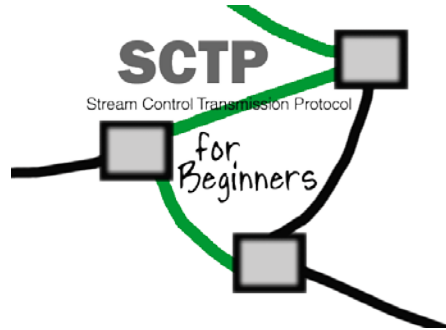


SCTP for Beginners

Erwin P. Rathgeb



Introduction

SCTP is a reliable transport protocol operating on top of a potentially unreliable connectionless packet service such as IP. It offers acknowledged error-free non-duplicated transfer of datagrams (messages). Detection of data corruption, loss of data and duplication of data is achieved by using checksums and sequence numbers. A selective retransmission mechanism is applied to correct loss or corruption of data.

Applicability

Originally, SCTP was designed to provide a general-purpose transport protocol for message-oriented applications, as is needed for the transportation of signalling data. It has been designed by the IETF SIGTRAN working group, which has released the SCTP standard draft document (RFC2960) in October 2000. Its design includes appropriate congestion avoidance behavior and resistance to flooding and masquerade attacks.

The decisive difference to TCP is multihoming and the concept of several streams within a connection (which will be referred to as association in the rest of these documents). Where in TCP a stream is referred to as a sequence of bytes, an SCTP stream represents a sequence of messages (and these may be very short or long).

SCTP can be used as the transport protocol for applications where monitoring and detection of loss of session is required. For such applications, the SCTP path/session failure detection mechanisms, especially the heartbeat, will actively monitor the connectivity of the session.

An SCTP association generally looks like this, so the services of SCTP are naturally at the same layer as TCP or UDP services:



SCTP Node A |<----- Network transport ----->| SCTP Node B

Diagram Showing the Concept of an SCTP Association

SCTP for Beginners

Erwin P. Rathgeb

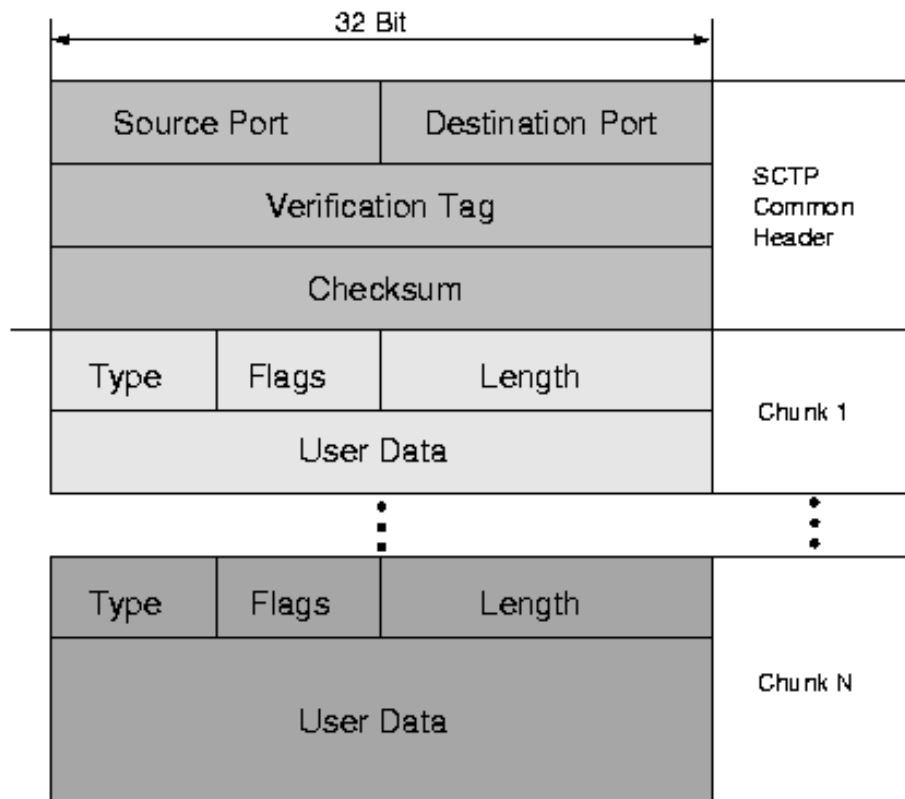
Overview

The following section SCTP Packets will explain shortly the structure of network packets as sent by an SCTP endpoint, as well as the different types of data that can be sent. Section SCTP States will then detail the way an association is established or taken down and present some typical message flow scenarios, explaining the states of the respective endpoints. In section SCTP Data Exchange we will present the rules for reliable transmission, selective acknowledgement and flow and congestion control. Then we will look at another very interesting SCTP feature: Multihoming. The requirements and implications of the RFC2960 are mentioned, and some examples presented.

Finally in the last sections we will present the concept of SCTP streams as opposed to TCP bytestreams, talk about the API that is described in RFC2960 , and explain the general SCTP terminology where needed. For all those that are interested in learning more about SCTP, our Link Section gives interesting pointers and links related to SCTP, applications, papers and much more.

SCTP Packets

The protocol data units (PDU) of SCTP are called SCTP packets. If SCTP runs over IP (as described in RFC2960), an SCTP packet forms the payload of an IP packet. An SCTP packet is composed of a common header and chunks. Multiple chunks may be multiplexed into one packet up to the Path-MTU size. A chunk may contain either control information or user data.



An SCTP-Protocol Data Unit With Several Chunks

The Common Header

The common header consists of 12 bytes. For the identification of an association, SCTP uses the same port concept as TCP and UDP. For the detection of transmission errors, each SCTP packet is protected by a 32 bit checksum (Adler-32 algorithm), which is more robust than the 16 bit checksum of TCP and UDP. SCTP packets with an invalid checksum are silently discarded. The common

SCTP for Beginners

Erwin P. Rathgeb

header also contains a 32 bit value called verification tag. The verification tag is association specific, and exchanged between the endpoints at association startup. So there are two tag values used in one association. See section SCTP states for detailed information on tags.

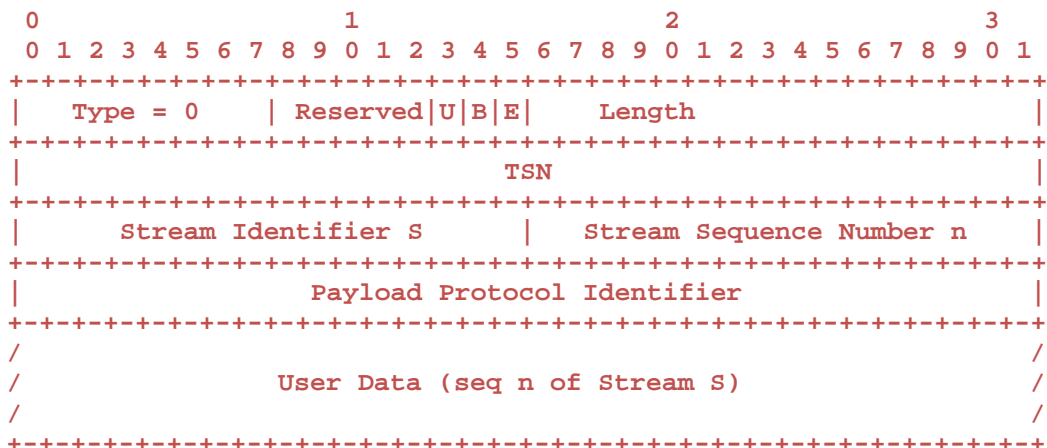
Chunks

Each chunk begins with a chunk type field, which is used to distinguish data chunks and different types of control chunks, followed by chunk specific flags and a chunk length field needed because chunks have a variable length. The value field contains the actual payload of the chunk. So far there are 13 chunk types defined for standard use. They are listed here. For the sake of simplicity, their definitions have been copied here from RFC2960 :

ID Value	Chunk Type
0	- Payload Data (DATA)
1	- Initiation (INIT)
2	- Initiation Acknowledgement (INIT ACK)
3	- Selective Acknowledgement (SACK)
4	- Heartbeat Request (HEARTBEAT)
5	- Heartbeat Acknowledgement (HEARTBEAT ACK)
6	- Abort (ABORT)
7	- Shutdown (SHUTDOWN)
8	- Shutdown Acknowledgement (SHUTDOWN ACK)
9	- Operation Error (ERROR)
10	- State Cookie (COOKIE ECHO)
11	- Cookie Acknowledgement (COOKIE ACK)
12	- Reserved for Explicit Congestion Notification Echo (ECNE)
13	- Reserved for Congestion Window Reduced (CWR)
14	- Shutdown Complete (SHUTDOWN COMPLETE)
15 to 62	- reserved by IETF
63	- IETF-defined Chunk Extensions
64 to 126	- reserved by IETF
127	- IETF-defined Chunk Extensions
128 to 190	- reserved by IETF
191	- IETF-defined Chunk Extensions
192 to 254	- reserved by IETF
255	- IETF-defined Chunk Extensions

Payload Data

The following format MUST be used for the DATA chunk:

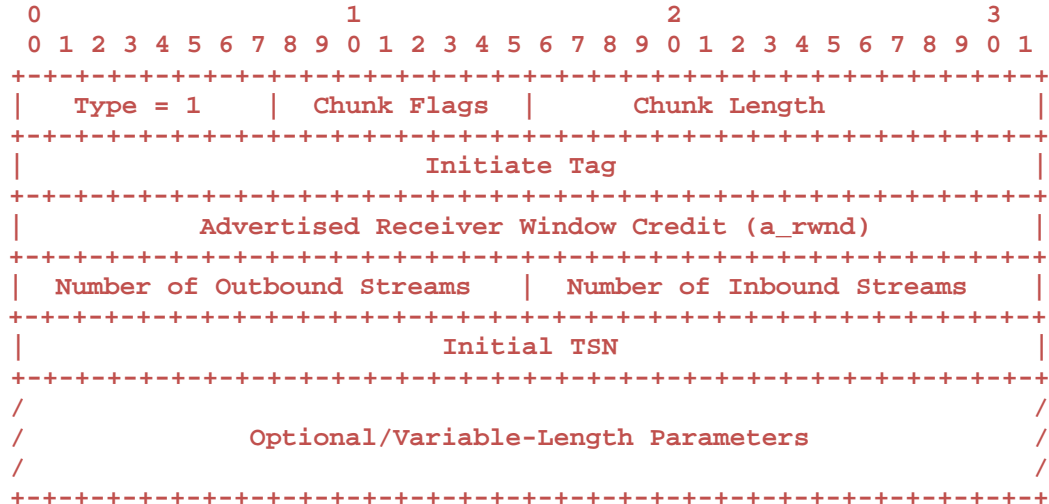


SCTP for Beginners

Erwin P. Rathgeb

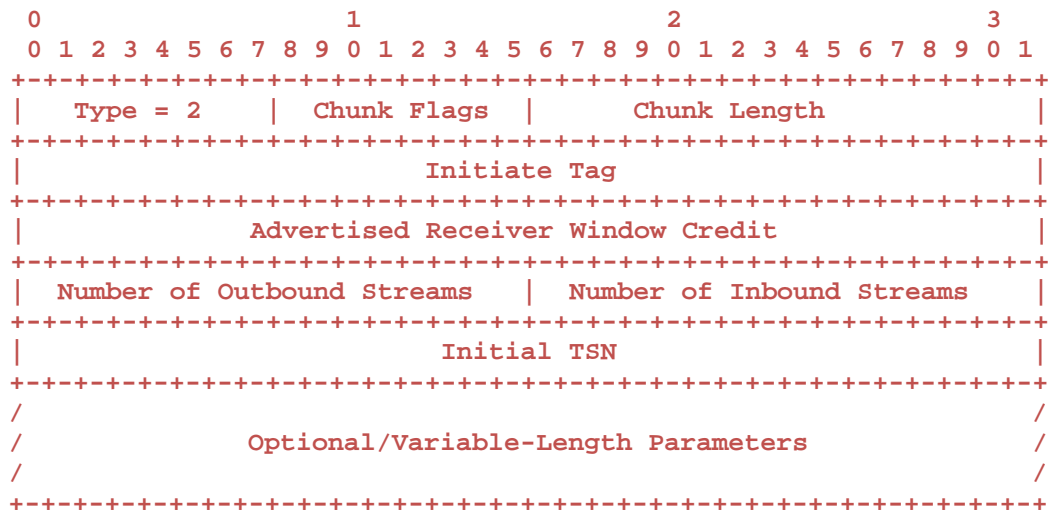
Initiation (INIT)

This chunk is used to initiate a SCTP association between two endpoints. The format of the INIT chunk is shown below:



Initiation Acknowledgement (INIT ACK)

The INIT ACK chunk is used to acknowledge the initiation of an SCTP association. The parameter part of INIT ACK is formatted similarly to the INIT chunk. It uses two extra variable parameters: The State Cookie and the Unrecognized Parameter:



Selective Acknowledgement (SACK)

This chunk is sent to the peer endpoint to acknowledge received DATA chunks and to inform the peer endpoint of gaps in the received subsequences of DATA chunks as represented by their TSNs. The SACK MUST contain the Cumulative TSN Ack and Advertised Receiver Window Credit (a_rwnd) parameters.

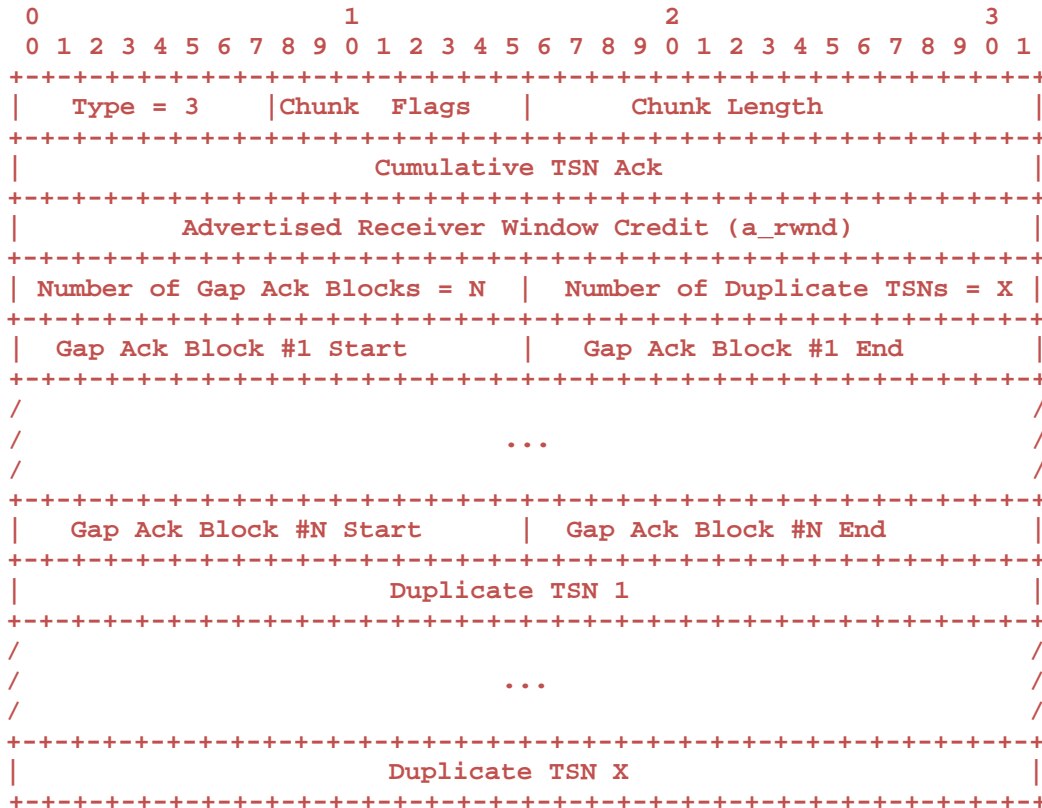
By definition, the value of the Cumulative TSN Ack parameter is the last TSN received before a break in the sequence of received TSNs occurs; the next TSN value following this one has not yet been

SCTP for Beginners

Erwin P. Rathgeb

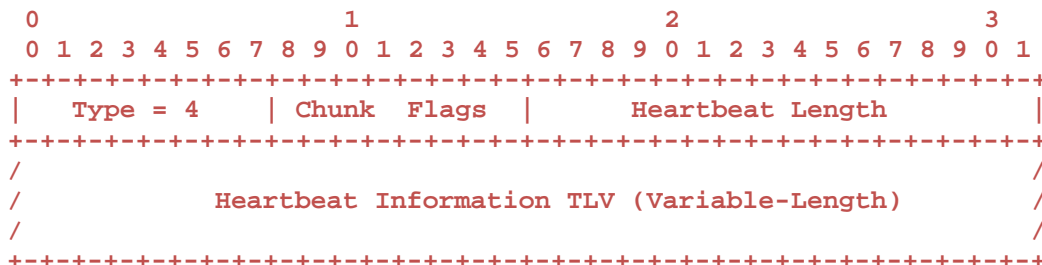
received at the endpoint sending the SACK. This parameter therefore acknowledges receipt of all TSNs less than or equal to its value.

The SACK also contains zero or more Gap Ack Blocks. Each Gap Ack Block acknowledges a subsequence of TSNs received following a break in the sequence of received TSNs. By definition, all TSNs acknowledged by Gap Ack Blocks are greater than the value of the Cumulative TSN Ack.



Heartbeat Request (HEARTBEAT)

An endpoint should send this chunk to its peer endpoint to probe the reachability of a particular destination transport address defined in the present association. The parameter field contains the Heartbeat Information which is a variable length opaque data structure understood only by the sender.

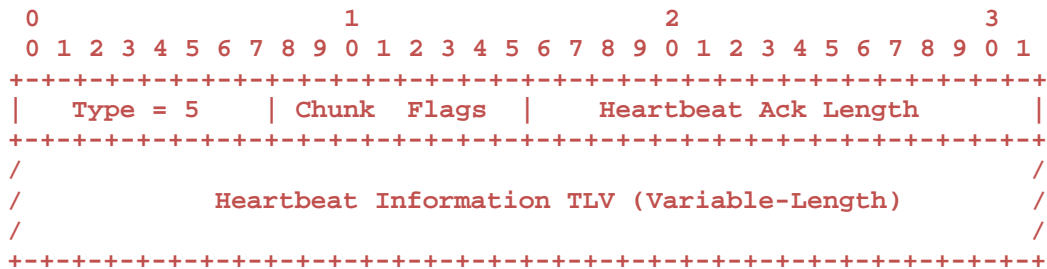


Heartbeat Acknowledgement (HEARTBEAT ACK)

An endpoint should send this chunk to its peer endpoint as a response to a HEARTBEAT chunk (see Section 8.3). A HEARTBEAT ACK is always sent to the source IP address of the IP datagram containing the HEARTBEAT chunk to which this ack is responding. The parameter field contains a variable length opaque data structure.

SCTP for Beginners

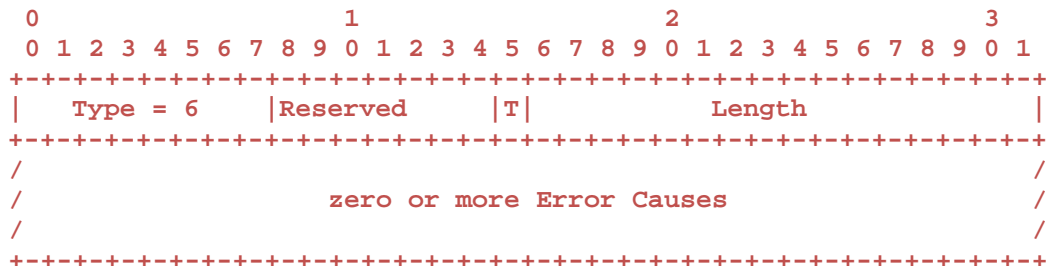
Erwin P. Rathgeb



Abort Association (ABORT)

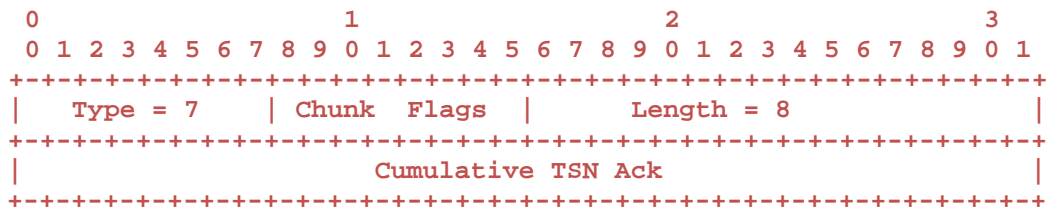
The ABORT chunk is sent to the peer of an association to close the association. The ABORT chunk may contain Cause Parameters to inform the receiver the reason of the abort. DATA chunks MUST NOT be bundled with ABORT. Control chunks (except for INIT, INIT ACK and SHUTDOWN COMPLETE) MAY be bundled with an ABORT but they MUST be placed before the ABORT in the SCTP packet, or they will be ignored by the receiver.

If an endpoint receives an ABORT with a format error or for an association that doesn't exist, it MUST silently discard it. Moreover, under any circumstances, an endpoint that receives an ABORT MUST NOT respond to that ABORT by sending an ABORT of its own.



Shutdown Association (SHUTDOWN)

An endpoint in an association MUST use this chunk to initiate a graceful close of the association with its peer. This chunk has the following format.



Shutdown Acknowledgement (SHUTDOWN ACK)

This chunk MUST be used to acknowledge the receipt of the SHUTDOWN chunk at the completion of the shutdown process, see Section 9.2 for details. The SHUTDOWN ACK chunk has no parameters.

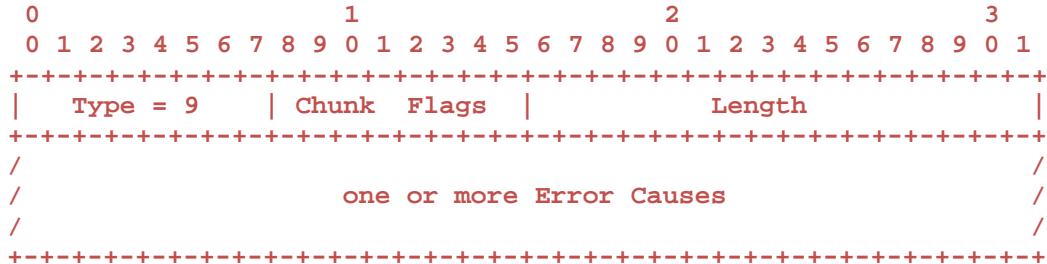


SCTP for Beginners

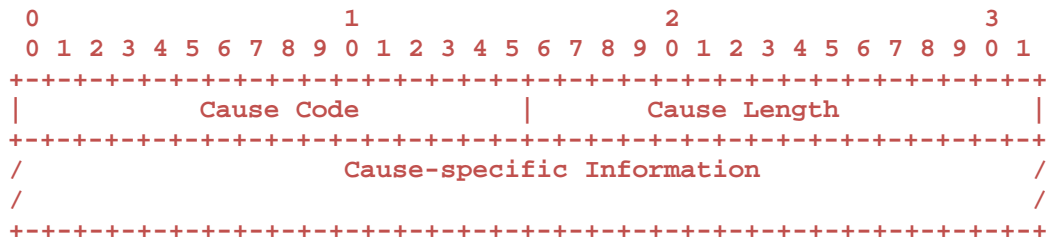
Erwin P. Rathgeb

Operation Error (ERROR)

An endpoint sends this chunk to its peer endpoint to notify it of certain error conditions. It contains one or more error causes. An Operation Error is not considered fatal in and of itself, but may be used with an ABORT chunk to report a fatal condition.



It contains as parameters a variable length field containing the type of error that has occurred:

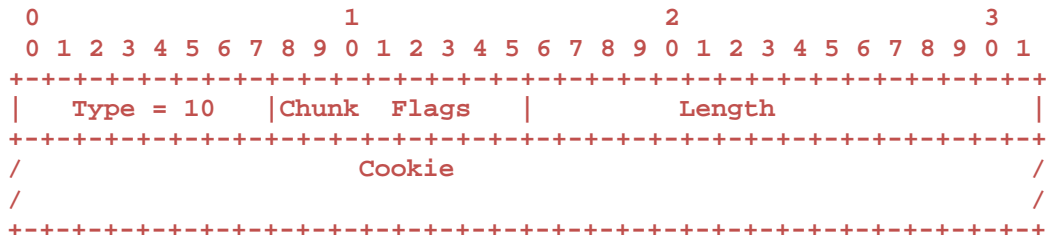


Each error cause may carry its own set of parameters. The error causes that have been defined are:

Cause Code Value	Cause Code
1	Invalid Stream Identifier
2	Missing Mandatory Parameter
3	Stale Cookie Error
4	Out of Resource
5	Unresolvable Address
6	Unrecognized Chunk Type
7	Invalid Mandatory Parameter
8	Unrecognized Parameters
9	No User Data
10	Cookie Received While Shutting Down

Cookie Echo (COOKIE ECHO)

This chunk is used only during the initialization of an association. It is sent by the initiator of an association to its peer to complete the initialization process. This chunk MUST precede any DATA chunk sent within the association, but MAY be bundled with one or more DATA chunks in the same packet.



SCTP for Beginners

Erwin P. Rathgeb

Cookie Acknowledgement (COOKIE ACK)

This chunk is used only during the initialization of an association. It is used to acknowledge the receipt of a COOKIE ECHO chunk. This chunk MUST precede any DATA or SACK chunk sent within the association, but MAY be bundled with one or more DATA chunks or SACK chunk in the same SCTP packet.

```

      0                               1                               2                               3
0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8  9  0  1
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|  Type = 11  |Chunk  Flags  |      Length = 4      |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Shutdown Complete (SHUTDOWN COMPLETE)

This chunk MUST be used to acknowledge the receipt of the SHUTDOWN ACK chunk at the completion of the shutdown process. The SHUTDOWN COMPLETE chunk has no parameters.

```

      0                               1                               2                               3
0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8  9  0  1
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|  Type = 14  |Reserved  |T|      Length = 4      |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

SCTP States

This section describes the states that an instance of the SCTP protocol enters while an association is established, and when it is taken down again. There are a couple of concepts that will need to be explained here. The initialization of an association is completed on both sides after the exchange of four messages. The passive side (let's call it server) does not allocate resources until the third of these messages has arrived and been validated. That is to ensure that the association setup request really does originate from the right peer (without the possibility of blind spoofing).

Normal Association Establishment

The Server Side

The server receives an association setup request (an INIT chunk) usually in the CLOSED state, and analyzes the data contained in that chunk. From that it generates all the values needed at its side to enter an established association, and generates a secure hash of these values and a secret key (e.g. with the MD5 or SHA-1 algorithms). The values are then put into the so-called COOKIE, along with the derived message authentication code (MAC). This COOKIE is returned to the sender of the INIT chunk in an INIT-ACK chunk. The server remains in the CLOSED state, and forgets all about the received INIT chunk.

Upon reception of a COOKIE-ECHO chunk (which contains a COOKIE data structure as parameter), the server unpacks the data contained in this COOKIE, and uses again the MAC contained therein to verify whether it was the originator of this COOKIE. If the MAC computes okay, it is a valid COOKIE that this server had created before, and the data values contained in the COOKIE are used to initialize the SCTP instance. The server will send a COOKIE-ACK to the client (optionally bundling a data chunk with this COOKIE-ACK chunk) and enter the ESTABLISHED state. It is then ready to accept data or send data chunks itself.

The Client Side

When an Upper Layer (ULP) wants to start an association, it calls the ASSOCIATE primitive (see SCTP API) and all necessary data structures are initialized in order to assemble an INIT chunk. This

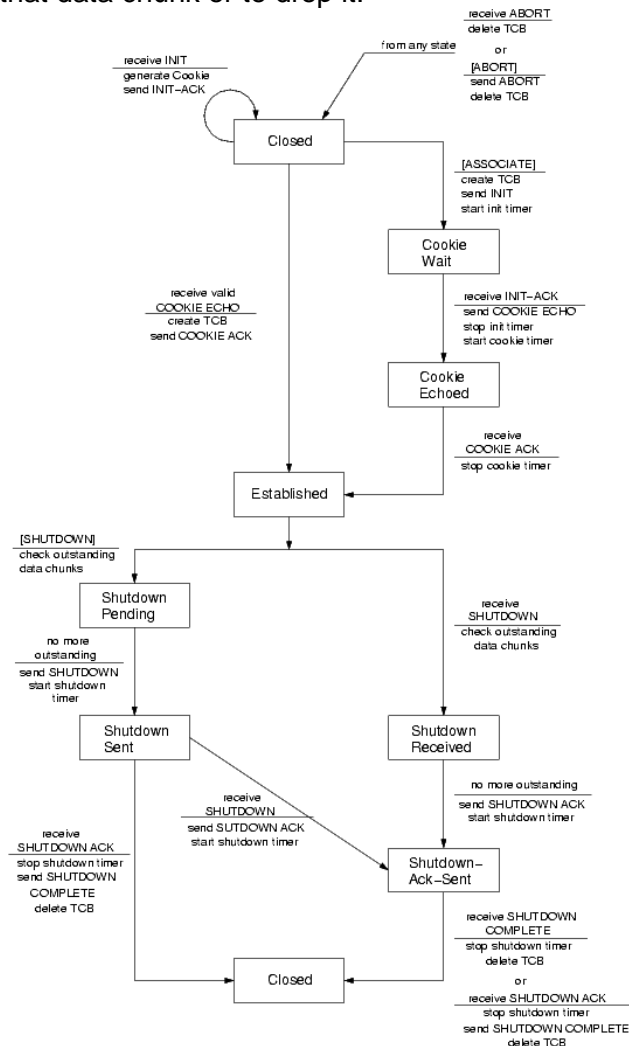
SCTP for Beginners

Erwin P. Rathgeb

INIT chunk is sent to one transport address (i.e. combination of IP-address and port) of a server. An init timer is started that triggers repetitive sending of the INIT chunk when it expires before an INIT-ACK chunk was received from the server. If after a configurable number of send events no INIT-ACK was received, an error is reported to the ULP and peer endpoint is reported unreachable. After the client has sent the first INIT chunk, it enters the COOKIE-WAIT state.

When the client receives an INIT-ACK chunk from the server in the COOKIE-WAIT state, it stops the init timer, assembles an COOKIE-ECHO chunk, puts the server's COOKIE from the received INIT-ACK chunk into the COOKIE-ECHO chunk, and returns it to the server. It then starts a cookie timer, that triggers repetitive sending of this COOKIE-ECHO, until a COOKIE-ACK is received from the server. After sending the first COOKIE-ECHO, the protocol instance enters the COOKIE-ECHOED state. If no COOKIE-ACK is received after a configurable number of COOKIE-ECHO send events, the server endpoint is reported unreachable.

After reception of a COOKIE-ACK chunk from the server, the client enters the ESTABLISHED state. Note that the COOKIE-ECHO may already be accompanied by a bundled data chunk. It is up to the server whether to accept that data chunk or to drop it.



Created by Andreas Jungmaier
with XFig

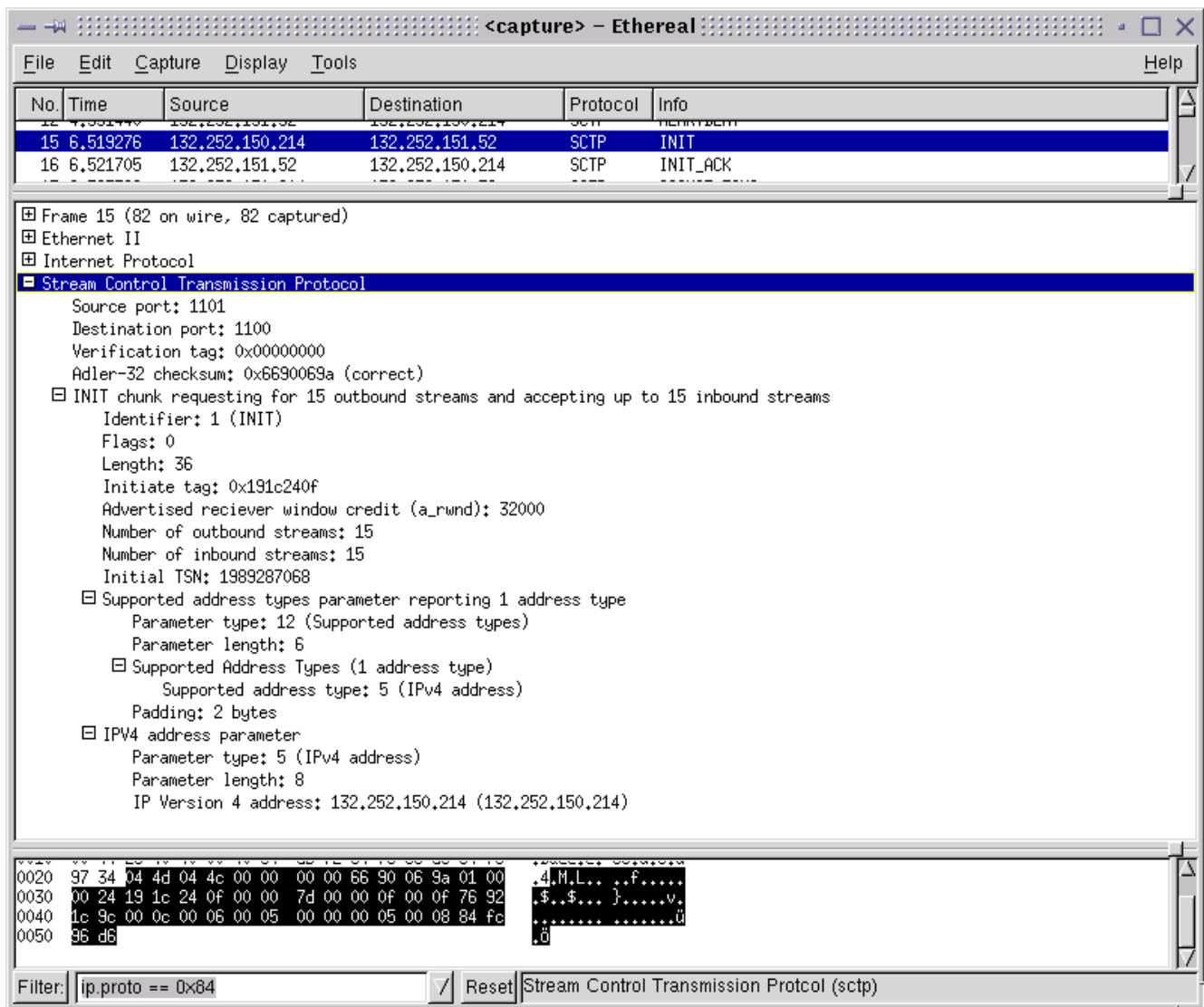
State Diagram of an SCTP Protocol Instance

SCTP for Beginners

Erwin P. Rathgeb

Real Life Examples - Initialization

The picture to the right displays an INIT chunk that is sent by a client to a server. The image has been taken from the program Ethereal (also see software list on the SCTP Links page). For details, please select the image (e.g. by clicking on it). It displays an INIT chunk that has been sent by a host with the IP address 132.252.150.214 to one with the address 132.252.151.52. The INIT chunk carries the initiation tag 0x191c240f and requests 15 outbound and 15 inbound streams. Along with it, it carries the "Supported Address Types"-TLV parameter as well as an address parameter with the IPv4 address of the client sending this chunk.



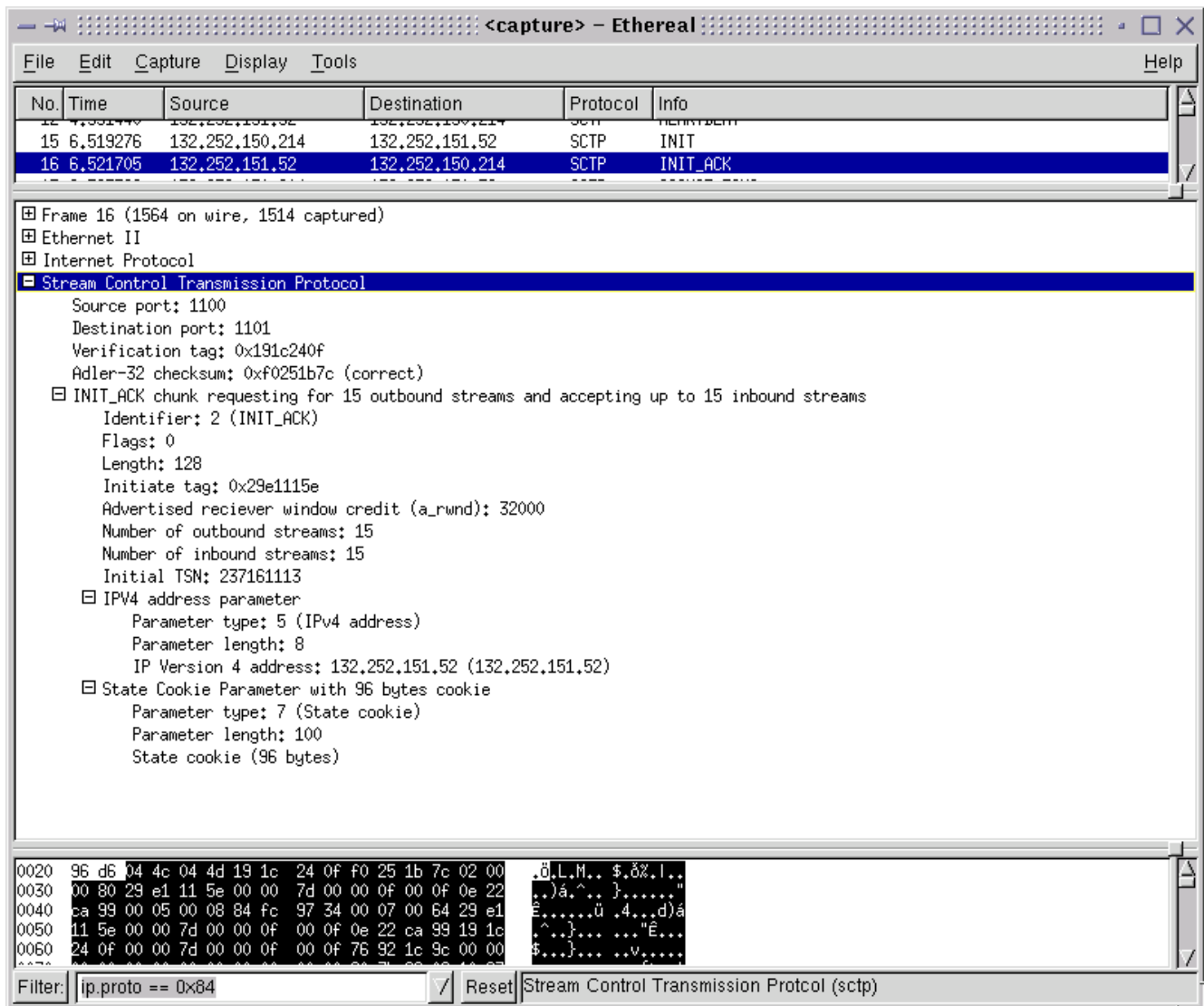
The response to an INIT chunk is a chunk that acknowledges it, and is called INIT-ACK chunk. Along with the INIT-ACK chunk, very similar parameters may be transmitted. In this example the server on IP address 132.252.151.52 returns his own tag 0x29e1115e in the INIT-ACK chunk, along with the number of inbound and outbound streams it is willing to accept (both are also 15 here).

Note that the tag in the SCTP common header is the same as the tag advertised by the client in its

SCTP for Beginners

Erwin P. Rathgeb

INIT chunk. Along with the INIT-ACK chunk a variable length parameter is sent, the so-called COOKIE data structure. The COOKIE must contain all the data that is needed by a server to initialize a new association. When this data structure is created, the server must not actually allocate any resources. Only when the very same COOKIE structure is returned to the server by the client that wants to establish the association, the server may actually allocate resources for the new association. To avoid possible tampering with any data contained in the COOKIE, this structure also contains a secure message authentication code (i.e. an MD5 or SHA-1 hash over the data structure and a secret key).



The client must then return the COOKIE data structure to the server in a so-called COOKIE-ECHO chunk. The COOKIE-ECHO chunk contains the very same variable length parameter as the INIT-ACK chunk. According to the RFC2960 the client may already transmit a DATA chunk after the COOKIE-ECHO chunk, which is not done here, though. As soon as the COOKIE-ECHO chunk is received by the server, it verifies the COOKIE structure (using the hash function and its secret key), and uses the data contained therein to initialize an appropriate association structure. It then notifies its user process with a COMMUNICATION-UP notification, and returns an acknowledgement to the client.

SCTP for Beginners

Erwin P. Rathgeb

The screenshot shows the Ethereal network protocol analyzer interface. The top menu bar includes File, Edit, Capture, Display, Tools, and Help. Below the menu is a table of captured packets:

No.	Time	Source	Destination	Protocol	Info
92	40.967037	132.252.150.214	132.252.151.52	SCTP	COOKIE_ECHO
94	40.970702	132.252.151.52	132.252.150.214	SCTP	COOKIE_ACK

The main pane displays the details for the selected packet (Frame 92):

- Frame 92 (146 on wire, 146 captured)
- Ethernet II
- Internet Protocol
- Stream Control Transmission Protocol
 - Source port: 1101
 - Destination port: 1100
 - Verification tag: 0x67dc95f9
 - Adler-32 checksum: 0x21941bed (correct)
 - COOKIE ECHO chunk containing a cookie of 96 bytes**
 - Identifier: 10 (COOKIE_ECHO)
 - Flags: 0
 - Length: 100
 - Cookie (96 bytes)

The bottom pane shows the raw packet data in hexadecimal and ASCII:

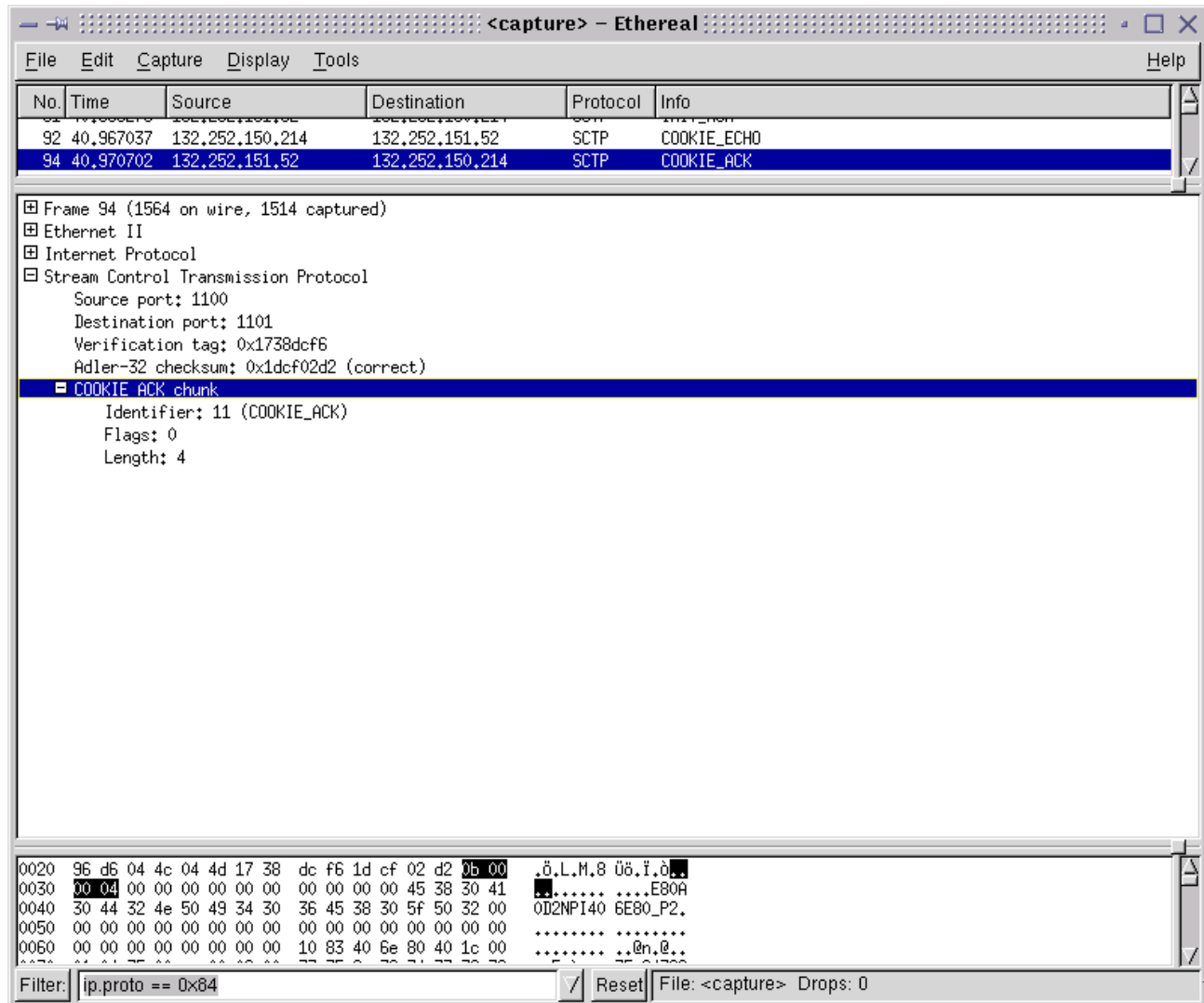
```
0020 97 34 04 4d 04 4c 67 dc 95 f9 21 94 1b ed 0a 00 .4.M.Lgü ,ù!..i..
0030 00 64 67 dc 95 f9 00 00 7d 00 00 0f 00 0f 53 1b .dgü.ü..}....S.
0040 64 c0 17 38 dc f6 00 00 7d 00 00 0f 00 0f 12 cd dA.8Ü6..}.....i
0050 58 f7 00 00 00 00 00 00 00 00 00 00 00 0d 02 8f.....*.....
0060 27 08 10 27 00 00 d1 41 63 77 9b a5 97 c3 75 74 ..NA cw.%Aut
```

The filter bar at the bottom shows the filter: `ip.proto == 0x84` and the status: `File: <capture> Drops: 0`.

The server sends back a very simple acknowledgement of the COOKIE reception to the client, a so-called COOKIE-ACK chunk. Along with this chunk, it may already transmit data chunks, which is, however, not done here in the example to the left. After the client has received a COOKIE-ACK chunk, it will notify its user process of the successful association establishment with a COMMUNICATION-UP notification. At this point, both sides know that the association is established and may start normal data transmission and heartbeating (see the SCTP Data Transmission and SCTP Multihoming pages).

SCTP for Beginners

Erwin P. Rathgeb



Association Termination

Both sides may decide to terminate an SCTP association for a number of reasons, and can do so practically at any time (provided they are in a state that is not CLOSED :-). There is the possibility of a graceful shutdown, ensuring that no data is lost, or hard termination, not taking care of the peer.

Graceful Termination of an Association

Upon receiving the SHUTDOWN primitive from its upper layer user process, an SCTP instance should stop accepting data from this process, and start sending a SHUTDOWN chunk, as soon as all of its outstanding data has been acknowledged. This process is secured by a timer, that repeats this process, should the SHUTDOWN be lost.

The peer will, at one point, receive the SHUTDOWN, and reply by sending a SHUTDOWN ACK chunk, as soon as all of its data has been acknowledged (also secured by a timer !). When the first peer (that started the shutdown procedure) receives the SHUTDOWN ACK, it will stop the timer, send a SHUTDOWN COMPLETE, and remove all data still belonging to that association, and enter the CLOSED state.

The peer that receives this SHUTDOWN COMPLETE chunk may then also remove all record of this

SCTP for Beginners

Erwin P. Rathgeb

association, and enter the CLOSED state. Should the last SHUTDOWN COMPLETE message be lost, the peer will repeat sending SHUTDOWN ACK chunks, until an error counter has been exceeded, which reports the other peer unreachable.

Aborting the Association

An endpoint may also decide to abort an existing association, taking into account that data still in flight may not be acknowledged, by sending an ABORT chunk to its peer endpoint. The sender MUST fill in the peer's Verification Tag in the outbound packet and MUST NOT bundle any DATA chunk with the ABORT.

The receiver of the ABORT does not reply, but validates the chunk, and removes the association, if the ABORT contains the correct tag value. If so, it also reports termination to its uppler layer process. Should the ABORT be lost, and the endpoint sending it terminate directly after sending it, it will take a rather long time to determine that the peer has gone (i.e. after the Peer Error Counter has been exceeded).

Special Cases

There are a number of special cases that need to be considered. These occur, when one endpoint is interrupted, restarted etc.

Sections 5.2.4 and 9 of RFC2960 describe the handling of these cases:

- Peer restart case, where the peer uses a new tag value.
- Cross initialization, where both peers send an INIT chunk at about the same time.
- Excessive delay of COOKIE chunks, etc.
- One peer trying to re-establish an association, while the other one tries to terminate it.
- SCTP Data Transmission

SCTP implementations must have flow and congestion control mechanisms according to RFC2960 , which ensures that SCTP can be introduced without problems in networks where TCP is in widespread use (see also Performance Evaluation of the Stream Control Transmission Protocol).

General Concepts

SCTP distinguishes different streams of messages within one SCTP association. This enables a delivery scheme where only the sequence of messages needs to be maintained per stream (partial in-sequence delivery) which reduces unnecessary head-of-line blocking between independent streams of messages (see also SCTP Streams).

SCTP operates on two levels:

- Within an association the reliable transfer of datagrams is assured by using a checksum, a sequence number and a selective retransmission mechanism. Without taking the initial sequence into account, every correctly received data chunk is delivered to a second, independent level.
- The second level realises a flexible delivery mechanism which is based on the notion of several independent streams of datagrams within an association. Chunks belonging to one or several streams may be bundled and transmitted in one SCTP packet provided they are not longer than the current path MTU.

SCTP for Beginners

Erwin P. Rathgeb

Detection of loss and duplication of data chunks is enabled by numbering all data chunks in the sender with the so-called Transport Sequence Number (TSN). The acknowledgements sent from the receiver to the sender are based on these sequence numbers.

Retransmissions are timer-controlled. The timer duration is derived from continuous measurements of the round trip delay. Whenever such a retransmission timer expires, (and congestion control allows transmissions) all non-acknowledged data chunks are retransmitted and the timer is started again doubling its initial duration (like in TCP).

When the receiver detects one or more gaps in the sequence of data chunks, each received SCTP packet is acknowledged by sending a Selective Acknowledgement (SACK) which reports all gaps. The SACK is contained in a specific control chunk. Whenever the sender receives four consecutive SACKs reporting the same data chunk missing, this data chunk is immediately retransmitted (fast retransmit). Most up-to-date operating systems already support a similar optional extension to TCP (see RFC 2018).

Flow Control

SCTP uses an end-to-end window based flow and congestion control mechanism similar to the one that is well known from TCP (see RFC 2581 - TCP Congestion Control). The receiver of data may control the rate at which the sender is sending by specifying an octet-based window size (the so-called Receiver Window), and returning this value along with all SACK chunks. The sender itself keeps a variable known as Congestion Window (short: CWND) that controls the maximum number of outstanding bytes (i.e. bytes that may be sent before they are acknowledged). Each received data chunk must be acknowledged, and the receiver may wait a certain time (usually 200 ms) before that is done. Should there be a larger number of SCTP packets with data received within this period of, every second SCTP packet containing data is to be acknowledged at once by sending a SACK chunk back to the sender.

Selective Acknowledgement

The acknowledgements carry all TSN numbers that have been received by one side with them. That is, there is a so called Cumulative TSN Ack value, that indicates all the data that has successfully been reassembled at the receivers side, and has either already been delivered to the receiving Upper Layer Process, or may readily be delivered upon request.

Moreover, there are so-called Gap Blocks that indicate that segments of data chunks have arrived, with some data chunks missing in between. Should some data chunks have been lost in the course of transmission, they will either be retransmitted after the transmission timer has expired, or after four SACK chunks have reported gaps with the same data chunk missing. In the latter case, the missing data is retransmitted via the Fast Retransmit mechanism.

In case a retransmission occurs which signals packet loss, the implementation must appropriately update congestion and flow control parameters.

Flow Control for Multihomed Endpoints

By default, all transmission is done to a previously selected address from the set of destination addresses, which is called the Primary Address. Retransmissions should be done on different paths, so that if one path is overloaded, retransmissions do not affect this path (unless the network topology is such that retransmissions hit the same point in the network where the data was dropped due to congestion). For certain network topologies that may have beneficial effects on overall throughput. Acknowledgements shall be sent to the transport address from which originated the data. Should the active path have a high number of failures and its error counter exceed a boundary, the

SCTP for Beginners

Erwin P. Rathgeb

SCTP implementation notifies its upper layer process that the path has become inactive. Then a new primary path may (and probably should) be chosen by the application (for more information on this, see SCTP Multihoming).

Congestion Control

The congestion control behaviour of an SCTP implementation according to RFC2960 may have an impact where timely delivery of messages is required (i.e. transport of signalling data). However, this ensures the proper behaviour of SCTP when it is introduced on a large scale into existing packet switched networks such as the Internet. The congestion control mechanisms for SCTP have been derived from RFC 2581 - TCP Congestion Control), and been adapted for multihoming. For each destination address (i.e. each possible path) a discrete set of flow and congestion control parameters is kept, such that from the point of view of the network, an SCTP association with a number of paths may behave similarly as the same number of TCP connections.

Slow Start and Congestion Avoidance

As in TCP, SCTP has two modes, Slow Start and Congestion Avoidance. The mode is determined by a set of congestion control variables, and as already mentioned, these are path specific. So, while the transmission to the primary path may be in the Congestion Avoidance mode, the implementation may still use Slow Start for the backup path(s).

For successfully delivered and acknowledged data the congestion window variable (CWND) is steadily increased, and once it exceeds a certain boundary (called Slow Start Threshold, SSTRESH), the mode changes from Slow Start to Congestion Avoidance. Generally, in Slow Start, the CWND is increased faster (roughly one MTU per received SACK chunk), and in Congestion Avoidance mode, it is only increased by roughly one MTU per Round Trip Time (RTT).

Events that trigger retransmission (timeouts or fast retransmission) cause the SSTHRESH to be cut down drastically, and reset the CWND (where a timeout causes a new Slow Start with CWND=MTU, and a Fast Retransmit sets CWND=SSTHRESH).

Path MTU Discovery

Since the Path MTU is such an important variable (influencing the congestion control), an SCTP implementations should keep a variable for the estimate of the current Maximum Transmission Unit (Path MTU) for each path. How this is done, is closer described in RFC 1191 - Path MTU Discovery and RFC 1981 - Path MTU Discovery for IP Version 6.

SCTP Multihoming

An essential property of SCTP is its support of multi-homed nodes, i.e. nodes which can be reached under several IP addresses. If the SCTP nodes and the according IP network are configured in such a way that traffic from one node to another travels on physically different paths if different destination IP address are used, associations become tolerant against physical network failures and other problems of that kind.

Address Management at Association Setup

If a client is multi-homed, it informs the server about all its IP addresses with the INIT chunk's address parameters. Thereby, the client is only required to know one IP address of the server because the server provides all its IP addresses to the client in the INIT-ACK chunk. SCTP is able to handle IP version 4 and IP version 6 addresses (even mixed). An SCTP instance regards each IP address of its peer as one "transmission path" towards this endpoint.

If no explicit IP addresses are contained in the INIT or INIT-ACK chunk, the source IP address of the

SCTP for Beginners

Erwin P. Rathgeb

IP packet which carries the SCTP datagram is used. This eases application of SCTP when Network Address Translation, (NAT), e.g. at the edge of large private IP networks, is involved. To facilitate this further, an additional optional feature has been introduced into the RFC2960 which allows the usage of host names in addition to or instead of IP addresses.

Path and Peer Monitoring

An SCTP instance monitors all transmission paths to the peer instance of an association. To this end, HEARTBEAT chunks are sent over all paths which are currently not used for the transmission of data chunks. Each HEARTBEAT chunk has to be acknowledged by a HEARTBEAT-ACK chunk.

Each path is assigned a state: it is either active or inactive. A path is active if it has been used in the recent past to transmit an (arbitrary) SCTP datagram which has been acknowledged by the peer. If transmissions on a certain path seem to fail repeatedly, the path is regarded as inactive.

The number of events where heartbeats were not acknowledged within a certain time, or retransmission events occurred is counted on a per association basis, and if a certain limit is exceeded (the value of which may be configurable), the peer endpoint is considered unreachable, and the association will be terminated.

Path Selection

At the set-up of an SCTP association, one of the IP addresses from the returned list is selected as initial primary path. Data chunks are transmitted over this primary transmission path by default. For retransmissions however, another active path may be selected, if one is available. To support the measurement of round trip delays, SACK chunks should be sent to the source address of the IP packet which carried the data chunk that triggered the SACK.

The users of SCTP are informed about the status (state and measurements) of a transmission path on request or when a transmission path changes its state. They may then instruct the local SCTP instance to use a new primary path.

SCTP Streams

While TCP couples the reliable transfer of user data and the strict order-of-transmission delivery of such data, SCTP separates the reliable transfer of datagrams from the delivery mechanism. This makes it possible to adapt protocol usage to the specific needs of the applications using SCTP. Some applications may only need partial ordering of datagrams while others might even be satisfied with a reliable transfer that does not guarantee any in-sequence maintenance at all.

Flexible Datagram Delivery

The user of SCTP may assign each datagram to one of several streams within an association. When an association is set-up, the number of available streams per direction is exchanged between the peer entities. Within each stream, SCTP assigns independent Stream Sequence Numbers (SSN) to the user datagrams. These numbers are used at the receiver to determine the sequence of delivery. SCTP performs in-sequence delivery per stream (for all datagrams which are not marked for out-of-order delivery). This mechanism avoids head-of-line blocking between independent streams of datagrams within one association. With TCP, this could only be achieved by setting-up several connections (one per stream) which would lead to additional cost and overhead.

As already mentioned, SCTP allows to mark datagrams for order-of-arrival delivery. This could be used for important messages which may by-pass others, like e.g. transaction abort messages of an application. If no sequence maintenance is required, all datagrams could be marked accordingly.

SCTP for Beginners

Erwin P. Rathgeb

SCTP API

We have been working on a library implementing the SCTP protocol as a user land implementation in the programming language C. This implementation has been developed in cooperation of the Computer Networking Group (at the IEM) of the University of Essen (Germany) and Siemens AG (ICN), Munich (Germany), and is currently available under the GNU Public License (GPL) from <http://www.sctp.de>. The RFC2960 contains a section that describes an example application programming interface (API) which has been used as a basis for the functions of this library. Note that other implementations may have similar, but still different APIs (such as the reference implementation from R. Stewart). Kernel implementations that are currently worked on will likely have an API based on the standard POSIX/Berkeley socket interface, which is still being discussed.

API According to RFC 2960

The RFC2960 considers a so-called Upper Layer Protocol (ULP) that is a user of SCTP services in section 10. As such, there is a communication between these two in both directions. The ULP may call the following primitives:

1. Initialize
2. Associate
3. Shutdown
4. Abort
5. Send
6. Set Primary
7. Receive
8. Status
9. Change Heartbeat
10. Request HeartBeat
11. Get SRTT Report
12. Set Failure Threshold
13. Set Protocol Parameters
14. Receive unsend message
15. Receive unacknowledged message
16. Destroy SCTP instance

The ULP may be notified by an SCTP instance of a number of events. The following are mentioned in the RFC2960 :

1. DATA ARRIVE notification
2. SEND FAILURE notification
3. NETWORK STATUS CHANGE notification
4. COMMUNICATION UP notification
5. COMMUNICATION LOST notification
6. COMMUNICATION ERROR notification
7. RESTART notification
8. SHUTDOWN COMPLETE notification

The meaning of some of these will be shortly described here, and we will show how it is implemented in our SCTP library implementation.

The SCTP Library - Overview

The complete interface to the SCTP library (see also the page SCTP Links) is contained in one file, that an application using SCTP will have to include. It is named `sctp.h`.

SCTP for Beginners

Erwin P. Rathgeb

Notifications to an application are handled by function callbacks which are initially passed to the library, and will be called after a certain event has occurred.

Alongside the API declarations for the above mentioned functions, our implementation also offers a number of helper routines for timer based execution of functions, and registration of callback functions for events on a UDP socket. As such the library also allows sending, receiving and processing data from UDP sockets. However, these will not be mentioned here since they are described in detail in the manual that comes with this implementation.

Function Reference

In the following you will find a short description of the API that our library implementation is offering. In case you want to use this library, you will also find a number of simple example applications in the distribution.

Please note that most of these issues are implementation specific, but the basic API adheres to the example API mentioned in the RFC2960 .

Initialize

Name: `sctp_registerInstance`

Description: Initializes SCTP instances (associated with a port number)

Syntax: `sctp_registerInstance(unsigned short localPort, unsigned short noOfInStreams, unsigned short noOfOutStreams, unsigned int noOfLocalAddresses, unsigned char localAddressList[][SCTP_MAX_IP_LEN], SCTP_ulpCallbacks ULPcallbackFunctions)`

Default: `unsigned short: instance name`

Is called to initialize one SCTP instance. On the very first call, it will open raw sockets for capturing SCTP packets (IPv4 and if possible, IPv6, too) from the network. An application may register several instances with different sets of callback functions, but there should not be several instances with the same port. If the port is set to zero, the instance will not be able to receive any datagrams, unless the `sctp_associate()` function is called. So it does not make sense to specify 0 as port number, unless the application is a client. A server needs a port number greater than 0 here.

Associate

Name: `sctp_associate`

Description: Sets up a new association

Syntax: `sctp_associate(unsigned short SCTP_InstanceName, unsigned short noOfOutStreams, unsigned char destinationAddress[], unsigned short destinationPort, void* ulp_data)`

Default: `unsigned int: association id`

This function is called to set up an association. It triggers sending of an INIT chunk to a server, and when the association gets established, the `CommunicationUp` notification is called. The ULP must specify the SCTP instance which this association belongs to.

Shutdown

SCTP for Beginners

Erwin P. Rathgeb

Name: `sctp_shutdown`

Description: Terminates an association gracefully

Syntax: `sctp_shutdown(unsigned int associationID)`

Default: `int: error code`

Initiates the shutdown of the specified association. Basically triggers sending of a SHUTDOWN chunk, and changes the association state to SHUTDOWN PENDING or SHUTDOWN SENT (see SCTP States). After the shutdown procedure is completed, the ShutdownComplete notification is called.

Abort

Name: `sctp_abort`

Description: Terminates an association

Syntax: `sctp_abort(unsigned int associationID)`

Default: `int: error code`

Initiates the abort of the specified association. Basically triggers sending of an ABORT chunk and closed the current association.

Send

Name: `sctp_send`

Description: Sends data as data chunk

Syntax: `sctp_send(unsigned int associationID, unsigned short streamID, unsigned char *buffer, unsigned int length, unsigned int protocolId, short path_id, void *context, unsigned int lifetime, int unorderedDelivery, int dontBundle)`

Default: `int: error code`

`sctp_send()` is used by the ULP to send data as data chunks. There are quite a few parameters that can be or must be passed along. For most of these, `sctp.h` does specify constants that may be used as default values.

Set Primary

Name: `sctp_setPrimary`

Description: Sets a primary path

Syntax: `sctp_setPrimary(unsigned int associationID, short path_id)`

Default: `int: error code`

SCTP for Beginners

Erwin P. Rathgeb

This function sets a new primary address. It is given the index of the new primary path and the association ID as parameter, and returns 0 on success, or 1 on error. After initialization, the index of the default primary path is always zero (because there will always be at least one path!).

Receive

Name: `sctp_receive`

Description: Copies received data chunks to user specified buffer

Syntax: `sctp_receive(unsigned int associationID, unsigned short streamID, unsigned char *buffer, unsigned int *length)`

Default: `unsigned short: error code`

After a DataArrive notification for a certain stream, a ULP may call this function with a specified buffer address and length to copy received data into user memory. After returning, the actual length is contained in the length field.

Status

The SCTP library offers quite a number of functions to get at association specific data. This data can be retrieved, and some parameters also be set. Our implementation defines parameters that are instance specific, association specific and (within an association) path specific. For these, the following data structures have been defined:

Instance specific parameters (and association defaults) :

```
struct SCTP_Instance_Parameters {
    unsigned int rtoInitial;
    unsigned int rtoMin;
    unsigned int validCookieLife;
    unsigned int assocMaxRetransmits;
    unsigned int pathMaxRetransmits;
    unsigned int maxInitRetransmits;
    unsigned int myRwnd;
    unsigned int delay;
    unsigned char ipTos;
    unsigned int maxSendQueue;
    unsigned int maxRecvQueue;
} SCTP_InstanceParameters;
```

Association specific parameters :

```
struct SCTP_Association_Status
{
    unsigned short state;
    unsigned short numberOfAddresses;
    unsigned char primaryDestinationAddress[SCTP_MAX_IP_LEN];
    unsigned short outStreams;
    unsigned short inStreams;
    unsigned short primaryAddressIndex;
    unsigned int currentReceiverWindowSize;
    unsigned int outstandingBytes;
    unsigned int noOfChunksInSendQueue;
    unsigned int noOfChunksInRetransmissionQueue;
    unsigned int noOfChunksInReceptionQueue;
}
```

SCTP for Beginners

Erwin P. Rathgeb

```
unsigned int    rtoInitial;
unsigned int    rtoMin;
unsigned int    validCookieLife;
unsigned int    assocMaxRetransmits;
unsigned int    pathMaxRetransmits;
unsigned int    maxInitRetransmits;
unsigned int    myRwnd;
unsigned int    delay;
unsigned char    ipTos;
unsigned int    maxSendQueue;
unsigned int    maxRecvQueue;
} SCTP_AssociationStatus;
```

Path specific parameters :

```
struct SCTP_Path_Status
{
    unsigned char destinationAddress[SCTP_MAX_IP_LEN];
    short state;
    unsigned int srtt;
    unsigned int rto;
    unsigned int rttvar;
    unsigned int heartbeatIntervall;
    unsigned int cwnd;
    unsigned int cwnd2;
    unsigned int partialBytesAacked;
    unsigned int ssthresh;
    unsigned int outstandingBytesPerAddress;
    unsigned int mtu;
    unsigned char ipTos;
}SCTP_PathStatus;
```

The following functions have been defined for getting the state parameters :

Name: `sctp_getAssocDefaults`

Description: Gets current default values for new associations

Syntax: `sctp_getAssocDefaults(unsigned short SCTP_InstanceName,
SCTP_InstanceParameters* params)`

Default: `int: error code`

Name: `sctp_getAssocStatus`

Description: Gets current parameters of an association

Syntax: `sctp_getAssocStatus(unsigned int associationID, SCTP_AssociationStatus*
status)`

Default: `int: error code`

Name: `sctp_getPathStatus`

Description: Gets current values for an association path

SCTP for Beginners

Erwin P. Rathgeb

Syntax: `sctp_getPathStatus(unsigned int associationID, short path_id, SCTP_PathStatus* status)`

Default: `int: error code`

Set Protocol Parameters

Name: `sctp_setAssocDefaults`

Description: Sets current default values for new associations

Syntax: `sctp_setAssocDefaults(unsigned short SCTP_InstanceName, SCTP_InstanceParameters* params)`

Default: `int: error code`

Name: `sctp_setAssocStatus`

Description: Sets current parameters of an association

Syntax: `sctp_setAssocStatus(unsigned int associationID, SCTP_AssociationStatus* status)`

Default: `int: error code`

Name: `sctp_setPathStatus`

Description: Sets current values for an association path

Syntax: `sctp_setPathStatus(unsigned int associationID, short path_id, SCTP_PathStatus* status)`

Default: `int: error code`

These functions are the pendants of the functions to retrieve the association status (see section Status). Except for the last function, all of these are currently implemented, allowing for modifying large parts of the protocol behavior (without breaking protocol compliance).

Request Heartbeat

Name: `sctp_changeHeartBeat`

Description: Turns heartbeat on or off, and sets time interval

Syntax: `sctp_changeHeartBeat(unsigned int associationID, short path_id, int heartbeatON, unsigned int timeIntervall)`

Default: `int: error code`

Turns the heartbeat of an association on or off, and sets the interval, if it is turned on. Our implementation is currently missing the jitter, that is required by the RFC2960 , but that will be added soon.

Request Heartbeat

SCTP for Beginners

Erwin P. Rathgeb

Name: `sctp_requestHeartbeat`

Description: Sends an on-demand heartbeat message

Syntax: `sctp_requestHeartbeat(unsigned int associationID, short path_id)`

Default: `int: error code`

Sends a heartbeat to the given destination address (i.e. path) of an association. This is an explicit request from the ULP to the SCTP instance to perform an on-demand heartbeat.

Get SRTT Report

Name: `sctp_getSrttReport`

Description: Returns the SRTT value of a certain path

Syntax: `sctp_getSrttReport(unsigned int associationID, short path_id)`

Default: `unsigned int: Smoothed roundtrip time in msec`

Returns the smoothed round trip time in milliseconds for a certain destination path within an association, or zero on error.

Set Failure Threshold

Name: `sctp_setFailureThreshold`

Description: Sets threshold for retransmission failures

Syntax: `sctp_setFailureThreshold(unsigned int associationID, unsigned short pathMaxRetransmissions)`

Default: `int: error code`

Is used to set the threshold for retransmissions of the given association. If the retransmission counter exceeds this threshold, the the destination address is considered unreachable.

Recieve Unsent

Name: `sctp_receiveUnsent`

Description: Returns messages that were not sent

Syntax: `sctp_receiveUnsent(unsigned int associationID, unsigned char *buffer, unsigned int *length, unsigned short *streamID, unsigned short *streamSN, unsigned int* protocolId)`

Default: `int: error code`

Will return messages that could not be sent for some reason, e.g. because association was aborted before they could be transmitted for the first time. Is NOT implemented in the current version of our library! Will return messages that were sent, but have not been acknowledged by the peer before the association was terminated (or aborted). Is NOT implemented in the current version of our library !

SCTP for Beginners

Erwin P. Rathgeb

Recieve Unacknowledged

Name: `sctp_receiveUnacked`

Description: Returns messages that were sent, but not acknowledged by the peer

Syntax: `sctp_receiveUnacked(unsigned int associationID, unsigned char *buffer, unsigned int *length, unsigned short *streamID, unsigned short *streamSN, unsigned int* protocolId)`

Default: `int: error code`

Destroy SCTP Instance

Name: `sctp_unregisterInstance`

Description: Un-registers SCTP instance and releases resources

Syntax: `sctp_unregisterInstance(unsigned short instance_name)`

Default: `int: error code`

Is called to release resources used by a previously registered SCTP instance. If no instances are registered for either IPv4 or IPv6 raw sockets any more, callbacks for these sockets are also removed.

Name: `sctp_deleteAssociation`

Description: Removes association data and releases resources

Syntax: `sctp_deleteAssociation(unsigned int associationID)`

Default: `int: error code`

This function may be called to remove all the data structures belonging to an association. It should be called some time after a ShutdownComplete or a CommunicationLost notification has been received, and the ULP has retrieved all the data that may still be in the queues using the (currently unimplemented) functions `sctp_receiveUnacked()` and `sctp_receiveUnsent()`. A call to this function finally frees all association resources.

Callback Notifications

The notifications are realized via function callbacks, and the functions for an SCTP instance are passed upon initialization (see function `sctp_registerInstance()`) in the struct `SCTP_ulpCallbacks`:

```
structSCTP_ulp_Callbacks
{
    void (*dataArriveNotif) (unsigned int, unsigned int,
                           unsigned int, unsigned int,
                           unsigned int, void*);
    void (*sendFailureNotif) (unsigned int, unsigned char *,
                             unsigned int, unsigned int *, void*);
    void (*networkStatusChangeNotif) (unsigned int, short,
                                       unsigned short, void*);
    void* (*communicationUpNotif) (unsigned int, unsigned short,
```

SCTP for Beginners

Erwin P. Rathgeb

```
        int, unsigned short,  
        unsigned short, void*);  
void (*communicationLostNotif) (unsigned int,  
                                unsigned short, void*);  
void (*communicationErrorNotif) (unsigned int,  
                                  unsigned short, void*);  
void (*restartNotif) (unsigned int, void*);  
void (*shutdownCompleteNotif) (unsigned int, void*);  
} SCTP_ulpCallbacks;
```

All of these functions pass the association id as parameter, and return all necessary parameters along. For a more detailed explanation of these parameters you can have a look at the accompanying library manual, which describes the library API in detail.

SCTP Terminology

This document uses certain terms and abbreviations that have a special meaning which is defined in RFC2960, and may be confusing at first. Take, e.g. a connection between two SCTP instances, which has been named an SCTP association. Most of these terms are explained here. In case you are missing something (or we have missed something) please contact us. For addresses see below.

SCTP Key Terms

These terms are taken straight from the RFC2960.

Active destination transport address: A transport address on a peer endpoint which a transmitting endpoint considers available for receiving user messages.

Bundling: An optional multiplexing operation, whereby more than one user message may be carried in the same SCTP packet. Each user message occupies its own DATA chunk.

Chunk: A unit of information within an SCTP packet, consisting of a chunk header and chunk-specific content.

Congestion Window (cwnd): An SCTP variable that limits the data, in number of bytes, a sender can send to a particular destination transport address before receiving an acknowledgement.

Cumulative TSN Ack Point: The TSN of the last DATA chunk acknowledged via the Cumulative TSN Ack field of a SACK.

Idle destination address: An address that has not had user messages sent to it within some length of time, normally the HEARTBEAT interval or greater.

Inactive destination transport address: An address which is considered inactive due to errors and unavailable to transport user messages.

Message Authentication Code (MAC): An integrity check mechanism based on cryptographic hash functions using a secret key. Typically, message authentication codes are used between two parties that share a secret key in order to validate information transmitted between these parties. In SCTP it is used by an endpoint to validate the State Cookie information that is returned from the peer in the COOKIE ECHO chunk. The term "MAC" has different meanings in different contexts. SCTP uses this term with the same meaning as in RFC2104.

SCTP for Beginners

Erwin P. Rathgeb

Ordered Message: A user message that is delivered in order with respect to all previous user messages sent within the stream the message was sent on.

Outstanding TSN (at an SCTP endpoint): A TSN (and the associated DATA chunk) that has been sent by the endpoint but for which it has not yet received an acknowledgement.

Path: The route taken by the SCTP packets sent by one SCTP endpoint to a specific destination transport address of its peer SCTP endpoint. Sending to different destination transport addresses does not necessarily guarantee getting separate paths.

Primary Path: The primary path is the destination and source address that will be put into a packet outbound to the peer endpoint by default. The definition includes the source address since an implementation MAY wish to specify both destination and source address to better control the return path taken by reply chunks and on which interface the packet is transmitted when the data sender is multi-homed.

Receiver Window (rwnd): An SCTP variable a data sender uses to store the most recently calculated receiver window of its peer, in number of bytes. This gives the sender an indication of the space available in the receiver's inbound buffer.

SCTP association: A protocol relationship between SCTP endpoints, composed of the two SCTP endpoints and protocol state information including Verification Tags and the currently active set of Transmission Sequence Numbers (TSNs), etc. An association can be uniquely identified by the transport addresses used by the endpoints in the association. Two SCTP endpoints MUST NOT have more than one SCTP association between them at any given time.

SCTP endpoint: The logical sender/receiver of SCTP packets. On a multi-homed host, an SCTP endpoint is represented to its peers as a combination of a set of eligible destination transport addresses to which SCTP packets can be sent and a set of eligible source transport addresses from which SCTP packets can be received. All transport addresses used by an SCTP endpoint must use the same port number, but can use multiple IP addresses. A transport address used by an SCTP endpoint must not be used by another SCTP endpoint. In other words, a transport address is unique to an SCTP endpoint.

SCTP packet: The unit of data delivery across the interface between SCTP and the connectionless packet network (e.g., IP). An SCTP packet includes the common SCTP header, possible SCTP control chunks, and user data encapsulated within SCTP DATA chunks.

SCTP user application: The logical higher-layer application entity which uses the services of SCTP, also called the Upper-layer Protocol (ULP).

Slow Start Threshold (ssthresh): An SCTP variable. This is the threshold which the endpoint will use to determine whether to perform slow start or congestion avoidance on a particular destination transport address. Ssthresh is in number of bytes.

Stream: A uni-directional logical channel established from one to another associated SCTP endpoint, within which all user messages are delivered in sequence except for those submitted to the unordered delivery service.

SCTP for Beginners

Erwin P. Rathgeb

Stream Sequence Number: A 16-bit sequence number used internally by SCTP to assure sequenced delivery of the user messages within a given stream. One stream sequence number is attached to each user message.

Transmission Sequence Number (TSN): A 32-bit sequence number used internally by SCTP. One TSN is attached to each chunk containing user data to permit the receiving SCTP endpoint to acknowledge its receipt and detect duplicate deliveries.

Transport address: A Transport Address is traditionally defined by Network Layer address, Transport Layer protocol and Transport Layer port number. In the case of SCTP running over IP, a transport address is defined by the combination of an IP address and an SCTP port number (where SCTP is the Transport protocol).

Unacknowledged TSN (at an SCTP endpoint): A TSN (and the associated DATA chunk) which has been received by the endpoint but for which an acknowledgement has not yet been sent. Or in the opposite case, for a packet that has been sent but no acknowledgement has been received.

Unordered Message: Unordered messages are "unordered" with respect to any other message, this includes both other unordered messages as well as other ordered messages. Unordered message might be delivered prior to or later than ordered messages sent on the same stream.

Verification Tag: A 32 bit unsigned integer that is randomly generated. The Verification Tag provides a key that allows a receiver to verify that the SCTP packet belongs to the current association and is not an old or stale packet from a previous association.

Abbreviations

The following list of abbreviations is also largely taken from RFC2960.

MAC: Message Authentication Code
RTO: Retransmission Time-out
RTT: Round-trip Time
RTTVAR: Round-trip Time Variation
SCTP: Stream Control Transmission Protocol
SRTT: Smoothed RTT
TCB: Transmission Control Block
TLV: Type-Length-Value Coding Format
TSN: Transmission Sequence Number
ULP: Upper-layer Protocol