Internet Draft R. Braden
Expiration: May 2003 USC/ISI
File: draft-braden-2level-signal-arch-01.txt B. Lindell
USC/ISI

A Two-Level Architecture for Internet Signaling

November 3, 2002

Status of Memo

Abstract

This memo defines an architectural framework for a wide variety of Internet signaling protocols. This framework has a two-level organization: a common lower layer "transport" protocol together with a suite of upper-level signaling protocols. The common lower level protocol CSTP (Common Signaling Transport Protocol) provides a transport-like service that may include reliable delivery and soft state management. The upper layer protocols, which implement algorithms and data structures specific to particular signaling applications, are generically called ULSPs (Upper-layer Signaling Protocols). This memo motivates the two-level design and describes the service model, API, and operation of the lower level CSTP.

Table of Contents

0. Changes in This Version

- The text now foregrounds the support for two different signaling models, with and without soft state. The previous version had both models, but it was not somewhat buried.

- The term ALSP is replaced by ULSP. We considered adopting one of the recently proposed sets of names for the CSTP and ULSP layers, but after careful thought decided that for now CSTP and ULSP are the best terms we can find.

- We included some NSIS working group issues, such as path-coupled signaling. (With respect to "peer" vs. "neighbor", see the first sentence of Section 2.)

- To make the job of the NSIS working group both harder (!), this revision introduces the alternative to basing CSTP either on TCP (section 3.3) or on the RSVP V1 mechanism (section 3.2). This choice does not affect the API or the ULSPs.

- We made several additions and corrections pointed out by Xingguo Song (see Acknowledgments.)

- We interchanged the terms INFO and EVENT to provide more intuitive terminology, and supplied some missing API calls.

1. Introduction

This memo presents the "Internet signaling protocol suite" (ISPS) framework, a unified architectural framework for the specification and implementation of a wide variety of Internet signaling protocols.

The ISPS framework composes Internet signaling protocols using two protocol levels: (1) a common lower level protocol and (2) a set of upper-level signaling functions specific to particular signaling applications. In particular, ISPS includes a common lower-level protocol called CSTP ("Common Signaling Transport Protocol") to implement transport and state-management functions, plus a suite of higher-level "User-Layer Signaling Protocols" (ULSPs). Each ULSP implements the algorithms and data structures for a particular signaling task.

The remainder of this section presents background and motivation and then introduces some terminology. Section 2 defines the functions and API that CSTP provides to a ULSP. Section 3 describes two proposals for the CSTP protocol, CSTP/IP and CSTP/TCP.

This memo makes several references to the RSVP Version 1 specifications [RFC2205, RFC2961]. Familiarity with these specifications may be useful but is not required to read the present memo.

1.1 Background

Under the basic Internet architecture, routers are unaware of individual user flows or even flow aggregates; routers are stateless except for routing tables that are used by all data packets equally. While this basic model has proven extremely powerful, it has become necessary to engineer into network nodes some flow awareness for particular functions. These functions include support for Quality-of-Service (QoS), control of middleboxes, VPN control, and access-link management, for example. Such flow-dependent functions generally require that some control state be installed into network nodes, either statically by configuration or dynamically using a "signaling" protocol.

The IETF defined RSVP Version 1 [RFC2205,Refresh00] specifically for signaling to support the Integrated Services QoS model [ISint93], but many RSVP extensions have been developed or proposed to support a variety of other Internet signaling applications. These applications include: QoS setup across diff-serv clouds [intdiff00], setting up MPLS paths with QoS

[mpls00], provisioning VPNs [aggr01], QoS setup for access networks [PCQoS99], NAT and firewall provisioning [TIST02], and active interest filtering for distributed simulation [AIF01]. With these extensions, RSVP Version 1 has in effect been expanded to define a suite of Internet signaling protocols.

Basing all of these protocols on RSVP brings some unity that is highly desirable. For example, the various signaling applications benefit from RSVP's transport, routing, and soft-state mechanisms as well as from its strongly-typed encoding. Using a common protocol base also has benefits in design economy and documentation. On the other hand, the complexity of the resulting multi-featured RSVP implementations and the confusion of feature interactions are the source of considerable complexity and some confusion.

The unified ISPS framework described in this memo is designed to organize and simplify the design and implementation of a wide variety of signaling applications, while building on the most successful aspects of RSVP V1. The two levels provide the software engineering advantages of modularity, including commonality, clarity, and reusability. For example, the framework should allow the transport functions of CSTP to evolve independently of the signaling application protocols. In particular, this document proposes two quite different approaches to CSTP in Section 3, a choice that should be transparent to every ULSP.

The two-level decomposition of the ISPS framework could be the first step towards a broader goal for unification: building the various ULSPs using a common set software building blocks. For example, it is possible that some sub-layering would be desirable within the ULSP level. However, we don't yet undersand how to take significant furthers step in this direction.

The Appendix A sketches how one would define a ULSP for QoS signaling with all the functions and features of RSVP V1. Although this member of the ISPS would not directly interoperate with RSVP Version 1, a signaling gateway could be developed to translate between RSVP Version 1 signaling messages and ISPS messages.

1.2 Terminology

We first introduce some useful terminology.

- Network Nodes

  We use the general term "network node", or simply "node", for a router or middlebox.

- Flow

  A flow is simply a distinguishable subset of the packet stream.

- Signaling

  The function of signaling is to set up state in one or more network nodes, to provide some desired service for user data flows.

  This definition makes no assumption about the degree of aggregation; a signaled flow may range from a micro-flow to all the traffic in a tunnel or trunk. The definition also does not assume that the endpoints of the signaling are end systems, or that state must be installed in every node along a path.

  By this definition, signaling is concerned with state setup along the path of some flow, rather than for example configuring an entire region of the network. It may be that some of the mechanisms for flow-related signaling would also be useful for regional state setup (i.e., network configuration), but regional state setup is outside the scope of the present ISPS framework.

- Path-Coupled Signaling

  Even for flow-related signaling, there is an engineering choice about whether the signaling is primarily performed in-line by the nodes through which the data flows, or whether it is performed by a distinct set of signaling engines. The first case is called "path-coupled signaling", while the second is "path-uncoupled".

- Signaled path

  Path-coupled signaling operates in the nodes along a "signaled path" between two (or more, for multicast)

"signaling endpoints". A signaling endpoint at which user data enters (or leaves) the signaled path is called " p-src" (or "p-sink", respectively). The p-src and p-sink nodes for a particular signaling instance might be end systems that are the ultimate sources and destinations of the data packets that establish the path, or they might be intermediate nodes such as border routers or aggregation points or tunnel endpoints.

Note that "src" (source) and "sink" terms are relative to the data flow, not to the flow to signaling messages. Similarly, in each node along the signaled path the directions "upstream" and "downstream" are defined relative to the user data flow that defines the path.

- ISPS Neighbors

  We define two CSTP-capable nodes as (ISPS) "neighbors" if they are connected by at least one path that includes no other CSTP-capable nodes. Neighbors that are directly connected, i.e., that have no nodes intervening, are "direct neighbors". A CSTP-capable node may have at most one neighbor through each point-to-point interface, but it may have multiple neighbors through a broadcast or NBMA interface.

  Signaling messages are generally (but not necessarily) sent hop-by-hop. Each hop is between neighbors, from an "h-src" (hop source) node to a neighbor node called "h-sink" (hop sink).

- SAPU

  A "Signaling Application Protocol Unit" (SAPU) is the basic transmission unit for signaling. A SAPU is derived from the signaled state in the h-src node and it is used to set, modify, or delete state in the h-sink node.

- Trigger, Refresh Messages

  A "trigger message" installs, modifies, or deletes signaled state, while a "refresh message" only refreshes existing state, i.e., prevents it from timing out.

2. The CSTP Service Model

Under the two-level architecture, corresponding ULSP modules in neighbor nodes are peers that communicate using the CSTP layer. Roughly, ULSP and CSTP correspond respectively to application-layer and transport layer protocols in the Internet stack. However, this memo uses the term "level" rather than "layer" for the ULSP/CSTP split, because they are more intertwined than strict protocol layering allows. This is reflected in the API to be described in Section 2.3.

Each ISPS message includes a ULSP identifier that selects a particular ULSP. We assume that there will be a simple registration space for ULSP identifiers. A major problem in developing particular ULSPs will be to choose an appropriate functional modularity. There might be a few very general and flexible ULSPs; at the other extreme, there might be a great many ULSPs that differ only in particular details. This choice is an engineering tradeoff whose criteria are not yet clear.

The partition of functionality between CSTP and ULSP is a tradeoff between generality and unity. A "thicker" CSTP level, i.e., one that has more function, would provide greater unity among signaling tasks. On the other hand, a "thicker" CSTP would also be less general and more likely to constrain the range of signaling protocols that can be achieved by any ULSP. This memo suggests a fairly "thin" CSTP, which includes a set of functions that are closely interlinked and that are generally useful for a broad range of signaling applications. For example, this CSTP will support signaling tasks that require simplex or full-duplex signaling, and it will support receiver- or sender-initiated signaling.

DISCUSSION

Suppose that the the current Version 1 RSVP functionality were to be mapped into a (CSTP, ULSP) pair (see Appendix A.) Neither RSVP's receiver-oriented operation nor its reservation styles [RFC2205] should appear in CSTP; these features would be implemented only in the RSVP-specific ULSP module.

CSTP has only hop-by-hop semantics; it handles the (reliable and secure) transmission of signaling state between neighbors and (optionally) managing this as soft state. End-to-end signaling semantics must be realized by the actions of the ULSP, which is responsible for maintaining consistent signaled state along the

path.
   Upon receiving a new or modified SAPU, a ULSP module may send
   appropriate SAPUs to other neighbors, to keep the state consistent
   end-to-end (on the other hand, it may not, depending upon the

function to be performed.)

CSTP must not constrain the granularity of the data flow that defines a signaling path (although an ULSP might.) The flow granularity might range from micro-flows that are created by particular user applications to highly-aggregated flows. On the other hand, each ULSP is likely to be optimized for a particular flow granularity or range of granularities.

It should be possible for signaling protocols supported by CSTP to operate correctly through CSTP-incapable nodes. This requirement, together with support for path-coupled signaling, can be met by sending signaling messages downstream using the destination address of the data. Such messages will automatically be forwarded correctly through CSTP-incapable nodes. This mechanism in turn requires that each CSTP hop intercept signaling messages from the data stream [Waypoint00], process and perhaps modify them, and then forward them.

2.1 CSTP Functions

The CSTP level performs the following functions. These functions are in general tightly coupled with each other, so they represent a logical set for CSTP to implement.

- Reliable Delivery of Signaling Messages

  Signaling operation must not be threatened by packet loss or reordering. Therefore, CSTP provides reliable delivery of trigger messages so that state can be reliably and promptly added, changed, and explicitly removed.

  DISCUSSION

  The early design of RSVP Version 1 made the optimistic assumption that signaling traffic could be protected by QoS and that reordering would be rare. Experience later showed that these assumptions could be violated unacceptably often, so a reliable delivery mechanism [Refresh00] was pasted onto RSVP Version 1. Reliable delivery of trigger messages is a fundamental objective for CSTP, although a particular ULSP may choose to not use it.

- Ordered Delivery of SAPUs

  The original RSVP v1 protocol spec [RFC2205] allowed network reordering of signaling packets to create significant (e.g.,

30 second) periods of erroneous reservation. The addition of reliable delivery prevents this particular failure mode, but it introduces the problem of delayed delivery of old duplicate packets. Therefore, CSTP includes a mechanism to ignore out-of-order trigger messages.

- o Soft State Support

  When signaling explicitly installs state in a node, there is cause for concern about the robustness with which this state will be removed. Besides system crashes, there is always the possibility of programming errors that "leak" state. In the somewhat chaotic multi-vendor environment of the Internet, it is unwise to assume error-free interoperation of many different implementations. CSTP therefore includes soft state -- removing state that is not periodically refreshed or explicitly torn down -- as a fundamental robustness mechanism, although a particular ULSP may choose to not use it.

- o Fragmentation, Reassembly, and Bundling of SAPUs

  CSTP must be able to fragment and reassemble SAPUs that exceed one MTU.

  DISCUSSION

  We expect that elementary ISPS messages will be only a little bit larger than the corresponding RSVP Version 1 messages; the majority of SAPUs should be under 200 bytes.

  The addition of security credentials may lead to some SAPUs O(1000) bytes, but SAPUs significantly larger than this are expected to be rare.

  Bundling -- carrying multiple small SAPUs in a single IP datagram -- may be desirable for performance within CSTP. It may be useful when cryptographic integrity checking is in use, as it allows a single cryptographic checksum to be used across all bundled messages. This is discussed further in subsections 3.2 and 3.3.

- o Congestion Control

  It would seem that the signaling protocol and the network configuration could ensure that signaling traffic will almost

always be small relative to the data flow. However, in general all Internet traffic must be able to slow down in response to congestion (in the absence of static or dynamic

partitioning of network bandwidth, e.g., by QoS.)
DISCUSSION

The flow of SAPUs normally has the general characteristics of media streams: long-lived (in fact, never-ending), somewhat bursty, streams of bytes. It should be possible to throttle back signaling bandwidth between a pair of nodes by slowing soft-state refreshes and by capping the rate of change of existing state, for example. In this regime, the techniques of TCP-friendly congestion control may be applicable to CSTP. However, bursts of trigger messages and retransmissions can also occur, so CSTP can also have TCP-like characteristics. Thus, reliable delivery introduces the need to dynamically compute the appropriate value for retransmission timers, and this computation must consider the round trip time (RTT) and network congestion.

The two-level ISPS framework centralizes issues relating to the volume and timing of network signaling traffic within the common CSTP protocol. The CSTP module is in a position to perform complex scheduling of signaling message transmissions, taking into account the congestion at each target node and the signaling load. For example, CSTP might limit the rate of signaling traffic but still allow a burst of signaling traffic when a route changes.

- Hop-by-Hop Security

Since the CSTP operates strictly hop/hop, CSTP is a natural place to implement (optional) hop-by-hop integrity. We suggest that the RSVP hop-by-hop integrity algorithms [Integrity00] be used in CSTP.

- Neighbor List

A CSTP module maintains state that lists the node's neighbors. This state may include the IP address of the neighbor, the local interface used to reach it, and Boolean flags giving important properties of the neighbor: ISPS-capable and Direct-Neighbor. A node builds the neighbor list as a result of receiving CSTP messages. The neighbor list should be implemented as soft state that is deleted if it is not refreshed.

An open issue is whether CSTP needs to provide an explicit neighbor-discovery mechanism or even an up/down protocol distinct from that provided by IP routing.

- Interface to Routing

  In order to perform path-related signaling, it is necessary that the signaling protocol be able to discover the route taken by the corresponding data flow. This should be true regardless of whether the signaling is path-coupled or path-decoupled. It would clearly be an architectural mistake for the signaling protocol to perform its own independent routing calculation, so signaling must be able to query (and perhaps influence, as in route pinning) IP routing. It makes sense to centralize this interface to routing in the CSTP module, to avoid replicating it in each ULSP.

  Note that it would be useful to be able to hide the complexities of multicast routing [Sections 3.3 and 3.9 of RFC2205] within the CSTP level, to simplify ULSPs that need to support multicast. However, the functionality does not seem to divide cleanly across the CSTP/ULSP boundary, so that a ULSP that supports multicast may have to cope with some of the messy details of multicast routing.

2.2 General ISPS Operation

The ISPS framework operates in the following general manner.

- Suppose that an ULSP in the h-src node S needs to send an SAPU containing signaled state to a peer ULSP on a neighbor h-sink node T. The h-src ULSP issues a downcall to its local CSTP module, passing the SAPU and a target IP address.

  This target address may explicitly name node T, or T may be determined implicitly because it intercepts the message that was addressed to some downstream node, e.g., to p-dest or to the ultimate destination address if different from p-dest.

- The CSTP level reliably delivers the SAPU to the corresponding CSTP level in T, which then upcalls to the h-dest ULSP to deliver the SAPU.

- At the request of the h-src ULSP, the SAPU contents can be treated as soft state. In this case, the CSTP level in S sends periodic refresh messages for the SAPU (unless the message was deleting state). The CSTP level in T will automatically time out the state and notify its local ULSP via an upcall if the state is not refreshed in time.

- On the other hand, the SAPU contents may be "information"

that needs to be reliably communicated to a peer ULSP but not retained as independent (soft) state in the h-sink CSTP. For example, information state might be a QoS request that is used for an admission control decision in a core node, which does not retain the individual requests but only the cumulative reservation (in the ULSP).

DISCUSSION

In this example of "stateless" admission control in the core, the ULSP would need to keep track of the individual requests somewhere at the edge of the network, in order to reverse a reservation when a flow ceases.

Also note that a ULSP could use this information (non-soft-state) option to transmit SAPUs to the peer ULSP and then implement its own soft state mechanism at the ULSP level. Bypassing the mechanism built into the CSTP in this manner is generally undersirable, but it does provide an escape for some unforeseen signaling requirement.

- The information included in an SAPU is logically a (<key>, <value>) pair. The <key> part distinguishes the state specified by the <value> part from other state sent between the same pair of neighbors. However, the distinction between <key> and <value> within the SAPU is known only to the ULSP module; CSTP treats the SAPU as opaque.

  DISCUSSION: EXAMPLE FROM RSVP V1

  For the equivalent of an RSVP Resv message, the <key> part of the SAPU would consist of the SESSION and NHOP objects and perhaps (depending upon the STYLE) the FILTER_SPEC objects. Other fields -- e.g., STYLE and FLOWSPEC -- would be in the <value> part. These complex rules on RSVP V1 <key>s would not be known by CSTP.

- The format of an SAPU is specific to the particular ULSP that sends and receives it. However, many ULSPs will benefit from using the typed "object" syntax and the object encoding rules of RSVP Version 1, encoding an SAPU as a sequence of elementary (type, length, value) triplets.

2.3 CSTP/ULSP API

This section defines a generic interface between CSTP and ULSP, i.e., the generic ULSP API.

For simplicity we assume that the implementations of the two levels are distinct, sharing no data structures. This means that data structures must be passed across this interface by value and that the CSTP must keep a shadow copy of the SAPU state to be retransmitted. An actual implementation is likely to share data structures between the two levels to avoid this inefficiency. (An analogous relationship occurs between IP and TCP in most protocol implementations).

Note that the CSTP level in designed to handle all of the event timing, so the ULSP can be event-driven by upcalls from the CSTP.

2.3.1 Downcalls from the ULSP

An ULSP may issue the following downcalls to the CSTP.

- SendNewSAPU(SAPU, IP-target [, OIf], burst_flag)
  -> SAPUid

  This downcall causes the specified SAPU to be transmitted reliably to the h-sink node specified or implied by address IP-target; it also allocates and returns a unique identifier SAPUid to the ULSP. If reliable delivery fails, the CSTP level issues an asynchronous SendFailed() upcall to the ULSP. If the SAPU is delivered and acknowledged, the CSTP level sends periodic soft-state refresh messages for it, until the ULSP makes a SendModSAPU() or sendTearSAPU() downcall for the same SAPUid.

  In the downstream direction, IP-target may be the signaling destination's IP address; the neighbor node on the path to IP-target will intercept and process the message. Otherwise, IP-target it must be the IP address of a neighbor (h-sink). For a multicast IP-target address, the caller may specify the outgoing interface OIf to be used.

  In order to retransmit for reliable delivery, the CSTP may cache a copy of the SAPU. If an SAPU to be retransmitted is not in the cache, the CSTP can issue a RegenSAPU() upcall (see below) to ask the ULSP to regenerate the SAPU.

If a route change later causes loss of state in a neighbor, CSTP will make a RegenSAPU() upcall to ask the ULSP to reconstruct the original SAPU, and then send this CSTP in a NEW trigger message containing a new SAPUid. The upcall will also transmit a revised SAPUid to the ULSP.

The burst_flag parameter is a boolean flag that can be used by the CSTP level as a "hint" about when it can efficiently bundle a set of successive calls (see Sections 3.2.3 and 3.3). When CSTP issues a burst of successive calls to SendNewSAPU(), all except the last should have this flag set to True. CSTP will make the decision about when to bundle. This allows the CSTP to avoid the introduction of substantial bundling delays.

- SendModSAPU(mod-SAPU, old-SAPUid, burst_flag) -> mod-SAPUid

  Modify an existing SAPU that had identifier old-SAPUid to be mod-SAPU with identifier mod-SAPUid.

  Mod-SAPU will be reliably delivered and refreshed at the neighbor specified or implied by IP-target, or else CSTP will issue a SendFailed(mod-SAPUid, reason) upcall to the ULSP.

- SendTearSAPU(SAPUid)

  Tear down (remove) the SAPU state that corresponds to SAPUid.

- SendInfoSAPU(SAPU, IP-target [, OIf], burst_flag)

  This call is used to send state to the specified target, without treating it as soft state. This call is identical to SendNewSAPU(), except the h-src CSTP does not retain state after the transmission is acknowledged and does not refresh the state, and the h-sink CSTP does not timeout the state.

- SendEventSAPU(SAPU, IP-target [, OIf], burst_flag)

  This call sends an SAPU with neither reliable delivery nor refreshing, i.e., it is sent as a datagram. This is called an "event" message.

2.3.2 Upcalls to the ULSP

The CSTP level may issue the following upcalls to the ULSP.

- SendFailed(SAPUid, reason)

  This upcall reports that the SendNewSAPU() or SendModSAPU() operation failed for the specified SAPUid.

- RecvNewSAPU(SAPU, SAPUid, h-src)

  A new SAPU has been received from the node whose IP address is h-src. SAPU is passed up for subsequent use in a RecvTearSAPU upcall.

- RecvModSAPU(SAPU, SAPUid, h-src)

  An existing SAPU has been modified.

  Note that the new/mod distinction here may not be needed; the ULSP will discover the status when it looks up the <key>. However, the mod upcall is included in the interface as a consistency check.

- RecvTearSAPU(SAPUid, h-src)

  This upcall may result from receiving a TEAR message for the specified state or from a local soft-state timeout. In either case, this call is a signal to the ULSP that the specified SAPUid is henceforth invalidated.

- RecvInfo(SAPU, SAPUid, h-src)

  This upcall delivers an SAPU that has been reliably transmitted but is not retained in the CSTP level as soft state. No refresh messages will be received for it, but a subsequent TEAR message may result in a RecvTearSAPU upcall for the same SAPUid.

- RecvEvent(SAPU, h-src)

  This upcall delivers an Event SAPU, i.e., without reliable delivery and without soft state refresh.

- RegenSAPU(SAPUid [, new-SAPUid]) -> SAPU

  This upcall requests that the ULSP regenerate and return the SAPU corresponding to SAPUid. If present, the

optional new-SAPUid parameter is used to replace SAPUid as
the internal handle for this atom of signaled state.

Note: this list is incomplete. For example, API calls are required for the routing interface (the RSRR interface of RSVP V1 may be a useful guide here) and for the neighbor list.

3. The CSTP Protocol

There are two basic design choices for transporting ISPS messages: use TCP connections, or explicitly program the required semantics within CSTP. We refer to these alternatives as CSTP/TCP and CSTP/IP, respectively; they are described in subsections 3.2 and 3.3. In either case, a common message format, described in subsection 3.1, is used.

3.1 Common Message Format

The basic CSTP message consists of a CSTP header, or "M-header", and a payload that may include an SAPU. The M-header contains a specification of the message type that determines the contents and format of the payload.

CSTP transports SAPUs in DnSig (down-stream signaling) messages and UpSig (upstream signaling) messages. We use the term "xSig" to denote an elementary CSTP signaling message without specifying the direction.

Each trigger message includes a unique identifier, the SAPUid. The SAPUid is used as a handle on the SAPU that is known to the CSTP (as opposed to the <key>, buried within the SAPU, that the CSTP cannot see). A SAPUid is used for for efficiently refreshing the corresponding state and as a handle for state

The M-header includes:

- The length of the message, including the M-header and the payload.
- A ULSP identifier
- The CSTP message type for this message (see below).
- The IP address h-src of the node that sent this message.
- A list of zero or more SAPUids

The first two bytes of the SAPU must be its length in bytes; otherwise, the SAPU format is entirely opaque to CSTP.

The nine currently-defined CSTP message types are as follows. They are shown schematically in functional notation with the type as the first parameter. In practice most the parameters listed here are carried explicitly in the M-header.

xSig(NEW, h-src, SAPUid, SAPU, R)

xSig(MOD, h-src, SAPUid, SAPU, old-SAPUid, R)

xSig(TEAR, h-src, SAPUid)

xSig(REFRESH, h-src, SAPUid, R)

xSig(ACK, h-dest, SAPUid-list)

xSig(NACK, h-src, SAPUid)

xSig(INFO, h-src, SAPUid, SAPU)

xSig(EVENT, h-src, SAPUid, SAPU)

xSig(CHALLENGE, h-src, challenge-object)

xSig(RESPONSE, h-src, challenge-object)

xSig(ERROR, h-dest, SAPUid)

Here:

- Every message contains the IP address of its originator, h-src. In most but not all cases this address is the same as the source IP address of the ISPS packet. For simplicity we specify that h-src will always appear explicitly in a CSTP header. It is used to build neighbor state.
- R specifies the refresh time for the SAPU (see [RFC2205]).
- For the MOD message, the sending ULSP must ensure that the new SAPU with identifier SAPUid and the old SAPU with identifier old-SAPUid share the same <key> parts.
- The NEW and MOD messages send soft state, and REFRESH messages refresh that state. The INFO message sends an SAPU reliably but does not retain or it as soft state. The EVENT message sends an SAPU on-time and unreliably.

- The CHALLENGE and RESPONSE messages are used to initialize the keyed hash integrity check [Integrity00]. The <challenge-object> is carried as a CSTP-level SAPU, which is a special case; all other SAPUs are opaque to CSTP and carried on behalf of an ULSP. <challenge-object> is defined in [Integrity00].

  Figures 1a and 1b show a state diagram for operation of CSTP at an h-src node, and Figure 2 summarizes the corresponding states at the receiver node h-sink. Here SendNewSAPU(), SendModSAPU(), and SendTearSAPU() represent down calls from the ULSP to the CSTP to install a new SAPU, modify an existing SAPU, or delete an SAPU, respectively. xSig(type) represents a CSTP message of a specific type. TO-R and TO-T refer to refresh and state timeouts, respectively.

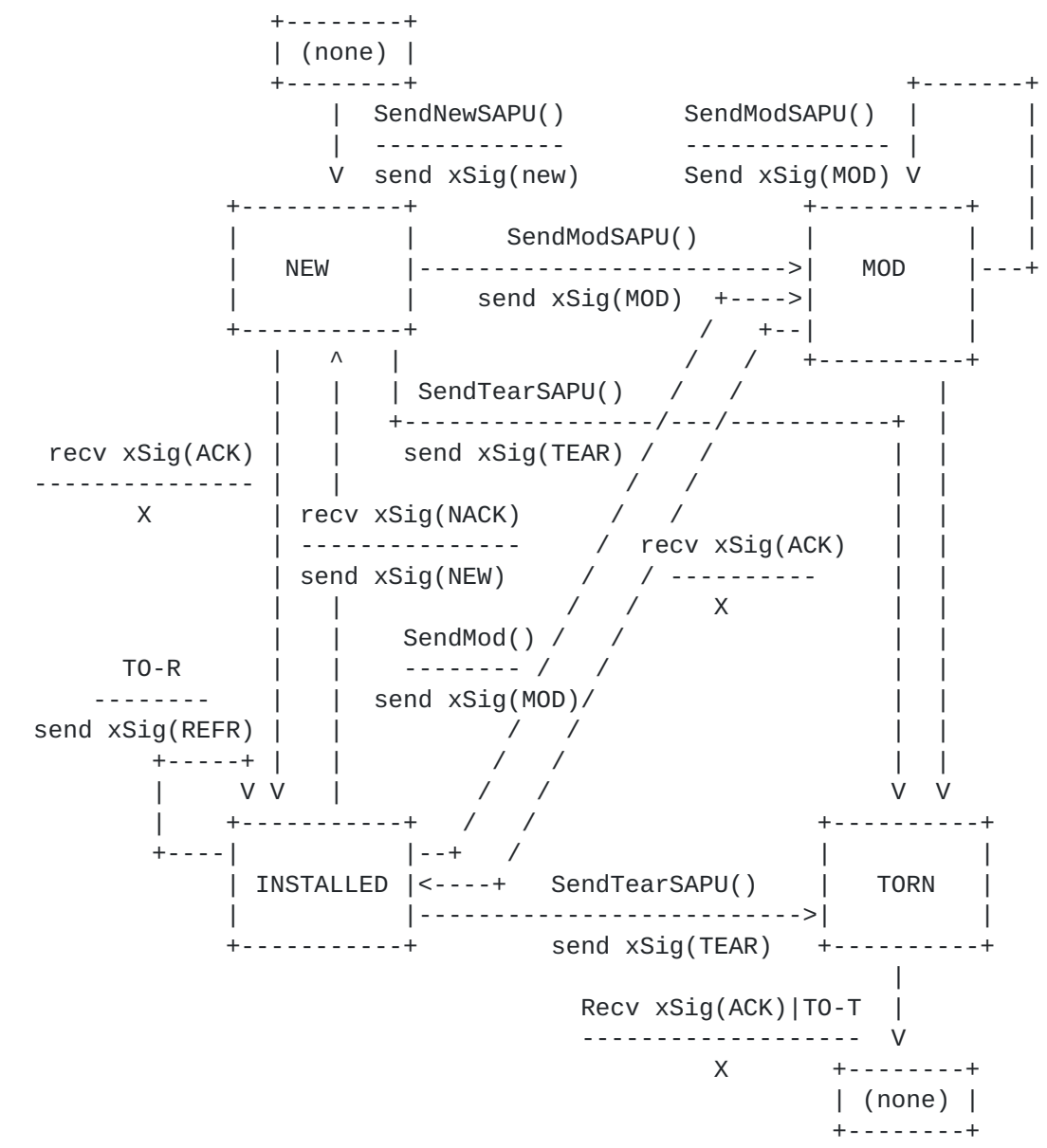

Figure 1a: H-Src CSTP State Diagram (Soft State)

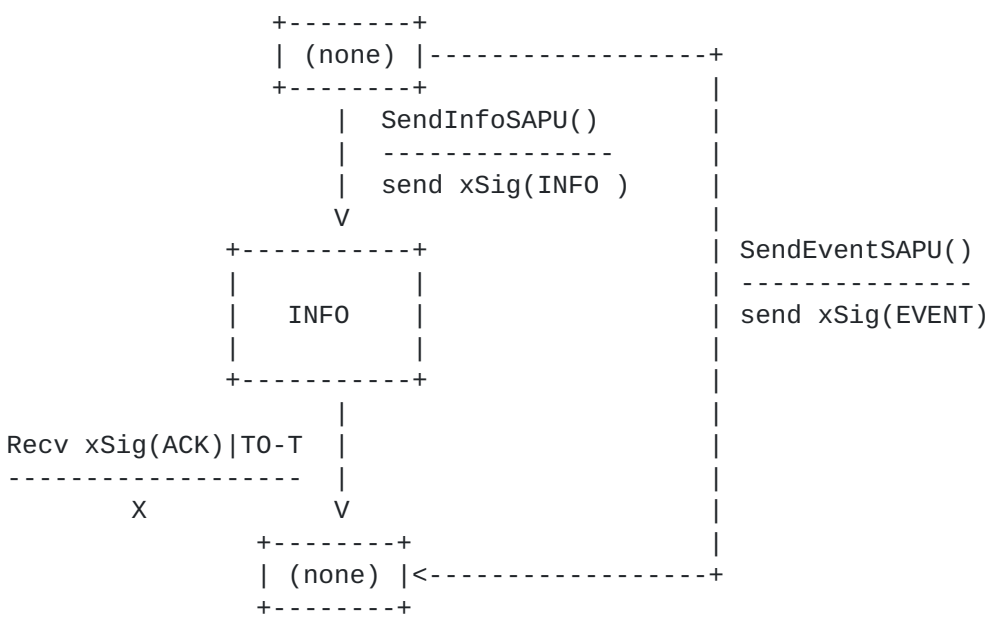

Figure 1b: H-Src CSTP State Diagram (Hard State and Datagrams)

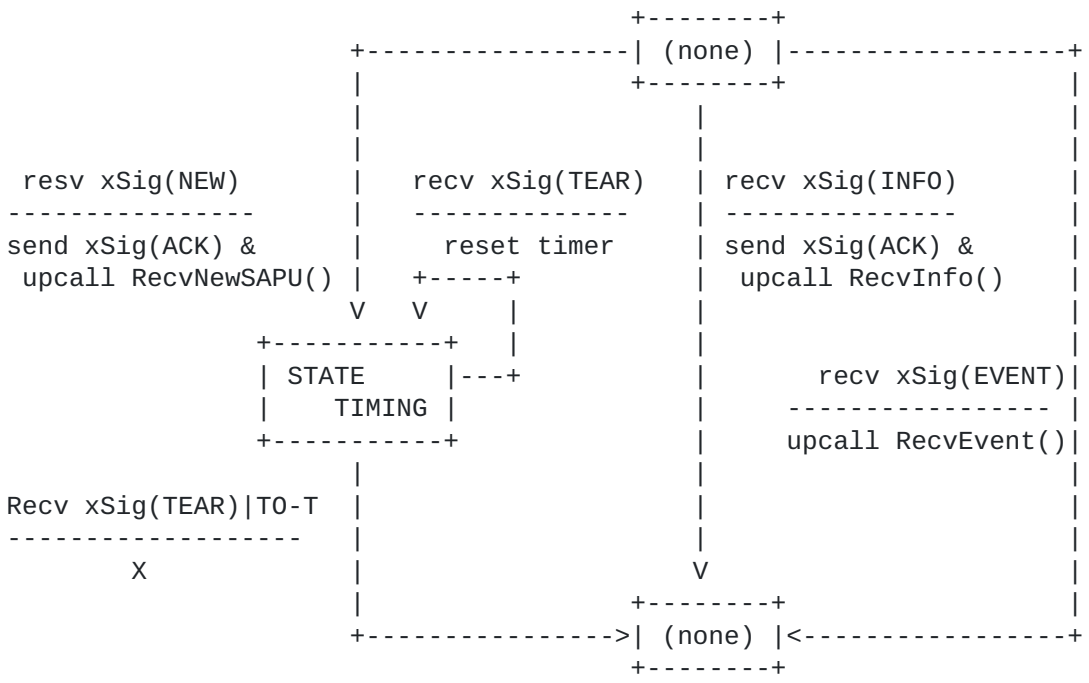

Figure 2: H-Sink CSTP State Diagram)

3.2 CSTP/IP

CSTP/IP uses the RSVP V1 signaling message paradigm. It includes a version of the RSVP "refresh reduction" extensions [Refresh00] to provide reliable delivery of trigger messages, rejection of old duplicates, and refreshing of state.

These mechanisms use the SAPUid as handle on the state. Note that we are overloading this unique identifier by using it both for (1) transmitting and refreshing SAPUs and for (2) local handles in the API interfaces of h-src and h-sink nodes. In an actual implementation distinct SAPUids could be used in the API, if that were more efficient.

3.2.1 Example: Sending New State

Sending new signaled state involves the following sequence of steps. Some secondary parameters are omitted here for simplicity.

1. The local ULSP issues the following downcall to its CSTP, passing the new SAPU:

   SendNewSAPU(SAPU, IP-target, [OIf]) -> SAPUid

   For downstream transmission, the target IP address P-target will be either the target signaling destination address p-dest or the address h-sink of a neighbor. For upstream transmission, it must be a neighbor address h-sink. The optional Outgoing InterFace (OIf) parameter is needed when IP-target is a multicast address.

   The CSTP:

   - generates an SAPUid,
   - creates a local send state block,
   - builds and sends the trigger message:

     xSig(NEW, h-src, SAPUid, SAPU)

     to the IP-target address,
   - sets a retransmit timer,
   - and returns the SAPUid to the ULSP, which records this handle.

2. If the retransmit timer goes off before the NEW message is acknowledged, the local CSTP retransmits the trigger message. This is repeated until either an ACK is received or a limit is reached. In the latter case, the CSTP issues the upcall:

SendFailed(SAPUid, SAPU)

and deletes the send state block.

3. Otherwise, when the CSTP receives a xSig(ACK, SAPUid) message, it stops retransmitting and starts sending periodic refresh messages to IP-target:

xSig(REFRESH, h-src, SAPUid)

4. If the CSTP receives a xSig(NACK, SAPUid) message, it returns to step 2 to (re-)transmit the trigger message.

5. When the NEW message is received at the h-sink node that was implied or specified by IP-target, the remote CSTP:

- Creates a local receive state block,
- passes the SAPU to the remote ULSP via an upcall:

RecvNewSAPU(SAPU, h-src)

- and returns an ACK message.

3.2.2 Ordered Delivery in CSTP/IP

Under soft-state signaling, old trigger messages should always be ignored. This can be accomplished by introducing a monotone-increasing sequence number in trigger messages. Following the example of the Refresh Reduction extensions to RSVP V1 [Refresh00], we can overload the SAPUid to serve as a sequence number as well as a handle on reservation state. An h-src node generates monotone increasing values for new SAPUids to be sent to a given h-sink. The h-sink node then:

(1) remembers the largest SAPUid seen so far from h-src;

(2) processes as a trigger message a SAPU received with a larger SAPUid;

(3) treats the message as a refresh if the received SAPUid

matches that of existing state from h-src; and otherwise,

(4) ignores the message and sends a NACK.

When a node crashes and restarts, losing its state, some mechanism is required to reliably instruct its neighbors to reset their latest sequence numbers. When a route changes and a REFRESH message is answered with a NACK, h-src must send the new trigger message with a new SAPUid; h-src must also upcall to inform its ULSP that the SAPUid has changed for the existing state.

An alternative approach to ordered delivery would be to use the sequence number that is already present in the hop-by-hop cryptographic integrity check mechanism [Integrity00]. The integrity mechanism also includes a Challenge/Response mechanism to robustly (and securely) reset the sequence number in neighbors at startup.

If a route change later causes loss of state in a neighbor, CSTP will make a RegenSAPU() upcall to ask the ULSP to reconstruct the original SAPU, and then send this CSTP in a NEW trigger message containing a new SAPUid. The upcall will also transmit the revised SAPUid to the ULSP.

3.2.3 Fragmentation and Bundling

In order to handle both fragmentation and bundling, an additional CSTP/IP header is prepended to each bundled message or fragment of a large message. This outer header is called the FB-header (fragment/bundle). Then a bundle of small messages has the form:

<FB-header> <CSTP message> <CSTP message>*

(where star denotes none or more), and a fragment of a large message has the form:

<FB-header> <CSTP message>

The BF-header contains:

- The total length of the datagram in bytes
- A fragment offset and MF ("More Fragments") bit
- A checksum or keyed hash integrity object

3.3 CSTP/TCP

An alternative to building a reliable, ordered delivery mechanism into CSTP, as in RSVP v1, would be to use TCP for delivery of CSTP messages. Using this CSTP/TCP, each CSTP module would open a TCP connection to each of its neighbors and use it for all signaling traffic. This traffic would be a series of CSTP messages as <M-header>, <payload> pairs, defined in subsection 3.1.

TCP would provide reliable and ordered delivery, fragmentation and reassembly, and congestion control. This should considerably simplify the CSTP level of the ISPS framework compared to CSTP/IP. On the other hand, using TCP may give the CSTP less control over exactly how it reacts to congestion or to a burst of traffic.

We believe that the API described in subsection 3.1 can be made to work equally well for CSTP/TCP and CSTP/IP, allowing the same ULSP to operate over either lower-level protocol. It is unclear whether only one or both of these CSTP protocols should be standardized. It may be that different situations will favor one or the other approach. If both are defined, then there must be some interoperability mechanism to allow a particular neighbor pair to agree on which is to be used.

It might seem that bundling would add no functionality to CSTP/TCP. However, performance may be significantly improved by including in each TCP segment all the small CSTP messages that will fit. If cryptographic integrity is in use, it will be important to compute a single cryptographic hash across each segment, and a new per-segment header must be introduced to carry this hash. This is analogous to the FB header introduced in Section 3.2.3, except that under CSTP/TCP it will not have a fragmentation function, only a bundling function.

4. Open Issues

A number of issues are left unresolved in this memo. In the following list of these issues, the first three are fundamental issues of the NSIS working group agenda. The rest are more specific technical issues.

1. A broad design question is how to partition the space of signaling applications into ULSPs (Section 2.)

2. This memo describes two alternative approaches to CSTP, CSTP/IP and CSTP/TCP (Section 3). Should one, or both, be

standardized?

3. Section 3.1 describes a generic API, which would be mapped into various implementation-specific interfaces. However, if it is desirable to create a market in third-party ULSP software, it will be necessary to standardize on a real API. Should we define a real API now?

4. The ULSP API defined in Section 3.1 is incomplete. It omits a way to communicate neighbor information to a ULSP, and it also omits the common interface to routing.

o Is an explicit neighbor discovery mechanism necessary or desirable (Section 2.1), or can CSTP simply learn of neighbors from signaling traffic and verify their status from routing?

5. Should CSTP support another delivery mode for NEW and MOD: unreliable delivery but with refresh? (Note that this would correspond to the service provided by the version of RSVP defined in [RFC2205], before the Refresh Reduction Extensions were defined.) Similarly, should CSTP support the option of unreliable delivery for TEAR?

6. Is MOD logically necessary, and is it useful?

7. The spec is currently missing a preemption mechanism, which can do a reverse teardown. That is, it should be possible to initiate a teardown in the direction counter to the setup direction.

8. Possible support for bidirectional reservations needs further thought.

5. Security Considerations

The CSTP protocol may support hop-by-hop integrity using the algorithms of RSVP version 1 [Integrity00]. Policy issues -- e.g., user authentication and access control as well as accounting -- are the province of each ULSP. Some ULSPs will wish to incorporate the COPS mechanisms for secure end-to-end authentication and access control [COPS00].

6. Acknowledgments

The conception behind this memo is not original. One of the advances in STream protocol II (ST-II) [RFC1191] over its predecessor
ST was the explicit definition of a reliable hop-by-hop control sub-protocol called ST Control Message Protocol (SCMP). We believe that CSTP reflects some important advances over SCMP, for example soft

state management.

We are grateful for several Xingguo Song of Concordia University for pointing out several errors and omissions in the previous version of this memo. He discovered these problems in the course of validating CSTP using the formal specification language SDL.

APPENDIX A. RSVP Version 1 as an ULSP

To write an ULSP specification for the base Version 1 RSVP protocol of RFC 2205, we can adopt nearly all of RFC2205. This is largely because many of the issues handled by CSTP are dealt with in the Refresh Reduction extension document [Refresh00], not in RFC 2205. The Refresh Reduction document [Refresh00] would be entirely obsoleted by our ISPS proposal, although we have suggested adopting its basic concepts.

Looking at RFC 2205 in detail, we find the following.

- Section 1 of RFC 2205 would be little changed. This section discusses the objectives of RSVP and defines a session, a flowspec, a filterspec, receiver-initiated reservations, scope, reservation merging, and styles.

- Section 2 of RFC 2205 which describe the RSVP protocol mechanisms in general terms, would be changed only where it describes soft state and specific RSVP Version 1 message types. RSVP Version 1 message types would become a combination of SAPU type and CSTP message types, as shown in the table below. Note that a few of the RSVP Version 1 message types, e.g., Bundle, simply disappear into mechanisms included in CSTP.

- Section 3 of RFC 2205 contains the functional specification of RSVP Version 1, and section 3.1 defines RSVP Version 1 message syntax and semantics. Each <xxx Message> definition that maps into ISPS becomes a <yyy SAPU> definition. The Common Header is replaced by an SAPU header that contains only a length and an SAPU type. The INTEGRITY object is omitted since it will now appear in the CSTP header. Otherwise, Section 3.1 would be unchanged.

- Some discussion would be required of exactly how the RSVP ULSP should invoke the downcalls to CSTP and the upcalls from CSTP.

The message types of RSVP Version 1 will be mapped as follows, using the ISPS design of this memo.

RSVP Version 1 Message Type	SAPU Type	CSTP Message Type
Path	Path	NEW or MOD
Resv	Resv	NEW or MOD
Srefresh	Path or Resv	REFRESH
ACK	Path or Resv	ACK or NACK
PathTear	Path	TEAR
ResvTear	Resv	TEAR
PathErr	PathErr	EVENT
ResvErr, ResvConf	ResvErr	EVENT
DREQ	DiagReq	EVENT
DREP	DiagRep	EVENT
Integrity Challenge	(none)	CHALLENGE
Integrity Response	(none)	RESPONSE
Bundle	(none)	(CSTP header)
ResvTearConf ??		

References

[aggr01] Baker, F. et. al., "Aggregation of RSVP for IPv4 and IPv6 Reservations", RFC 3175, September 2001.

[AIF01] Keaton, M., Lindell, R., Braden, R., and S. Zabele, "Active Multicast Information Dissemination", submitted to conference, April 2001.

[CM01] Balakrishnan, H. and S. Seshan, "The Congestion Manager", RFC 3124, June 2001.

[COPS00] Durham, D., Ed., Boyle, J., Cohen, R., Herzog, S., Rajan, R., and A. Sastry, "The COPS (Common Open Policy Service) Protocol", RFC 2748, January 2000.

[intdiff00] Bernet, Y. et al, "A Framework for Integrated Services Operation over Diffserv Networks", RFC 2998, November 2000.

[Integrity00] Baker, F., Lindell, R., and M. Talwar, "RSVP Cryptographic Authentication", RSVP 2747, January 2000. 1996.

[ISInt93] Braden, R., Clark, D., and S. Shenker, "Integrated Services in the Internet Architecture: an Overview", RFC 1633, June 1994.

[ISrsvp96] Wroclawski, J., "The Use of RSVP with Integrated Services", RFC 2210, September 1997.

[mpls00] Swallow, G., et al, "RSVP-TE: Extensions to RSVP for LSP Tunnels", <draft-ietf-mpls-rsvp-lsp-tunnel-09.txt>, IETF, Sept 2001.

[optical00] Rajagopalan, B., "LMP, LDP and RSVP Extensions for Optical UNI Signaling", <draft-bala-uni-signaling-extensions-00.txt>, IETF, October 2001.

[PCQoS99] "PacketCable(tm) Dynamic Quality-of-Service Specification", PKT-SP-DQOS-I01-991201, Cable Television Laboratories, Inc., 1999.

[Refresh00] Berger, L., et. al., "RSVP Refresh Overhead Reduction Extensions", <draft-ietf-rsvp-refresh-reduct-05.txt>, IETF, June 2000.

[RFC2205] Braden., R. Ed., et. al., "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205, September 1997.

[TIST02] Shore, M., "The TIST (Topology-Insensitive Service Traversal) Protocol", <draft-shore-tist-prot-00.txt>, IETF, May 2002.

[Waypoint00] The path-oriented concept was explored in an expired Internet Draft: Lindell, B., "Waypoint -- A Path Oriented Delivery Mechanism for IP based Control, Measurement, and Signaling Protocols"", <draft-lindell-waypoint-00.txt>, IETF, November 2000.

[XSong02] Xingguo Song, "Specification and Validation of the Common Signaling Transport Protocol in SDL", Thesis, Concordia University, Montreal, Canada, September 2002.

Authors' Addresses

Bob Braden
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

Phone: (310) 448-9173
EMail: Braden@ISI.EDU

Bob Lindell
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

Phone: (310) 448 8727
EMail: Lindell@ISI.EDU