SCTP

STREAM CONTROL TRANSMISSION PROTOCOL

INTRODUCTION TO SCTP, A GENERAL PURPOSE TRANSPORT PROTOCOL SUITED FOR HIGH RELIABILITY APPLICATIONS

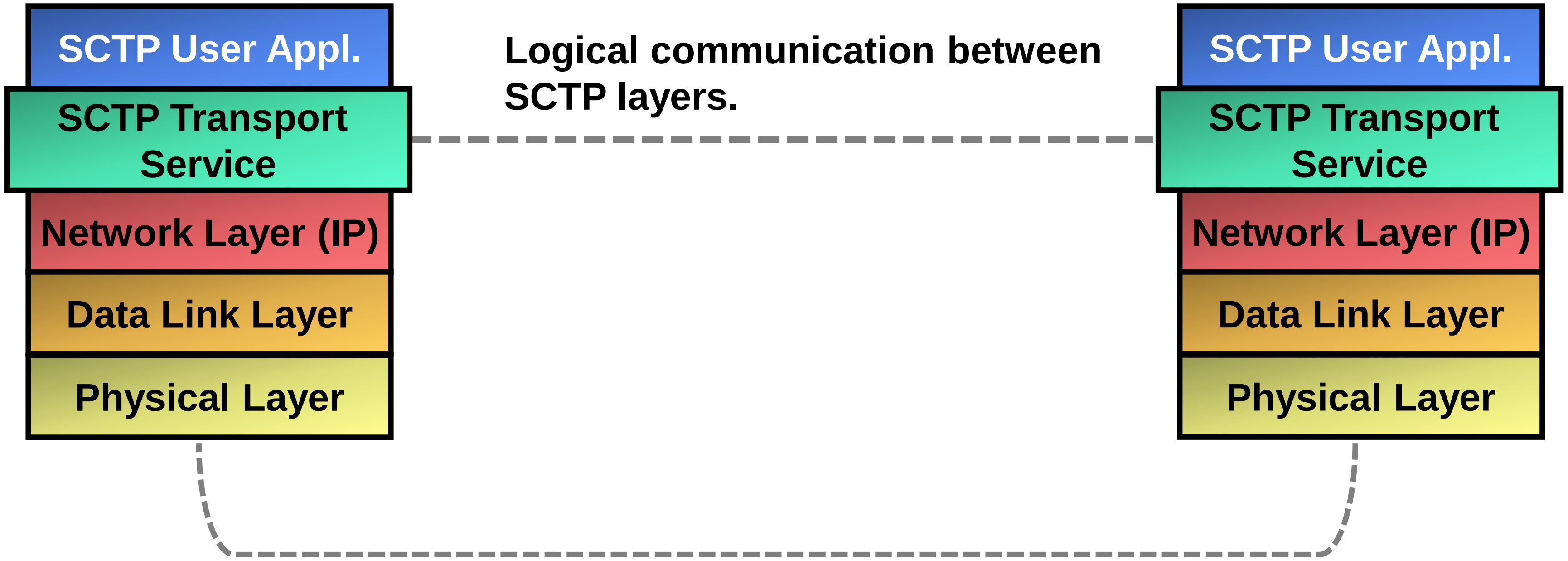
Peter R. Egli
peteregli.net

Contents

1. What is SCTP?

SCTP (Stream Control Transmission Protocol, RFC4960) is a transport protocol on OSI layer 4 (like TCP or UDP).
SCTP was specifically designed as a transport protocol for public telephony network signalling message transport. However, SCTP is generic and may supersede TCP in other applications as well.

Simplified OSI stack (session and presentation layers omitted):



2. Why use SCTP? (1/2)

SCTP has similar characteristics and applications as TCP (RFC793), but includes some important improvements:

1. No head-of-line blocking:

TCP imposes a strict data ordering. However, if a user data message is lost during transit, all subsequent user data messages are delayed until the lost message has been retransmitted (= head-of-line blocking).
Some applications do not require a strict ordering of messages. E.g. applications that exchange unrelated application messages could cope with out-of-order messages (messages are simply processed as they arrive at the receiver). But TCP does not allow messages to pass each other.

2. No stream-oriented data transfer:

TCP is stream-oriented. This means that TCP treats data chunks transmitted by an application as an ordered stream of bytes (=octets in network speak). While this concept supports a wide range of applications (message oriented like email, character oriented like TELNET, stream oriented like video), it is unsuited in most applications because these exchange application level messages.
SCTP preserves application level message boundaries, thus liberating applications from implementing a framing protocol on top of the transport protocol for delineating messages.
SCTP simply maps application messages to chunks on the transmit path and back to application messages on the receive path.

2. Why use SCTP? (2/2)

3. Multihoming:

Since a TCP connection is defined by the quadruple source IP, destination IP, source port and destination port, TCP does not support multihoming (use of multiple IP addresses on either side of the connection to allow multiple transmission paths through the network thus increasing reliability).
SCTP has built-in support for multihoming which offloads high-availability applications from implementing this feature.

4. Certain protection against denial of service attacks:

The connection setup of TCP allows denial of service attacks, particularly SYN attacks. Each time the TCP layer receives a SYN packet for setting up a new connection, it allocates a data structure for storing connection parameters (called Transport Control Block). Flooding with a high number of such SYN packets may lead to memory exhaustion.
SCTP implements a procedure to avoid or at least make it more difficult for an attacker to launch a connection denial of service attack (4-way connection setup with state cookie).

3. Main features of SCTP

Multiple data stream support:
Support for multiple logical streams of application messages. Ordering of messages within a stream. Avoidance of head-of-line blocking.

Message oriented data transfer:
Transport of user data as messages, preservation of application level message boundaries.

Multihoming for network redundancy:
Use of multiple IP addresses per SCTP endpoint to allow transmission of data chunks through different network paths.

Denial of service attack protection:
Some measures to protect against denial of service attacks such as connection setup flooding.

Fragmentation:
Detection of path MTU and fragmentation of data chunks to fit into the available path MTU.

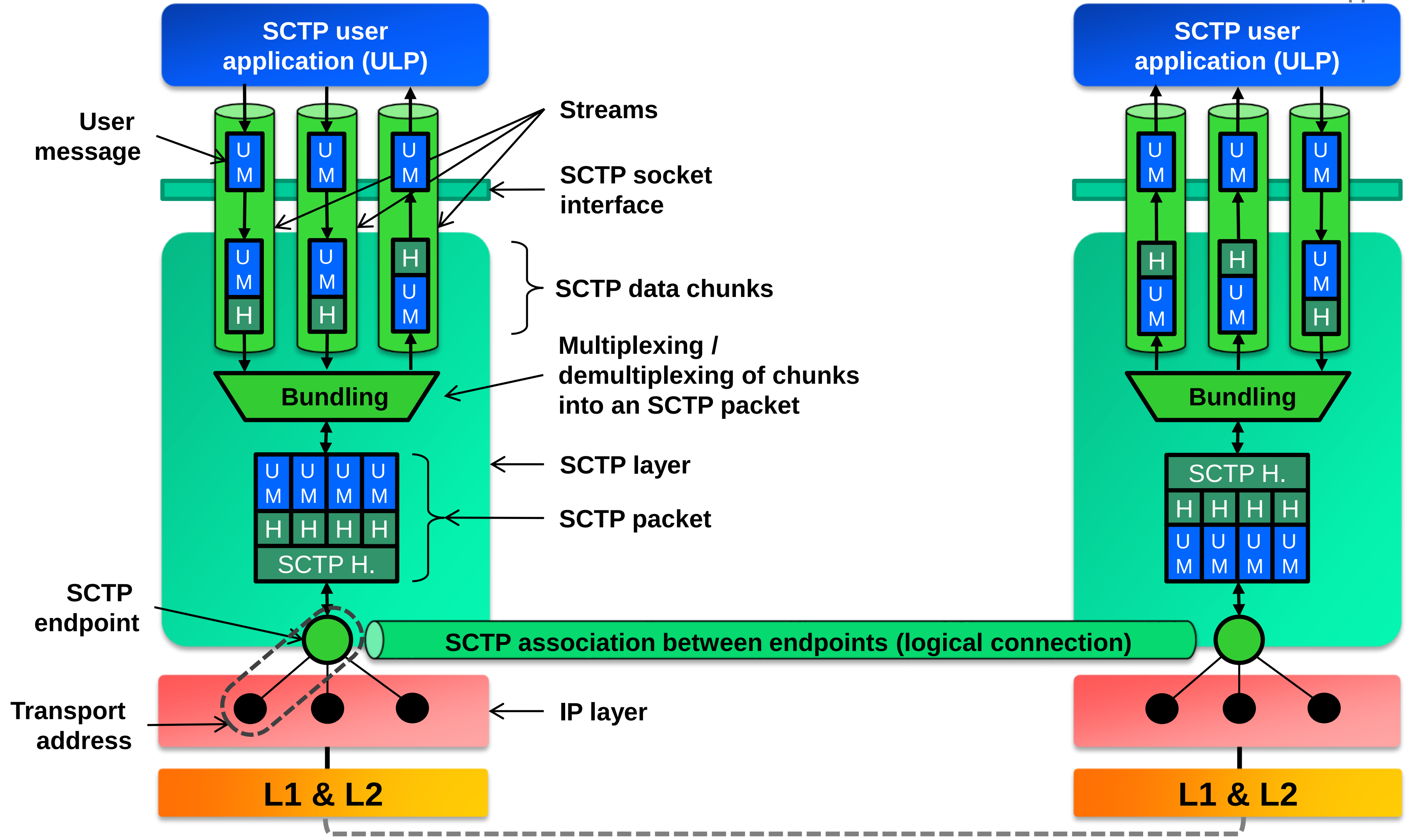
Error correction:
Acknowledged error-free, non-duplicated data transfer.

Congestion avoidance:
Similar functionality as in TCP to avoid congestion to build up in the network.

4. The SCTP model (1/4)

Concepts and terms of SCTP and their relation:

UM:	User Message
H:	Header
L1,L2:	Layer 1, Layer 2
ULP:	Upper Layer Protocol


SCTP user application (ULP)
User message
Streams
SCTP socket interface
UM
H
SCTP data chunks
Multiplexing / demultiplexing of chunks into an SCTP packet
Bundling
SCTP layer
SCTP packet
SCTP H.
SCTP endpoint
SCTP association between endpoints (logical connection)
Transport address
IP layer
L1 & L2

4. The SCTP model (2/4)

Association:

An SCTP association is the logical relationship between 2 SCTP endpoints (≈ SCTP connection).

Bundling:

Bundling packs multiple chunks (user data chunks and SCTP control chunks) into an SCTP packet.

Chunk:

Unit of information within an SCTP packet. Chunks contain either user data (user data chunk) or SCTP control information (control chunk). Each chunk has its own header (chunk header).

Endpoint:

An SCTP endpoint is an addressable logical endpoint of an SCTP association. An SCTP endpoint represents exactly 1 SCTP port number, but may contain multiple transport addresses in case of multihoming (1 SCTP port number, multiple IP addresses).

Stream:

A stream is a logical channel transporting in-order application messages.
Streams are unidirectional. If an application requires a bidirectional stream, it must open 1 outgoing and 1 incoming unidirectional stream and treat them together as a bidirectional stream.

4. The SCTP model (3/4)

SCTP packet:
SCTP transport PDU that contains multiple chunks and an SCTP header. The SCTP packet is encapsulated into an IP packet.

Socket interface:
Platform specific API (OS) for opening an SCTP association and sending and receiving user messages.

Transport address:
An SCTP transport address contains the SCTP port number and one specific IP address.

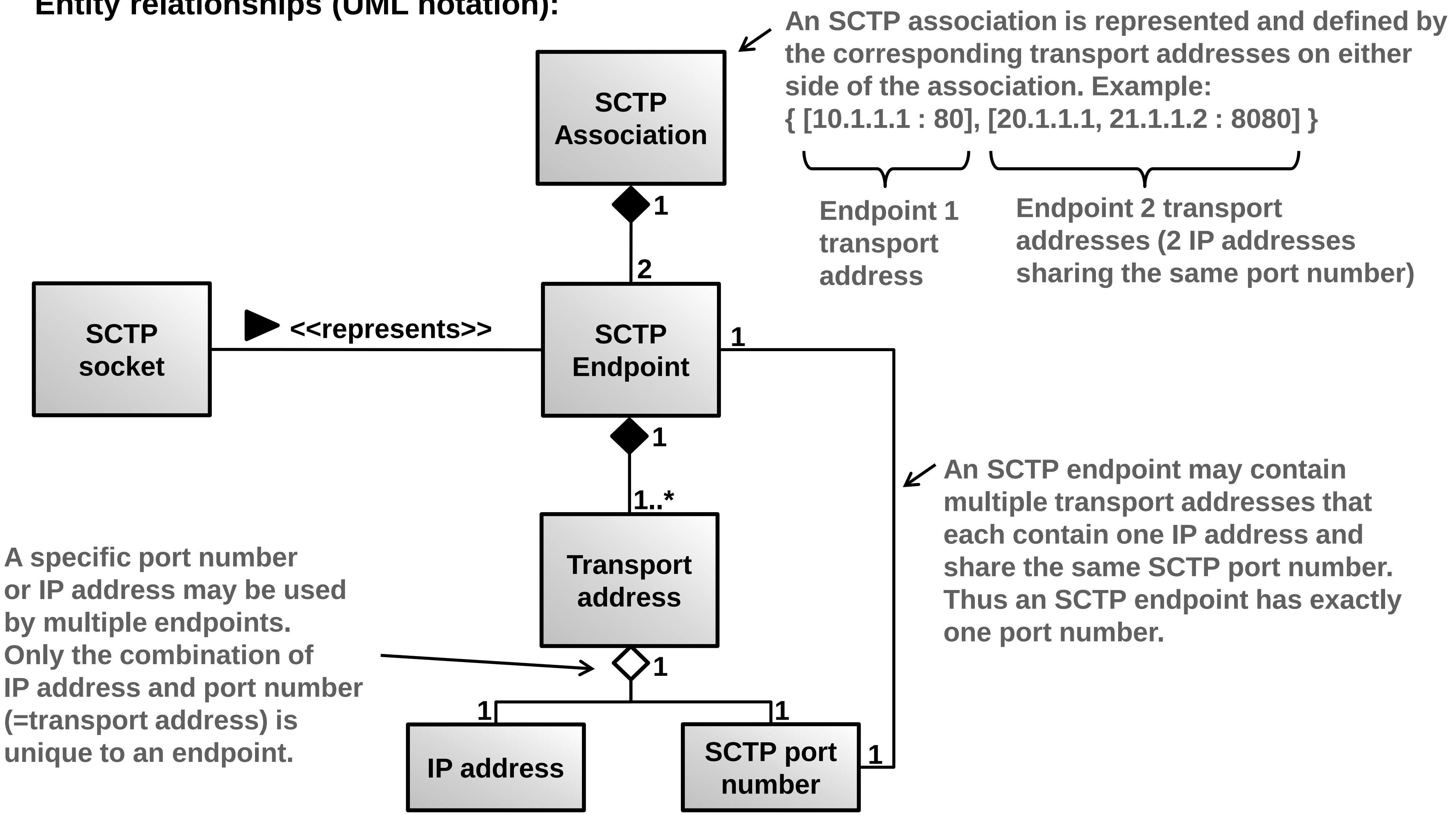
Transport control block (TCB):
Data structure containing connection information on either side of an SCTP connection, maintained by the SCTP layer.

User application:
Application that opens an SCTP association to a peer SCTP user application and exchanges messages with the peer application through the SCTP socket interface.
Also called ULP (Upper Layer Protocol).

User message:
Unit of data sent by the SCTP user application over the SCTP socket interface. In protocol speek, a user message is an 'APDU' (Aplication Protocol Data Unit).

4. The SCTP model (4/4)

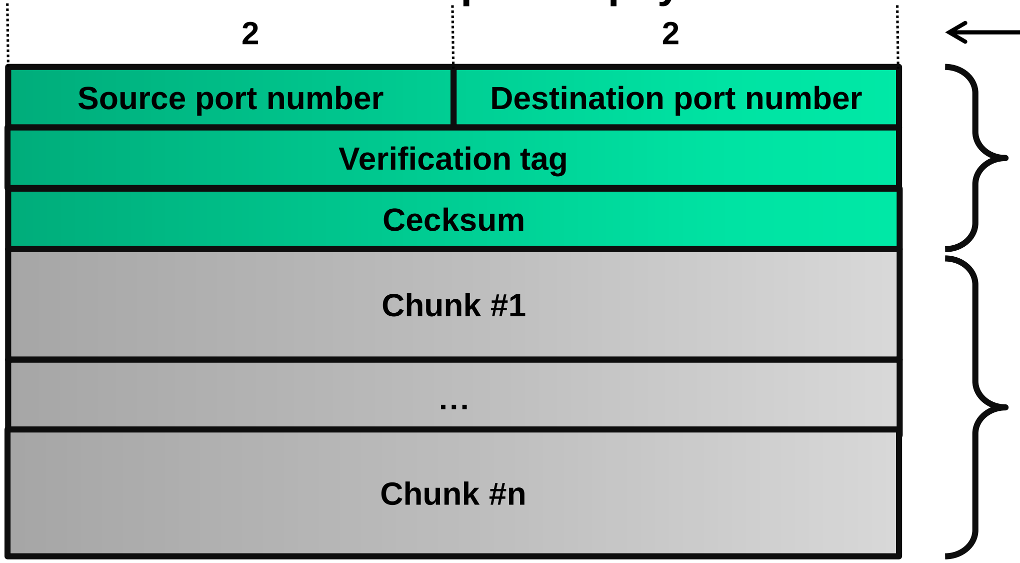
Entity relationships (UML notation):


An SCTP association is represented and defined by the corresponding transport addresses on either side of the association. Example:
{ [10.1.1.1 : 80], [20.1.1.1, 21.1.1.2 : 8080] }
Endpoint 1 transport address
Endpoint 2 transport addresses (2 IP addresses sharing the same port number)
SCTP Association
1
2
SCTP socket
<<represents>>
SCTP Endpoint
1
1
1..*
Transport address
1
1
1
1
IP address
SCTP port number
An SCTP endpoint may contain multiple transport addresses that each contain one IP address and share the same SCTP port number. Thus an SCTP endpoint has exactly one port number.
A specific port number or IP address may be used by multiple endpoints. Only the combination of IP address and port number (=transport address) is unique to an endpoint.

5. SCTP packet format (1/3)

SCTP packet header and chunk format:

The SCTP header contains only a limited set of fields.
More SCTP control information is contained in control chunks that are bundled with user data chunks into the SCTP packet payload.

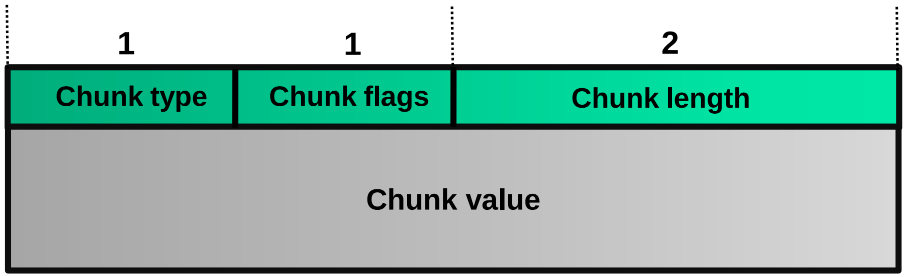


SCTP common header:

Source port #:	Sender's port number
Dest. port #:	Receiver's port number
Verif. tag:	Used by receiver to validate the sender
Checksum:	CRC32 checksum over entire SCTP packet

SCTP payload:

Sequence of user data and control chunks.
Each chunk consists of a chunk header and a chunk value (chunk format see below).



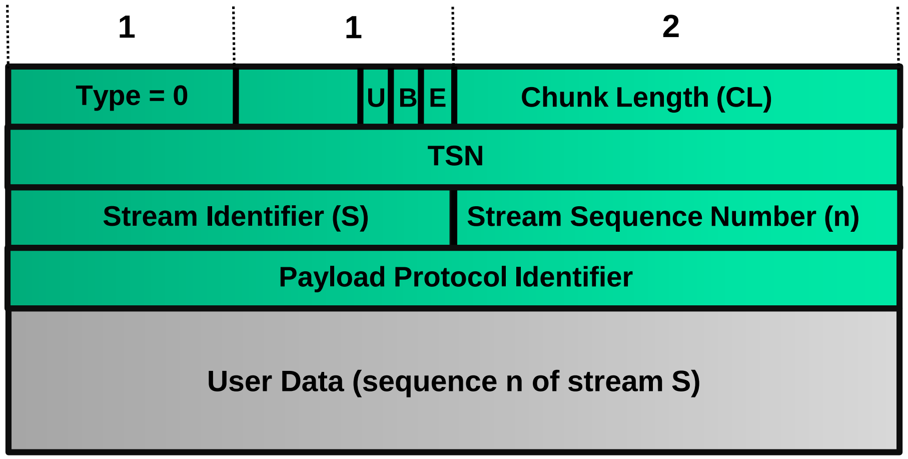
Chunk:

Chunk type:	DATA or control (INIT, INITACK etc.)
Chunk flags:	Chunk-specific bits
Chunk length:	Size of chunk including chunk header
Chunk value:	Chunk-specific data

5. SCTP packet format (2/3)

SCTP DATA chunk format:

User data chunks carry application data along with some chunk and stream management data.


1
1
2
Type = 0
U
B
E
Chunk Length (CL)
TSN
Stream Identifier (S)
Stream Sequence Number (n)
Payload Protocol Identifier
User Data (sequence n of stream S)

Data chunk:

U bit:	If set to 1, indicates that this is an unordered chunk. Unordered chunks are transmitted and sent to the receiving application as is without re-ordering based on the sequence number.
B bit:	Begin of fragment bit. If set to 1 this is the first fragment of a larger fragmented user data message.
E bit:	End of fragment bit. If set to 1 this is the last fragment of a larger fragmented user data message.
Stream identifier:	Identifies the stream to which this chunk belongs.
Stream seq. no.:	Sequence number of user data within this stream. In case of fragmentation this numer is identical for all fragments.
Payload proto. id.:	Identifies the upper (application) layer protocol (e.g. HTTP).
User data:	Application user data (e.g. HTTP header and payload).

5. SCTP packet format (3/3)

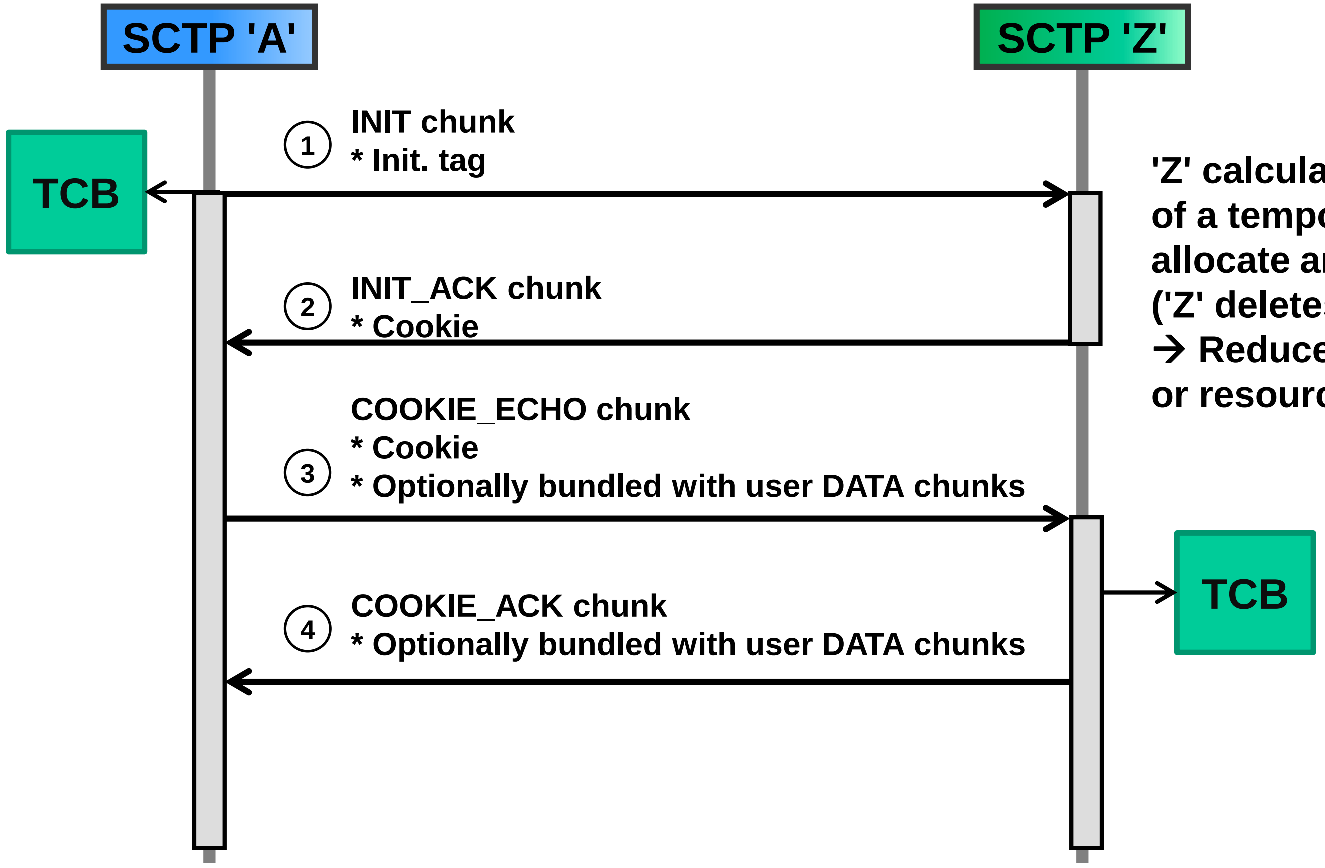
SCTP chunk types:

SCTP packet chunks are either data (DATA) or control chunks.
These chunks may be bundled in an SCTP packet in any order.

Chunk Type	ID	Description
DATA	0	User payload data.
INIT	1	Used for the initiation of an SCTP association.
INIT_ACK	2	Initiation Acknowledegment.
SACK	3	Selective Acknowledegment of DATA chunks.
HEARTBEAT	4	Heartbeat request for probing reachability of a peer transport address.
HEARTBEAT_ACK	5	Heartbeat Acknowledgement.
ABORT	6	Used to immediately close an association. Subsequent control chunks are ignored by SCTP.
SHUTDOWN	7	Used for gracefully shutting down an association.
SHUTDOWN_ACK	8	Shutdown Acknowledgement.
SHUTDOWN_COMPLETE	14	Acknowledge receipt of SHUTDOWN_ACK and completion of shutdown process.
ERROR	9	Used for reporting an error condition to the SCTP peer. An ERROR chunk may be followed by an ABORT (fatal error).
COOKIE_ECHO	10	Chunk containing the State Cookie in the association setup.
COOKIE_ACK	11	Cookie Acknowledgement.

6. Association setup with SCTP (1/4)

SCTP uses a 4-way handshake for establishing an association (= connection).
This scheme helps reducing vulnerability to resource attacks (akin to TCP SYN floods).



'Z' calculates a state cookie (hashing of a temporary TCB), but does NOT allocate any connection resources ('Z' deletes the temporary TCB).
→ Reduce vulnerability to DoS or resource attacks.

'Z' calculates the cookie from the connection parameters and compares it with the received cookie.
In case of a match, 'Z' allocates a TCB for this new SCTP association.

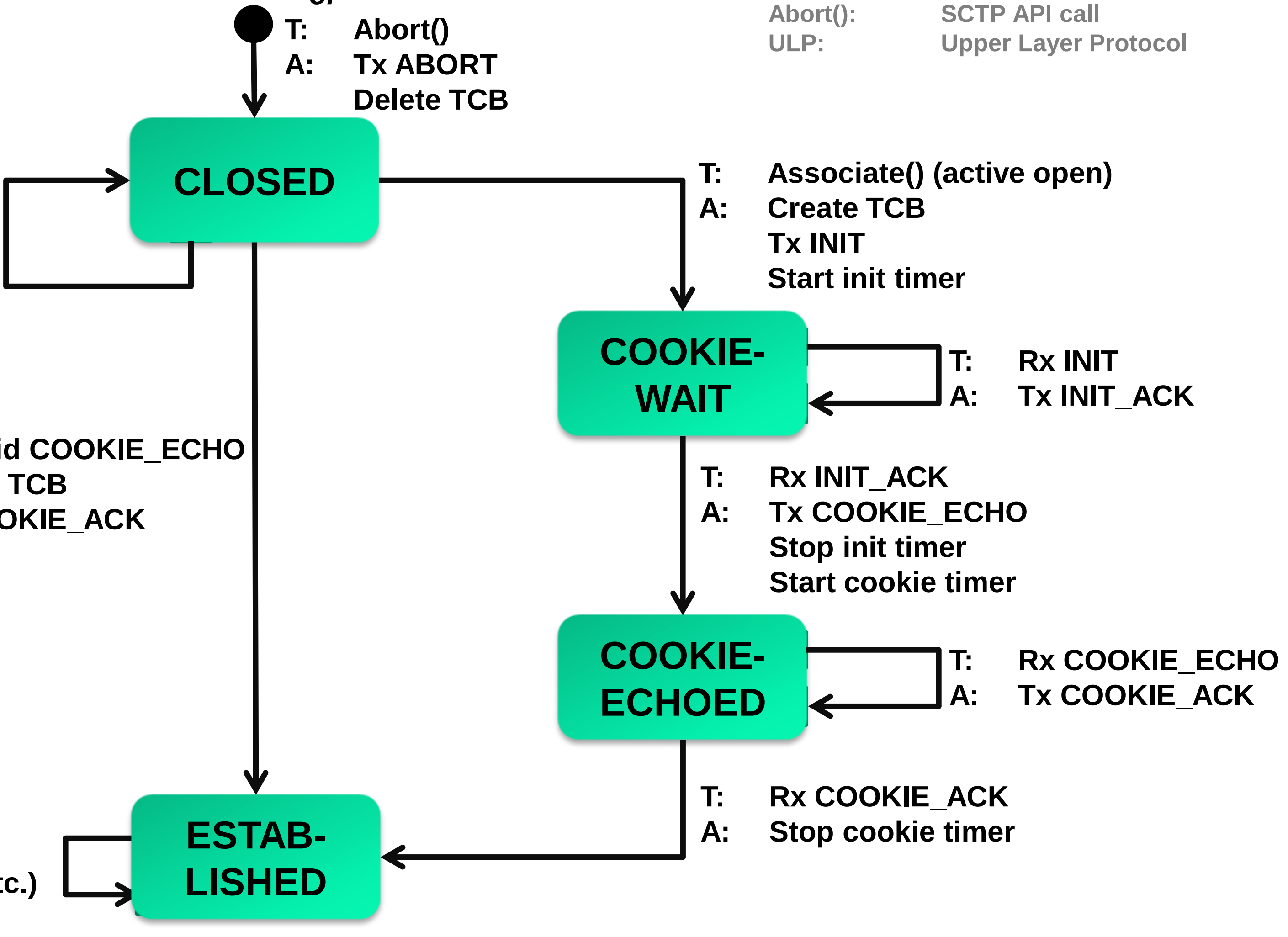
Key:

TCB:	Transport Control Block

6. Association setup with SCTP (2/4)

State diagram for an association setup:

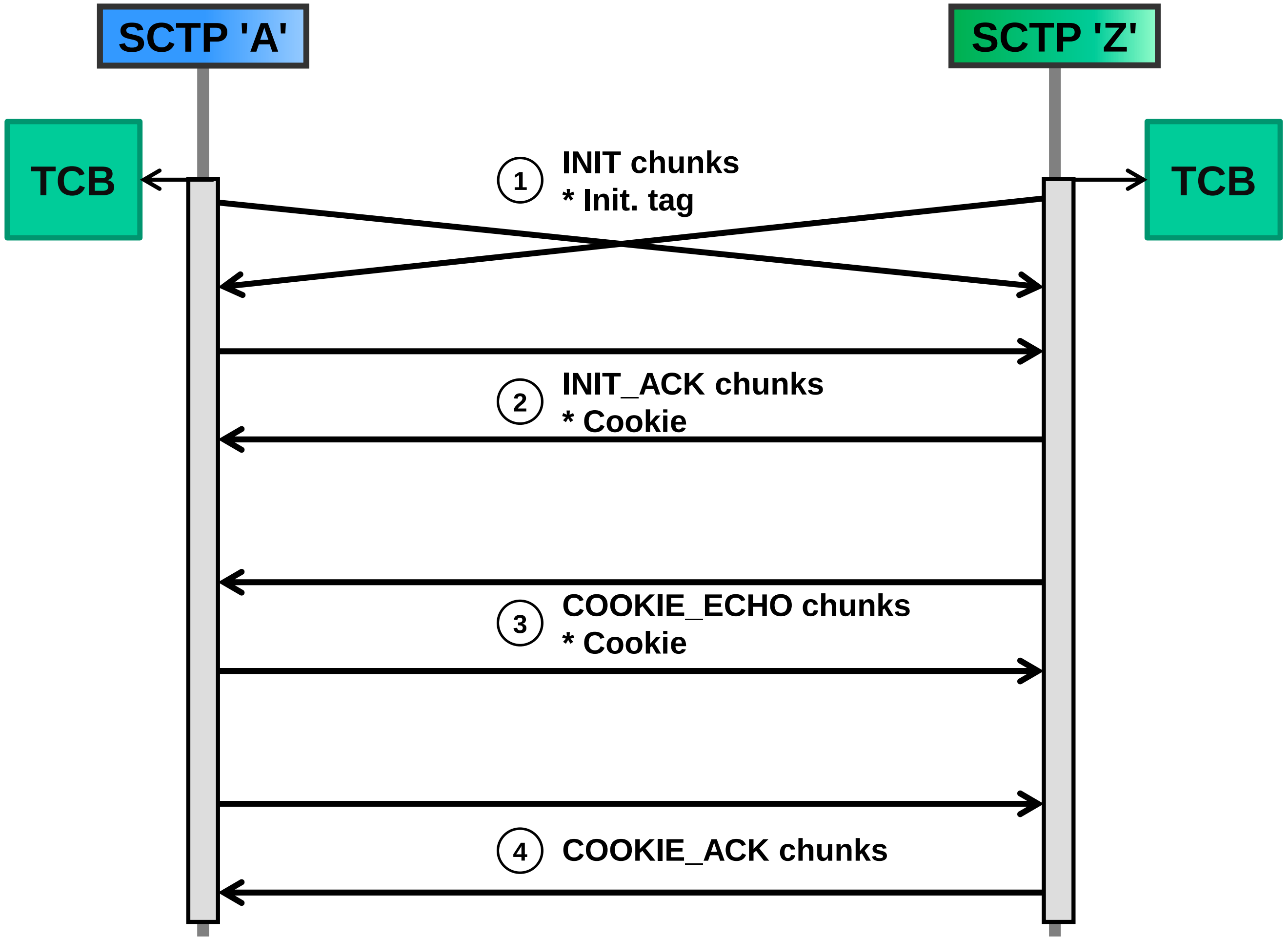
Key:	
T:	Trigger causing the transition
A:	Action to be carried out
Rx:	Receive
Tx:	Transmit
ABORT:	SCTP chunks are in uppercase
Abort():	SCTP API call
ULP:	Upper Layer Protocol



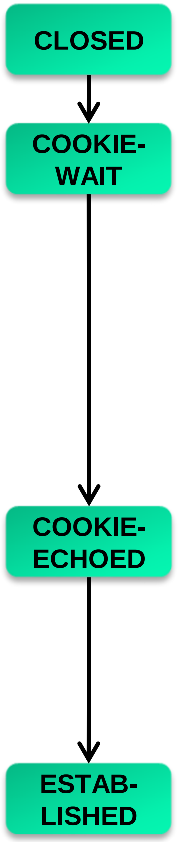
6. Association setup with SCTP (3/4)

Setup collision:

Similar to TCP, an association setup collision (both SCTP 'A' and 'Z' attempt to open an association to each other at roughly the same time) will result in a single association.



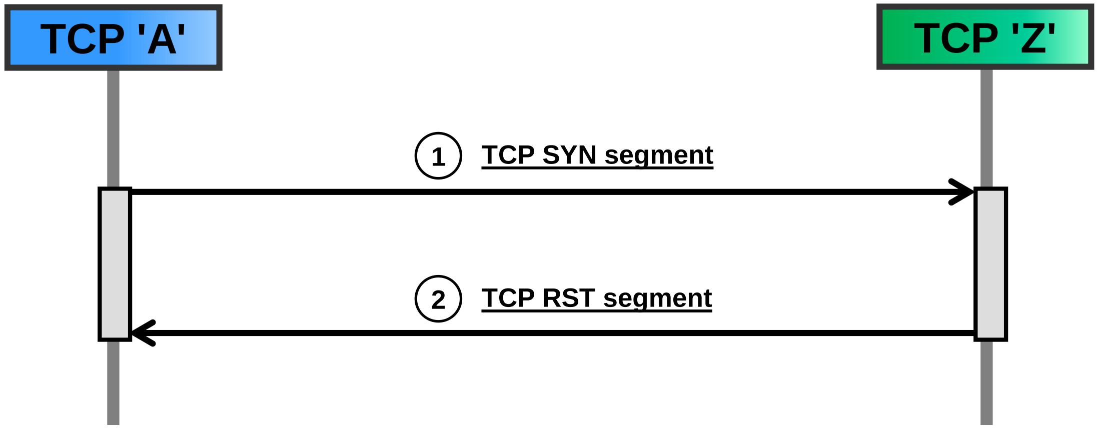
State transitions 'A' & 'Z':



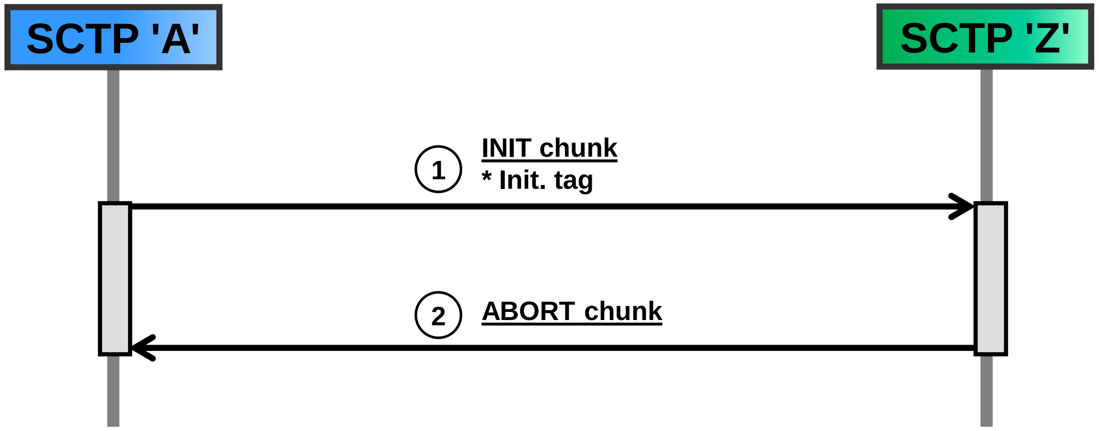
6. Association setup with SCTP (4/4)

Opening an association to a closed port:

Similar to TCP where a SYN to a closed port is answered with a RST packet, SCTP sends back an ABORT chunk. This signals to the INIT-chunk sender that the addressed port is closed.



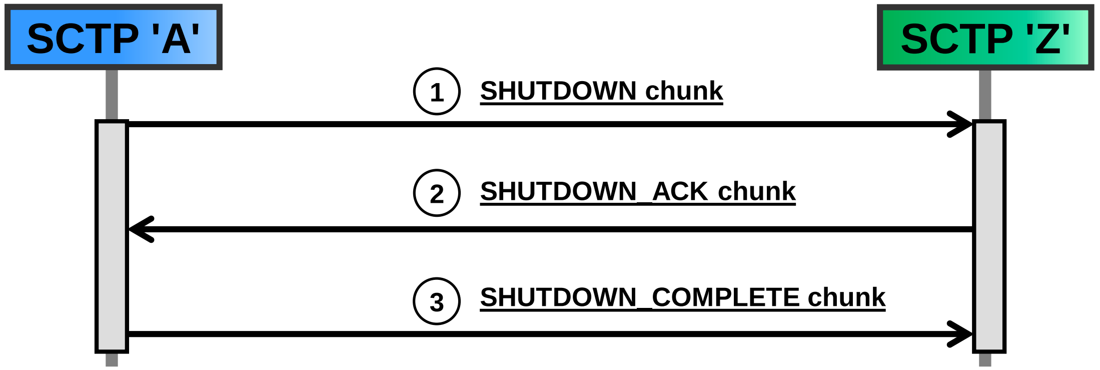
The addressed TCP port is closed on 'Z'.



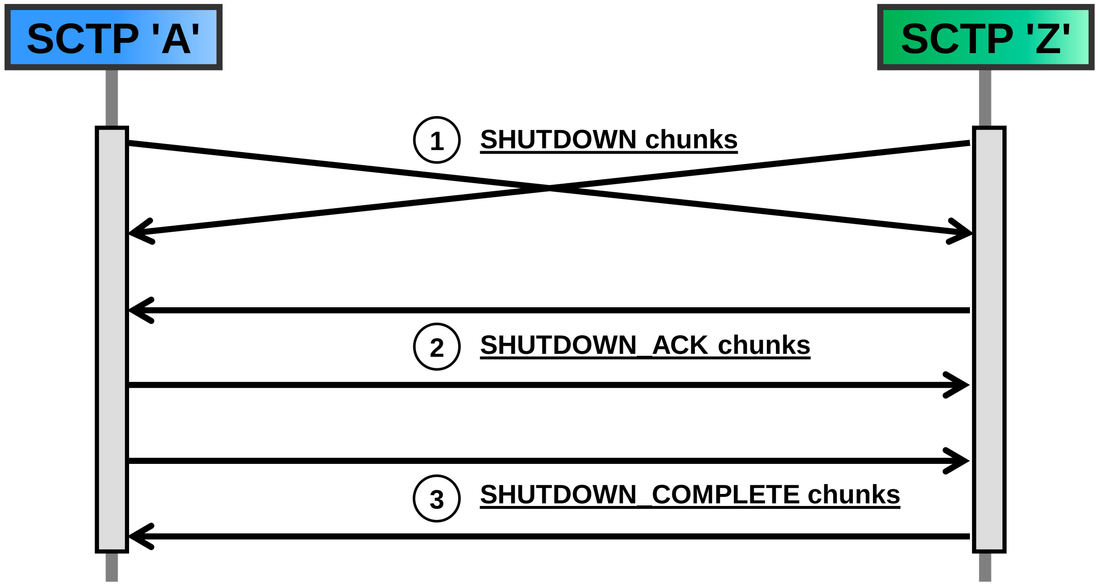
The addressed SCTP port is closed on ‘Z’.

7. Association shutdown with SCTP (1/2)

Unlike TCP, SCTP does not support half-open associations (half-close). A shutdown sequence always closes all streams irrespective of their direction.



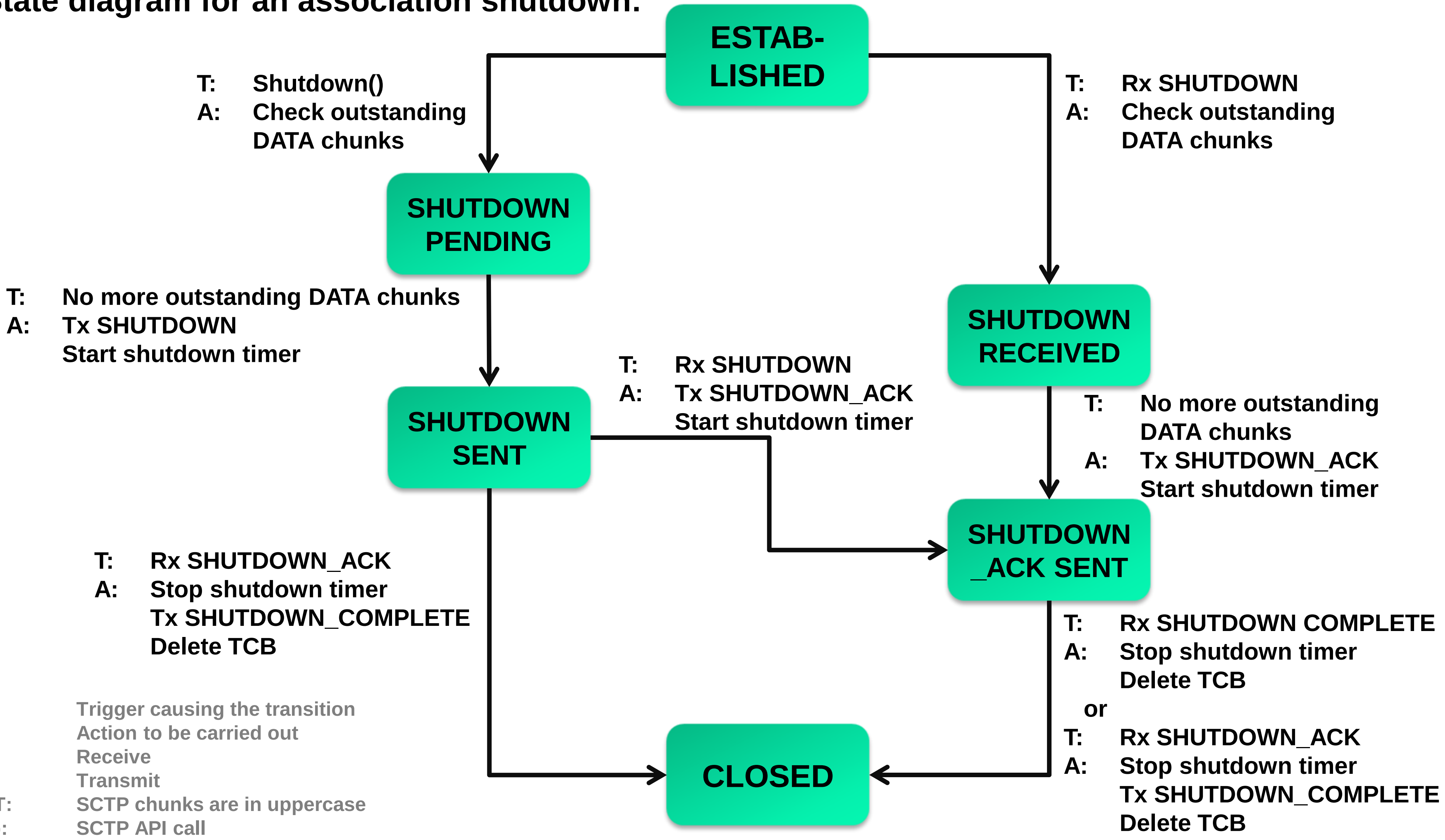
Normal shutdown procedure where either side initiates the shutdown by sending a SHUTDOWN chunk.



Shutdown collision: Both sides of the association send a SHUTDOWN chunk at around the same time.

7. Association shutdown with SCTP (2/2)

State diagram for an association shutdown:



Key:

T:	Trigger causing the transition
A:	Action to be carried out
Rx:	Receive
Tx:	Transmit
ABORT:	SCTP chunks are in uppercase
Abort():	SCTP API call

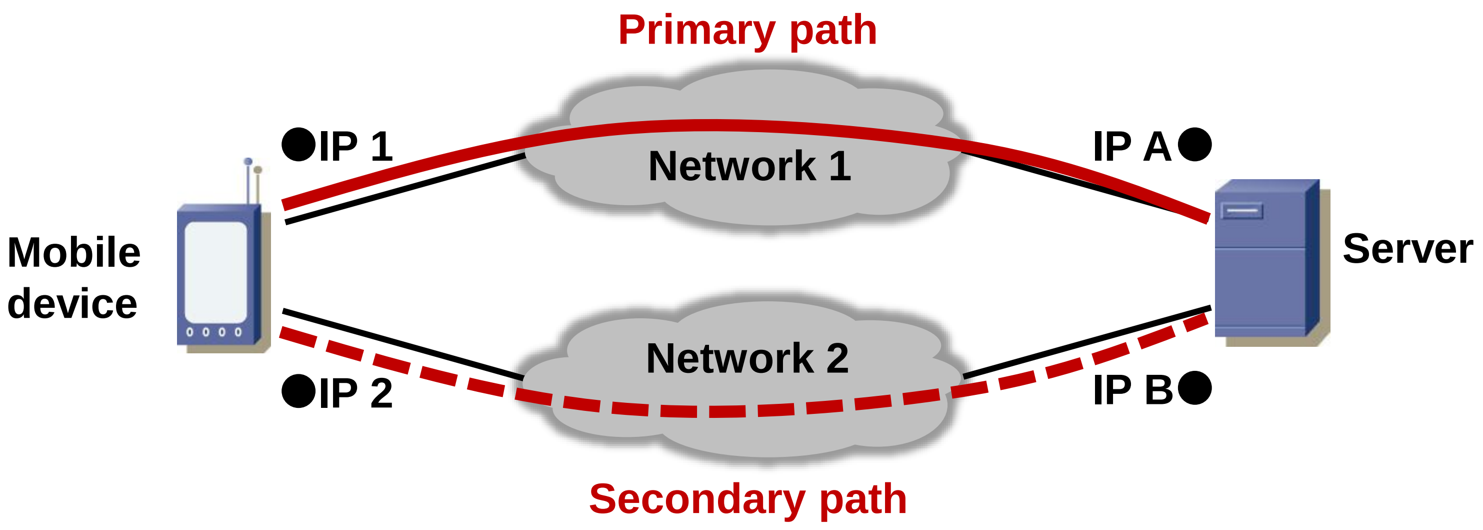
8. Multihoming with SCTP

Multihoming in SCTP allows using multiple transport addresses in an association. Typically, different transport addresses (IP addresses) are bound to different networks thus providing resiliency in case of a network failure.

**One of the transmission paths is the *primary path*. If connectivity over the primary path fails (timeouts of sent packets), SCTP falls back to a secondary (alternate) path for transmitting.
N.B.: SCTP does not perform load balancing or load sharing.**

The ULP can request SCTP to send data to a specific destination transport address.

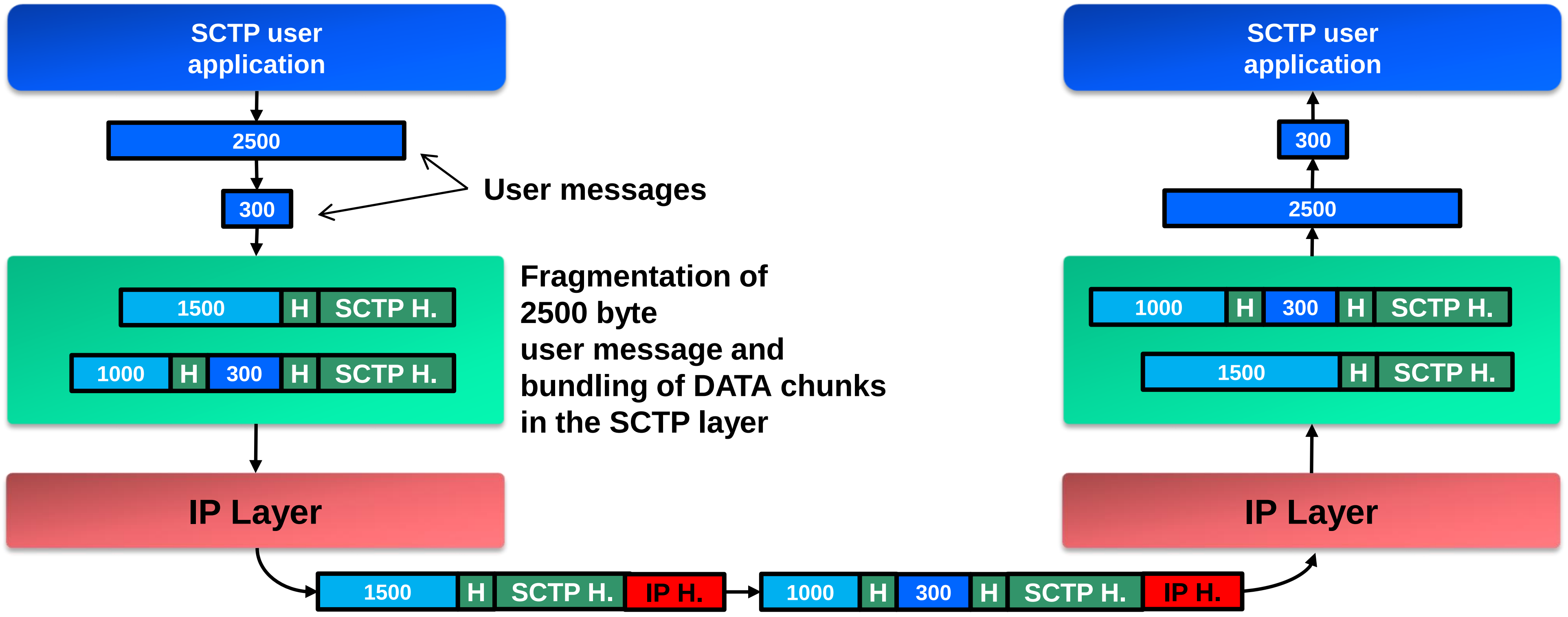
Multihoming scenario with a mobile device and a primary and secondary path:



9. SCTP fragmentation

SCTP includes an (optional) fragmentation on the transmit side to avoid fragmentation in the IP layer. An SCTP receiver must support fragmentation (be able to reassemble a fragmented packet).

An SCTP sender must fragment DATA chunks if they do not fit into an SCTP packet including chunk headers (exceed the association path MTU = smallest MTU of all paths to all peers in a multihomed scenario).



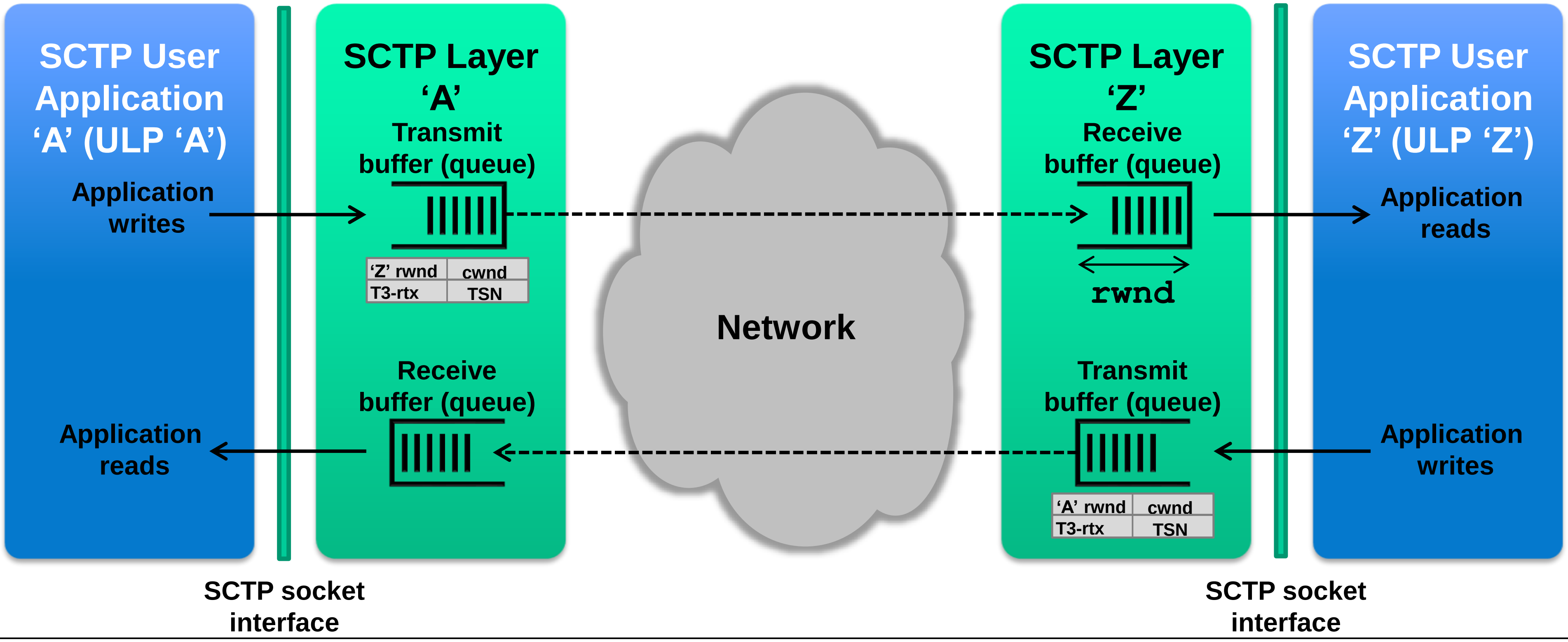
10. SCTP flow control (1/7)

Akin to TCP, SCTPs flow control mechanism is based on a receiver window size (`rwnd`).

The SCTP flow control algorithm guarantees that the receive buffer never experiences overflow (sent data always fits into the receive buffer).

The following scenarios exemplify and explain the various algorithms and mechanisms used in the SCTP flow control.

The examples use the following conceptual model:



10. SCTP flow control (2/7)

Receive window (`rwnd`):

The recieve window size `rwnd` indicates the receive buffer size. The advertised receive window size `a_rwnd` is the value of `rwnd` sent by a receiver SCTP layer to its peer STCP layer to inform it about its receive buffer size.
A sender must not send more DATA chunks than fit into `rwnd`. If `a_rwnd` reaches zero, the sender must stop sending DATA chunks.
Exception: 1 DATA chunk may be in flight (underway) if allowed by `cwnd` for window probing (see below).
The default initial value for rwnd is 1500 bytes.
An SCTP receiver increases `a_rwnd` by the amount of data delivered to the ULP and advertises new values to the SCTP transmitter.

TSN – DATA chunk counter:

Every DATA chunk has a Transmission Sequence Number (TSN). As opposed to TCP, SCTP counts full DATA chunks and not bytes (SCTP is message oriented, not stream oriented). TSNs are used by the receiver to acknowledge successful receipt of DATA chunks (in SACK chunks).

Congestion control window (`cwnd`):

The size of the congestion control window is determined by the congestion control algorithm (see below). At any given time, the amount of outstanding data (DATA chunks in flight but not yet acknowledged) must not exceed the congestion window size (`cwnd`).
If `cwnd=0`, the sender must stop sending SCTP packets.

10. SCTP flow control (3/7)

Zero window probe / window probing:

When a receiver has advertised a zero receive window (`a_rwnd=0`) and a subsequent SACK advertising a non-zero receive window is lost, the sender will be blocked forever from sending since only a new DATA chunk would trigger a new SACK packet with the non-zero rwnd advertisement.

To overcome this potential deadlock situation, a sender may send a zero window probe after RTO time elapsed if it received `a_rwnd=0`. After that, the sender should exponentially increase the zero window probe intervals. Again, if cwnd=0, the sender must not send anything (not even zero window probes).

A zero window probe may only be sent when all DATA chunks have been acknowledged and no DATA chunks are inflight.

Bundling policy:

Bundling is the process of packing different chunks (DATA chunks, control chunks such as SACK) into an SCTP packet. An SCTP sender must heed the following policies:

a. The SCTP packet size must not exceed the *association path MTU*, i.e. the smallest MTU of all paths of an SCTP association (in multihomed scenarios there are multiple paths between the SCTP peers).

b. Bundle SACK chunks with highest priority.

c. After bundling SACK chunks, if both `rwnd` and `cwnd` permit, bundle DATA chunks eligible for retransmission into the remaining space of an SCTP packet.

d. Bundle the remaining space with new DATA chunks (again if `rwnd` and `cwnd` permit).

10. SCTP flow control (4/7)

Reassembly in the receiver:

An SCTP receiver first reassembles DATA chunks into the original ULP messages, then delivers these messages to the ULP.

Silly window syndrome prevention:

The Silly Window Syndrome (SWS) can occur if both of the following are true:
a. Receive buffer almost full, receiver advertises small `rwnd` values.
b. Sender sends tinygrams, i.e. very small packets as allowed by `rwnd`.
This results in very poor transmission performance. To avoid SWS, an SCTP sender has to apply the mechanisms set forth in RFC1122 (basically the receiver has to restrain from advertising small `rwnd` values and the sender has to avoid sending tinygrams).

Transmit buffer size:

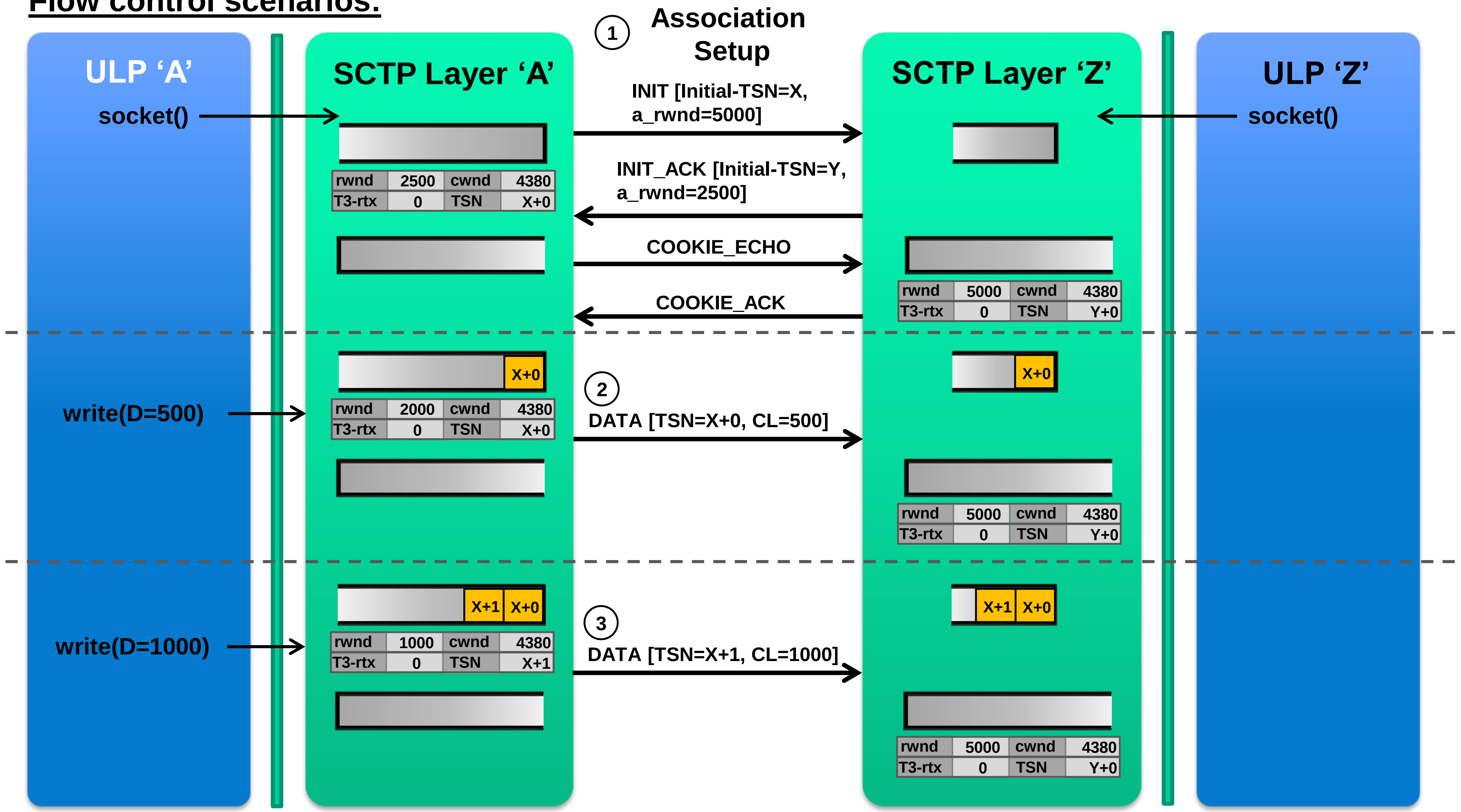
The transmit buffer in the SCTP sender may accept more data than is advertised by the SCTP reciever with `a_rwnd`. An SCTP sender reports transmit buffer overflow back to the ULP by some appropriate means (socket callback etc.). The handling of such situations is up to the ULP (outside of scope of SCTP).

Delayed ack:

The delayed ack guidelines set forth in RFC2581 should be followed by an SCTP sender (send acknowledge for every other SCTP DATA packet, no later than 200ms after reception of a DATA packet).

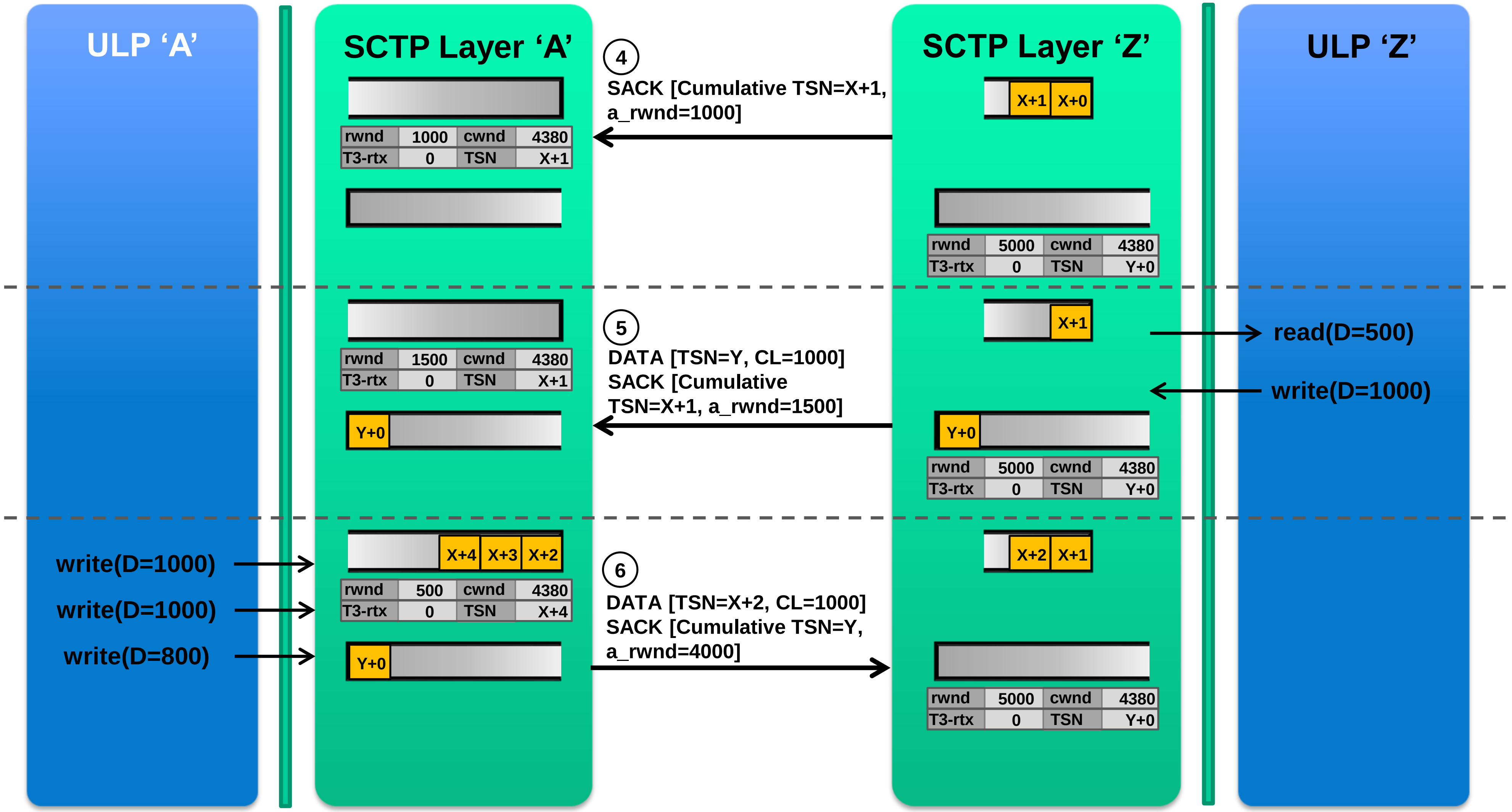
10. SCTP flow control (5/7)

Flow control scenarios:



10. SCTP flow control (6/7)

Flow control scenarios:



10. SCTP flow control (7/7)

1. SCTP association setup:
Both SCTP peers communicate their respective receive buffer size as `a_rwnd` (advertised receive window) in the assocation setup. Both SCTP peers initialize their `rwnd` variable with the `a_rwnd` advertised by the peer.

2. & 3. Application writes:
ULP ‘A’ writes an application message to the SCTP layer through the socket API.
SCTP ‘A’ stores the message in the transmit buffer along with the TSN of this chunk (X) and decrements `rwnd` by the size of the data (500 bytes).
SCTP ‘Z’ receives the DATA chunk and stores it in its receive buffer.
Since SCTP mandates the implementation of delayed acks as per RFC1122, SCTP ‘Z’ does not immediately send back a SACK chunk. Instead it waits up to 200ms in order to receive another DATA chunk so as to acknowledge both chunks with one SACK.
In the meantime, ULP ‘A’ writes another message over the API into the SCTP layer which delivers it as another DATA chunk with TSN=X+1.

4. SACK chunk:
SCTP ‘Z’ received 2 DATA chunks before the delayed ack timer expired (200ms), so it returns a SACK chunk with cumulative TSN = X+1 singaling that it successfully received all DATA chunks up until and including TSN=X+1.
The `a_rwnd` value is still 1000 because both DATA chunks X and X+1 still occupy space in the receive buffer because they have not yet been delivered to the ULP ‘Z’.

5. ULP ‘Z’ reads and writes data:
ULP ‘Z’ reads 500 bytes from the SCTP layer and thus frees up 500 bytes in the receive buffer. Right afterwards, ULP ‘Z’ writes a 1000 byte message to the SCTP layer which bundles it as a DATA chunk with a SACK chunk advertising a new `a_rwnd` value (1500 bytes free in receive buffer) and sends it to the SCTP ‘A’ layer.

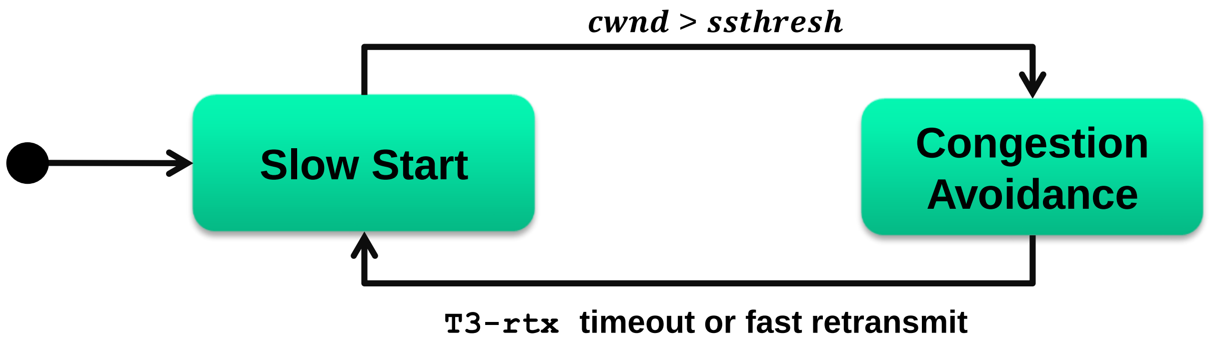
6. ULP ‘A’ writes multiple messages:
ULP ‘A’ writes two 1000 byte and one 800 byte messages to its SCTP layer. Since `a_rwnd` advertised by SCTP ‘Z’ is 1500, SCTP ‘A’ only sends the first 1000 byte message as a DATA chunk. In order to implement the delayed ack algorithm (RFC1122), SCTP ‘A’ bundles a SACK chunk advertising `a_rwnd`=4000.

11. SCTP congestion control (1/6)

Congestion control tries avoiding overload situations in network components like routers. Congestion in network components can lead to packet loss which is handled by the error control function of SCTP (see below). The goal of congestion control is to avoid packet loss in the first place.

SCTP congestion control key facts:

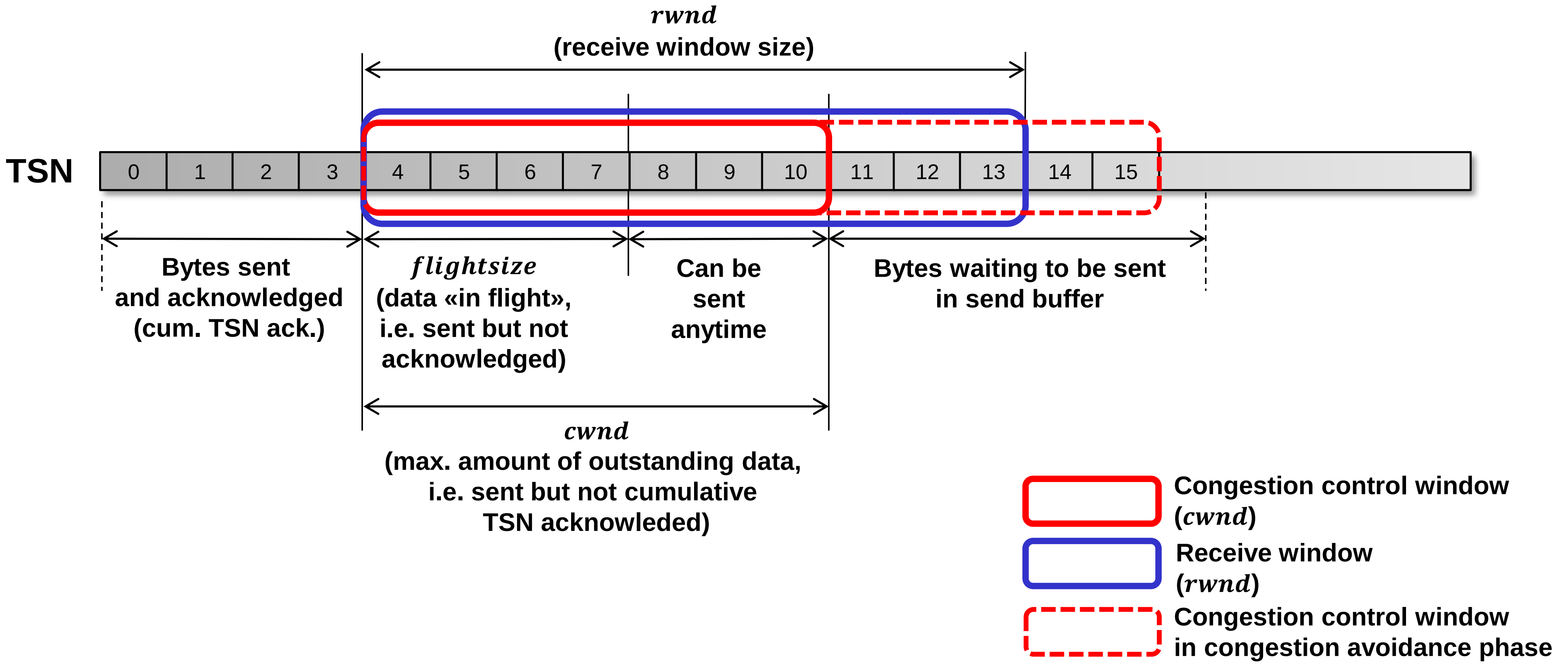
- **SCTP congestion control is based on RFC2581 with some minor modifications.**
- **Congestion control is applied to the entire association, not individual streams.**
- **SCTP maintains a separate $cwnd$ parameter for each peer destination address in multihomed scenarios.**
- **As defined in RFC2581, the transmission rate starts slowly at the beginning (slow start phase), based on feedback provided by received SACK chunks. After the slow start phase, SCTP enters the congestion avoidance phase. In case of congestion in the network, SCTP immediately reverts back to the slow start phase.**



11. SCTP congestion control (2/6)

SCTP congestion control window $cwnd$:

$cwnd$ and $rwnd$ (or a_rwnd = advertised receiver window size) define 2 windows where the smaller of the 2 determines the maximum amount of data that can be sent.
After the slow start phase, $cwnd$ is large so that $rwnd$ becomes the dominant window size.

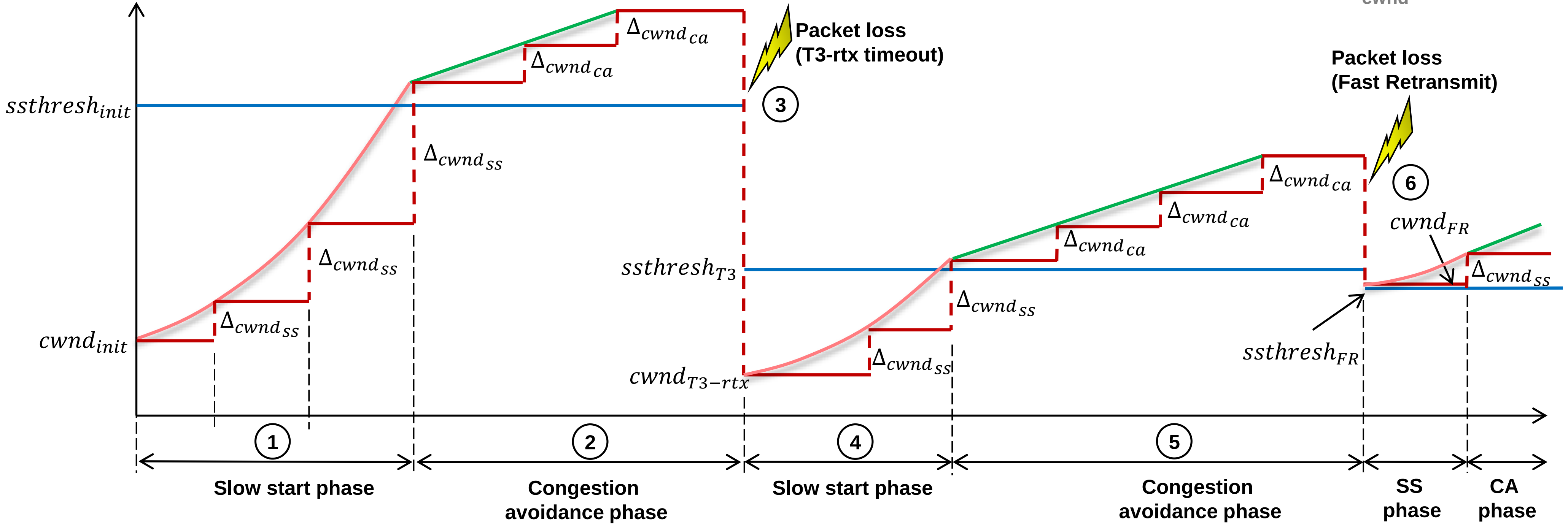

$rwnd$
(receive window size)
TSN
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Bytes sent and acknowledged (cum. TSN ack.)
$flightsize$ (data «in flight», i.e. sent but not acknowledged)
Can be sent anytime
Bytes waiting to be sent in send buffer
$cwnd$
(max. amount of outstanding data, i.e. sent but not cumulative TSN acknowleded)
Congestion control window ($cwnd$)
Receive window ($rwnd$)
Congestion control window in congestion avoidance phase

11. SCTP congestion control (3/6)

Slow start and congestion avoidance phases:

Key:

PMTU:	Association Path MTU
SS:	Slow Start
CA:	Congestion Avoidance
🟦	ssthresh
🟥	cwnd



$cwnd_{init} = min(4 * PMTU, max(2 * PMTU, 4380bytes))$

$ssthresh_{init} = \infty \ (or\ a\_rwnd)$

$\Delta_{cwnd_{ss}} = min(DATA_{acked}, PMTU)$

$\Delta_{cwnd_{ca}} = 1 * PMTU\ per\ RTT$

$cwnd_{T3-rtx} = 1 * PMTU$

$cwnd_{FR} = ssthresh_{FR}$

$ssthresh_{T3} = ssthresh_{FR} = max\left(cwnd/_{2}, 4 * PMTU\right)$

$burst_{max} = 4$

11. SCTP congestion control (4/6)

General mechanism of congestion control:

The general mechanism applied in SCTP congestion control (as per RFC2581) is to slowly increase the congestion window size $cwnd$, but to rapidly collapse the window when there are signs of congestion.
Packet loss is deemed a sign of congestion. Note, however, that this is not always true (e.g. on wireless links there may be packet loss due to radio signal interferences).
Congestion control is applied to the entire association, not individual streams. Nevertheless, $cwnd$ is maintained per destination transport address (multihomed scenarios).

Maximum burst limitation:

At any instance of time, the maximum burst limit is 4 (a maximum of 4 packets may be sent at any opportunity to send).
This may be accomplished by regulating $cwnd$ as follows:

$$if\ \left((flightsize + burst_{max} * PMTU) < cwnd\right)\ then\ cwnd = flightsize + burst_{max} * PMTU$$

Congestion control parameters:

The following parameters are involved in congestion control:

Parameter	Description
$ssthresh$	Slow start threshold (threshold between slow start and congestion avoidance phase). If $cwnd \le ssthresh$, SCTP is in the slow start phase. If $cwnd > ssthresh$, SCTP is in the congestion avoidance phase.
$cwnd$	Congestion window size. Maximum permissible number of outstanding bytes (sent data which are not yet acknowledged).
$flightsize$	Actually sent data which are not yet acknowledged.
pba	Partially bytes acknowledged. Total number of bytes acknowledged including SACK gap ack blocks.

11. SCTP congestion control (5/6)

1. Slow Start Phase:

Before any data transfer takes place, $\boldsymbol{cwnd}$ is initialized to $cwnd_{init}$.
$\boldsymbol{ssthresh}$ is initialized to some arbitrarily large value, e.g. the receiver window size ($\boldsymbol{a\_rwnd}$).
The slow start phase is defined by $\boldsymbol{cwnd \leq ssthresh}$.
$\boldsymbol{cwnd}$ is incremented by $\Delta_{cwnd_{ss}}$ when:

a. Current $\boldsymbol{cwnd}$ is fully utilized
b. Incoming SACK advances the cumulative TSN ack point
c. Sender is not in fast recovery

The slow start phase is a self clocking mechanism in that only successfully received SACK chunks increase $\boldsymbol{cwnd}$. In case of congestion (packet loss), SACKs will not be received thus $\boldsymbol{cwnd}$ is not increased.

2. Congestion Avoidance Phase:

The congestion avoidance phase is defined by $\boldsymbol{cwnd > ssthresh}$.
In this phase, $\boldsymbol{cwnd}$ is incremented linearly by $\boldsymbol{PMTU}$ as follows:

$\boldsymbol{pba = 0}$ (initial $\boldsymbol{pba}$).
Increment $\boldsymbol{pba}$ by the total amount of data bytes acknowledged in SACK chunks.
If $\boldsymbol{pba \geq \mathrm{cwnd}}$:

$\boldsymbol{\mathrm{cwnd} = \mathrm{cwnd} + PMTU}$
$\boldsymbol{pba = pba - \mathrm{cwnd}}$
When all sent data sent has been acknowledged by receiver: $\boldsymbol{pba = 0}$.

11. SCTP congestion control (6/6)

3. & 4. Packet loss (retransmission timeout) with ensuing slow start phase:

**If the retransmission timer `T3-rtx` times out due to packet loss,
$cwnd$ is immediately reduced to $cwnd_{T3-rtx}$. This limits the number of sent packets in bursts which in turn alleviates congestion in the network.
This way, SCTP reverts back to the slow start phase.
$ssthresh$ is reduced to half the last known good $cwnd$ or $4 * \mathrm{PMTU}$, whichever is larger. This makes SCTP enter the subsequent congestion avoidance phase earlier, i.e. SCTP stops incrementing $cwnd$ exponentially earlier, thus again alleviating the congestion.**

5. Congestion Avoidance Phase:

Again, once $cwnd > ssthresh$, SCTP enters the congestion avoidance phase.

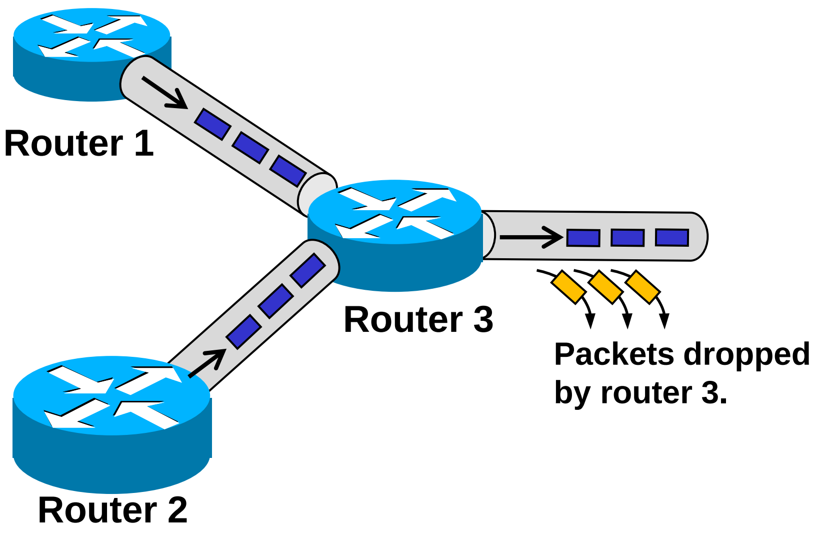
6. Packet loss with fast retransmit:

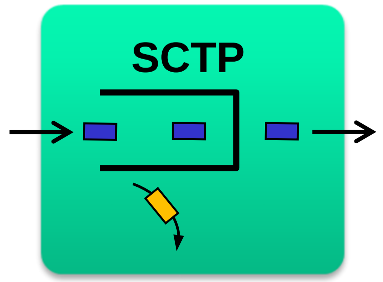
**Fast retransmit indicates slight congestion in the network.
Again, $ssthresh$ is reduced to half the last known good $cwnd$ or $4 * \mathrm{PMTU}$, whichever is larger.
In such situations, the congestion control algorithm reacts more gently in that $cwnd$ is cut in half and not down to PMTU.
Subsequently, SCTP enters the slow start phase for a short period of time (slow start phase if $cwnd \leq ssthresh$).
As soon as $cwnd > ssthresh$, SCTP enters the congestion avoidance phase again.**

12. SCTP error control (1/8)

Retransmissions of lost packets:

Packets may be lost due to buffer overflows in transit or due to bit errors in the packet itself.





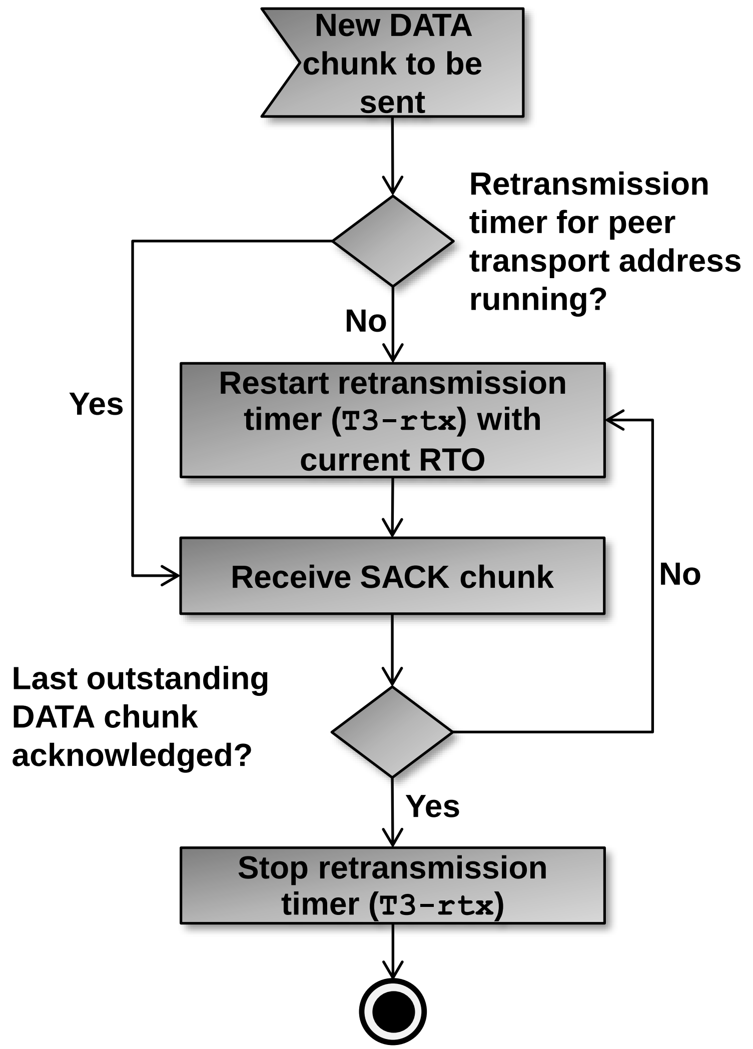
If the aggregate bandwidth temporarily exceeds the available bandwidth of an output link, the router drops excess packets.

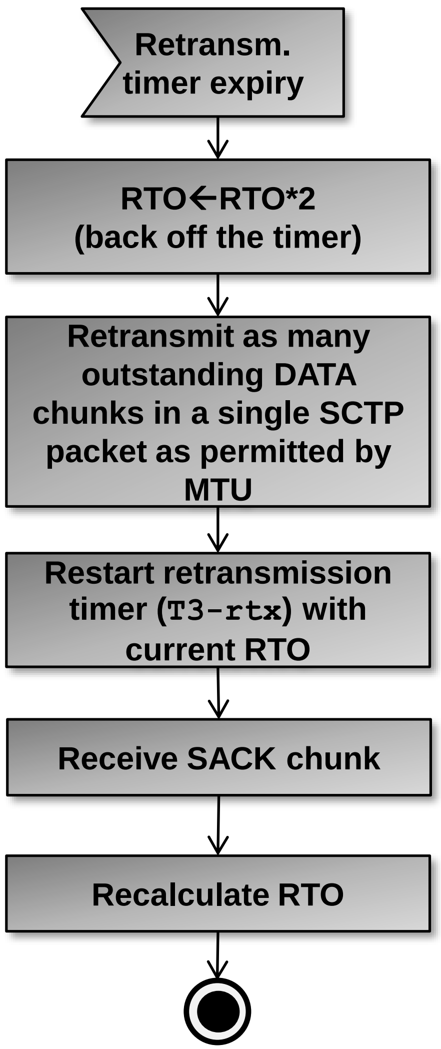
SCTP silently discards packets with an incorrect checksum (CRC32).

SCTP handles lost packets through retransmission (akin to TCP).
A retransmission timer is started for each chunk (see procedure below).
SCTP maintains a separate retransmission timer RTO for each peer transport address.
The calculation of RTO closely follows the definitions of the TCP retransmission timer value.

12. SCTP error control (2/8)

Retransmission procedure:

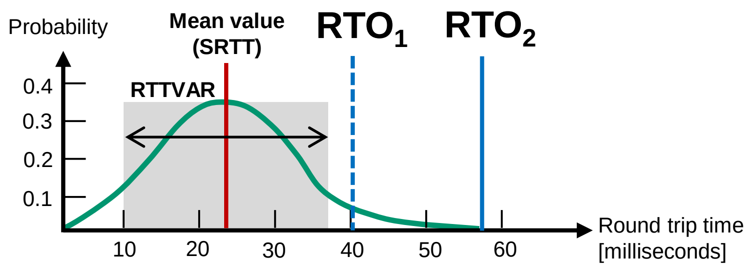

New DATA chunk to be sent
Retransmission timer for peer transport address running?
No
Yes
Restart retransmission timer (T3-rtx) with current RTO
Receive SACK chunk
No
Last outstanding DATA chunk acknowledged?
Yes
Stop retransmission timer (T3-rtx)


Retransm. timer expiry
RTO←RTO*2 (back off the timer)
Retransmit as many outstanding DATA chunks in a single SCTP packet as permitted by MTU
Restart retransmission timer (T3-rtx) with current RTO
Receive SACK chunk
Recalculate RTO

12. SCTP error control (3/8)

Retransmission timer value:

SCTP maintains a retransmission timer RTO for each peer transport address.
Since the actual round trip time is subject to constant changes, it needs to be constantly adjusted.



RTO$_1$ is too small. Many packets experience round trip times larger than RTO$_1$ which would result in inadvertent retransmissions.
RTO$_2$ is a good choice since it is slightly larger than the largest round trip time but still as small as possible.

$$SRTT_{new} = (1 - RTO_\alpha) * SRTT_{old} + RTO_\alpha * R'$$

$$RTTVAR_{new} = (1 - RTO_\beta) * RTTVAR_{old} + RTO_\beta * |SRTT_{old} - R'|$$

$$RTO_{new} = SRTT_{new} + 4 * RTTVAR_{new}$$

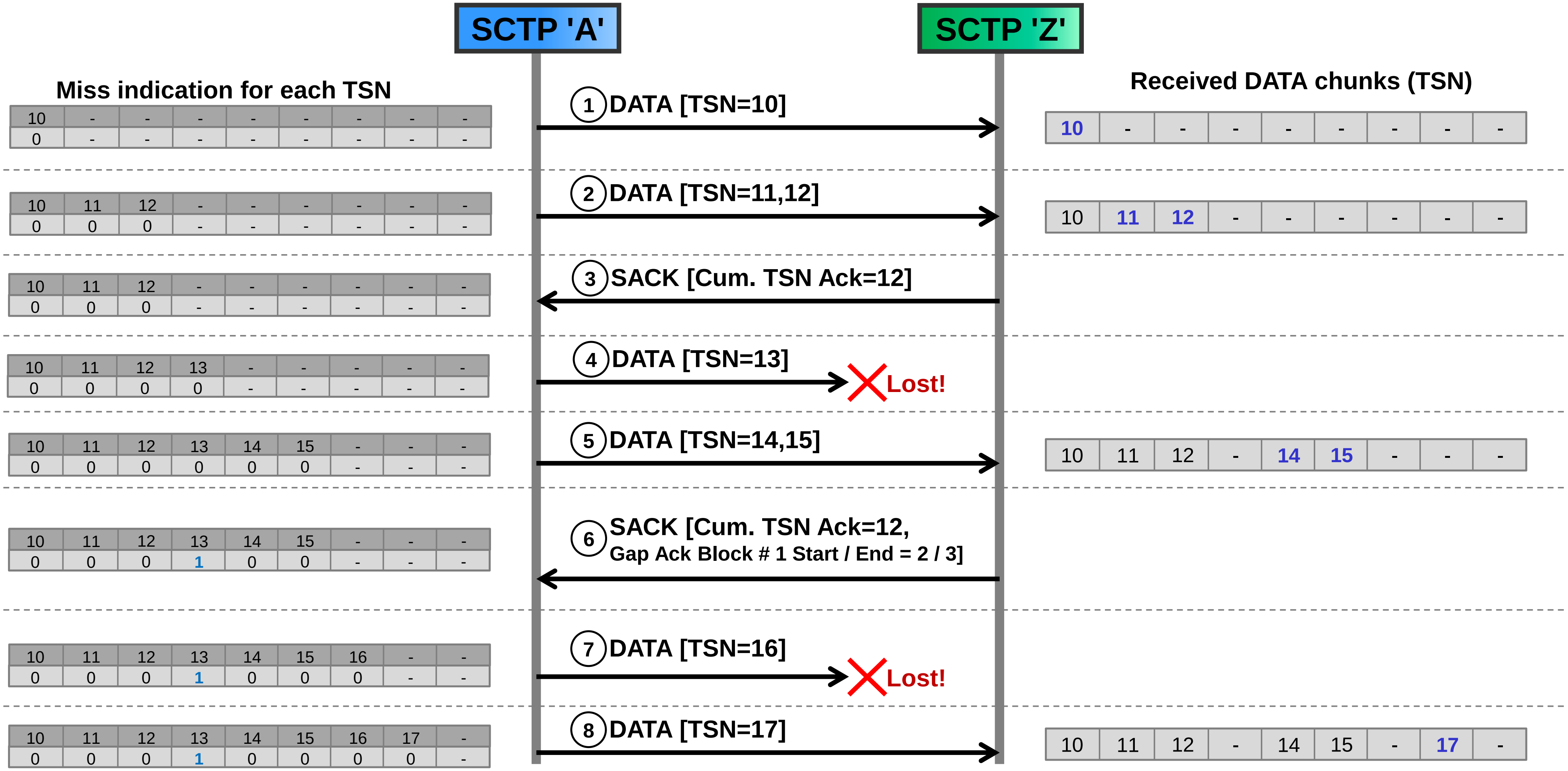
RTO is recalculated factoring in the currently smoothed («averaged») round trip time and its variation.
The weighing factors (RTO_α and RTO_β) determine the speed of adjustment of RTO to new measurement values for the round trip time (R').

where
$RTTVAR = current\ Round\ Trim\ Time\ variation$
$SRTT = current\ Smoothed\ Round\ Trip\ Time$
$RTO = current\ Retransmission\ TimeOut\ timer\ value$
$RTO_\alpha = weighing\ factor\ (default\ value = 1/8)$
$RTO_\beta = weighing\ factor\ (default\ value = 1/4)$
$R' = current\ measured\ round\ trip\ time$

12. SCTP error control (4/8)

Fast Retransmit:

SCTP employs a similar mechanism for fast retransmits as TCP as shown below.



Miss indication for each TSN

10	-	-	-	-	-	-	-	-
0	-	-	-	-	-	-	-	-

10	11	12	-	-	-	-	-	-
0	0	0	-	-	-	-	-	-

10	11	12	-	-	-	-	-	-
0	0	0	-	-	-	-	-	-

10	11	12	13	-	-	-	-	-
0	0	0	0	-	-	-	-	-

10	11	12	13	14	15	-	-	-
0	0	0	0	0	0	-	-	-

10	11	12	13	14	15	-	-	-
0	0	0	1	0	0	-	-	-

10	11	12	13	14	15	16	-	-
0	0	0	1	0	0	0	-	-

10	11	12	13	14	15	16	17	-
0	0	0	1	0	0	0	0	-

Received DATA chunks (TSN)

10	-	-	-	-	-	-	-	-

10	11	12	-	-	-	-	-	-

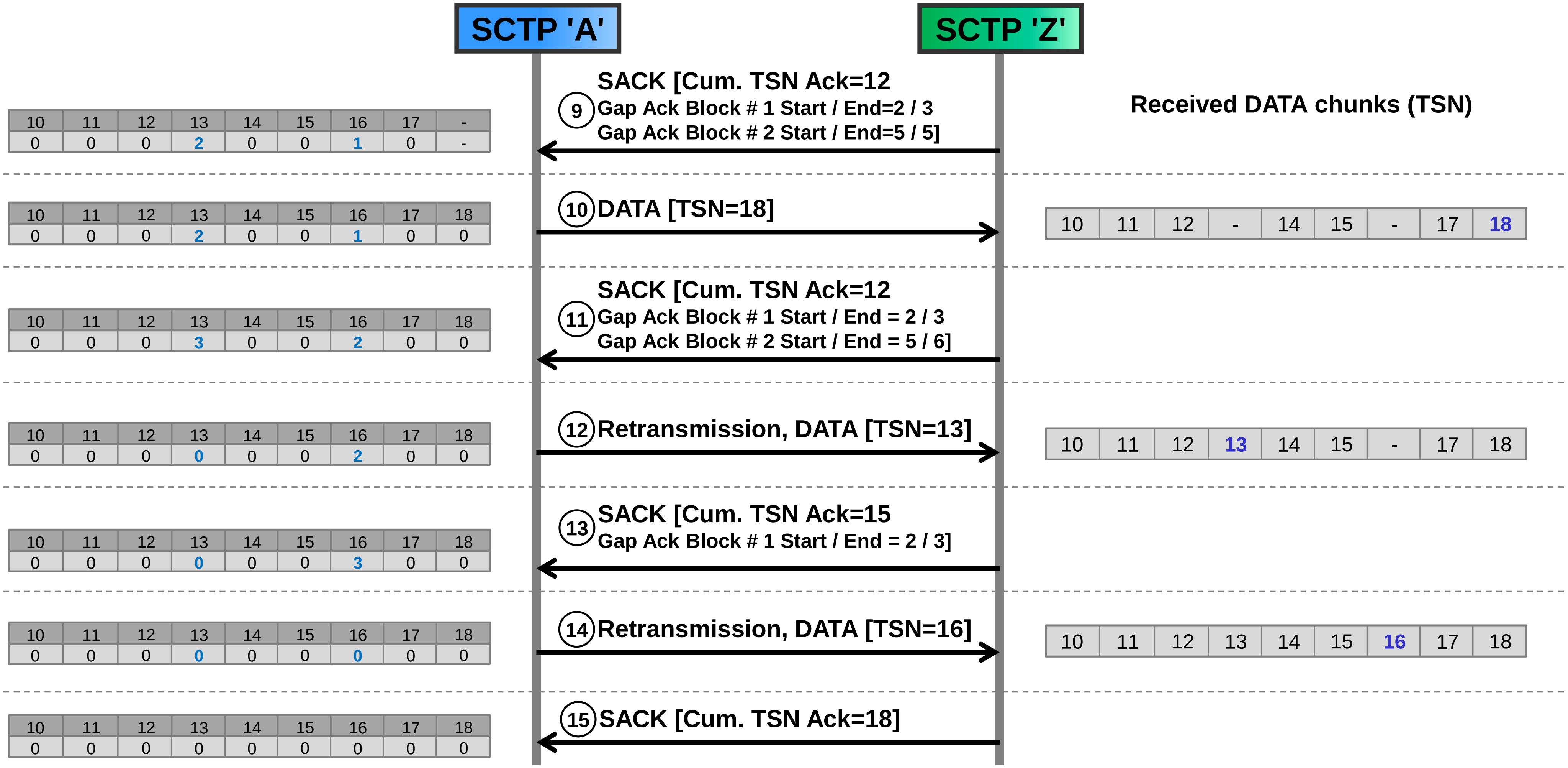
10	11	12	-	14	15	-	-	-

10	11	12	-	14	15	-	17	-

12. SCTP error control (5/8)

Fast Retransmit:

SCTP employs a similar mechanism for fast retransmits as TCP as shown below.


SCTP 'A'
SCTP 'Z'
Received DATA chunks (TSN)
9 SACK [Cum. TSN Ack=12 Gap Ack Block # 1 Start / End=2 / 3 Gap Ack Block # 2 Start / End=5 / 5]
10 DATA [TSN=18]
11 SACK [Cum. TSN Ack=12 Gap Ack Block # 1 Start / End = 2 / 3 Gap Ack Block # 2 Start / End = 5 / 6]
12 Retransmission, DATA [TSN=13]
13 SACK [Cum. TSN Ack=15 Gap Ack Block # 1 Start / End = 2 / 3]
14 Retransmission, DATA [TSN=16]
15 SACK [Cum. TSN Ack=18]

10	11	12	13	14	15	16	17	-
0	0	0	2	0	0	1	0	-

10	11	12	13	14	15	16	17	18
0	0	0	2	0	0	1	0	0

10	11	12	-	14	15	-	17	18

10	11	12	13	14	15	16	17	18
0	0	0	3	0	0	2	0	0

10	11	12	13	14	15	16	17	18
0	0	0	0	0	0	2	0	0

10	11	12	13	14	15	-	17	18

10	11	12	13	14	15	16	17	18
0	0	0	0	0	0	3	0	0

10	11	12	13	14	15	16	17	18
0	0	0	0	0	0	0	0	0

10	11	12	13	14	15	16	17	18

10	11	12	13	14	15	16	17	18
0	0	0	0	0	0	0	0	0

12. SCTP error control (6/8)

Fast Retransmit:

Step by step explanation of a *Fast Retransmit* procedure.

1. & 2. DATA chunks:
SCTP 'A' sends DATA chunks with TSNs 10, 11 and 12 in 2 separate SCTP packets. Both packets are correctly received by SCTP 'Z'.

3. SACK chunk:
Following the guidelines set forth in RFC2581 (Delayed Ack: Acknowledge for every other SCTP DATA packet, no later than 200ms after reception of a DATA packet), SCTP 'Z' sends back a SACK chunk acknowledging the correct reception of DATA chunks 10, 11, and 12 (Cumulative TSN Ack equals 12 thus acknowledging correct reception of all TSNs up until and including DATA chunk 12).

4. Lost SCTP packet:
The SCTP packet with TSN=13 is lost. Such packet losses are typical of routers experiencing slight congestion on their output buffers where they drop a few excess packets.

5. Successful delivery of next DATA chunk:
SCTP DATA chunks with TSNs 14 and 15 are again successfully received by SCTP 'Z'.

6. SCTP 'Z' signals missing TSNs:
SCTP 'Z' sends back a SACK with cumulative TSN Ack = 12 (all TSNs including 12 correctly received) and a gap ack block with start=2 and end=3 indicating that TSN=14 (12+2) and TSN=15 (12+3) have been correctly received. This in turn signals to SCTP 'A' that TSN=13 has been lost (gap). SCTP 'A' increments the miss indication counter for TSN=13.

7. Lost SCTP packet again:
The SCTP packet with TSN=16 is lost.

8. Successful reception of next DATA chunk:
The DATA chunk with TSN=17 is correctly received again by SCTP 'Z'.

12. SCTP error control (7/8)

Fast Retransmit:

Step by step explanation of a Fast Retransmit procedure.

9. SCTP 'Z' signals missing TSNs:
SCTP 'Z' sends back another SACK, now with 2 gap ack blocks.
The first gap ack block acknowledges the reception of TSN=14 and TSN=15, the second gap ack block acknowledges the reception of TSN=17 (12+5). SCTP 'A' increments the miss indication counters for TSN=13 and TSN=16.

10. Successful reception of next DATA chunk:
The DATA chunk with TSN=18 is correctly received again by SCTP 'Z'.

11. SACK including acknowledgement for TSN=18:
SCTP 'Z' sends back another SACK with 2 gap ack blocks, the first with the same start and end numbers as in step 9, the second with start=5 (12+5) and end=6 (12+6). SCTP 'A' again increments the miss indication counters for TSN=13 and TSN=16.

12. Retransmission of TSN=13:
The miss indication counter for TSN=13 reaches 3, thus TSN=13 is eligible for retransmission. SCTP 'A' retransmits the DATA chunk with TSN=13 which is correctly received by SCTP 'Z'.

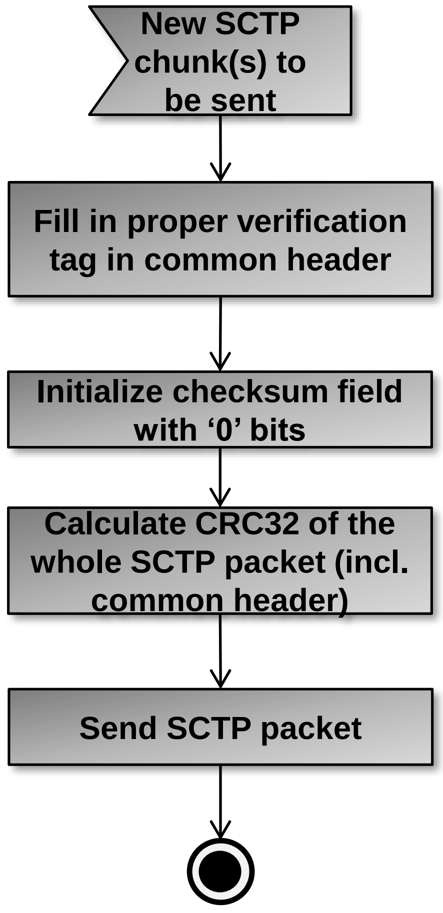
13. & 14. SACK with Cumulative Ack TSN=15 and retransmission of TSN=16:
SCTP 'Z' signals the correct reception of all DATA chunks up until and including TSN=15. The gap ack block with start=2 (15+2) and end=3 (15+3) indicates that TSN=17 and TSN=18 were successfully received and thus that TSN=16 is still missing. SCTP 'A' increments the miss indication counter for TSN=16 which reaches 3 thus triggering the retransmission of TSN=16.

15. Last SACK of fast retransmission procedure:
Finally, SCTP 'Z' sends back a SACK signalling the correct reception of all DATA chunks up until and including TSN=18.
SCTP 'Z' reverts to delayed acknowledge mode (RFC2581) and SCTP 'A' exits the fast retransmit mode.

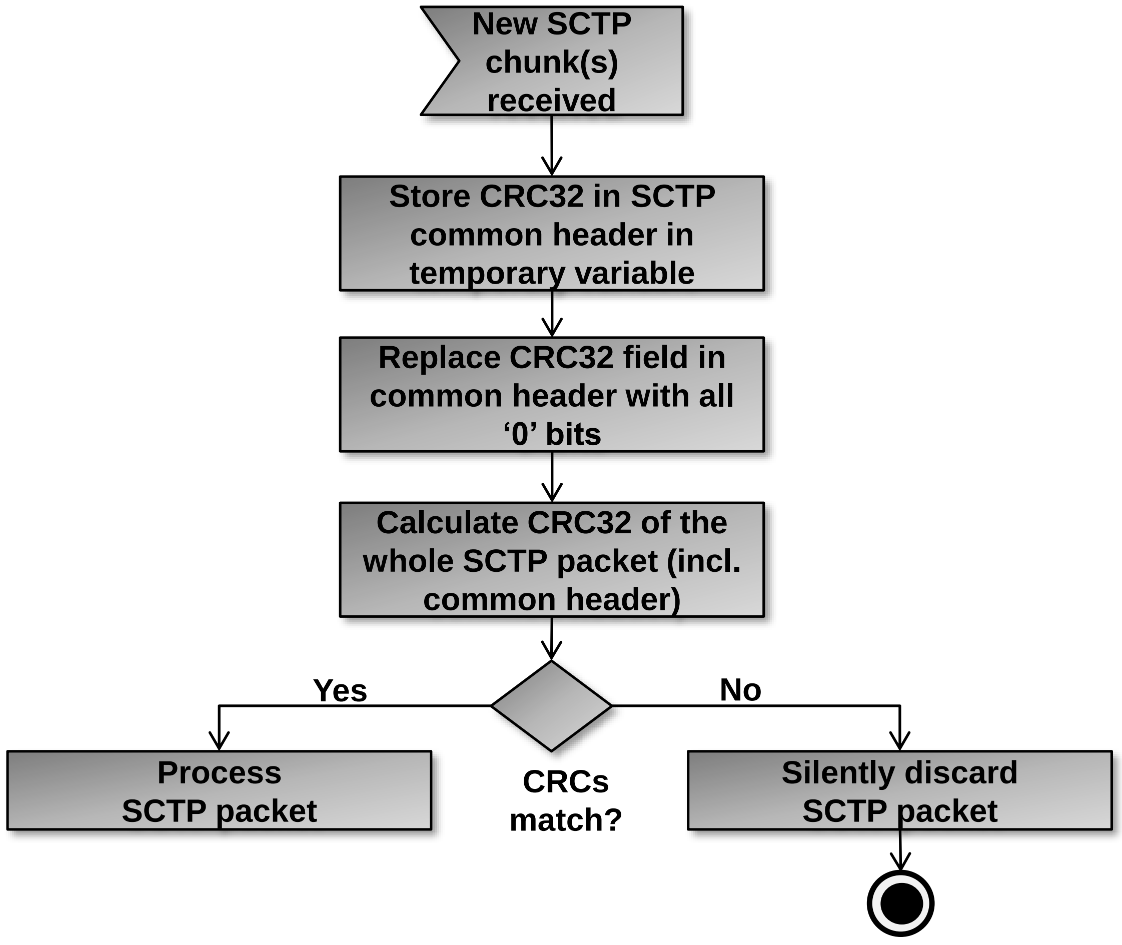
12. SCTP error control (8/8)

SCTP uses a CRC32 checksum as per RFC3309 to provide data integrity of SCTP packets.

SCTP sender CRC32 handling:


New SCTP chunk(s) to be sent
Fill in proper verification tag in common header
Initialize checksum field with '0' bits
Calculate CRC32 of the whole SCTP packet (incl. common header)
Send SCTP packet

SCTP receiver CRC32 handling:


New SCTP chunk(s) received
Store CRC32 in SCTP common header in temporary variable
Replace CRC32 field in common header with all '0' bits
Calculate CRC32 of the whole SCTP packet (incl. common header)
Yes
No
CRCs match?
Process SCTP packet
Silently discard SCTP packet

13. Security with SCTP

Security (encryption, authentication etc.) was not initially designed into SCTP.
2 first approaches (SCTP over IPSec and TLS over IPSec) have limitations and are only stop-gap solutions.
Recent activities in integrating security into the SCTP protocol itself are funneled into an IETF draft and may eventually become a standard.
DTLS over SCTP is an alternative based on the adoption of DTLS.

Security Protocol	Standard	Description
SCTP over IPSec	RFC3554	• Does not support all SCTP features (no multihoming). • 1 IPSec Security Association per IP address needed (scalability issue).
TLS over SCTP	RFC3436	• Does not support all SCTP features (e.g. no unordered delivery). • Control chunks and SCTP header not secured, only DATA chunks. • 1 TLS connection per SCTP stream (scalability issue). • Each DATA chunk (ULP message) individually secured (performance issue).
Secure SCTP (S-SCTP)	IETF Draft	• No TLS or IPSec needed. • Security built into the SCTP protocol. • Fully compatible with RFC4960.
DTLS over SCTP	RFC6083	• Datagram Transport Layer Security for SCTP • Provides almost all transport features of SCTP • Limitations: • Limitation of maximum user message size of 2^14 bytes • SCTP-AUTH not possible since authentication done by DTLS

14. SCTP support in different OSs and platforms

There is varying support for SCTP by the different OSs and platforms.

Where native (kernel) support is not available, user space SCTP libraries such as `sctplib` (Linux, FreeBSD, Mac OS X, Solaris, Windows) or `SctpDrvcan` (Windows) can be used instead.

OS / Platform	SCTP Support	Description
Linux	Yes	SCTP supported since kernel 2.4. `lksctp-tools` is a user space library that provides a specific SCTP socket interface. Without `lksctp-tools`, only the standard (SCTP agnostic) socket interface is available.
FreeBSD / OpenBSD	Yes	Since version 7.0.
Mac OSX	Yes	Since OS X version 10.7 Lion.
Solaris	Yes	Since Solaris 10.
Windows	No	No Windows version including Windows 7, 8, 10 and Windows RT supports SCTP. Microsoft does not have any plans to add native SCTP (lack of customer demand they say).
.Net	No	No support for SCTP in .Net framework including version 4.6.2.
Java	(Yes)	API available under com.sun.nio.sctp since Java 7. Requires native SCTP (kernel) support by the underlying OS.
Cisco IOS	Yes	Since IOS 12.
QNX Neutrino Real Time OS	Yes	Since 6.3.0.
Android	Yes	Needs the activation of the `lksctp` libary in the Linux kernel.

15. SCTP API in Linux (socket interface)

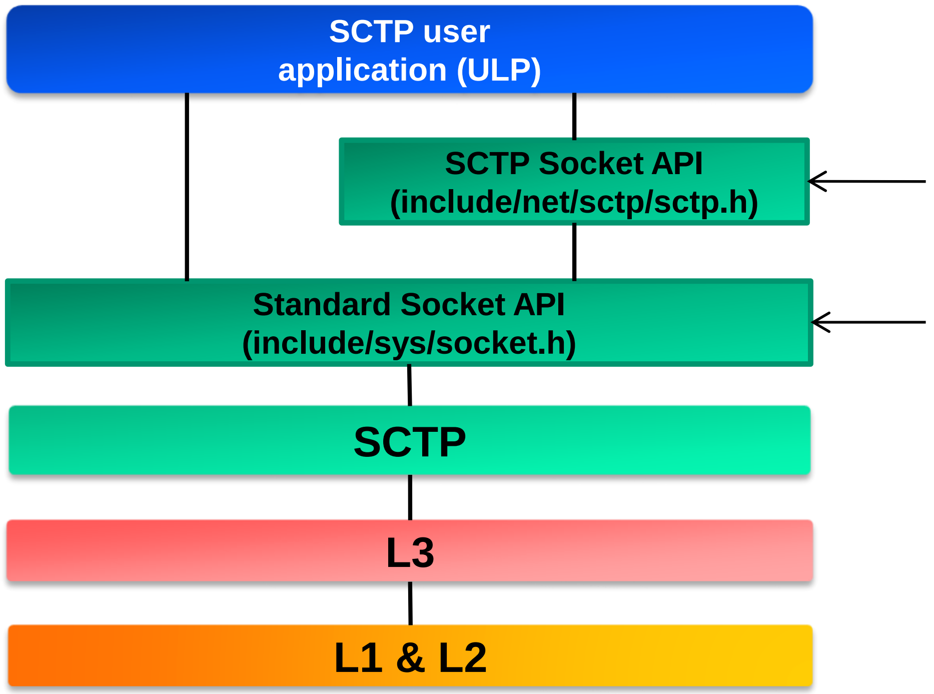
On Linux, 2 APIs are available for using SCTP sockets.

1. Standard socket interface

- **Linux header file: include/sys/socket.h**
- `listenSock = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP);`

2. Socket extensions for SCTP:

- **Defined in RFC6458**
- **Linux header file: include/net/sctp/sctp.h**


SCTP user application (ULP)
SCTP Socket API (include/net/sctp/sctp.h)
Standard Socket API (include/sys/socket.h)
SCTP
L3
L1 & L2

SCTP specific API.
Full support of all SCTP features.

Standard socket features only.
SCTP features like streams not available.

SCTP – Stream Control Transmission Protocol

Glossary

Term	Description	Term	Description
APDU	Application PDU	PDU	Protocol Data Unit
API	Application Programming Interface	PMTU	Path MTU
CA	Congestion Avoidance	RTO	Retransmission TimeOut
CRC	Cyclic Redundancy Check	SACK	Selective ACK
DTLS	Datagram TLS	SCTP	Stream Control Transmission Protocol
HTTP	Hypertext Transmission Protocol	SS	Slow Start
IETF	Internet Engineering Task Force	TCB	Transport Control Block
IP	Internet Protocol	TCP	Transmission Control Protocol
IPSec	IP Security	TLS	Transport Layer Security
L1, L2, L3	Layer 1, Layer 2, Layer 3	TSN	Transmission Sequence Number
MTU	Maximum Transfer Unit	ULP	Upper Layer Protocol
OS	Operating System	UM	User Message
OSI	Open Systems Interconnect		

Change History

Version	Date of Publishing	Description
2.10	2017-05-30	Previous version
2.20	2017-10-13	Page 43 corrected RFC number for SCTP over IPSec. Page 43 added DTLS over SCTP RFC6083. Page 44 added latest Windows and .Net versions lacking support for SCTP. Page 46 added glossary.