once in the Establd state the previous hop may relay a CONNECT message. A transition to Add will create a CONNECT message that is placed in the FIFO queue between the PHSM and the MS/C box. The CONNECT message contains the SID, an updated FlowSpec, and a TargetList. The MS/C box will then make a copy of the CONNECT message, partition the Targetlist parameter and place it the NHSM and/or TSM queues.The spliting (or separating) information is derived from the implementation's routing and LRM functions.

Once in the **Add** state the OSM waits to get ACCEPT or REFUSE responses. The stream will not transition back to the **Establd** state until all Targets have responded. The expiration of the retry timer and count (if the next hop is not ACKing the request) or the expiration of the end-to-end timer will be interpreted as an implicit refuse.

Change:The Application at the Origin may wish to change the FlowSpec of an established stream. To do so, it informs the ST2 Agent at the Origin of the new FlowSpec and of the list of Targets relative to the change and this message will be propagated to through the NHSMs to the PHSMs and TSMs. The control flow to the **Change** state is very similar to the previous FSM discussions.

Figure 9. Previous Hop State Machine (PHSM)



PHSM	Init	Estbld	Add	Change
+ prevhop_open	>> Estbld	-	-	-

Table 4: PHSM

PHSM	Init	Estbld	Add	Change
+ACCEPT	-	-	>> Self -ACCEPT	>> Self -ACCEPT
+ACCEPT_LAST	-	-	>> Estbld -ACCEPT	>> Estbld -ACCEPT
+CHANGE	-	>> Change -CHANGE	>> Self -queue	>> Self -queue
+CONNECT	-	>> Add -CONNECT	>> Self -queue	>> Self -queue
+JOIN_AUTHORIZED	-	>> Add -CONNECT	>> Self -queue	>> Self -queue
+JOIN_NOT_AUTHORIZED	-	>> Self -JOIN-REJ	>> Self -queue	>> Self -queue
+DISCONNECT	-	>> Self -DISCON	>> Self -DISCON	>> Self -DISCON
+DISCONNECT_LAST	-	-	>>Establd -DISCON	>>Establd -DISCON
+DISCONNECT_ALL	-	>>Init -DISCON	>>Init -DISCON	>>Init -DISCON
+REFUSE	-	>> Self -REFUSE	>> Self -REFUSE	>> Self -REFUSE (-CH_DISC) (-CH_REF)
+REFUSE_LAST	-	-	>> Estbld -REFUSE	>> Estbld -REFUSE (-CH_DISC) (-CH_REF)
+REFUSE_ALL	-	>>Init -REFUSE	>>Init -REFUSE	>>Init -REFUSE (-CH_DISC) (-CH_REF)
+RetryTimeout	-	-	>> Self -REFUSE	>> Self -REFUSE (-CH_DISC) (-CH_REF)

Table 4: PHSM

PHSM	Init	Estbld	Add	Change
+RetryTimeout_Last	-	-	>> Estbld -REFUSE	>> Estbld -REFUSE (-CH_DISC) (-CH_REF)
+E2ETimeout_last	-	-	>> Estbld -DISCON -NOTIFYi	-

Table 4: PHSM

3.5 The Target State Machine (TSM)

The Target State Machine (TSM) is a high level state machine which communicates with a PHSM, or OSM if residing the same Agent as the Origin. The TSM also talks to the Upper Layer Protocol via primitives. The TSM consists of a small number of states: **Init, Establd, Add and Change.**

Init: The Application initially takes control from the **Init** state to the **Establd** state via the **tsm_open** predicate. 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 ST2 Agent together with the FlowSpec. The Application directs the TSM to generate a JOIN message containing the Application's request to join the stream and sends it to the PHSM which in turn sends it upstream toward the stream Origin.

An ST2 Agent receiving a JOIN message for which that Agent has a matching stream , 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 ST2 Agent is not traversed by the stream that has to be joined, it propagates the JOIN message toward the stream's Origin. Eventually, an ST2 Agent traversed by the stream or the stream's Origin itself is reached. In any case, the TSM will eventually receive a JOIN-REJECT or CONNECT response. This is shown as transitions to the **Establd** state and the **Add** state respectively.

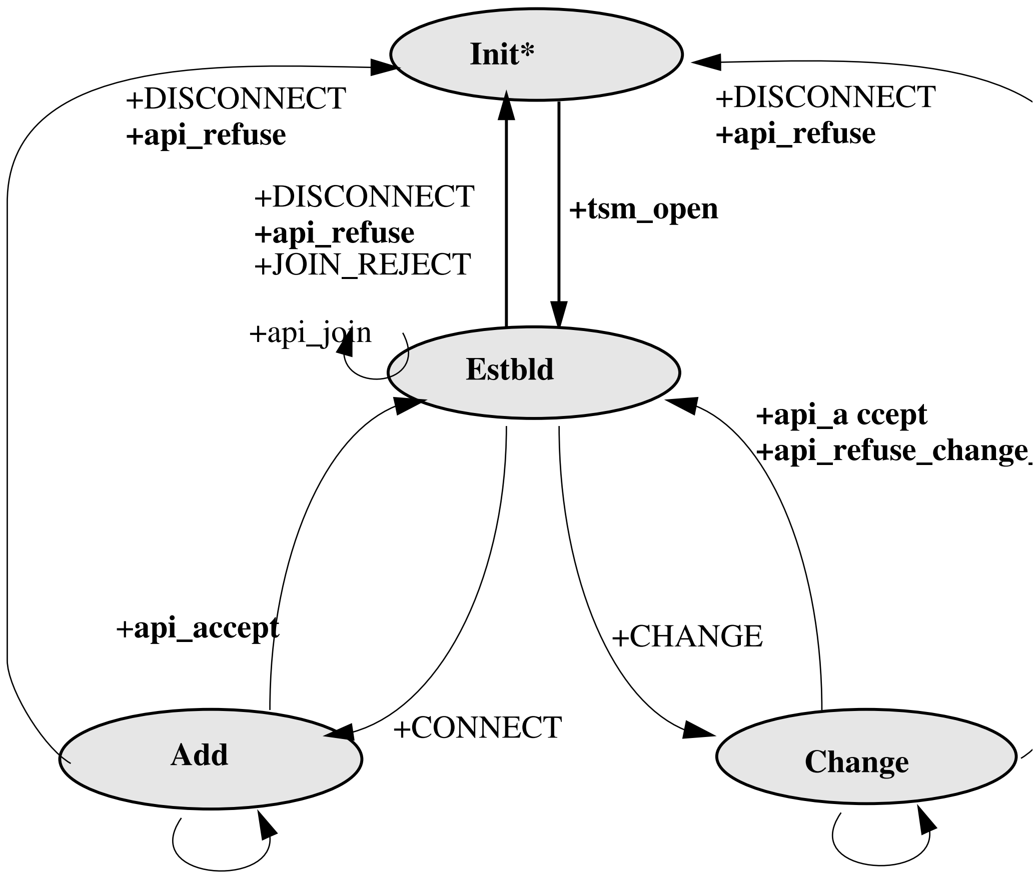
Add: The TSM may receive a +CONNECT message any time . The ST2 Agent reserves local resources and inquires from the specified Application process whether or not it is willing to accept the connection. In particular, the Application must be presented with parameters from the CONNECT, such as the SID, 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.

The TSM responds with an ACCEPT or REFUSE - a result of the Upper Layer Protocol decision.

Change: The TSM may receive a CHANGE message any time it is in a **Establd** state. This happens always after a CONNECT. The TSM again responds with an ACCEPT or REFUSE after informing the Upper Layer Protocol.

The TSM may any time want to terminate its membership in the stream. This is handled by the TSM sending out a REFUSE message. On the other hand it is possible for an Origin or IntermediateAgent to disconnect the Target from the stream. This is accomplished by the Agent or Origin sending a -DISCONNECT message.

Figure 10. Target State Machine (TSM)



TSM	Init	Estbld	Add	Change
+tsm_open	>>Estbld	-	-	-

Table 5: TSM

TSM	Init	Estbld	Add	Change
+api_accept	-	-	>> Estbld -ACCEPT	>> Estbld -ACCEPT
+CHANGE	-	>> Change -change_api	>>self -queue	>>self -queue
+CONNECT	-	>> Add -connect_api	-	-
+DISCONNECT	-	>> Init -disconnect_api	>> Init -discon_api	>> Init -discon_api
+JOIN_REJECT	-	>> Init -join_reject_api	-	-
+api_join	-	>> Self -JOIN	-	-
+api_refuse	-	>> Init -REFUSE	>> Init -REFUSE	>> Init -REFUSE
+api_refuse_change	-	-	-	>> Estbld -REFUSE

Table 5: TSM

SECTION 4

ST2 Agent FSMs

Section 2 summarized the ST2 Agent architecture, describing the ST2 Dispatcher and MS/C Box roles in the filtering, validation, queuing and creation of ST2 PDUs, as well as the CFSM relationships between the individual stream OSM, NHSM, PHSM and TSM partners. Section 3 detailed the normal operations of these individual stream FSMs. The roles of the Retry FSM and the Control FSM will now be discussed in more detail. First, the SCMP messages will described according to how they traverse the ST2 Agents and impact the individual Agent databases and states.

4.1 Control Message Traversal and Agent Database Context

ST2 Agent stream database entries are intitated by the first CONNECT for that Stream Id. The information initially correlated to each StreamId entry includes:

ST2 Neighbor Previous Hop and Next Hops

FlowSpec, Group, MulticastAddress, Origin, TargetList, ACK and Response timers

Stream Options for NoRecovery(S-bit) and Join Authorization Level (J-bit, N-bit)

Routing results for each Target's Next Hop

LRM results for each Next Hop's resource allocation

Each Agent database is modified when the CONNECT Responses indicate some variation specified by a downstream Agent or Target Response. Subsequent Requests can also modify the database and include additional CONNECT, JOIN and CHANGE requests. Origin, Network or Agent Recovery and LRM initiated stream teardown can occur in the form of explicit DISCONNECTs, REFUSEs and Recovery initiated CONNECTs or implicit conditions detected through the HELLO, STATUS, STATUS-RESPONSE and NOTIFY messages. The database context is then augmented with the history of Reason Codes, prior stream characteristics, G-bit (Global stream TargetList), I-bit and E-bit (CHANGE characteristics), J-bit and N-bit (Join level)and R-bit (Restarted Agent) values.

While all control messages may have an indirect effect on stream state and databases , only ACCEPT, CHANGE, CONNECT, DISCONNECT, JOIN-REJECT, JOIN and REFUSE directly affect each Agent's defintion of each stream. ACK, ERROR, HELLO, NOTIFY, STATUS and STATUS-RESP are the control messages that are primarily used to maintain ST2 Agent databases for datalink, neighbor and network management functions.

Thus as the ST2 control messages traverse the Agents, the databases are updated to include the additional information as approriate to the Agent and stream FSM implementations. Table 6. lists the control message with the scope of the control message flow in the following categories :

End-to-End - messages flow between the Origin and the Targets.

End-to-Intermediate & Intermediate-to-End- control messages can flow between a Target and an Intermediate (both directions) or between an Intermediate and Origin (both directions)

Local - a message traverses Upstream or Downstream but limited to one hop only.

Table 6: Control Message Direction

	Message Direction		
	End-to-End	End-to-Intermediate & Intermediate-to-End	Local
Message	Origin-to-Target (D) or Target-to-Origin (U)	Target-to-Interm (T->I) or Interm-to-Target (I->T) or Interm-to-Origin (I->O) or Origin-to-Interm (O->I)	U = {O, I} D = {I, T}
ACCEPT	Upstream	T->I, I->O	
ACK			Either
CHANGE	Downstream	O->I, I->T	
CONNECT	Downstream	O->I, I->T	
DISCONNECT	Downstream	O->I, I->T	
ERROR			Either
HELLO			Either
JOIN-REJECT	Downstream	O->I, I->T	
JOIN	Upstream	T->I	
NOTIFY	Either	Either	Either
REFUSE	Upstream	T->I, I->O	
STATUS			Either
STATUS-RESP			Either

As the ST2 PDUs traverse the network, each Agent presumably has a platform specific interface-to-packet-switching function that must intercept the ST2 packets for ST2 functions. The ST2 Dispatcher represents this PDU validation, filtering and packetswitching. The ST2 Dispatcher in this model is organized as the Agent packet-switcher, rather than as a per-interface or per-nex-hop packet-switcher. This function may be reorganized as a distributed function if the Agent platform architecture requires such distribution.

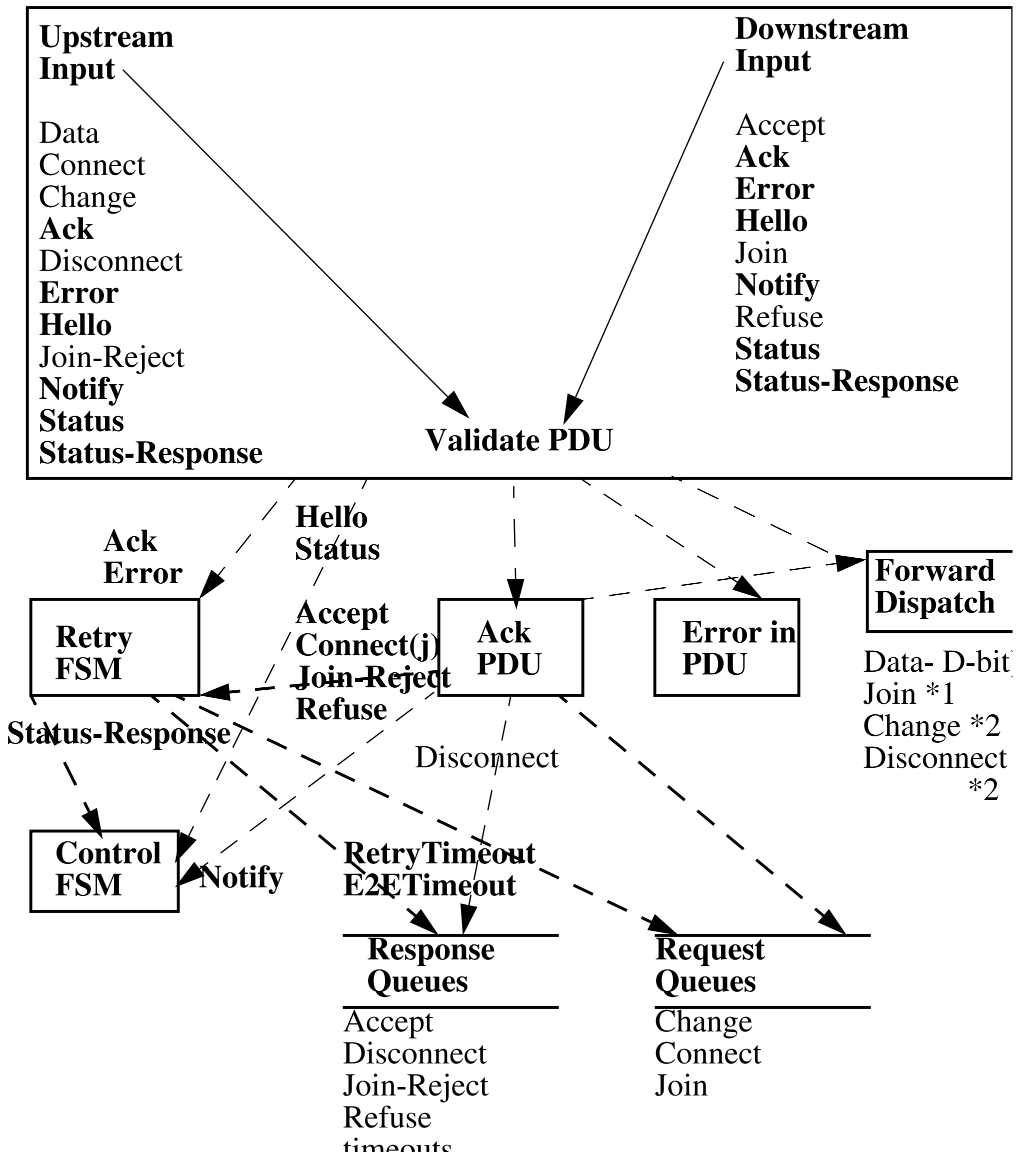
4.2 ST2 Dispatcher role for incoming Packet-switching, ACKnowledgement and PDU validation

An ST2 Dispatcher can validate an ST2 PDU for ST2 header and PDU semantic validity and then rapidly switch Data packets , .i.e to a local Target application SAP or to the appropriate next hop interface for remote Targets.

When the PDU semantics are in error, an ERROR PDU with the corresponding Reason Code and the offending PDU contents are returned to the SenderIpAddress (instead of an ACK for those SCMP messages that require an ACK). The PDU in ERROR is then discarded. The following Reason Codes detail the inconsistencies reported in an ERROR Response:

	2	ErrorUnknown	An error not contained in this list has been detected.
	8	AuthentFailed	The authentication function failed.
	13	CksumBadCtl	Control PDU has a bad message checksum.
	14	CksumBadST	PDU has a bad ST Header checksum.
	23	InvalidSender	Control PDU has an invalid SenderIPAddress field.
	24	InvalidTotByt	Control PDU has an invalid TotalBytes field.
*	26	LnkRefUnknown	Control PDU contains an unknown LnkReference.
	31	OpCodeUnknown	Control PDU has an invalid OpCode field.
	32	PCodeUnknown	Control PDU has a parameter with an invalid PCode.
	33	ParmValueBad	Control PDU contains an invalid parameter value.
	35	ProtocolUnknown	Control PDU contains an unknown next-higher layer protocol identifier.
	37	RefUnknown	Control PDU contains an unknown Reference.
	45	SAPUnknown	Control PDU contains an unknown next-higher layer SAP (port).
*	46	SIDUnknown	Control PDU contains an unknown SID.
	48	STVer3Bad	A received PDU is not ST Version 3.
	54	TruncatedCtl	Control PDU is shorter than expected.
	55	TruncatedPDU	A received ST PDU is shorter than the ST Header indicates.

Figure 11.
ST2 Dispatcher
Input



*1 SID not active on this Agent - forward toward Origin specified
***2** SID active,JOIN level 2, Target notlisted - forward to all ST2 nbrs

In some cases, SCMP PDUs may also be forwarded without impacting the local stream FSMs. A JOIN whose SID is not active on this Agent is sent to the appropriate neighbor, as indicated by the Origin to which the JOIN is directed. Also, when the PDU is a CHANGE or DISCONNECT for an active SIDand an unknown Target, but with a JOIN level of 2,, the PDU is forwarded to all of the stream next hops. This satisfies the flooding mechanism required to support a stream whose Origin and branches do not explicitly know all downstream Targets.

The next level of PDU analysis involves Agent and stream consistency. The PDU is examined for content consistency with both Agent and stream database information.The following detected inconsistencies may result:

3 AccessDenied Access denied.

4 AckUnexpected An unexpected ACK was received.

15 DuplicateIgn Control PDU is a duplicate and is being acknowledged.

16 DuplicateTarget Control PDU contains a duplicate target, or an attempt to add an existing target.

49 StreamExists A stream with the given SID already exists.

51 TargetExists A CONNECT was received that specified an existing target.

52 TargetUnknown A target is not a member of the specified stream.

53 TargetMissing A target parameter was expected and is not included, or is empty.

Most SCMP PDUs (except ACK, ERROR, HELLO, STATUS, STATUS-RESPONSE,) will trigger an ACK to the ST2 neighbor that sent the PDU.

CONNECT, CHANGE and JOIN Requests will be directed to the appropriate stream PHSM.

Incoming Responses are first correlated with any corresponding Request Reference so that the appropriate next hop or Response timer may be terminated. Then ACCEPT and REFUSE messages are queued up to the appropriate stream NHSM, while DISCONNECT and JOIN-REJECT messages arequeued up to the appropriate stream PHSM.

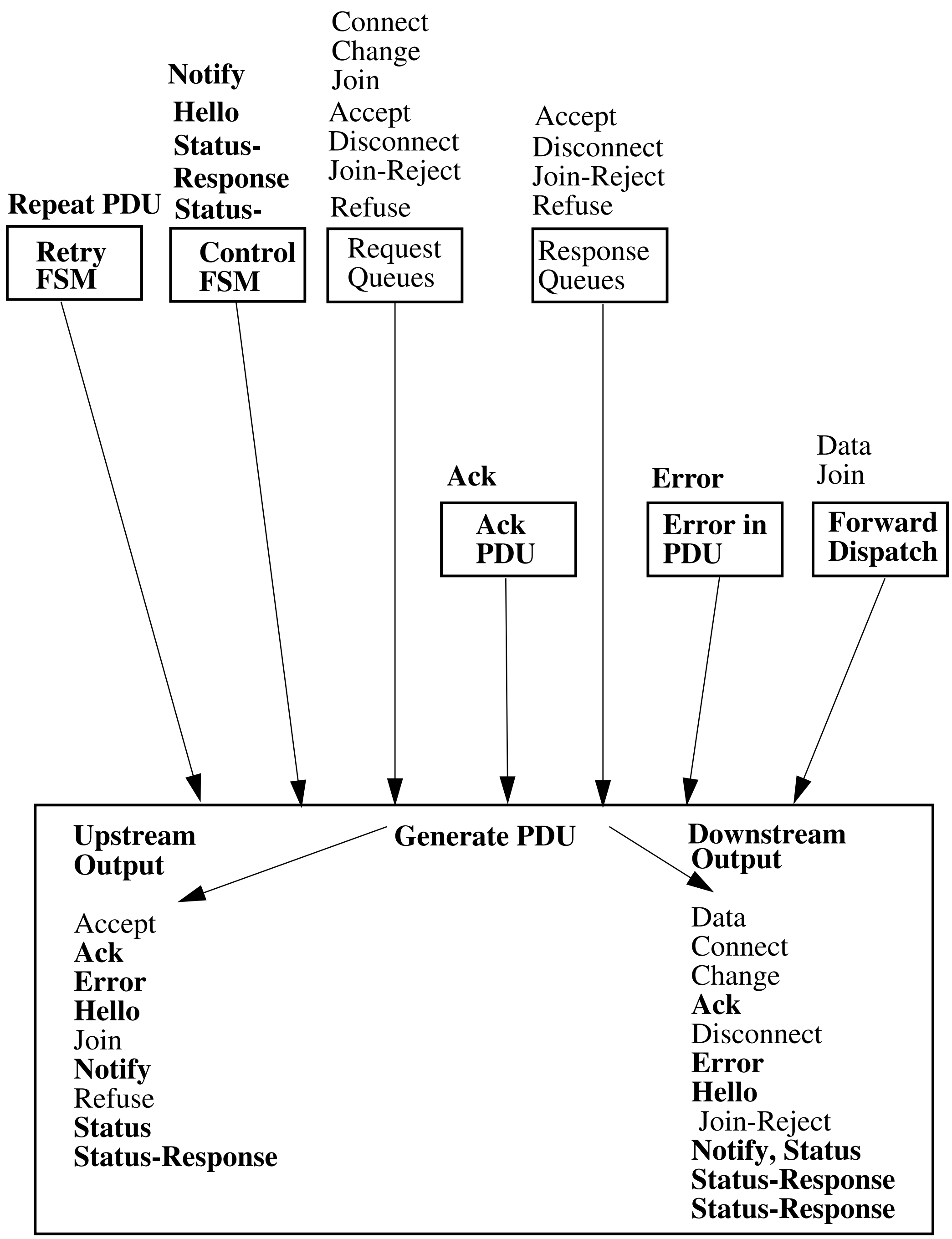
ACK and ERROR messages are correlated with a PDU Reference, terminating the appropriate timers, and then queued up to the stream Retry FSM.

HELLO, STATUS and STATUS-RESPONSE messages are correlated with a PDU Reference so that the appropriate timers may be terminated and then queued up to the Control FSM.

4.3 ST2 Dispatcher functions for outgoing Packet switching, timer and retry settings

Figure 12.

ST2 Dispatcher Output

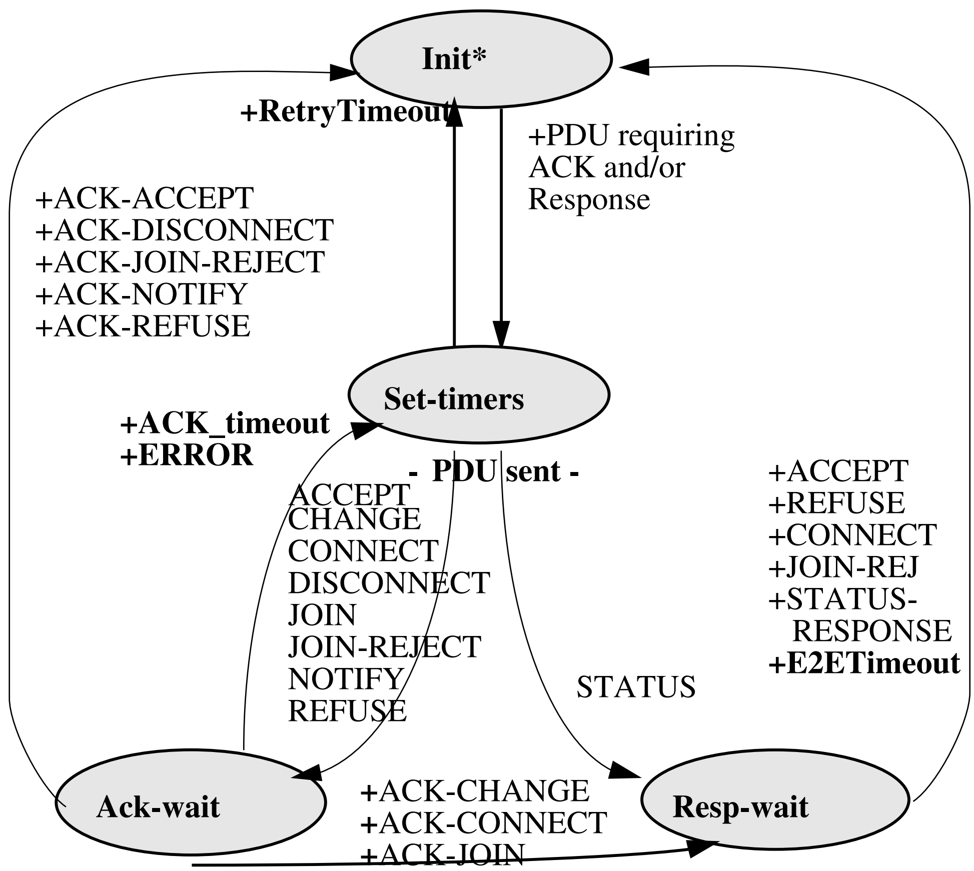


The ST2 Dispatcher also has the role of packaging and forwarding outgoing PDUs to the appropriate interfaces. The outgoing PDU must be given it's own PDU Reference number and any correlated PDU Referencer number, as well as the semantics and context of the PDU database entries. This Agent architecture model assumes that the Agent and stream databases are the intra-Agent repository of all activities, such that the ST2 Dispatcher can efficiently create and distribute the PDUs. However, it is entirely possible that the accumulated contents of a PDU has exceeded an outgoing MTU restriction and the PDU would be trunkated with the following Reason Codes:

6	UserDataSize	UserData parameter too large to permit a message to fit into a network's MTU.
36	RecordRouteSize	RecordRoute parameter is too long to permit message to fit a network's MTU.

4.4 Retry FSM- RFSM for datalink reliability of PDU transmissions

Figure 13. **Retry State Machine (RFSM)**



The Retry FSM has four states - **Init, Set-timers** (and check retry count), **Ack-wait** and **Resp-wai**t. The general paradigm for the Retry FSM is to move from the **Init** state to the **Set-timers** state whenthere is a PDU requiring an ACK and/or a Response, then transition to the appropriate

state to wait for the resultant ACKS, Responses and/or timeouts. PDUs requiring ACKS cycle through resends by NAccept, NChange, NConnect, NDisconnect, NJoin, NJoinReject, NNotify and NRefuse configured counts.

Since all ACKs and Responses are correlated by PDU Reference numbers, packets are correlated to the outstanding Retry FSM by the same mechanism. Either an ACK or a RetryTimeout that is correlated to an ACCEPT, DISCONNECT, JOIN-REJECT, NOTIFY or REFUSE results in the Retry FSM transitionsto Init. Such PDUs have no end to end response rquirements and generally have no secondary error processing when it can be assumed that the neighbor Agent and/or link level reliability is gone.. An ACCEPT is an exception. The failure of an ACCEPT is an implicit REFUSE upstream and DISCONNECT downstream.since the end to end Response has now failed to completely traverse the stream Agents.

An ACK Response to a CHANGE, CONNECT or JOIN results in a transtion to the Resp-wait state until either a Response is received or the Respsonse timers expire.

A legitimate Response, or a RetryTimeout or an E2ETimeout on CHANGE, CONNECT or JOIN causes the transition to the Init state with the signal to be replicated for individaul stream FSM.

In fact, only an OSM or a PHSM acting as an Origin for a JOIN will instantiate the Response-wait state of the Retry FSM. This could be inidcated by the implementation’s databases, or as some implementor’s prefer, variations of the Retry FSM could be incorporated into each of the OSM, NHSM, PHSM, TSM and Control FSMs. Section 5 will discuss some of the secondary and tertiary exception processing issues that may influence this implementation decision.

RFSM	**Init**	**Set-timersd**	**Ack-wait**	**Response-wait**
+Ack_timeout			>>Set-timer -resend PDU	
+ERROR			>>Set-timer -resend PDU	
+ACK-ACCEPT			>>Init	
+ACK-CHANGE			>>Response-wait	
+ACK-CONNECT			>>Response-wait	
+ACK-DISCONNECT			>>Init	

Table 7: PHSM

RFSM	Init	Set-timersd	Ack-wait	Response-wait
+ACK-JOIN			>>Response-wait	
+ACK-JOIN-REJECT			>>Init	
+ACK-NOTIFY			>>Init	
+ACK-REFUSE			>>Init	
+RetryTimeout-ACCEPT			>>Init -DISCON-NECT -REFUSE	
+RetryTimeout-CHANGE			>>Init -RetryTime-out	
+RetryTimeout-CONNECT			>>Init -RetryTime-out	
+RetryTimeout-DISCONNECT			>>Init	
+RetryTimeout-JOIN			>>Init -RetryTime-out	
+RetryTimeout-JOIN-REJECT			>>Init	
+RetryTimeout-NOTIFY			>>Init	
+RetryTimeout-REFUSE			>>Init	
+E2ETimeout		>>Init -E2ETimeout	>>Init -E2ETimeout	>>Init -E2ETimeout
+ACCEPT	-	>> Init -ACCEPT	>> Init -ACCEPT	>>Init -ACCEPT
+REFUSE	-	>> Init -REFUSE	>>Ini -REFUSE	>> Init -REFUSE
+CONNECT		>> Init -CONNECT	>> Init -CONNECT	>> Init -CONNECT
+JOIN-REJECT		>> Init -JOIN-REJECT	>> Init -JOIN-REJECT	>> Init -JOIN-REJECT

Table 7: PHSM

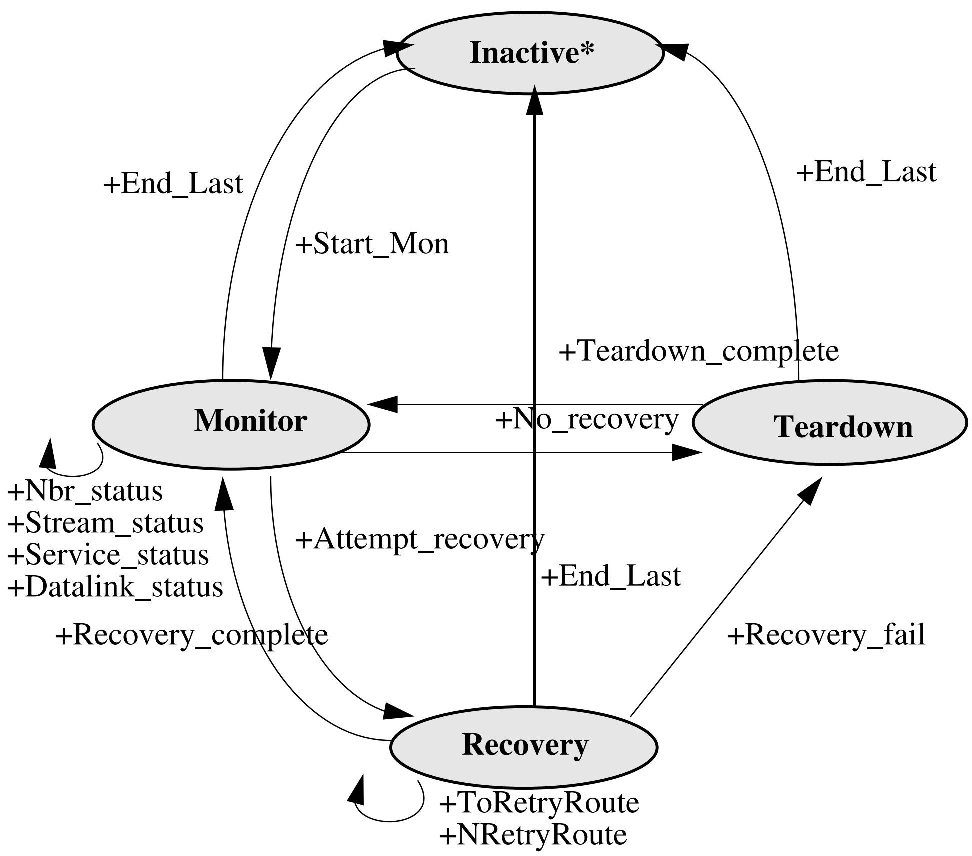
RFSM	Init	Set-timersd	Ack-wait	Response-wait
+STATUS-RESPONSE		>> Init -STATUS-RESPONSE	>> Init -STATUS-RESPONSE	>> Init -STATUS-RESPONSE

Table 7: PHSM

4.5 Agent , Neighbor and Stream Supervision

4.5.1 The Control FSM for Agent and Stream Supervision

Figure 14. .CFSM



Each ST2 Agent must monitor its own status, network conditions, neighbor Agent status and supervise the Recovery of streams whenever required and possible during a network failure.This

CFSM is intended to be a general approach to these issues, rather than a fully specified FSM since the particular network, platform and implementation architecture will determine detail FSM considerations.

What this CFSM model does suggest is that the CFSM provides a superstructure for the management of the Neighbor Detection Failure FSM, any Agent NOTIFY, STATUS or STATUS-RESPONSE implications, Service Model management (including application issues, routing and LRM or other Agent implementation specific issues), as well as datalink statistics analysis (e.g., broken or dropped PDUs or accumulated routing errors).

At the very least, stream Recovery requires careful analysis of the possible recursions in Agent, neighbor failure detection, routing and LRM conditions. The ST2+ specification defines parameters for a configured number of times that Recovery should be attempted (NRetryRoute), the configured time to wait for each Response (ToRetryRoute) and variations in the exception processing.

During the course of a stream setup, the CONNECT contains a Recovery Timeout, as specified by the Origin. The resultant ACCEPTs contain the individual Agent's "supportable" Recovery Timeout such that the stream Recovery Timeout becomes the smallest Recovery Timeout for all Targets. The HELLO timer must be smaller than the smallest Recovery Timeout for all streams between these Agents, but an Agent may have various HELLO timers between different Agents, such that the management funtion of such timers should fall into the CFSM also. A Round Trip Time (RTT) estimation function is available with STATUS and STATUS_RESPONSE messages to aid in this area.

The CFSM relies on the Nieghbor Detection Failure FSM (NDFSM) as the primary notification vehicle for stream and neighbor management. During the initial stream setup of any stream NHSM and PHSM, the CFSM is signalled to begin monitoring of the FSM neighbor Agents involved in the stream. The sending of HELLOs is begun once an ACCEPT is forwarded upstream. The receiving of HELLOs is acceptable as soon as an ACCEPT is received. HELLOs are terminated once an ACK is sent or received for the DISCONNECT or REFUSE associated with the last of all streams and Targets for that neighbor.This requires signalling and coordination with the ST2 Dispatcher, Retry FSM and database context, especially when the Restarted bit is active for either the local Agent of a neighbor.

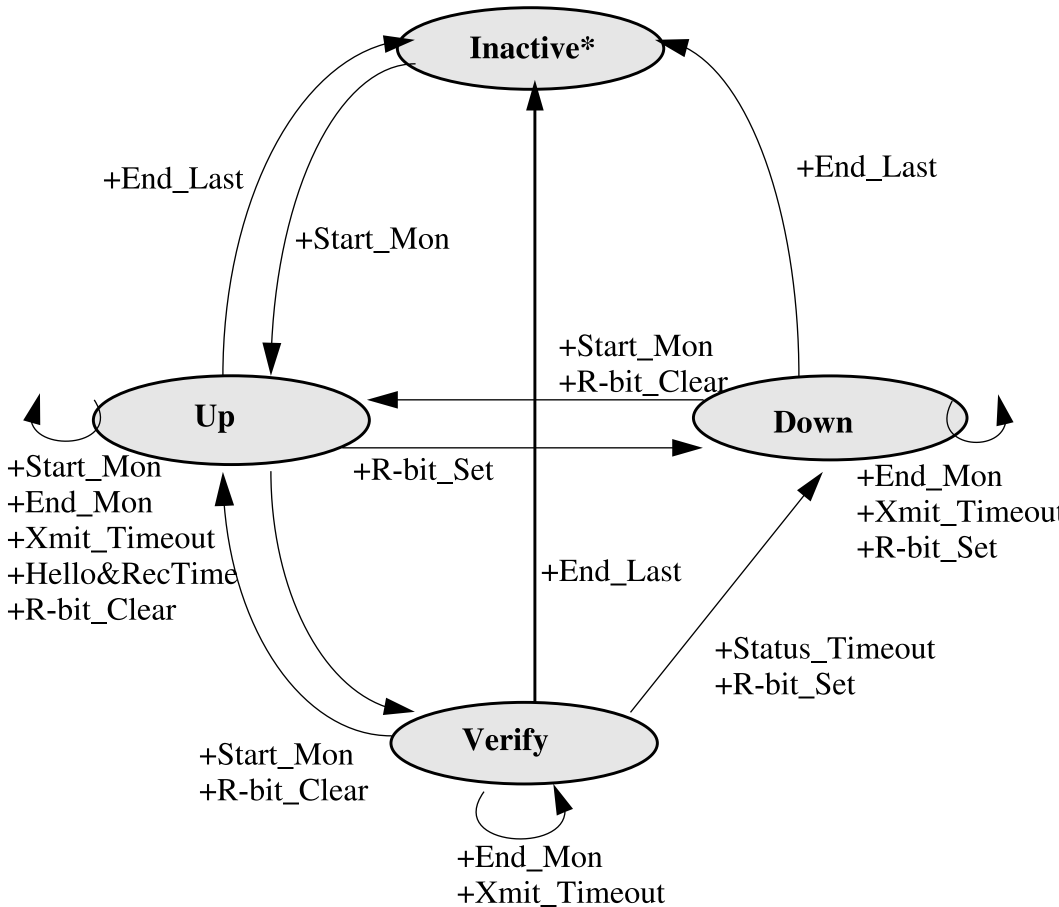
Agent network "inspection and repair" functions might also exist in the CFSM to extend the mechanisms of the NDFSM before attempting Recovery and/or stream teardown.

Group management for Bandwidth-sharing, Fate-sharing, Path-sharing and Subnet resource- sharing can be intiated by any ST2 Agent and it may be adviseable to incorporate optimization algorithms in the CFSM tointeract with routing and LRM function, thus allowing the CFSM to monitor and gauge the impact on the stream Recovery analysis.

4.5.2 The Nieghbor Detection Failure FSM for Neighbor Management

This FSM has a more atomic focus in that ST2 neighbor HELLOs are maintained and monitored only while there are one or more shared streams active. When the neighbor HELLOs and subsequent STATUS inquiry fails or the neighbor R-bit has been set, the neighbor is considered down and the streams involved in that neighbor relationship must be examined for Recovery conditions.

Figure 15. NFDSM



NFDSM	Inactive	Up	Verify	Down
+Start Monitoring	>> Up	>> Self	>> Up	>> Up
+End Monitoring	-	>>Self	>> Self	>> Self
+End Monitoring Last	-	>>Inactive	>>Inactive	>>Inactive
+Xmit Timeout	-	>> Self -Hello	>> Self -Hello	>> Self -Hello

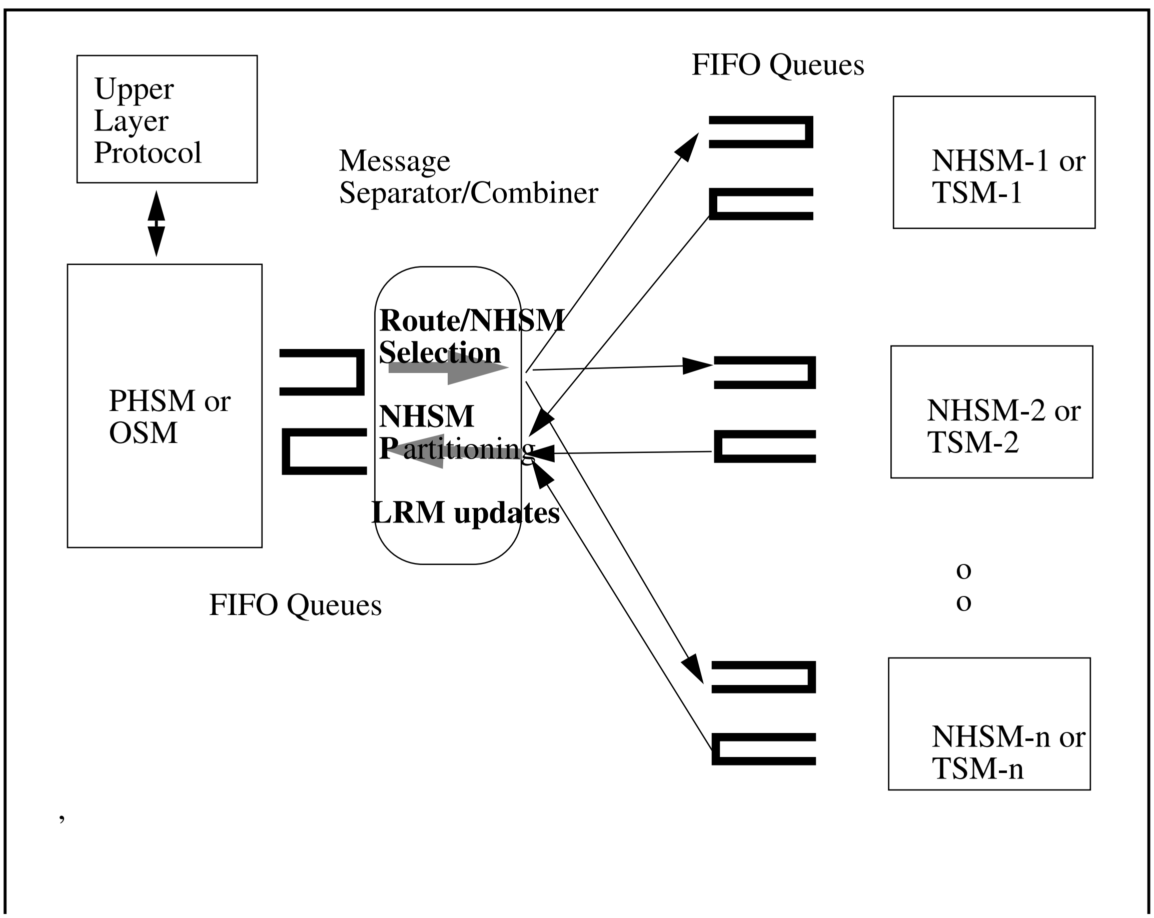
Table 8: NFDSM

NFDSM	Inactive	Up	Verify	Down
+Hello & Recovery Timeout	-	>> Self	-	-
+NO Hello & Recovery Timeout	-	>> Verify -Status (SID)	-	-
+Status Timeout	-	-	>> Down -N_Down	-
+R-bit Set	-	>> Down -N_Down	>> Down -N_Down	>> Self
+R-bit Clear	-	>> Self	>> Up	>> Up

Table 8: NFDSM

4.5.3 Service Model Interactions

Figure 16. MS/C Box Communications inside an Agent



The optimization of route and LRM functions can affect the selection from multiple path routes to a Target on initial CONNECTs, as well as CHANGE and Recovery procedures. This document's model follows a sequential process of integrating the route and LRM services with the MS/C Box. for the atomic individual stream FSMs.

Additional algorithms may be used in the CFSM, such that Options and Group factors may be optimized in relation to the stream Recovery decisions.

SECTION 5

Exception Processing

All the exception processing conditions have been referenced in the preceding sections. Not all have been spelled out in detail. The general paradigms fall into several categories and all of this document's models are based on a suggested approach. The secondary and tertiary conditions of some apects of exception processsing are especially subject to implementation preferences.

The first topic of discussion might be the category SCMP datalink reliability as generally characterized in the Retry FSM. This document favors maintaining a coordinating Retry FSM versus incorporating the Retry states in each of the OSM, NHSM, PHSM and TSM, which is naturally an alternative.

ERROR message generation for PDU semantics problems has discussed in Section 5 as an ST2 Dispatcher function. A special case occurs in PDU construction when the MTU size is exceeded, i.e.:

6 UserDataSize UserData parameter too large to permit a message to fit into a network's MTU.

36 RecordRouteSize RecordRoute parameter is too long to permit message to fit a network's MTU.

However, all of the analysis and potential REFUSE message or signal generation, still seems best suited to the ST2 Dispatcher.

5.1 Additional Exception Processing

5.1.1 ST2 Dispatcher detected inconsistencies with Reason Codes:

The following errors can also be detected by the ST2 Dispatcher with the careful analysis of all Agent and stream database values:

3 AccessDenied Access denied.

4 AckUnexpected An unexpected ACK was received.

15 DuplicateIgn Control PDU is a duplicate and is being acknowledged.

16 DuplicateTarget Control PDU contains a duplicate target, or an attempt to add an existing target.

49 StreamExists A stream with the given SID already exists.

51 TargetExists A CONNECT was received that specified an

existing target.

52 TargetUnknown A target is not a member of the specified stream.

53 TargetMissing A target parameter was expected and is not included, or is empty.

This means that the atomic FSMs do not have to incorporate this logic and simplifies core FSM paradigms.

5.1.2 Control FSM issues with neighbor failure and stream recovery with Reason Codes:

The details of these specific instances can be intertwined with Retry, Routing and LRM failures.

12 CantRecover Unable to recover failed stream.

22 IntfcFailure A network interface failure has been detected.

27 NetworkFailure A network failure has been detected.

39 RestartLocal The local ST agent has recently restarted.

40 RestartRemote The remote ST agent has recently restarted.

47 STAgentFailure An ST agent failure has been detected.

5.1.3 Retry and Timeout Failures with Reason Codes:

38 ResponseTimeout Control message has been acknowledged but n**ot** answered by an appropriate control message.

41 RetransTimeout An acknowledgment has not been received after several retransmissions.

5.1.4 Routing issues with Reason Codes:

Routing issues initiate special exception processing requirements. Some of these have been addressed in the ST2+ specification, but each implementation should consider the network and platform architecture, also.

9 BadMcastAddress IP Multicast address is unacceptable in CONNECT

28 NoRouteToAgent Cannot find a route to an ST agent.

29 NoRouteToHost Cannot find a route to a host.

30 NoRouteToNet Cannot find a route to a network.

34 PathConvergence Two branches of the stream join during the CONNECT setup.

42 RouteBack Route to next-hop through same interface as previous-hop and is not previous-hop.

43 RouteInconsist A routing inconsistency has been detected.

44 RouteLoop A routing loop has been detected.

5.1.5 LRM issues with Reason Codes:

Optimization of routing and LRM issues can also initiate special exception processing requirements. Some of these have been addressed in the ST2+ specification, but each implementation should also consider the network and platform architecture.

10	CantGetResrc	Unable to acquire (additional) resources.
11	CantRelResrc	Unable to release excess resources.
17	FlowSpecMismatch	FlowSpec in request does not match existing FlowSpec.
18	FlowSpecError	An error occurred while processing the FlowSpec.
19	FlowVerUnknown	Control PDU has a FlowSpec Version Number that is not supported.
20	GroupUnknown	Control PDU contains an unknown Group Name.
21	InconsistGroup	An inconsistency has been detected with the streams forming a group.
50	StreamPreempted	The stream has been preempted by one with a higher precedence.

SECTION 6

Appendix A

6.1 ST Control Message Flow

Control Message Types

ST2 control messages are generally of the request -response type. Table 1 summarizes these control messages alphabetically. The table has three major columns.

- Message type
- Response
- Possible causes for message

6.1.1 Message Type

Under the Message Type each control message is categorized either as a:

— Request message

— Response message

It is possible for a message to be more than one type depending on the usage, although this is not apparent from this table.

6.1.2 Response

The response to each control message is given in the next major column under Response. Note that the response to a message can be interpreted to mean either:

1. a response to another control message
2. a response to indicate the condition of receipt of the message, driven primarily by the error control function

The second interpretation of response includes positive acknowledgments and negative acknowledgments (error response). Thus, this major column has the following categories:

— Error Response

— Mandatory Response

— Other response following mandatory response.

An X or an entry in the table indicates classification of a message under a particular category shown under each major column.

6.1.3 Possible causes for message

Finally, a control message might have been sent in response to another control message. This is shown in the last column. Note that it is possible that independently a number of control messages may be the cause for this control message in question Note that an entry does not necessarily mean that is the only cause. A blank entry in this column for instance means that the message was not invoked by another message.

For example, an ACCEPT message is a Response Type message to either a CONNECT or a CHANGE message. It will be acknowledged (Mandatory response) with an ACK. It may be responded with an ERROR in case of error conditions. The state diagrams illustrate this sequencing more completely. It may be noted that the sequencing of messages gives the protocol semantics.

.

Table 9: Message Types: Requests, Responses and Others

	Type		Response			Possible causes for message:
Message	Req.	Resp.	Error Resp.	Man-datory Resp.	Other response following Mandatory Resp.	
ACCEPT		X	ERROR	ACK		1. CONNECT 2. CHANGE
ACK		X	ERROR			1. ACCEPT 2. CHANGE 3. CONNECT 4. DISCONNECT 5. JOIN 6.JOIN-REJECT 7. NOTIFY 8. REFUSE
CHANGE	X		ERROR	ACK	ACCEPT REFUSE	1.Origin Change stream
CONNECT	X	X	ERROR	ACK	ACCEPT REFUSE	1.Target JOIN 2. Origin Connect TargetList

Table 9: Message Types: Requests, Responses and Others

	Type		Response			
Message	Req.	Resp.	Error Resp.	Man-datory Resp.	Other response following Mandatory Resp.	Possible causes for message:
DISCONNECT	X		ERROR	ACK		1.Origin Disconnect TargetList 2. Intermediate Agent detects upstream failure or is acting as an Origin
ERROR		X				Errored mesgs.: 1. ACCEPT 2. ACK 3. CHANGE 4. CONNECT 5. DISCONNECT 6. HELLO 7. JOIN 8. JOIN-REJECT 9. NOTIFY 10.REFUSE 11.STATUS 12. STATUS-RESPONSE
HELLO	X		ERROR			1.Periodic Message
JOIN-REJECT		X	ERROR	ACK		1. JOIN
JOIN	X		ERROR	ACK	CONNECT JOIN-REJECT	Target joining stream
NOTIFY		X	ERROR	ACK		1. Information message 2. Notification upstream to JOIN for some authorization levels

Table 9: Message Types: Requests, Responses and Others

	Type		Response			Possible causes for message:
Message	Req.	Resp.	Error Resp.	Man-datory Resp.	Other response following Mandatory Resp.	
REFUSE	X	X	ERROR	ACK		1.Target leaving stream 2. CONNECT 3. CHANGE 4.Intermediate Agent detects downstream stream failure
STATUS	X		ERROR	STATUS-RESPONSE		
STATUS-RESPONSE		X	ERROR			1.STATUS

APPENDIX B

The following internetwork diagram of ST2 Agents indicates the Origin (O), Intermediate (I) and Target (T) roles of each ST2 Agent in relation to two ST2 conferences. Conference 1 (C1) has 4 participants, all of which are sending and receiving data from each other. Intermediate Agents 1, 2, 3 and 4 each have an ST2 application with an Origin sending data to all other members, and simultaneously receiving data as a Target for each of the other members.

Figure 17.

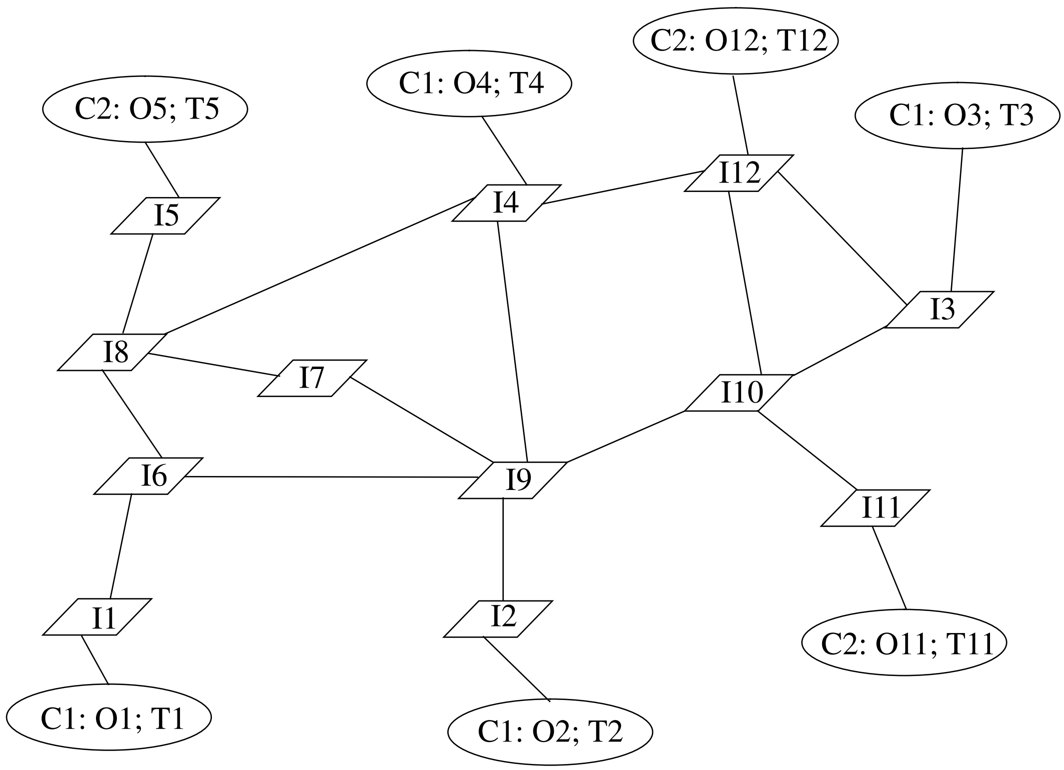
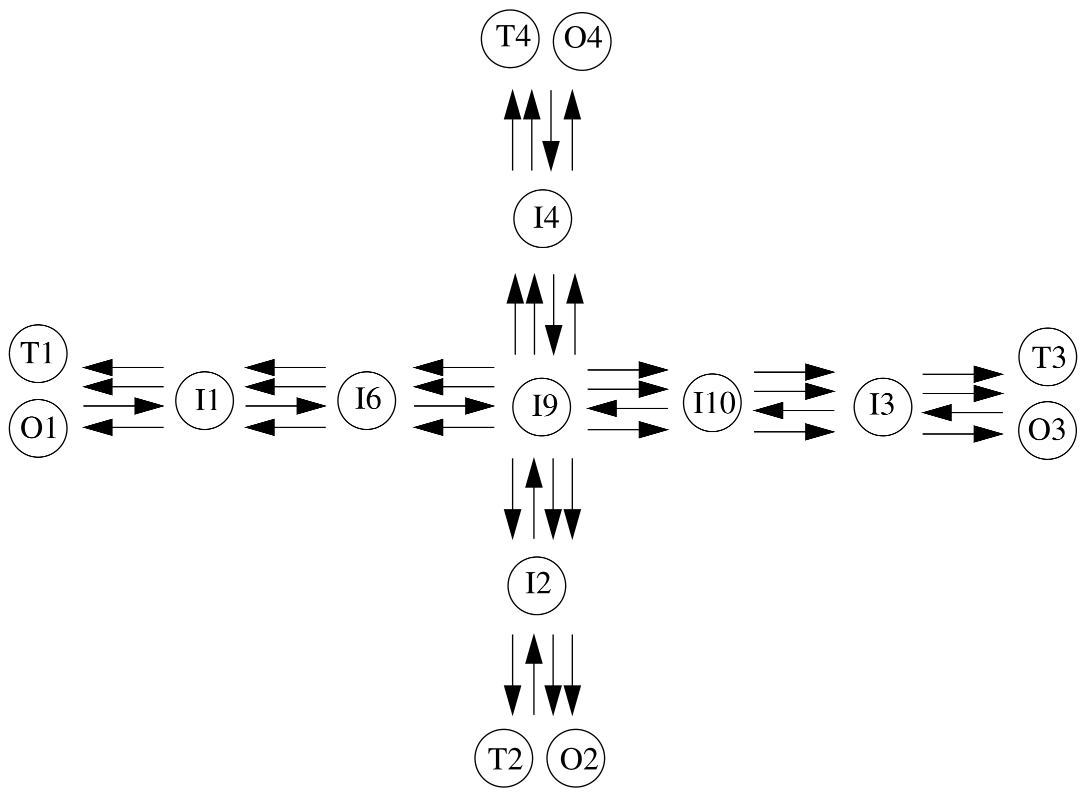


Figure 18.

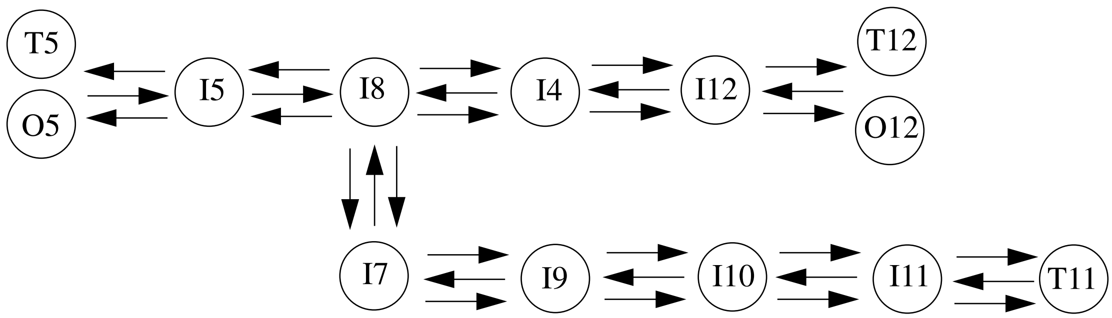


Each ST2 Agent participating in C1, as illustrated, has all four streams to manage, representing a fully meshed stream conference with Targets and Origins communicating along the same paths.

There are a number of other possible routes for each individual stream in C1. The above paths through Agents I6, I9 and I10 were chosen to illustrate a simple routing scheme for such a conference. Agents I8 and I12 could have just as easily been involved, if there were no other routing metrics to consider other than number of hops. However, it is also possible that the resources available at any Agent or interface may not actually be equal , such that Agents I7 and I12 became involved in branches of some of the streams. In such alternative routing and resource circumstances, some of the Intermediate Agents might only maintain one stream in the conference. However, in this illustration, Agent I9 happens to have an ST2 neighbor for every stream and the need to manage multiple targets for each stream.

Conference 2 (C2) has 3 participants and is also a fully meshed set of streams for each member of the conference. All ST2 Agents in the C2 illustration also have multiple ST2 neighbors, streams and interfaces to manage.

Figure 19. ,



In addition, since both conferences are being conducted simultaneously, several Agents are managing streams from both conferences, which may be Grouped for Resource or Fatesharing characteristics. The dynamics of such internetwork topology and resource issues can become complex stream management issues.

Internet-Draft **November, 1995**

ST Working Group M. Rajagopal and Sharon Sergeant

File: draft-ietf-st2-state-01.ps Expires: May, 1996

ACKNOWLEDGEMENTS and AUTHORS:

Many individuals have contributed to the work described in this memo. We thank the participants in the ST Working Group for their input, review, and constructive comments.

We would also like to thank Luca Delgrossi and Louis Berger for allowing us to adopt the text from their [1] document.

We would like to acknowledge inputs from Mark Pullen and his graduate students, Tim O'Malley, Eric Crowley, Muneyoshi Suzuki and many others.

Murali Rajagopal EMail: murali@fbcs.com, Phone: 714-764-2952

Sharon Sergeant EMail:sergeant@xylogics.com, Phone: 617-893-6142

LIST OF REFERENCES:

[1] L. Delgrossi and L. Berger: Internet STream Protocol Version 2 (ST2) - Protocol Specification-Version ST2+, RFC 1819 , August 1995.

[2]D. Brand and P. Zafiropulo: On Communicating Finite-State Machines, J.ACM, 30, No.2, April 1983