**Chapter 16: Stream Control Transmission Protocol**

Table of Contents

[16.2 SCTP Services 3](#_Toc92544604)

[Multiple Streams 3](#_Toc92544605)

[Multihoming 4](#_Toc92544606)

[16.3 SCTP Features 5](#_Toc92544607)

[Transmission Sequence Number 5](#_Toc92544608)

[Stream Identifier 5](#_Toc92544609)

[Stream Sequence Number 5](#_Toc92544610)

[Packets 6](#_Toc92544611)

[16.4 Packet Format 8](#_Toc92544612)

[Chunks 8](#_Toc92544613)

[16.5 An SCTP Association 10](#_Toc92544614)

[Association Establishment 10](#_Toc92544615)

[Association Termination 11](#_Toc92544616)

The **Stream Control Transmission Protocol** (SCTP) is a new transport layer protocol that is designed to improve upon TCP.

We originally used **UDP**, which was **message-oriented**, but **unreliable**. To deal with the unreliability, we moved to **TCP**, which was **reliable**, but **byte-oriented**. A message-oriented protocol preserves the **message boundaries**. When a process sends a message, that message is just encapsulated in a user datagram and forwarded. This is useful in many cases. **UDP** was capable of this, but TCP was not.

**SCTP** combines the best features of both UDP and TCP. It is **message-oriented** and also **reliable**. Over the following pages, we will see that SCTP is mostly the **same as TCP**, with very minor changes. There are a few new features added, which are the ones we will be discussing. Areas where SCTP is the same as TCP are not discussed here.

## 16.2 SCTP Services

SCTP offers several services:

* Process-to-Process Communication
* Multiple Streams
* Multihoming
* Full-Duplex Communication
* Connection-Oriented Service
* Reliable Service

As can be seen, the only differences from TCP are the Multiple Streams service and the Multihoming service.

### Multiple Streams

**TCP** is a **stream-oriented** protocol. This means that each connection between a **TCP client** and a **TCP server** is a **stream**. The problem with streams however, is that if there is any issue at any point, then the stream gets **blocked**.

**SCTP** solves this by allowing **multiple streams**. Data is sent by the processes to specific streams. If **one gets blocked**, communication does not completely stop, since the **others are still active**. All the streams are combined under a **single connection**, called an **association** in SCTP terminology.

### Multihoming

It is possible to connect a client or a server to **multiple IP addresses**. Such a host is called a **multihomed host**.

**TCP** only supports a **single source** and a **single destination** IP address, so for multihomed devices, just **one** of the IP addresses on either end will be used. An **SCTP association** on the other hand, supports **multihoming**, which allows a host to define **multiple IP addresses** on either end of the association. This is **fault-tolerant**, since if one path fails, another can be used. If there are 2 IP addresses on either end, there are a total of 4 possible paths.

Note however, that SCTP does not currently allow **load-sharing**, meaning multiple paths cannot be used at the same time to send data. Only **one path** is used, with the others serving as **backups**.

## 16.3 SCTP Features

### Transmission Sequence Number

In SCTP, the unit of data is a **data chunk**. One data chunk may or may not consist of one message, due to fragmentation. Each data chunk is numbered using a **Transmission Sequence Number** (TSN). This can be directly compared to the sequence numbers in TCP.

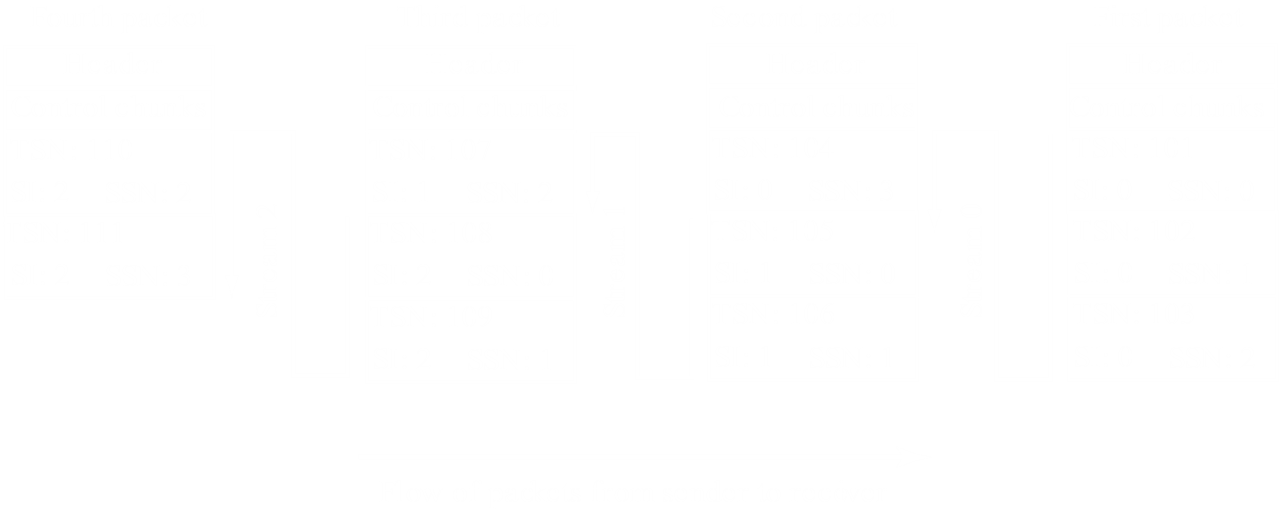
TSNs are **32 bits** long and **initialized randomly** between and . Each data chunk has its TSN in its **header**.

### Stream Identifier

Since there are **multiple streams** in SCTP, we need to identify which stream a particular data chunk belongs to. This is done using the **Stream Identifier** (SI). The SI value of a data chunk is included in its **header**. This is a **16 bit** value starting at .

### Stream Sequence Number

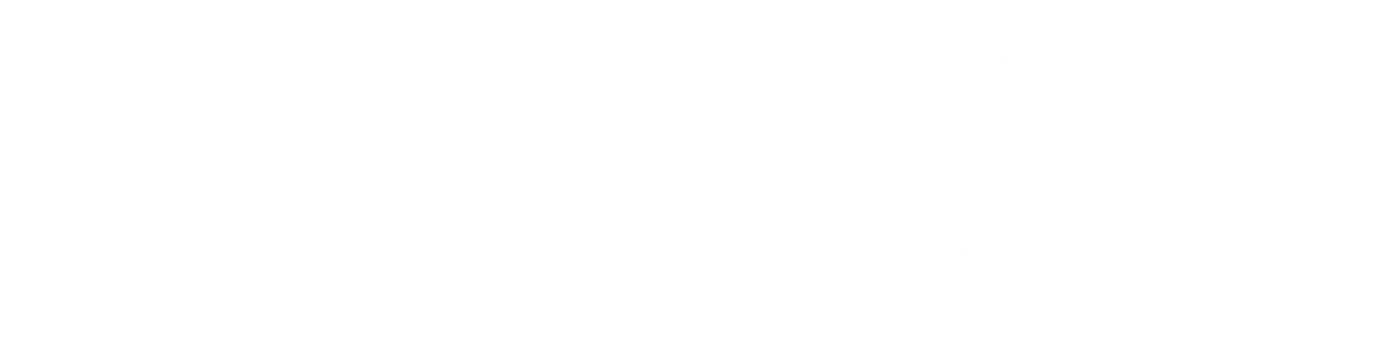
SCTP **maintains the order** of data chunks within each stream. This means that there is also a number that defines the position of each data chunk **within a stream**. This value is the **Stream Sequence Number** (SSN).



The diagram above should prove useful in understanding TSN, SI and SSN. In SCTP, we combine chunks into packets. However, notice that **not all chunks in a packet** necessarily belong to the **same stream**. This tells us that the actual **association** transfers **packets**, not chunks.

### Packets

In SCTP, instead of segments, we use **packets**. A single packet may contain **several data chunks** and **control chunks**, which contain control information.

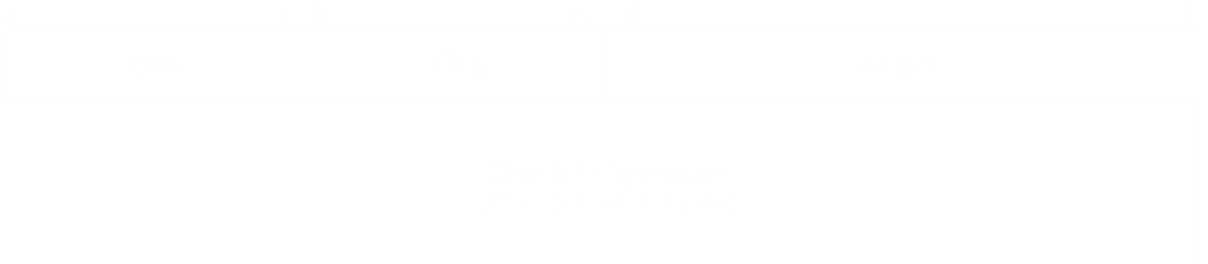


There are several changes in the format of SCTP packets compared to TCP segments:

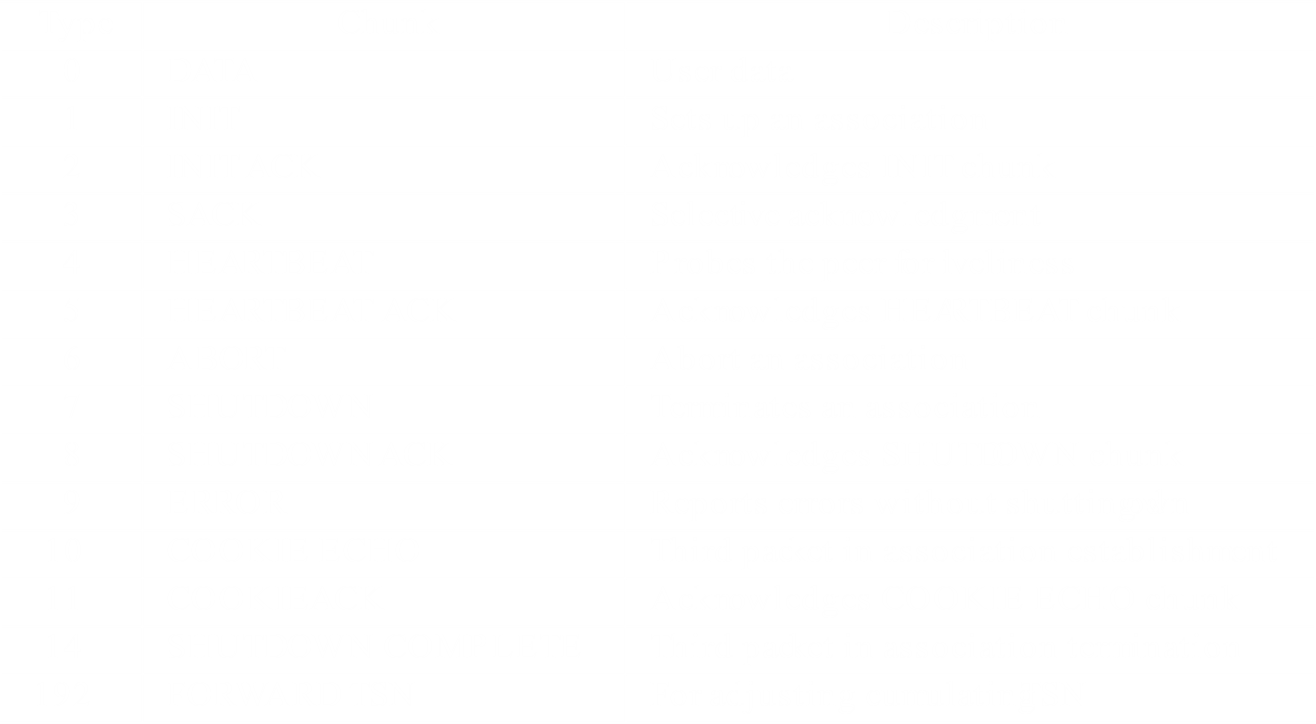
1. The **control information** is not directly included in the header, but instead added as separate **control chunks**. There can be multiple types of control chunks.
2. The **data** is not treated as a single entity, but as **separate chunks**, which can belong to **different streams**.
3. **Options** no longer exist. Options are handled using **new chunk types**.
4. The general **header** is **12 bytes**, compared to the 20 bytes of mandatory header in TCP. This header includes the **TSN**, **ACK number** and **window size**. There is **no header length** field, since the length is fixed and **no urgent pointer**.
5. The **checksum** is **32 bits**, compared to 16 bits in TCP.
6. The **verification tag** is new and is used to identify an **association**. In TCP, we used the IP address and port number to identify a connection, but we cannot do that here because **multihoming** exists. This also gets rid of an issue in TCP where successive connections using the same port number might cause older packets to go to the new connection. This was previously dealt with by using significantly different port numbers.
7. The **sequence number** from TCP is no longer necessary.
8. Some TCP segments carried control information, like SYN and FIN, which consumed a sequence number. **Control chunks** in SCTP **never use** TSN, IS or SSN numbers, since those belong to **data chunks**, not the packet.

## 16.4 Packet Format

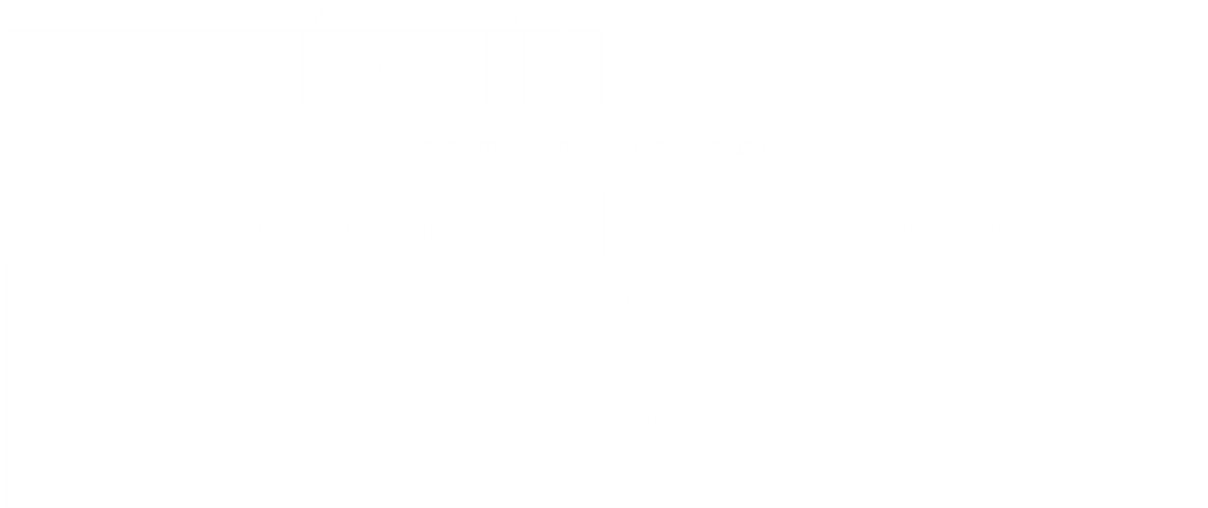
### Chunks



Chunks can be of **256 types**. Some are given below:



For a **data chunk** for example, we have the following format:

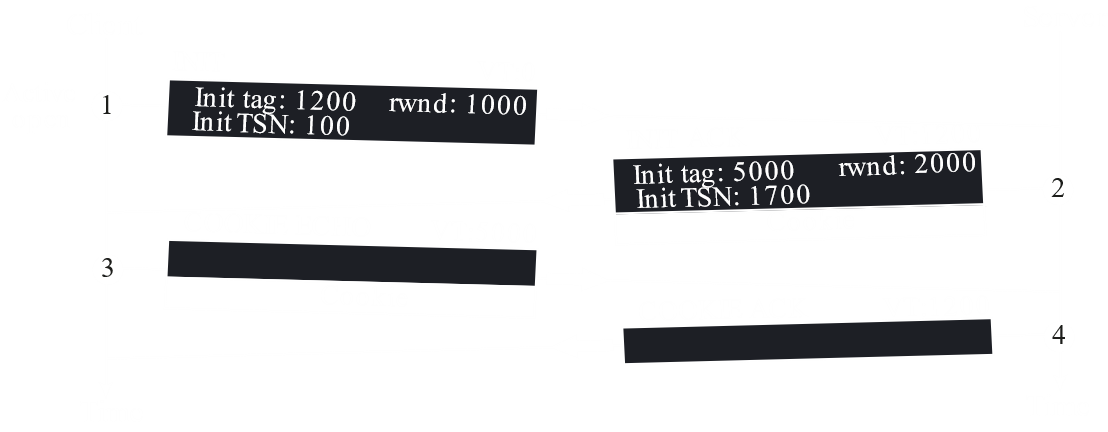


Notice that the **flags** are shown for a specific chunk type, meaning **different chunk types** can have **different flags**.

Here, the **U (unordered) flag** is set if data is **out of order**. This is similar to the **urgent flag** from TCP. The **B (beginning)** and **E (end)** flags tell us about **fragmentation**. A specific chunk could contain the entire message, the beginning of a message, the middle of a message or the end of a message.

## 16.5 An SCTP Association

### Association Establishment



**Association establishment** involves a **four-way handshake**.

1. Assuming that a **passive open** on the server side has already taken place, the connection begins with an **active open** on the client’s side. The first packet contains an **INIT** chunk. The **verification tag** is set to **0**, since no tag has yet been established for this direction. The **INIT tag** is given a value that will be used by the verification tags from the opposite direction. The **initial TSN** and **rwnd** values are also defined.
2. In response to the above, the server sends a packet with an **INIT ACK** chunk. Here, again, the **INIT tag** is set which will now be used as the verification tag from the opposite direction, and the **initial TSN** and **rwnd** values are also set. Additionally, it also sends a **cookie**, which is a **cryptographic** piece of information that can only be read by the server itself.

Note that the **ACK value** in SCTP indicates that packet number that has **already been received**, not what is being expected next.

1. The client now sends a **COOKIE ECHO** chunk, which is just the cookie that the server sent in the last step. This allows for **verification**, thus avoiding attacks like SYN attacks, which we saw with TCP.

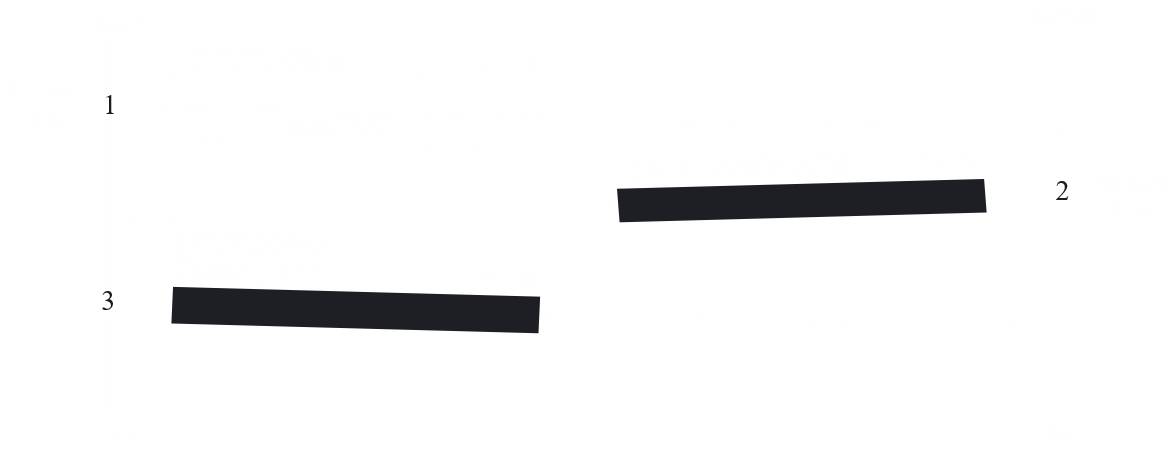
The server **does not reserve any resources** for the client until it receives this packet. It does not even store the cookie it sent. Instead, when it receives a COOKIE ECHO chunk, it **creates the cookie again** and verifies it. This means that the cookie is somehow connected to the client’s information.

1. Finally, the server sends a **COOKIE ACK**, completing the association establishment.

Note that both COOKIE ECHO and COOKIE ACK packets are allowed to also contain **data chunks**.

### Association Termination

Unlike TCP, a **half-closed association** cannot exist. Thus, we only need **three packets**, a SHUTDOWN chunk in one packet, a SHUTDOWN ACK chunk in another packet and a SHUTDOWN COMPLETE chunk in a third packet.



Instead of **graceful termination** like above, the association can also be **aborted** by either end for a variety of reasons. This just involves sending an ABORT chunk in a packet.

