Network Working Group
Request for Comments: 2334
Category: Standards Track

J. Luciani
Bay Networks
G. Armitage
Bellcore
J. Halpern
Newbridge
N. Doraswamy
Bay Networks

April 1998

Server Cache Synchronization Protocol (SCSP)

Status of this Memo

Copyright Notice

Abstract

This document describes the Server Cache Synchronization Protocol (SCSP) and is written in terms of SCSP's use within Non Broadcast Multiple Access (NBMA) networks; although, a somewhat straight forward usage is applicable to BMA networks. SCSP attempts to solve the generalized cache synchronization/cache-replication problem for distributed protocol entities. However, in this document, SCSP is couched in terms of the client/server paradigm in which distributed server entities, which are bound to a Server Group (SG) through some means, wish to synchronize the contents (or a portion thereof) of their caches which contain information about the state of clients being served.

1. Introduction

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [10].

It is perhaps an obvious goal for any protocol to not limit itself to a single point of failure such as having a single server in a client/server paradigm. Even when there are redundant servers, there

still remains the problem of cache synchronization; i.e., when one server becomes aware of a change in state of cache information then that server must propagate the knowledge of the change in state to all servers which are actively mirroring that state information. Further, this must be done in a timely fashion without putting undue resource strains on the servers. Assuming that the state information kept in the server cache is the state of clients of the server, then in order to minimize the burden placed upon the client it is also highly desirable that clients need not have complete knowledge of all servers which they may use. However, any mechanism for synchronization should not preclude a client from having access to several (or all) servers. Of course, any solution must be reasonably scalable, capable of using some auto-configuration service, and lend itself to a wide range of authentication methodologies.

This document describes the Server Cache Synchronization Protocol (SCSP). SCSP solves the generalized server synchronization/cache-replication problem while addressing the issues described above. SCSP synchronizes caches (or a portion of the caches) of a set of server entities of a particular protocol which are bound to a Server Group (SG) through some means (e.g., all NHRP servers belonging to a Logical IP Subnet (LIS)[1]). The client/server protocol which a particular server uses is identified by a Protocol ID (PID). SGs are identified by an ID which, not surprisingly, is called a SGID. Note, therefore, that the combination PID/SGID identifies both the client/server protocol for which the servers of the SG are being synchronized as well as the instance of that protocol. This implies that multiple instances of the same protocol may be in operation at the same time and have their servers synchronized independently of each other. An example of types of information that must be synchronized can be seen in NHRP[2] using IP where the information includes the registered clients' IP to NBMA mappings in the SG LIS.

The simplest way to understand SCSP is to understand that the algorithm used here is quite similar to that used in OSPF[3]. In fact, if the reader wishes to understand more details of the tradeoffs and reliability aspects of SCSP, they should refer to the Hello, Database Synchronization, and Flooding Procedures in OSPF [3].

As described later, the protocol goes through three phases. The first, very brief phase is the hello phase where two devices determine that they can talk to each other. Following that is database synchronization. The operation of SCSP assumes that up to the point when new information is received, two entities have the same data available. The database synchronization phase ensures this.

In database synchronization, the two neighbors exchange summary information about each entry in their database. Summaries are used since the database itself is potentially quite large. Based on these summaries, the neighbors can determine if there is information that each needs from the other. If so, that is requested and provided. Therefore, at the end of this phase of operation, the two neighbors have the same data in their databases.

After that, the entities enter and remain in flooding state. In flooding state, any new information that is learned is sent to all neighbors, except the one (if any) that the information was learned from. This causes all new information in the system to propagate to all nodes, thus restoring the state that everyone knows the same thing. Flooding is done reliably on each link, so no pattern of low rate packet loss will cause a disruption. (Obviously, a sufficiently high rate of packet loss will cause the entire neighbor relationship to come down, but if the link does not work, then that is what one wants.)

Because the database synchronization procedure is run whenever a link comes up, the system robustly ensures that all participating nodes have all available information. It properly recovers from partitions, and copes with other failures.

The SCSP specification is not useful as a stand alone protocol. It must be coupled with the use of an SCSP Protocol Specific specification which defines how a given protocol would make use of the synchronization primitives supplied by SCSP. Such specification will be done in separate documents; e.g., [8] [9].

2. Overview

SCSP places no topological requirements upon the SG. Obviously, however, the resultant graph must span the set of servers to be synchronized. SCSP borrows its cache distribution mechanism from the link state protocols [3,4]. However, unlike those technologies, there is no mandatory Shortest Path First (SPF) calculation, and SCSP imposes no additional memory requirements above and beyond that which is required to save the cached information which would exist regardless of the synchronization technology.

In order to give a frame of reference for the following discussion, the terms Local Server (LS), Directly Connected Server (DCS), and Remote Server (RS) are introduced. The LS is the server under scrutiny; i.e., all statements are made from the perspective of the LS when discussing the SCSP protocol. The DCS is a server which is directly connected to the LS; e.g., there exists a VC between the LS and DCS. Thus, every server is a DCS from the point of view of every other server which connects to it directly, and every server is an LS which has zero or more DCSs directly connected to it. From the perspective of an LS, an RS is a server, separate from the LS, which is not directly connected to the LS (i.e., an RS is always two or more hops away from an LS whereas a DCS is always one hop away from an LS).

SCSP contains three sub protocols: the "Hello" protocol, the "Cache Alignment" protocol, and the "Cache State Update" protocol. The "Hello" protocol is used to ascertain whether a DCS is operational and whether the connection between the LS and DCS is bidirectional, unidirectional, or non-functional. The "Cache Alignment" (CA) protocol allows an LS to synchronize its entire cache with that of the cache of its DCSs. The "Cache State Update" (CSU) protocol is used to update the state of cache entries in servers for a given SG. Sections 2.1, 2.2, and 2.3 contain a more in-depth explanation of the Hello, CA, and CSU protocols and the messages they use.

SCSP based synchronization is performed on a per protocol instance basis. That is, a separate instance of SCSP is run for each instance of the given protocol running in a given box. The protocol is identified in SCSP via a Protocol ID and the instance of the protocol is identified by a Server Group ID (SGID). Thus the PID/SGID pair uniquely identify an instance of SCSP. In general, this is not an issue since it is seldom the case that many instances of a given protocol (which is distributed and needs cache synchronization) are running within the same physical box. However, when this is the case, there is a mechanism called the Family ID (described briefly in the Hello Protocol) which enables a substantial reduction in maintenance traffic at little real cost in terms of control. The use of the Family ID mechanism, when appropriate for a given protocol which is using SCSP, will be fully defined in the given SCSP protocol specific specification.

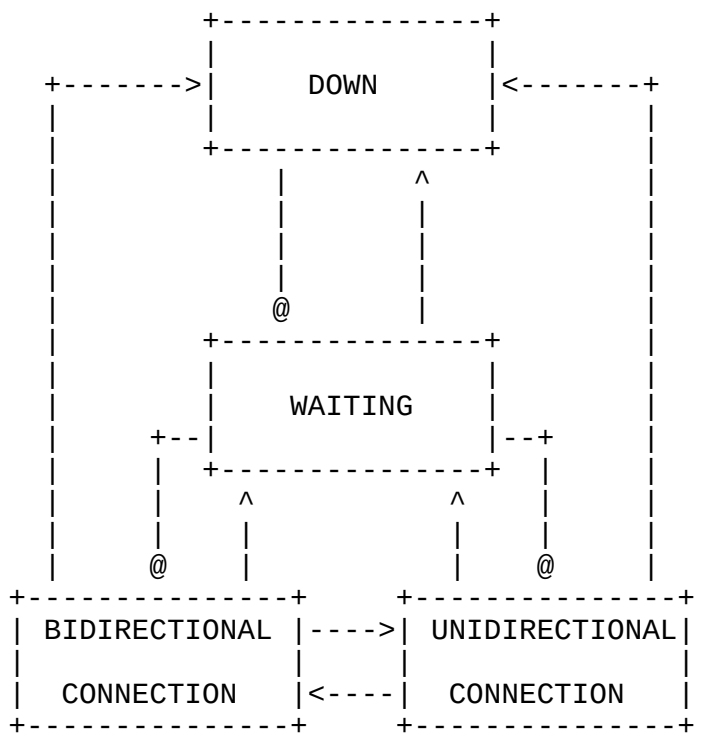


Figure 1: Hello Finite State Machine (HFSM)

2.1 Hello Protocol

"Hello" messages are used to ascertain whether a DCS is operational and whether the connections between the LS and DCS are bidirectional, unidirectional, or non-functional. In order to do this, every LS MUST periodically send a Hello message to its DCSs.

An LS must be configured with a list of NBMA addresses which represent the addresses of peer servers in a SG to which the LS wishes to have a direct connection for the purpose of running SCSP; that is, these addresses are the addresses of would-be DCSs. The mechanism for the configuration of an LS with these NBMA address is beyond the scope of this document; although one possible mechanism would be an autoconfiguration server.

An LS has a Hello Finite State Machine (HFSM) associated with each of its DCSs (see Figure 1) for a given SG, and the HFSM monitors the state of the connectivity between the servers.

The HFSM starts in the "Down" State and transitions to the "Waiting" State after NBMA level connectivity has been established. Once in the Waiting State, the LS starts sending Hello messages to the DCS. The Hello message includes: a Sender ID which is set to the LS's ID (LSID), zero or more Receiver IDs which identify the DCSs from which the LS has recently heard a Hello message (as described below), and a HelloInterval and DeadFactor which will be described below. At this point, the DCS may or may not already be sending its own Hello messages to the LS.

When the LS receives a Hello message from one of its DCSs, the LS checks to see if its LSID is in one of the Receiver ID fields of that message which it just received, and the LS saves the Sender ID from that Hello message. If the LSID is in one of the Receiver ID fields then the LS transitions the HFSM to the Bidirectional Connection state otherwise it transitions the HFSM into the Unidirectional Connection state. The Sender ID which was saved is the DCS's ID (DCSID). At some point before the next time that the LS sends its own Hello message to the DCS, the LS will check the saved DCSID against a list of Receiver IDs which the LS uses when sending the LS's own Hello messages. If the DCSID is not found in the list of Receiver IDs then it is added to that list before the LS sends its Hello message.

Hello messages also contain a HelloInterval and a DeadFactor. The Hello interval advertises the time (in seconds) between sending of consecutive Hello messages by the server which is sending the "current" Hello message. That is, if the time between reception of Hello messages from a DCS exceeds the HelloInterval advertised by that DCS then the next Hello message is to be considered late by the LS. If the LS does not receive a Hello message, which contains the LS's LSID in one of the Receiver ID fields, within the interval HelloInterval*DeadFactor seconds (where DeadFactor was advertised by the DCS in a previous Hello message) then the LS MUST consider the DCS to be stalled. At which point one of two things will happen: 1) if any Hello messages have been received during the last HelloInterval*DeadFactor seconds then the LS should transition the HFSM for that DCS to the Unidirectional Connection State; otherwise, the LS should transition the HFSM for that DCS to the Waiting State and remove the DCSID from the Receiver ID list.

Note that the Hello Protocol is on a per PID/SGID basis. Thus, for example, if there are two servers (one in SG A and the other in SG B) associated with an NBMA address X and another two servers (also one in SG A and the other in SG B) associated with NBMA address Y and there is a suitable point-to-point VC between the NBMA addresses then there are two HFSMs running on each side of the VC (one per PID/SGID).

Hello messages contain a list of Receiver IDs instead of a single Receiver ID in order to make use of point to multipoint connections. While there is an HFSM per DCS, an LS MUST send only a single Hello message to its DCSs attached as leaves of a point to multipoint connection. The LS does this by including DCSIDs in the list of Receiver IDs when the LS's sends its next Hello message. Only the DCSIDs from non-stalled DCSs from which the LS has heard a Hello message are included.

Any abnormal event, such as receiving a malformed SCSP message, causes the HFSM to transition to the Waiting State; however, a loss of NBMA connectivity causes the HFSM to transition to the Down State. Until the HFSM is in the Bidirectional Connection State, if any properly formed SCSP messages other than Hello messages are received then those messages MUST be ignored (this is for the case where, for example, there is a point to multipoint connection involved).

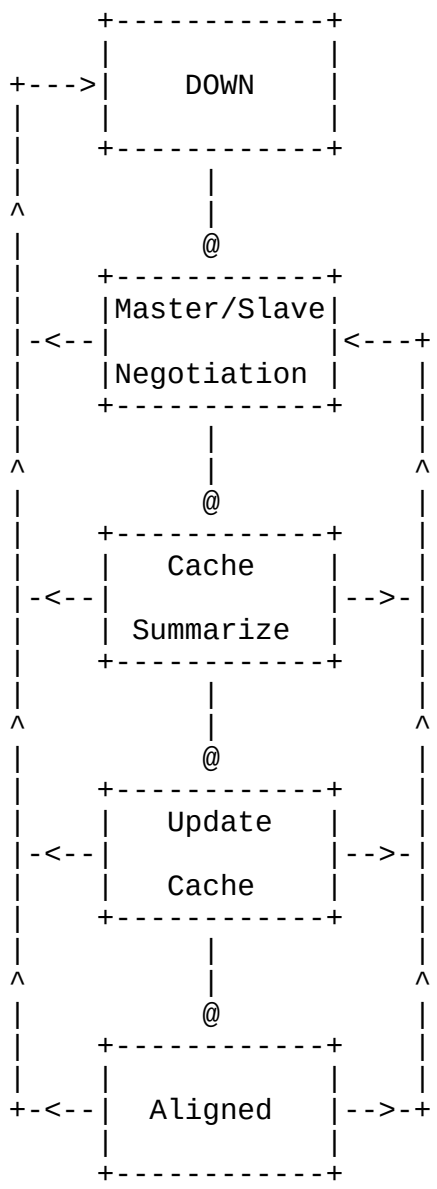


Figure 2: Cache Alignment Finite State Machine

2.2 Cache Alignment Protocol

"Cache Alignment" (CA) messages are used by an LS to synchronize its cache with that of the cache of each of its DCSs. That is, CA messages allow a booting LS to synchronize with each of its DCSs. A CA message contains a CA header followed by zero or more Cache State Advertisement Summary records (CSAS records).

An LS has a Cache Alignment Finite State Machine (CAFSM) associated (see Figure 2) with each of its DCSs on a per PID/SGID basis, and the CAFSM monitors the state of the cache alignment between the servers. The CAFSM starts in the Down State. The CAFSM is associated with an HFSM, and when that HFSM reaches the Bidirectional State, the CAFSM transitions to the Master/Slave Negotiation State. The Master/Slave Negotiation State causes either the LS or DCS to take on the role of master over the cache alignment process. In a sense, the master server sets the tempo for the cache alignment.

When the LS's CAFSM reaches the Master/Slave Negotiation State, the LS will send a CA message to the DCS associated with the CAFSM. The format of CA messages are described in Section B.2.1. The first CA message which the LS sends includes no CSAS records and a CA header which contains the LSID in the Sender ID field, the DCSID in the Receiver ID field, a CA sequence number, and three bits. These three bits are the M (Master/Slave) bit, the I (Initialization of master) bit, and the O (More) bit. In the first CA message sent by the LS to a particular DCS, the M, O, and I bits are set to one. If the LS does not receive a CA message from the DCS in CAReXmtInterval seconds then it resends the CA message it just sent. The LS continues to do this until the CAFSM transitions to the Cache Summarize State or until the HFSM transitions out of the Bidirectional State. Any time the HFSM transitions out of the Bidirectional State, the CAFSM transitions to the Down State.

2.2.1 Master Slave Negotiation State

When the LS receives a CA message from the DCS while in the Master/Slave Negotiation State, the role the LS plays in the exchange depends on packet processing as follows:

1) If the CA from the DCS has the M, I, and O bits set to one and there are no CSAS records in the CA message and the Sender ID as specified in the DCS's CA message is larger than the LSID then

   a) The timer counting down the CAReXmtInterval is stopped.
   b) The CAFSM corresponding to that DCS transitions to the Cache Summarize State and the LS takes on the role of slave.
   c) The LS adopts the CA sequence number it received in the CA message as its own CA sequence number.
   d) The LS sends a CA message to the DCS which is formated as follows: the M and I bits are set to zero, the Sender ID field is set to the LSID, the Receiver ID field is set to the DCSID, and the CA sequence number is set to the CA sequence number that appeared in the DCS's CA message. If there are CSAS records to be sent (i.e., if the LS's cache is not empty), and if all of them will not fit into this CA message then the O bit is set to

one and the initial set of CSAS records are included in the CA message; otherwise the O bit is set to zero and if any CSAS Records need to be sent then those records are included in the CA message.

2) If the CA message from the DCS has the M and I bits off and the Sender ID as specified in the DCS's CA message is smaller than the LSID then

   a) The timer counting down the CAReXmtInterval is stopped.
   b) The CAFSM corresponding to that DCS transitions to the Cache Summarize State and the LS takes on the role of master.
   c) The LS must process the received CA message. An explanation of CA message processing is given below.
   d) The LS sends a CA message to the DCS which is formated as follows: the M bit is set to one, I bit is set to zero, the Sender ID field is set to the LSID, the Receiver ID field is set to the DCSID, and the LS's current CA sequence number is incremented by one and placed in the CA message. If there are any CSAS records to be sent from the LS to the DCS (i.e., if the LS's cache is not empty) then the O bit is set to one and the initial set of CSAS records are included in the CA message that the LS is sending to the DCS.

3) Otherwise, the packet must be ignored.

2.2.2 The Cache Summarize State

At any given time, the master or slave have at most one outstanding CA message. Once the LS's CAFSM has transitioned to the Cache Summarize State the sequence of exchanges of CA messages occurs as follows:

1) If the LS receives a CA message with the M bit set incorrectly (e.g., the M bit is set in the CA of the DCS and the LS is master) or if the I bit is set then the CAFSM transitions back to the Master/Slave Negotiation State.

2) If the LS is master and the LS receives a CA message with a CA sequence number which is one less than the LS's current CA sequence number then the message is a duplicate and the message MUST be discarded.

3) If the LS is master and the LS receives a CA message with a CA sequence number which is equal to the LS's current CA sequence number then the CA message MUST be processed. An explanation of "CA message processing" is given below. As a result of having received the CA message from the DCS the following will occur:

a) The timer counting down the CAReXmtInterval is stopped.
b) The LS must process any CSAS records in the received CA message.
c) Increment the LS's CA sequence number by one.
d) The cache exchange continues as follows:

1) If the LS has no more CSAS records to send and the received CA message has the O bit off then the CAFSM transitions to the Update Cache State.
2) If the LS has no more CSAS records to send and the received CA message has the O bit on then the LS sends back a CA message (with new CA sequence number) which contains no CSAS records and with the O bit off. Reset the timer counting down the CAReXmtInterval.
3) If the LS has more CSAS records to send then the LS sends the next CA message with the LS's next set of CSAS records. If LS is sending its last set of CSAS records then the O bit is set off otherwise the O bit is set on. Reset the timer counting down the CAReXmtInterval.

4) If the LS is slave and the LS receives a CA message with a CA sequence number which is equal to the LS's current CA sequence number then the CA message is a duplicate and the LS MUST resend the CA message which it had just sent to the DCS.

5) If the LS is slave and the LS receives a CA message with a CA sequence number which is one more than the LS's current CA sequence number then the message is valid and MUST be processed. An explanation of "CA message processing" is given below. As a result of having received the CA message from the DCS the following will occur:

a) The LS must process any CSAS records in the received CA message.
b) Set the LS's CA sequence number to the CA sequence number in the CA message.
c) The cache exchange continues as follows:

1) If the LS had just sent a CA message with the O bit off and the received CA message has the O bit off then the CAFSM transitions to the Update Cache State and the LS sends a CA message with no CSAS records and with the O bit off.
2) If the LS still has CSAS records to send then the LS MUST send a CA message with CSAS records in it.

a) If the message being sent from the LS to the DCS does not contain the last CSAS records that the LS needs to send then the CA message is sent with the O bit on.
b) If the message being sent from the LS to the DCS does contain the last CSAS records that the LS needs to

send and the CA message just received from the DCS had the O bit off then the CA message is sent with the O bit off, and the LS transitions the CAFSM to the Update Cache State.

c) If the message being sent from the LS to the DCS does contain the last CSAS records that the LS needs to send and the CA message just received from the DCS had the O bit on then the CA message is sent with the O bit off and the alignment process continues.

6) If the LS is slave and the LS receives a CA message with a CA sequence number that is neither equal to nor one more than the current LS's CA sequence number then an error has occurred and the CAFSM transitions to the Master/Slave Negotiation State.

Note that if the LS was slave during the CA process then the LS upon transitioning the CAFSM to the Update Cache state MUST keep a copy of the last CA message it sent and the LS SHOULD set a timer equal to CAReXmtInterval. If either the timer expires or the LS receives a CSU Solicit (CSUS) message (CSUS messages are described in Section 2.2.3) from the DCS then the LS releases the copy of the CA message. The reason for this is that if the DCS (which is master) loses the last CA message sent by the LS then the DCS will resend its previous CA message with the last CA Sequence number used. If that were to occur the LS would need to resend its last sent CA message as well.

2.2.2.1 "CA message processing":

The LS makes a list of those cache entries which are more "up to date" in the DCS than the LS's own cache. This list is called the CSA Request List (CRL). See Section 2.4 for a description of what it means for a CSA (Client State Advertisement) record or CSAS record to be more "up to date" than an LS's cache entry.

2.2.3 The Update Cache State

If the CRL of the associated CAFSM of the LS is empty upon transition into the Update Cache State then the CAFSM immediately transitions into the Aligned State.

If the CRL is not empty upon transition into the Update Cache State then the LS solicits the DCS to send the CSA records corresponding to the summaries (i.e., CSAS records) which the LS holds in its CRL. The solicited CSA records will contain the entirety of the cached information held in the DCS's cache for the given cache entry. The LS solicits the relevant CSA records by forming CSU Solicit (CSUS) messages from the CRL. See Section B.2.4 for the description of the CSUS message format. The LS then sends the CSUS messages to the DCS. The DCS responds to the CSUS message by sending to the LS one or more

CSU Request messages containing the entirety of newer cached information identified in the CSUS message. Upon receiving the CSU Request the LS will send one or more CSU Replies as described in Section 2.3. Note that the LS may have at most one CSUS message outstanding at any given time.

Just before the first CSUS message is sent from an LS to the DCS associated with the CAFSM, a timer is set to CSUSReXmtInterval seconds. If all the CSA records corresponding to the CSAS records in the CSUS message have not been received by the time that the timer expires then a new CSUS message will be created which contains all the CSAS records for which no appropriate CSA record has been received plus additional CSAS records not covered in the previous CSUS message. The new CSUS message is then sent to the DCS. If, at some point before the timer expires, all CSA record updates have been received for all the CSAS records included in the previously sent CSUS message then the timer is stopped. Once the timer is stopped, if there are additional CSAS records that were not covered in the previous CSUS message but were in the CRL then the timer is reset and a new CSUS message is created which contains only those CSAS records from the CRL which have not yet been sent to the DCS. This process continues until all the CSA records corresponding CSAS records that were in the CRL have been received by the LS. When the LS has a completely updated cache then the LS transitions CAFSM associated with the DCS to the Aligned State.

If an LS receives a CSUS message or a CA message with a Receiver ID which is not the LS's LSID then the message must be discarded and ignored. This is necessary since an LS may be a leaf of a point to multipoint connection with other servers in the SG.

2.2.4 The Aligned State

While in the Aligned state, an LS will perform the Cache State Update Protocol as described in Section 2.3.

Note that an LS may receive a CSUS message while in the Aligned State and, the LS MUST respond to the CSUS message with the appropriate CSU Request message in a similar fashion to the method previously described in Section 2.2.3.

2.3 Cache State Update Protocol

"Cache State Update" (CSU) messages are used to dynamically update the state of cache entries in servers on a given PID/SGID basis. CSU messages contain zero or more "Cache State Advertisement" (CSA) records each of which contains its own snapshot of the state of a particular cache entry. An LS may send/receive a CSU to/from a DCS

only when the corresponding CAFSM is in either the Aligned State or the Update Cache State.

There are two types of CSU messages: CSU Requests and CSU Replies. See Sections B.2.2 and B.2.3 respectively for message formats. A CSU Request message is sent from an LS to one or more DCSs for one of two reasons: either the LS has received a CSUS message and MUST respond only to the DCS which originated the CSUS message, or the LS has become aware of a change of state of a cache entry. An LS becomes aware of a change of state of a cache entry either through receiving a CSU Request from one of its DCSs or as a result of a change of state being observed in a cached entry originated by the LS. In the former case, the LS will send a CSU Request to each of its DCSs except the DCS from which the LS became aware of the change in state. In the latter case, the LS will send a CSU Request to each of its DCSs. The change in state of a particular cache entry is noted in a CSA record which is then appended to the end of the CSU Request message mandatory part. In this way, state changes are propagated throughout the SG.

Examples of such changes in state are as follows:

1) a server receives a request from a client to add an entry to its cache,
2) a server receives a request from a client to remove an entry from its cache,
3) a cache entry has timed out in the server’s cache, has been refreshed in the server’s cache, or has been administratively modified.

When an LS receives a CSU Request from one of its DCSs, the LS acknowledges one or more CSA Records which were contained in the CSU Request by sending a CSU Reply. The CSU Reply contains one or more CSAS records which correspond to those CSA records which are being acknowledged. Thus, for example, if a CSA record is dropped (or delayed in processing) by the LS because there are insufficient resources to process it then a corresponding CSAS record is not included in the CSU Reply to the DCS.

Note that an LS may send multiple CSU Request messages before receiving a CSU Reply acknowledging any of the CSA Records contained in the CSU Requests. Note also that a CSU Reply may contain acknowledgments for CSA Records from multiple CSU Requests. Thus, the terms "request" and "reply" may be a bit confusing.

Note that a CSA Record contains a CSAS Record followed by client/server protocol specific information contained in a cache entry (see Section B.2.0.2 for CSAS record format information and

Section B.2.2.1 for CSA record format information). When a CSA record is considered by the LS to represent cached information which is more "up to date" (see Section 2.4) than the cached information contained within the cache of the LS then two things happen: 1) the LS's cache is updated with the more up to date information, and 2) the LS sends a CSU Request containing the CSA Record to each of its DCSs except the one from which the CSA Record arrived. In this way, state changes are propagated within the PID/SGID. Of course, at some point, the LS will also acknowledge the reception of the CSA Record by sending the appropriate DCS a CSU Reply message containing the corresponding CSAS Record.

When an LS sends a new CSU Request, the LS keeps track of the outstanding CSA records in that CSU Request and to which DCSs the LS sent the CSU Request. For each DCS to which the CSU Request was sent, a timer set to CSUReXmtInterval seconds is started just prior to sending the CSU Request. This timer is associated with the CSA Records contained in that CSU Request such that if that timer expires prior to having all CSA Records acknowledged from that DCS then (and only then) a CSU Request is re-sent by the LS to that DCS. However, the re-sent CSU Request only contains those CSA Records which have not yet been acknowledged. If all CSA Records associated with a timer becomes acknowledged then the timer is stopped. Note that the re-sent CSA Records follow the same time-out and retransmit rules as if they were new. Retransmission will occur a configured number of times for a given CSA Record and if acknowledgment fails to occur then an "abnormal event" has occurred at which point the then the HFSM associated with the DCS is transitioned to the Waiting State.

A CSA Record instance is said to be on a "DCS retransmit queue" when it is associated with the previously mentioned timer. Only the most up-to-date CSA Record is permitted to be queued to a given DCS retransmit queue. Thus, if a less up-to-date CSA Record is queued to the DCS retransmit queue when a newer CSA Record instance is about to be queued to that DCS retransmit queue then the older CSA Record instance is dequeued and disassociated with its timer immediately prior to enqueuing the newer instance of the CSA Record.

When an LS receives a CSU Reply from one of its DCSs then the LS checks each CSAS record in the CSU Reply against the CSAS Record portion of the CSA Records which are queued to the DCS retransmit queue.

1) If there exists an exact match between the CSAS record portion of the CSA record and a CSAS Record in the CSU Reply then that CSA Record is considered to be acknowledged and is thus dequeued from the DCS retransmit queue and is disassociated with its timer.

2) If there exists a match between the CSAS record portion of the CSA record and a CSAS Record in the CSU Reply except for the CSA Sequence number then

   a) If the CSA Record queued to the DCS retransmit queue has a CSA Sequence Number which is greater than the CSA Sequence Number in the CSAS Record of the the CSU Reply then the CSAS Record in the CSU Reply is ignored.
   b) If the CSA Record queued to the DCS retransmit queue has a CSA Sequence Number which is less than the CSA Sequence Number in the CSAS Record of the the CSU Reply then CSA Record which is queued to the DCS retransmit queue is dequeued and the CSA Record is disassociated with its timer. Further, a CSUS Message is sent to that DCS which sent the more up-to-date CSAS Record. All normal CSUS processing occurs as if the CSUS were sent as part of the CA protocol.

When an LS receives a CSU Request message which contains a CSA Record which contains a CSA Sequence Number which is smaller than the CSA Sequence number of the cached CSA then the LS MUST acknowledge the CSA record in the CSU Request but it MUST do so by sending a CSU Reply message containing the CSAS Record portion of the CSA Record stored in the cache and not the CSAS Record portion of the CSA Record contained in the CSU Request.

An LS responds to CSUS messages from its DCSs by sending CSU Request messages containing the appropriate CSA records to the DCS. If an LS receives a CSUS message containing a CSAS record for an entry which is no longer in its database (e.g., the entry timed out and was discarded after the Cache Alignment exchange completed but before the entry was requested through a CSUS message), then the LS will respond by copying the CSAS Record from the CSUS message into a CSU Request message and the LS will set the N bit signifying that this record is a NULL record since the cache entry no longer exists in the LS's cache. Note that in this case, the "CSA Record" included in the CSU Request to signify the NULL cache entry is literally only a CSAS Record since no client/server protocol specific information exists for the cache entry.

If an LS receives a CSA Record in a CSU Request from a DCS for which the LS has an identical CSA record posted to the corresponding DCS's DCS retransmit queue then the CSA Record on the DCS retransmit queue is considered to be implicitly acknowledged. Thus, the CSA Record is dequeued from the DCS retransmit queue and is disassociated with its timer. The CSA Record sent by the DCS MUST still be acknowledged by the LS in a CSU Reply, however. This is useful in the case of point

to multipoint connections where the rule that "when an LS receives a CSA record from a DCS, that LS floods the CSA Record to every DCS except the DCS from which it was received" might be broken.

If an LS receives a CSU with a Receiver ID which is not equal to the LSID and is not set to all 0xFFs then the CSU must be discarded and ignored. This is necessary since the LS may be a leaf of a point to multipoint connection with other servers in the LS's SG.

An LS MAY send a CSU Request to the all 0xFFs Receiver ID when the LS is a root of a point to multipoint connection with a set of its DCSs. If an LS receives a CSU Request with the all 0xFFs Receiver ID then it MUST use the Sender ID in the CSU Request as the Receiver ID of the CSU Reply (i.e., it MUST unicast its response to the sender of the request) when responding. If the LS wishes to send a CSU Request to the all 0xFFs Receiver ID then it MUST create a time-out and retransmit timer for each of the DCSs which are leaves of the point to multipoint connection prior to sending the CSU Request. If in this case, the time-out and retransmit timer expires for a given DCS prior to acknowledgment of a given CSA Record then the LS MUST use the specific DCSID as the Receiver ID rather than the all 0xFFs Receiver ID. Similarly, if it is necessary to re-send a CSA Record then the LS MUST specify the specific DCSID as the Receiver ID rather than the all 0xFFs Receiver ID.

Note that if a set of servers are in a full mesh of point to multipoint connections, and one server of that mesh sends a CSU Request into that full mesh, and the sending server sends the CSA Records in the CSU Request to the all 0xFFs Receiver ID then it would not be necessary for every other server in the mesh to source their own CSU Request containing those CSA Records into the mesh in order to properly flood the CSA Records. This is because every server in the mesh would have heard the CSU Request and would have processed the included CSA Records as appropriate. Thus, a server in a full mesh could consider the mesh to be a single logical port and so the rule that "when an LS receives a CSA record from a DCS, that LS floods the CSA Record to every DCS except the DCS from which it was received" is not broken. A receiving server in the full mesh would still need to acknowledge the CSA records with CSU Reply messages which contain the LSID of the replying server as the Sender ID and the ID of the server which sent the CSU Request as the Receiver ID field. In the time out and retransmit case, the Receiver ID of the CSU Request would be set to the specific DCSID which did not acknowledge the CSA Record (as opposed to the all 0xFFs Receiver ID). Since a full mesh emulates a broadcast media for the servers attached to the full mesh, use of SCSP on a broadcast medium might use this technique as well. Further discussion of this use of a full mesh or use of a broadcast media is left to the client/server protocol

specific documents.

2.4 The meaning of "More Up To Date"/"Newness"

During the cache alignment process and during normal CSU processing, a CSAS Record is compared against the contents of an LS's cache entry to decide whether the information contained in the record is more "up to date" than the corresponding cache entry of the LS.

There are three pieces of information which are used in determining whether a record contains information which is more "up to date" than the information contained in the cache entry of an LS which is processing the record: 1) the Cache Key, 2) the Originator which is described by an Originator ID (OID), and 3) the CSA Sequence number. See Section B.2.0.2 for more information on these fields.

Given these three pieces of information, a CSAS record (be it part of a CSA Record or be it stand-alone) is considered to be more "up to date" than the information contained in the cache of an LS if all of the following are true:

1) The Cache Key in the CSAS Record matches the stored Cache Key in the LS's cache entry,
2) The OID in the CSAS Record matches the stored OID in the LS's cache entry,
3) The CSA Sequence Number in the CSAS Record is greater than CSA Sequence Number in the LS's cache entry.

Discussion and Conclusions

While the above text is couched in terms of synchronizing the knowledge of the state of a client within the cache of servers contained in a SG, this solution generalizes easily to any number of database synchronization problems (e.g., LECS synchronization).

SCSP defines a generic flooding protocol. There are a number of related issues relative to cache maintenance and topology maintenance which are more appropriately defined in the client/server protocol specific documents; for example, it might be desirable to define a generic cache entry time-out mechanism for a given protocol or to advertise adjacency information between servers so that one could obtain a topo-map of the servers in a SG. When mechanisms like these are desirable, they will be defined in the client/server protocol specific documents.

Appendix A: Terminology and Definitions

CA Message - Cache Alignment Message
These messages allow an LS to synchronize its entire cache with that of the cache of one of its DCSs.

CAFSM - Cache Alignment Finite State Machine
The CAFSM monitors the state of the cache alignment between an LS and a particular DCS. There exists one CAFSM per DCS as seen from an LS.

CSA Record - Cache State Advertisement Record
A CSA is a record within a CSU message which identifies an update to the status of a "particular" cache entry.

CSAS Record - Cache State Advertisement Summary Record
A CSAS contains a summary of the information in a CSA. A server will send CSAS records describing its cache entries to another server during the cache alignment process. CSAS records are also included in a CSUS messages when an LS wants to request the entire CSA from the DCS. The LS is requesting the CSA from the DCS because the LS believes that the DCS has a more recent view of the state of the cache entry in question.

CSU Message - Cache State Update message
This is a message sent from an LS to its DCSs when the LS becomes aware of a change in state of a cache entry.

CSUS Message - Cache State Update Solicit Message
This message is sent by an LS to its DCS after the LS and DCS have exchanged CA messages. The CSUS message contains one or more CSAS records which represent solicitations for entire CSA records (as opposed to just the summary information held in the CSAS).

DCS - Directly Connected Server
The DCS is a server which is directly connected to the LS; e.g., there exists a VC between the LS and DCS. This term, along with the terms LS and RS, is used to give a frame of reference when talking about servers and their synchronization. Unless explicitly stated to the contrary, there is no implied difference in functionality between a DCS, LS, and RS.

HFSM - Hello Finite State Machine
An LS has a HFSM associated with each of its DCSs. The HFSM monitors the state of the connectivity between the LS and a particular DCS.

LS - Local Server
The LS is the server under scrutiny; i.e., all statements are made from the perspective of the LS. This term, along with the terms DCS and RS, is used to give a frame of reference when talking about servers and their synchronization. Unless explicitly stated to the contrary, there is no implied difference in functionality between a DCS, LS, and RS.

LSID - Local Server ID
The LSID is a unique token that identifies an LS. This value might be taken from the protocol address of the LS.

PID - Protocol ID
This field contains an identifier which identifies the client/server protocol which is making use of SCSP for the given message. The assignment of Protocol IDs for this field is given over to IANA as described in Section C.

RS - Remote Server (RS)
From the perspective of an LS, an RS is a server, separate from the LS, which is not directly connected to the LS (i.e., an RS is always two or more hops away from an LS whereas a DCS is always one hop away from an LS). Unless otherwise stated an RS refers to a server in the SG. This term, along with the terms LS and DCS, is used to give a frame of reference when talking about servers and their synchronization. Unless explicitly stated to the contrary, there is no implied difference in functionality between a DCS, LS, and RS.

SG - Server Group
The SCSP synchronizes caches (or a portion of the caches) of a set of server entities which are bound to a SG through some means (e.g., all servers belonging to a Logical IP Subnet (LIS)[1]). Thus an SG is just a grouping of servers around some commonality.

SGID - Server Group ID
This ID is a 16 bit identification field that uniquely identifies the instance client/server protocol for which the servers of the SG are being synchronized. This implies that multiple instances of the same protocol may be in operation at the same time and have their servers synchronized independently of each other.

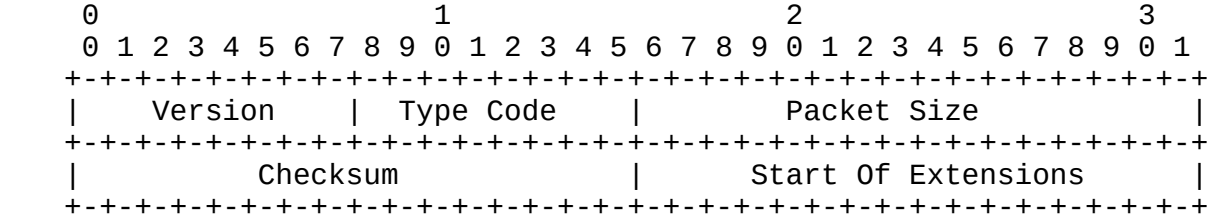
Appendix B: SCSP Message Formats

This section of the appendix includes the message formats for SCSP. SCSP protocols are LLC/SNAP encapsulated with an LLC=0xAA-AA-03 and OUI=0x00-00-5e and PID=0x00-05.

SCSP has 3 parts to every packet: the fixed part, the mandatory part, and the extensions part. The fixed part of the message exists in every packet and is shown below. The mandatory part is specific to the particular message type (i.e., CA, CSU Request/Reply, Hello, CSUS) and, it includes (among other packet elements) a Mandatory Common Part and zero or more records each of which contains information pertinent to the state of a particular cache entry (except in the case of a Hello message) whose information is being synchronized within a SG. The extensions part contains the set of extensions for the SCSP message.

In the following message formats, the fields marked as "unused" MUST be set to zero upon transmission of such a message and ignored upon receipt of such a message.

B.1 Fixed Part



Version
This is the version of the SCSP protocol being used. The current version is 1.

Type Code
This is the code for the message type (e.g., Hello (5), CSU Request(2), CSU Reply(3), CSUS (4), CA (1)).

Packet Size
The total length of the SCSP packet, in octets (excluding link layer and/or other protocol encapsulation).

Checksum
The standard IP checksum over the entire SCSP packet starting at the fixed header. If the packet is an odd number of bytes in length then this calculation is performed as if a byte set to 0x00 is appended to the end of the packet.

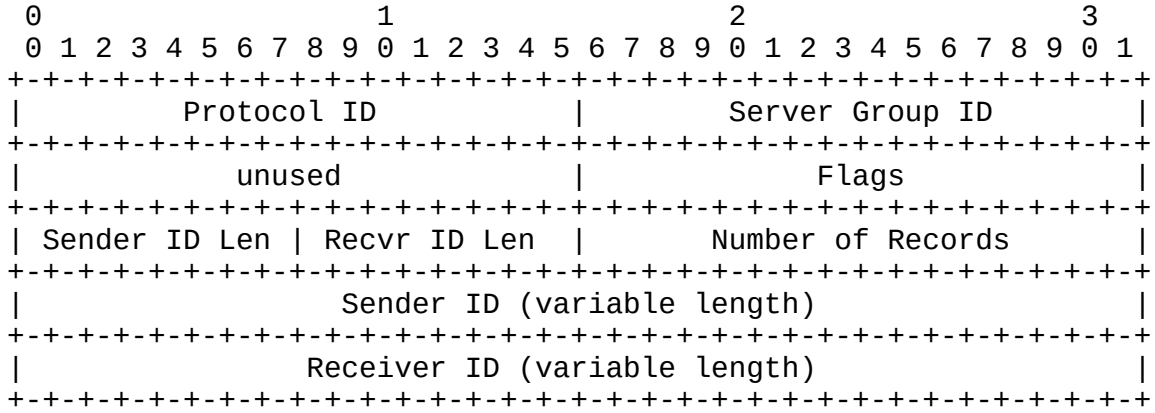
Start Of Extensions
This field is coded as zero when no extensions are present in the message. If extensions are present then this field will be coded with the offset from the top of the fixed header to the beginning of the first extension.

B.2.0 Mandatory Part

The mandatory part of the SCSP packet contains the operation specific information for a given message type (e.g., SCSP Cache State Update Request/Reply, etc.), and it includes (among other packet elements) a Mandatory Common Part (described in Section B.2.0.1) and zero or more records each of which contains information pertinent to the state of a particular cache entry (except in the case of a Hello message) whose information is being synchronized within a SG. These records may, depending on the message type, be either Cache State Advertisement Summary (CSAS) Records (described in Section B.2.0.2) or Cache State Advertisement (CSA) Records (described in Section B.2.2.1). CSA Records contain a summary of a cache entry's information (i.e., a CSAS Record) plus some additional client/server protocol specific information. The mandatory common part format and CSAS Record format is shown immediately below, prior to showing their use in SCSP messages, in order to prevent replication within the message descriptions.

B.2.0.1 Mandatory Common Part

Sections B.2.1 through B.2.5 have a substantial overlap in format. This overlapping format is called the mandatory common part and its format is shown below:



Protocol ID
This field contains an identifier which identifies the client/server protocol which is making use of SCSP for the given message. The assignment of Protocol IDs for this field is given over to IANA as described in Section C. Protocols with current documents have the following defined values:

1 - ATMARP
2 - NHRP
3 - MARS
4 - DHCP
5 - LNNI

Server Group ID
This ID is uniquely identifies the instance of a given client/server protocol for which servers are being synchronized.

Flags
The Flags field is message specific, and its use will be described in the specific message format sections below.

Sender ID Len
This field holds the length in octets of the Sender ID.

Recvr ID Len
This field holds the length in octets of the Receiver ID.

Number of Records
This field contains the number of additional records associated with the given message. The exact format of these records is specific to the message and will be described for each message type in the sections below.

Sender ID
This is an identifier assigned to the server which is sending the given message. One possible assignment might be the protocol address of the sending server.

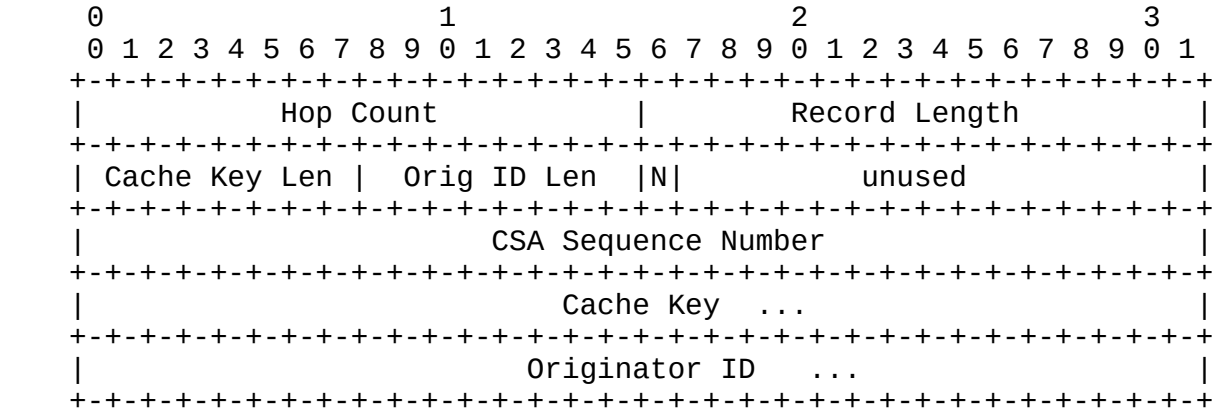
Receiver ID
This is an identifier assigned to the server which is to receive the given message. One possible assignment might be the protocol address of the server which is to receive the given message.

B.2.0.2 Cache State Advertisement Summary Record (CSAS record)

CSAS records contain a summary of information contained in a cache entry of a given client/server database which is being synchronized through the use of SCSP. The summary includes enough information for SCSP to look into the client/server database for the appropriate database cache entry and then compare the "newness" of the summary against the "newness" of the cached entry.

Note that CSAS records do not contain a Server Group ID (SGID) nor do they contain a Protocol ID. These IDs are necessary to identify which protocol and which instance of that protocol for which the summary is applicable. These IDs are present in the mandatory common part of each message.

Note also that the values of the Hop Count and Record Length fields of a CSAS Record are dependent on whether the CSAS record exists as a "stand-alone" record or whether the CSAS record is "embedded" in CSA Record. This is further described below.



Hop Count
This field represents the number of hops that the record may take before being dropped. Thus, at each server that the record traverses, the Hop Count is decremented. This field is set to 1 when the CSAS record is a "stand-alone" record (i.e., it is not embedded within a CSA record) since summaries do not go beyond one hop during the cache alignment process. If a CSAS record is "embedded" within a CSA record then the Hop Count is set to an administratively defined value which is almost certainly greater than or equal to the the cardinality of the SG minus one. Note that an exception to the previous rule occurs when the CSA Record is carried within a CSU Request which was sent in response to a solicitation (i.e., in response to a CSAS Record which was sent in a CSUS message); in which case, the Hop Count SHOULD be set to 1.

Record Length
If the CSAS record is a "stand-alone" record then this value is 12+"Cache Key Leng"+"Orig ID Len" in bytes; otherwise, this value is set to 12+"Cache Key Leng"+"Orig ID Len"+ sizeof("Client/Server Protocol Specific Part for cache entry"). The size of the Client/Server Protocol Specific Part may be obtained from the client/server protocol specific document for the given Protocol ID.

Cache Key Len
Length of the Cache Key field in bytes.

Orig ID Len.
Length of the Originator ID field in bytes.

N
The "N" bit signifies that this CSAS Record is actually a Null record. This bit is only used in a CSAS Record contained in a CSU Request/Reply which is sent in response to a CSUS message. It is possible that an LS may receive a solicitation for a CSA record when the cache entry represented by the solicited CSA Record no longer exists in the LS's cache (see Section 2.3 for details). In this case, the LS copies the CSAS Record directly from the CSUS message into the CSU Request, and the LS sets the N bit signifying that the cache entry does not exist any longer. The DCS which solicited the CSA record which no longer exists will still respond with a CSU Reply. This bit is usually set to zero.

CSA Sequence Number
This field contains a sequence number that identifies the "newness" of a CSA record instance being summarized. A "larger" sequence number means a more recent advertisement. Thus, if the state of part (or all) of a cache entry needs to be updated then the CSA record advertising the new state MUST contain a CSA Sequence Number which is larger than the one corresponding to the previous advertisement. This number is assigned by the originator of the CSA record. The CSA Sequence Number may be assigned by the originating server or by the client which caused its server to advertise its existence.

The CSA Sequence Number is a signed 32 bit number. Within the CSA Sequence Number space, the number -2^31 (0x80000000) is reserved. Thus, the usable portion of the CSA Sequence Number space for a given Cache Key is between the numbers -2^31+1 (0x80000001) and 2^31-1 (0x7fffffff). An LS uses -2^31+1 the first time it originates a CSA Record for a cache entry that it created. Each time the cache entry is modified in some manner and when that modification needs to be synchronized with the other servers in the SG, the LS increments the CSA Sequence number associated with the

given Cache Key and uses that new CSA Sequence Number when advertising the update. If it is ever the case that a given CSA Sequence Number has reached 2^31-2 and the associated cache entry has been modified such that an update must be sent to the rest of the servers in the SG then the given cache entry MUST first be purged from the SG by the LS by sending a CSA Record which causes the cache entry to be removed from other servers and this CSA Record carries a CSA Sequence Number of 2^31-1. The exact packet format and mechanism by which a cache entry is purged is defined in the appropriate protocol specific document. After the purging CSA Record has been acknowledged by each DCS, an LS will then send a new CSA Record carrying the updated information, and this new CSA Record will carry a CSA Sequence Number of -2^31+1.

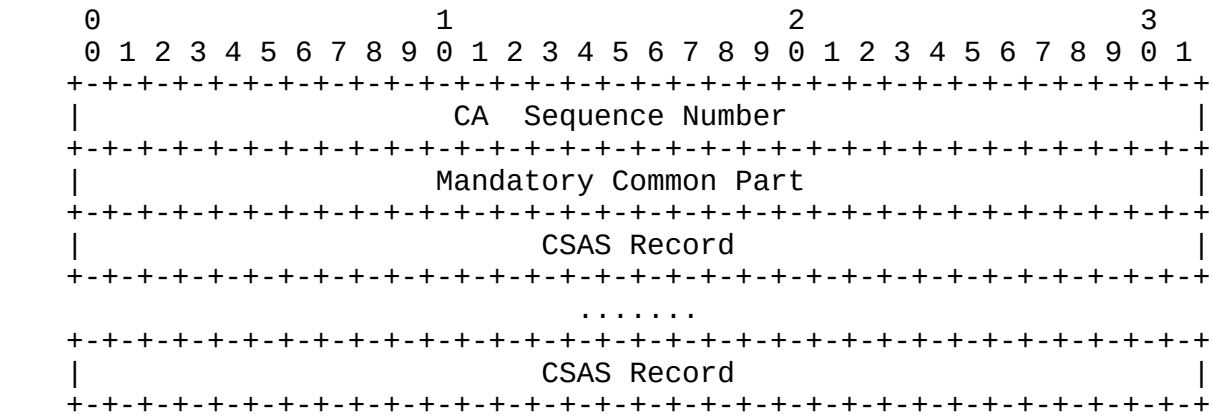
After a restart occurs and after the restarting LS's CAFSM has achieved the Aligned state, if an update to an existing cache entry needs to be synchronized or a new cache entry needs to be synchronized then the ensuing CSA Record MUST contain a CSA Sequence Number which is unique within the SG for the given OID and Cache Key. The RECOMMENDED method of obtaining this number (unless explicitly stated to the contrary in the protocol specific document) is to set the CSA Sequence Number in the CSA Record to the CSA Sequence Number associated with the existing cache entry (if an out of date cache entry already exists and zero if not) plus a configured constant. Note that the protocol specific document may require that all cache entries containing the OID of the restarting LS be purged prior to updating the cache entries; in this case, the updating CSA Record will still contain a CSA Sequence Number set to the CSA Sequence Number associated with the previously existing cache entry plus a configured constant.

Cache Key
This is a database lookup key that uniquely identifies a piece of data which the originator of a CSA Record wishes to synchronize with its peers for a given "Protocol ID/Server Group ID" pair. This key will generally be a small opaque byte string which SCSP will associate with a given piece of data in a cache. Thus, for example, an originator might assign a particular 4 byte string to the binding of an IP address with that of an ATM address. Generally speaking, the originating server of a CSA record is responsible for generating a Cache Key for every element of data that the the given server originates and which the server wishes to synchronize with its peers in the SG.

Originator ID
This field contains an ID administratively assigned to the server which is the originator of CSA Records.

B.2.1 Cache Alignment (CA)

The Cache Alignment (CA) message allows an LS to synchronize its entire cache with that of the cache of its DCSs within a server group. The CA message type code is 1. The CA message mandatory part format is as follows:



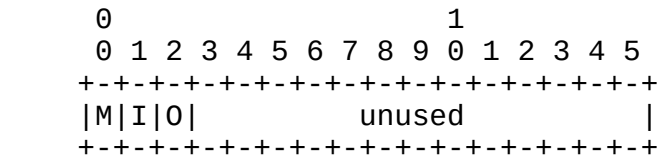
CA Sequence Number
A value which provides a unique identifier to aid in the sequencing of the cache alignment process. A "larger" sequence number means a more recent CA message. The slave server always copies the sequence number from the master server's previous CA message into its current CA message which it is sending and the the slave acknowledges the master's CA message. Since the initial CA process is lock-step, if the slave does not receive the same sequence number which it previously received then the information in the slave's previous CA message is implicitly acknowledged. Note that there is a separate CA Sequence Number space associated with each CAFSM.

Whenever it is necessary to (re)start cache alignment and the CAFSM enters the Master/Slave Negotiation state, the CA Sequence Number should be set to a value not previously seen by the DCS. One possible scheme is to use the machine's time of day counter.

Mandatory Common Part
The mandatory common part is described in detail in Section B.2.0.1. There are two fields in the mandatory common part whose codings are specific to a given message type. These fields are the "Number of Records" field and the "Flags" field.

Number of Records
The Number of Records field of the mandatory common part for the CA message gives the number of CSAS Records appended to the CA message mandatory part.

Flags
The Flags field of the mandatory common part for the CA message has the following format:



M
This bit is part of the negotiation process for the cache alignment. When this bit is set then the sender of the CA message is indicating that it wishes to lead the alignment process. This bit is the "Master/Slave bit".

I
When set, this bit indicates that the sender of the CA message believes that it is in a state where it is negotiating for the status of master or slave. This bit is the "Initialization bit".

O
This bit indicates that the sender of the CA message has more CSAS records to send. This implies that the cache alignment process must continue. This bit is the "mOre bit" despite its dubious name.

All other fields of the mandatory common part are coded as described in Section B.2.0.1.

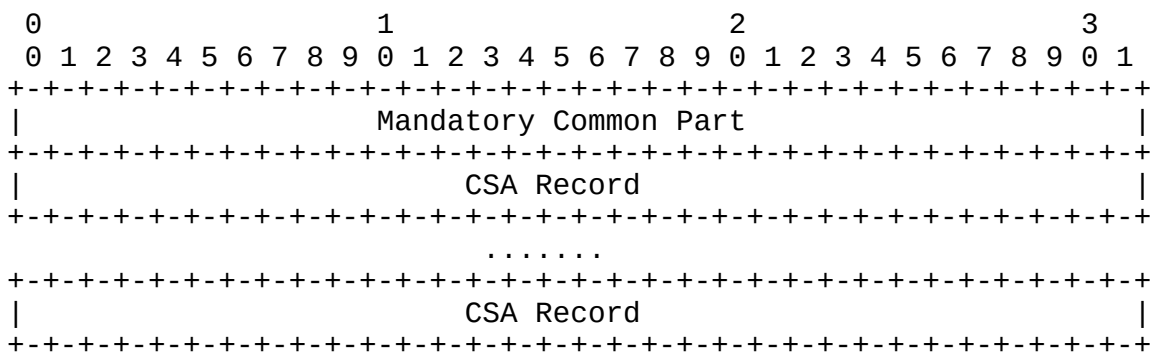
CSAS record
The CA message appends CSAS records to the end of its mandatory part. These CSAS records are NOT embedded in CSA records. See Section B.2.0.2 for details on CSAS records.

B.2.2 Cache State Update Request (CSU Request)

The Cache State Update Request (CSU Request) message is used to update the state of cache entries in servers which are directly connected to the server sending the message. A CSU Request message is sent from one server (the LS) to directly connected server (the DCS) when the LS observes changes in the state of one or more cache

entries. An LS observes such a change in state by either receiving a CSU request which causes an update to the LS's database or by observing a change of state of a cached entry originated by the LS. The change in state of a cache entry is noted in a CSU message by appending a "Cache State Advertisement" (CSA) record to the end of the mandatory part of the CSU Request as shown below.

The CSU Request message type code is 2. The CSU Request message mandatory part format is as follows:



Mandatory Common Part

The mandatory common part is described in detail in Section B.2.0.1. There are two fields in the mandatory common part whose codings are specific to a given message type. These fields are the "Number of Records" field and the "Flags" field.

Number of Records

The Number of Records field of the mandatory common part for the CSU Request message gives the number of CSA Records appended to the CSU Request message mandatory part.

Flags

Currently, there are no flags defined for the Flags field of the mandatory common part for the CSU Request message.

All other fields of the mandatory common part are coded as described in Section B.2.0.1.

CSA Record

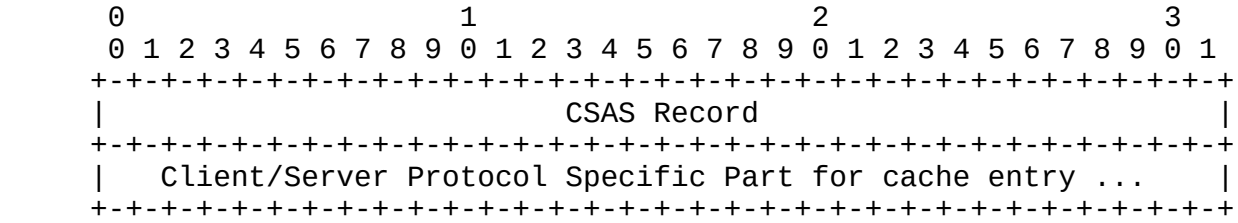
See Section B.2.2.1.

B.2.2.1 Cache State Advertisement Record (CSA record)

CSA records contain the information necessary to relate the current state of a cache entry in an SG to the servers being synchronized. CSA records contain a CSAS Record header and a client/server protocol specific part. The CSAS Record includes enough information for SCSP to look into the client/server database for the appropriate database cache entry and then compare the "newness" of the summary against the "newness" of the cached entry. If the information contained in the CSA is more new than the cached entry of the receiving server then the cached entry is updated accordingly with the contents of the CSA Record. The client/server protocol specific part of the CSA Record is documented separately for each such protocol. Examples of the protocol specific parts for NHRP and ATMARP are shown in [8] and [9] respectively.

The amount of information carried by a specific CSA record may exceed the size of a link layer PDU. Hence, such CSA records MUST be fragmented across a number of CSU Request messages. The method by which this is done, is client/server protocol specific and is documented in the appropriate protocol specific document.

The content of a CSA record is as follows:



CSAS Record
See Section B.2.0.2 for rules and format for filling out a CSAS Record when it is "embedded" in a CSA Record.

Client/Server Protocol Specific Part for cache entry
This field contains the fields which are specific to the protocol specific portion of SCSP processing. The particular set of fields are defined in separate documents for each protocol user of SCSP. The Protocol ID, which identifies which protocol is using SCSP in the given packet, is located in the mandatory part of the message.

B.2.3 Cache State Update Reply (CSU Reply)

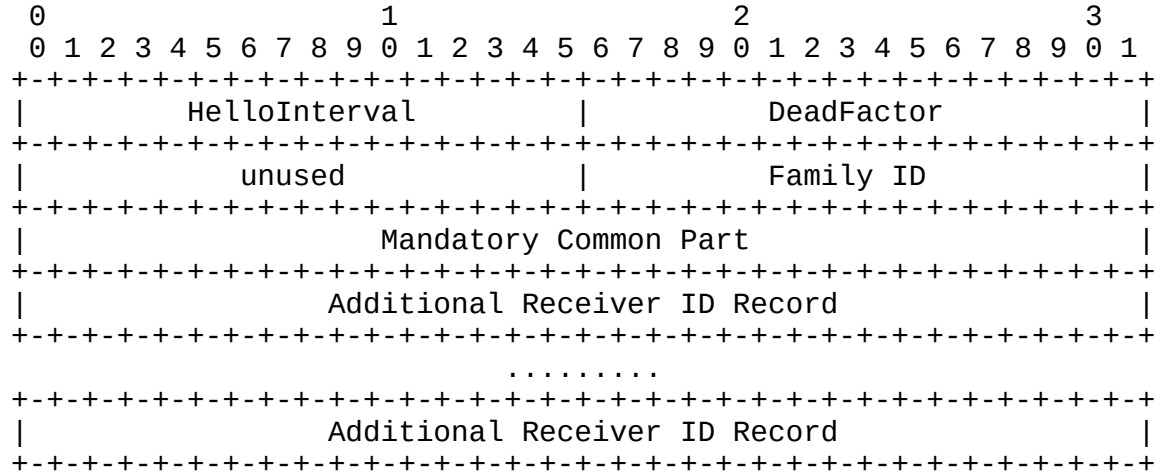
The Cache State Update Reply (CSU Reply) message is sent from a DCS to an LS to acknowledge one or more CSA records which were received in a CSU Request. Reception of a CSA record in a CSU Request is acknowledged by including a CSAS record in the CSU Reply which corresponds to the CSA record being acknowledged. The CSU Reply message is the same in format as the CSU Request message except for the following: the type code is 3, only CSAS Records (rather than CSA records) are returned, and only those CSAS Records for which CSA Records are being acknowledged are returned. This implies that a given LS sending a CSU Request may not receive an acknowledgment in a single CSU Reply for all the CSA Records included in the CSU Request.

B.2.4 Cache State Update Solicit Message (CSUS message)

This message allows one server (LS) to solicit the entirety of CSA record data stored in the cache of a directly connected server (DCS). The DCS responds with CSU Request messages containing the appropriate CSA records. The CSUS message type code is 4. The CSUS message format is the same as that of the CSU Reply message. CSUS messages solicit CSU Requests from only one server (the one identified by the Receiver ID in the Mandatory Part of the message).

B.2.5 Hello:

The Hello message is used to check connectivity between the sending server (the LS) and one of its directly connected neighbor servers (the DCSs). The Hello message type code is 5. The Hello message mandatory part format is as follows:



HelloInterval
The hello interval advertises the time between sending of consecutive Hello Messages. If the LS does not receive a Hello message from the DCS (which contains the LSID as a Receiver ID) within the HelloInterval advertised by the DCS then the DCS's Hello is considered to be late. Also, the LS MUST send its own Hello message to a DCS within the HelloInterval which it advertised to the DCS in the LS's previous Hello message to that DCS (otherwise the DCS would consider the LS's Hello to be late).

DeadFactor
This is a multiplier to the HelloInterval. If an LS does not receive a Hello message which contains the LS's LSID as a Receiver ID within the interval HelloInterval*DeadFactor from a given DCS, which advertised the HelloInterval and DeadFactor in a previous Hello message, then the LS MUST consider the DCS to be stalled; at this point, one of two things MUST happen: 1) if the LS has received any Hello messages from the DCS during this time then the LS transitions the corresponding HFSM to the Unidirectional State; otherwise, 2) the LS transitions the corresponding HFSM to the Waiting State.

Family ID
This is an opaque bit string which is used to refer to an aggregate of Protocol ID/SGID pairs. Only a single HFSM is run for all Protocol ID/SGID pairs assigned to a Family ID. Thus, there is a one to many mapping between the single HFSM and the CAFSMs corresponding to each of the Protocol ID/SGID pairs. This might have the net effect of substantially reducing HFSM maintenance traffic. See the protocol specific SCSP documents for further details.

Mandatory Common Part
The mandatory common part is described in detail in Section B.2.0.1. There are two fields in the mandatory common part whose codings are specific to a given message type. These fields are the "Number of Records" field and the "Flags" field.

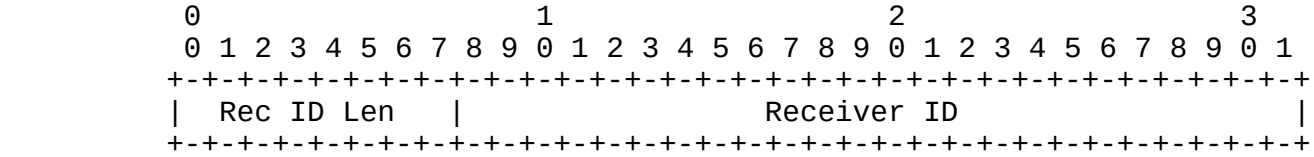
Number of Records
The Number of Records field of the mandatory common part for the Hello message contains the number of "Additional Receiver ID" records which are included in the Hello. Additional Receiver ID records contain a length field and a Receiver ID field. Note that the count in "Number of Records" does NOT include the Receiver ID which is included in the Mandatory Common Part.

Flags
Currently, there are no flags defined for the Flags field of the mandatory common part for the Hello message.

All other fields of the mandatory common part are coded as described in Section B.2.0.1.

Additional Receiver ID Record
This record contains a length field followed by a Receiver ID. Since it is conceivable that the length of a given Receiver ID may vary even within an SG, each additional Receiver ID heard (beyond the first one) will have both its length in bytes and value encoded in an "Additional Receiver ID Record". Receiver IDs are IDs of a DCS from which the LS has heard a recent Hello (i.e., within DeadFactor*HelloInterval as advertised by the DCS in a previous Hello message).

The format for this record is as follows:



If the LS has not heard from any DCS then the LS sets the Hello message fields as follows: Recvr ID Len is set to zero and no storage is allocated for the Receiver ID in the Common Mandatory Part, "Number of Records" is set to zero, and no storage is allocated for "Additional Receiver ID Records".

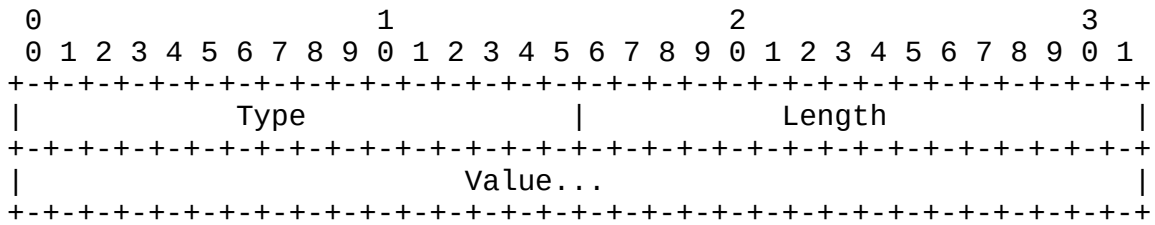
If the LS has heard from exactly one DCS then the LS sets the Hello message fields as follows: the Receiver ID of the DCS which was heard and the length of that Receiver ID are encoded in the Common Mandatory Part, "Number of Records" is set to zero, and no storage is allocated for "Additional Receiver ID Records".

If the LS has heard from two or more DCSs then the LS sets the Hello message fields as follows: the Receiver ID of the first DCS which was heard and the length of that Receiver ID are encoded in the Common Mandatory Part, "Number of Records" is set to the number of "Additional" DCSs heard, and for each additional DCS an "Additional Receiver ID Record" is formed and appended to the end of the Hello message.

B.3 Extensions Part

The Extensions Part, if present, carries one or more extensions in {Type, Length, Value} triplets.

Extensions have the following format:



Type
The extension type code (see below).

Length
The length in octets of the value (not including the Type and Length fields; a null extension will have only an extension header and a length of zero).

When extensions exist, the extensions part is terminated by the End of Extensions extension, having Type = 0 and Length = 0.

Extensions may occur in any order but any particular extension type may occur only once in an SCSP packet. An LS MUST NOT change the order of extensions.

B.3.0 The End Of Extensions

Type = 0
Length = 0

When extensions exist, the extensions part is terminated by the End Of Extensions extension.

B.3.1 SCSP Authentication Extension

Type = 1 Length = variable

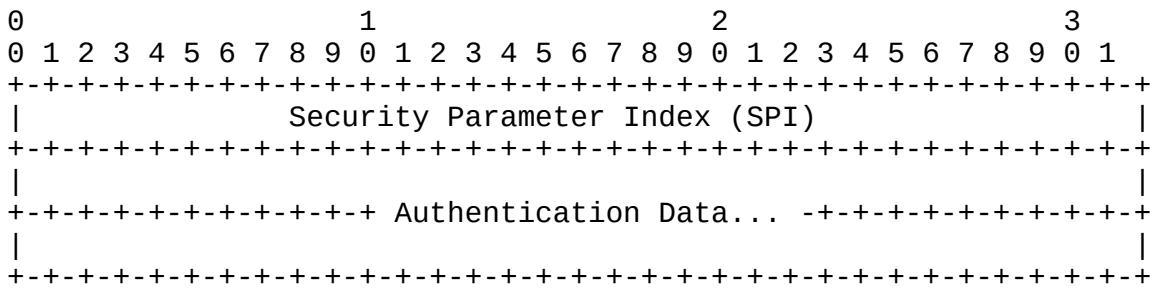
The SCSP Authentication Extension is carried in SCSP packets to convey the authentication information between an LS and a DCS in the same SG.

Authentication is done pairwise on an LS to DCS basis; i.e., the authentication extension is generated at each LS. If a received packet fails the authentication test then an "abnormal event" has occurred. The packet is discarded and this event is logged.

The presence or absence of authentication is a local matter.

B.3.1.1 Header Format

The authentication header has the following format:



Security Parameter Index (SPI) can be thought of as an index into a table that maintains the keys and other information such as hash algorithm. LS and DCS communicate either off-line using manual keying or online using a key management protocol to populate this table. The receiving SCSP entity always allocates the SPI and the parameters associated with it.

The authentication data field contains the MAC (Message Authentication Code) calculated over the entire SCSP payload. The length of this field is dependent on the hash algorithm used to calculate the MAC.

B.3.1.2 Supported Hash Algorithms

The default hash algorithm to be supported is HMAC-MD5-128 [11]. HMAC is safer than normal keyed hashes. Other hash algorithms MAY be supported by def.

IANA will assign the numbers to identify the algorithm being used as described in Section C.

B.3.1.3 SPI and Security Parameters Negotiation

SPI’s can be negotiated either manually or using an Internet Key Management protocol. Manual keying MUST be supported. The following parameters are associated with the tuple <SPI, DCS ID>- lifetime, Algorithm, Key. Lifetime indicates the duration in seconds for which

the key is valid. In case of manual keying, this duration can be infinite. Also, in order to better support manual keying, there may be multiple tuples active at the same time (DCS ID being the same).

Any Internet standard key management protocol MAY be used to negotiate the SPI and parameters.

B.3.1.4 Message Processing

At the time of adding the authentication extension header, LS looks up in a table to fetch the SPI and the security parameters based on the DCS ID. If there are no entries in the table and if there is support for key management, the LS initiates the key management protocol to fetch the necessary parameters. The LS then calculates the hash by zeroing authentication data field before calculating the MAC on the sending end. The result replaces in the zeroed authentication data field. If key management is not supported and authentication is mandatory, the packet is dropped and this information is logged.

When receiving traffic, an LS fetches the parameters based on the SPI and its ID. The authentication data field is extracted before zeroing out to calculate the hash. It computes the hash on the entire payload and if the hash does not match, then an "abnormal event" has occurred.

B.3.1.5 Security Considerations

It is important that the keys chosen are strong as the security of the entire system depends on the keys being chosen properly and the correct implementation of the algorithms.

SCSP has a peer to peer trust model. It is recommended to use an Internet standard key management protocol to negotiate the keys between the neighbors. Transmitting the keys in clear text, if other methods of negotiation is used, compromises the security completely.

Data integrity covers the entire SCSP payload. This guarantees that the message was not modified and the source is authenticated as well. If authentication extension is not used or if the security is compromised, then SCSP servers are liable to both spoofing attacks, active attacks and passive attacks.

There is no mechanism to encrypt the messages. It is assumed that a standard layer 3 confidentiality mechanism will be used to encrypt and decrypt messages. As integrity is calculated on an SCSP message

and not on each record, there is an implied trust between all the servers in a domain. It is recommend to use the security extension between all the servers in a domain and not just a subset servers.

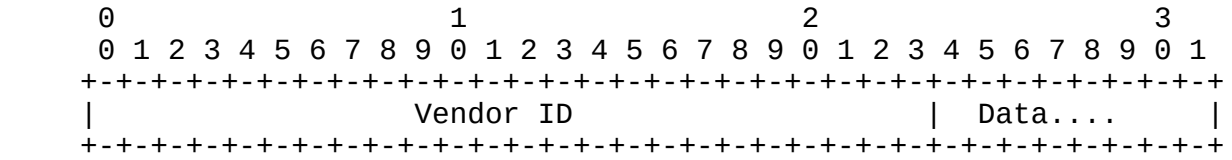
Any SCSP server is susceptible to Denial of Service (DOS) attacks. A rouge host can inundate its neighboring SCSP server with SCSP packets. However, if the authentication option is used, SCSP databases will not become corrupted, as the bogus packets will be discarded when the authentication check fails.

Due to the pairwise authentication model of SCSP, the information received from any properly authenticated server is trusted and propagated throughout the server group. Consequently, if security of any SCSP server is compromised, the entire database becomes vulnerable to curruption originating from the compromised server.

B.3.2 SCSP Vendor-Private Extension

Type = 2
Length = variable

The SCSP Vendor-Private Extension is carried in SCSP packets to convey vendor-private information between an LS and a DCS in the same SG and is thus of limited use. If a finer granularity (e.g., CSA record level) is desired then then given client/server protocol specific SCSP document MUST define such a mechanism. Obviously, however, such a protocol specific mechanism might look exactly like this extension. The Vendor Private Extension MAY NOT appear more than once in an SCSP packet for a given Vendor ID value.



Vendor ID
802 Vendor ID as assigned by the IEEE [7].

Data
The remaining octets after the Vendor ID in the payload are vendor-dependent data.

If the receiver does not handle this extension, or does not match the Vendor ID in the extension then the extension may be completely ignored by the receiver.

C. IANA Considerations

Any and all requests for value assignment from the various number spaces described in this document require proper documentation. Possible forms of documentation include, but are not limited to, RFCs or the product of another cooperative standards body (e.g., the MPOA and LANE subworking group of the ATM Forum). Other requests may also be accepted, under the advice of a "designated expert". (Contact the IANA for the contact information of the current expert.)

References

[1] Laubach, M., and J. Halpern, "Classical IP and ARP over ATM", Laubach, RFC 2225, April 1998.

[2] Luciani, J., Katz, D., Piscitello, D., Cole, B., and N. Doraswamy, "NMBA Next Hop Resolution Protocol (NHRP)", RFC 2332, April 1998.

[3] Moy, J., "OSPF Version 2", STD 54, RFC 2328, April 1998.

[4] "P-NNI V1", Dykeman, Goguen, 1996.

[5] Armitage, G., "Support for Multicast over UNI 3.0/3.1 based ATM Networks", RFC 2022, November 1996.

[6] Keene, "LAN Emulation over ATM Version 2 - LNNI specification", btd-lane-lnni-02.08

[7] Reynolds, J., and J. Postel, "Assigned Numbers", STD 2, RFC 1700, October 1994.

[8] Luciani, J., "A Distributed NHRP Service Using SCSP", RFC 2335, April 1998.

[9] Luciani, J., "A Distributed ATMARP Service Using SCSP", Work In Progress.

[10] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[11] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed Hashing for Message Authentication", RFC 2104, February 1997.

Acknowledgments

This memo is a distillation of issues raised during private discussions, on the IP-ATM mailing list, and during the Dallas IETF (12/95). Thanks to all who have contributed but particular thanks to following people (in no particular order): Barbara Fox of Harris and Jeffries; Colin Verrilli of IBM; Raj Nair, and Matthew Doar of Ascom Nexion; Andy Malis of Cascade; Andre Fredette of Bay Networks; James Watt of Newbridge; and Carl Marcinik of Fore.

Authors' Addresses

James V. Luciani
Bay Networks, Inc.
3 Federal Street, BL3-03
Billerica, MA 01821

Phone: +1-978-916-4734
EMail: luciani@baynetworks.com

Grenville Armitage
Bell Labs Lucent Technologies
101 Crawfords Corner Road
Holmdel, NJ 07733

Phone: +1 201 829 2635
EMail: gja@lucent.com

Joel M. Halpern
Newbridge Networks Corp.
593 Herndon Parkway
Herndon, VA 22070-5241

Phone: +1-703-708-5954
EMail: jhalpern@Newbridge.COM

Naganand Doraswamy
Bay Networks, Inc.
3 Federal St, BL3-03
Billerice, MA 01821

Phone: +1-978-916-1323
EMail: naganand@baynetworks.com

Full Copyright Statement