# SCTP

## IN THEORY AND PRACTICE

A QUICK INTRODUCTION TO THE SCTP PROTOCOL AND ITS SOCKET INTERFACE IN LINUX

TSVETOMIR DIMITROV

# SCTP in theory and practice

## A quick introduction to the SCTP protocol and its socket interface in Linux

Tsvetomir Dimitrov

This book is for sale at http://leanpub.com/sctp

This version was published on 2023-04-15


This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

# Contents

**Preface** . . . . . **1**
Who is this book for . . . . . 1
How to read the book . . . . . 2
Code samples and PCAP files . . . . . 2
THANK YOU . . . . . 2

**Chapter 1: Quick introduction to the SCTP protocol** . . . . . **3**
RFC 4960: The specification of the SCTP protocol . . . . . 3
Some SCTP related key terms and abbreviations . . . . . 4
Sample PCAP files . . . . . 5
SCTP packet structure . . . . . 6
An example of common header and chunks encoding . . . . . 8
Key take-aways . . . . . 11

**Chapter 2: Association initialisation** . . . . . **12**
Association initialisation . . . . . 12
INIT chunk . . . . . 13
INIT ACK chunk . . . . . 14
COOKIE ECHO chunk . . . . . 17
COOKIE ACK chunk . . . . . 18
Key take-aways . . . . . 18

**Chapter 3: Data transfer in SCTP** . . . . . **20**
DATA chunk . . . . . 20
SACK chunk . . . . . 20
Data transfer procedures . . . . . 20
Key take-aways . . . . . 21

**Chapter 4: Association teardown** . . . . . **22**
Failure detection . . . . . 22
Endpoint Failure Detection . . . . . 22
Path Failure Detection . . . . . 22
Path Heartbeat . . . . . 22
HEARTBEAT chunk . . . . . 23
HEARTBEAT ACK chunk . . . . . 23

Association teardown . . . 23
Association shutdown . . . 24
Key take-aways . . . 24

**Chapter 5: Multi-homing . . . 25**
How multi-homing works . . . 25
Time to check the specification . . . 25
Path verification . . . 25
Primary path . . . 26
Time to check the specification . . . 26
Association termination . . . 26
Example . . . 26
Key take-aways . . . 26

**Chapter 6: Sockets introduction . . . 27**
About the code . . . 27
SCTP sockets in Linux . . . 27
Building the code . . . 29
The code . . . 31
Key take-aways . . . 37

**Chapter 7: SCTP Linux API: One-to-many style interface . . . 38**
One-to-many in a nutshell . . . 38
The code . . . 38
Key take-aways . . . 39

**Chapter 8: Working with SCTP ancillary data . . . 40**
Introduction . . . 40
How ancillary data works . . . 40
Code snippets showing how to accomplish a few common tasks . . . 40
Ancillary data in SCTP protocol . . . 40
The code . . . 42
Key take-aways . . . 42

**Chapter 9: SCTP notifications in Linux . . . 43**
Introduction . . . 43
How notifications work . . . 43
Enabling SCTP notifications . . . 43
Notification structure . . . 43
Events . . . 43
The code . . . 45
Key take-aways . . . 45

**Chapter 10: SCTP specific socket functions in Linux . . . 46**

Obtaining association id . . . . . 46
Getting local and remote association addresses . . . . . 46
Sending and receiving data . . . . . 46
Peel off . . . . . 46
One-to-one server . . . . . 46
One-to-many server . . . . . 47
Peel-off server . . . . . 47
Conclusion . . . . . 47

**Chapter 11: Multi-homing in Linux . . . . . 48**
Binding and connecting . . . . . 48
How to implement multi-homing . . . . . 48
The server . . . . . 48
The client . . . . . 48
Running the code . . . . . 48
Few words about one-to-one style sockets . . . . . 49
Conclusion . . . . . 49

# Preface

I see the network protocols that we use every day as citizens of two quite different worlds. The first one is the world of the Internet while second one is the world of telecoms and specifically the mobile network operators. Both worlds have got the mission to keep us connected, but they feel quite different.

The Internet world is big, open and welcoming. There is a lot of information about it and with just a simple web search you can find tons of information, tutorials, sample code and even ready to use open source implementations. On the other side is the mobile core network world which I find exotic and mystique. The web is not your friend here. You can search a lot and find very little useful information.

This book is about the SCTP protocol, which I like to describe as an Internet protocol for the Telecom world. It is developed by IETF as an IP based replacement for the legacy SS7 network. My first encounters with this protocol involved a lot of struggle. Not that there is no information about it on the Internet - quite the opposite. But it either implies you should know some stuff (and you don't) or covers a basic case which you already know or something in between.

With this book I have got one purpose. I want to save you some time and help you find the right information faster. The book is split in two parts. The first one is purely about how SCTP is designed and how it works. This part is based on the SCTP specification and is full with links to relative sections of it. My purpose is to give you intuition and context about a few SCTP topics I find important and to point you to the relative sections from the specification if you want to learn more.

The second part of the book is more practical. You will see how to use the SCTP protocol implementation in the Linux kernel to write your own applications. Each chapter of this part shows a specific technique and contains sample code which demonstrates it. You will also find references to man pages and specifications covering the topics in greater detail.

This book doesn't contain any unique and secret information which you can't find in the SCTP specification or the Internet. It is just a very detailed map which hopefully will make your trip through this information faster and more pleasant.

## Who is this book for

The intended audience of this book is network engineers and software developers willing to learn how the protocol works and how to use it in Linux. It may be also useful for students.

# How to read the book

If you are a network engineer who wants to understand how the protocol works in great detail I advise you to focus on the first part of the book and follow all references to the specification. If your focus is just using the SCTP protocol as a developer the information in the first part should be enough. But I strongly advise you not to skip it and make sure you understand how the protocol works before moving on to the code samples.

The last chapter of the book contains a multi-homed client server application which you can use to multi-homing in practice even if you are not interested in coding.

# Code samples and PCAP files

All code samples and PCAP files used in the book are available as Extra content for this book. They are a very important so I strongly encourage you to download and use them while reading.

# THANK YOU

The decision to buy this book and spare your precious time reading it means a lot to me! I hope you will like it and will learn a lot from it. Don't hesitate to contact me if you have got any questions, you have found a mistake or just want to share your feedback. You can find my up to date contact information on https://petanode.com[1].

[1] https://petanode.com/

# Chapter 1: Quick introduction to the SCTP protocol

SCTP stands for Stream Control Transmission Protocol. The protocol is not very popular in the Internet world, but plays an important role in the telecommunication networks. It is used as a transport for the SIGTRAN[2] and it is the default transport protocol in 4G core network - EPC[3].
SCTP is connection oriented (like TCP) and message oriented (like UDP) transport layer protocol. Let's see what these terms mean:

- **Connection oriented protocol**. Means that the protocol maintains logical connection. Usually these protocols are reliable, they handle packet delivery confirmations, retransmissions, how much data the receiver can handle and so on. Connection oriented protocols ensure that the messages are delivered in-order without any losses. TCP is a classical example for such protocol.
- **Message oriented protocol**. The data is transmitted in separate messages and they are received one by one from the peer. The counterpart of these protocols are called **Stream oriented**. They transmit the data as a continuous stream without any logical separators. It's a responsibility of the upper layer to read enough data and to know how to interpret and separate it. These protocols usually use predefined header which contains the length of the data or some sort of separator. HTTP for example uses CRLF as separator and the Content-Length header field specifies the length of the message body.

Back to SCTP. What does connection oriented and message oriented means? SCTP establishes a logical connection between the client and server, handles rate at which the sender transmits data to the receiver and handles the retransmissions of lost packets. At the same time SCTP transmits messages (called chunks) between the peers and the receiver can easily distinguish them. Additionally SCTP supports some useful features like multiple streams and multi-homing, but more on this later.

## RFC 4960: The specification of the SCTP protocol

Each protocol in the Internet world is specified by IETF in a technical specification called "Request For Comment" (or in short RFC) with a unique number. SCTP is specified by RFC 4690[4]. Although it is the most complete source of information about the protocol reading a RFC is not an easy task,

[2] http://en.wikipedia.org/wiki/SIGTRAN
[3] http://en.wikipedia.org/wiki/System_Architecture_Evolution
[4] http://tools.ietf.org/html/rfc4960

especially when you do it for the first time. The document contains all details one might need to create working SCTP implementation and it's easy to get lost.

In this book I try to sacrifice some of the details and to produce an easy to follow guide to SCTP. In each chapter I try to explain a concept about the protocol and point to the exact sections of the specification where this topic is covered. I hope this approach will save you some time and get you a smoother learning experience. Don't feel obligated to read each section of the specification, unless you really need the details. Just reading the book should be enough to understand how the protocol works.

## Some SCTP related key terms and abbreviations

Before moving on with the protocol, let's review some common SCTP terms. They are used a lot throughout the specification and are fundamental for the protocol.

**Association**. SCTP is connection-oriented protocol. The logical connection between the client and the server is called **association**. Think about it as the alternative of a connection in TCP.

**Multi-homing**. An SCTP association can use more than one IP address on either sides (client, server or both). This feature of the protocol is called **multi-homing**. Let's see an example on fig. 1. Host A has got three Ethernet interfaces, with three different IP addresses. Host B has got also three interfaces, but only two are used for the association. The interfaces, part of the association, are shown in dark purple. The addresses for each peer are announced during association establishment. On each peer all addresses use the same port number. Initially a single pair of addresses between the client and the server is picked as main path. In case it fails - alternative one is selected. I don't want to go in much details because there is a dedicated chapter for multi-homing, but for now I want you to know what's the idea behind it.

**Stream**. An SCTP stream is a logical unidirectional channel in the association. The number of streams in each direction (client to server and server to client) is negotiated during the association initialisation and it may or may not match in each direction. For example in fig. 1, from host A to host B there are 5 streams, but from host B to host A - only 3.

The purpose of the streams is to provide logical channels for in-order transfer of data/messages. Let's say SCTP is used for signalling between two nodes in a telecom network. In this scenario there can be a single association between the hosts and the signalling messages for different phone calls can be transmitted over different streams. This approach provides logical separation of the signalling data. Additionally if a message in stream X is lost and needs to be retransmitted, it won't delay the message in stream Y, because they are independent. This feature is used to mitigate the "head-of-line blocking" issue observed in stream protocols like TCP.

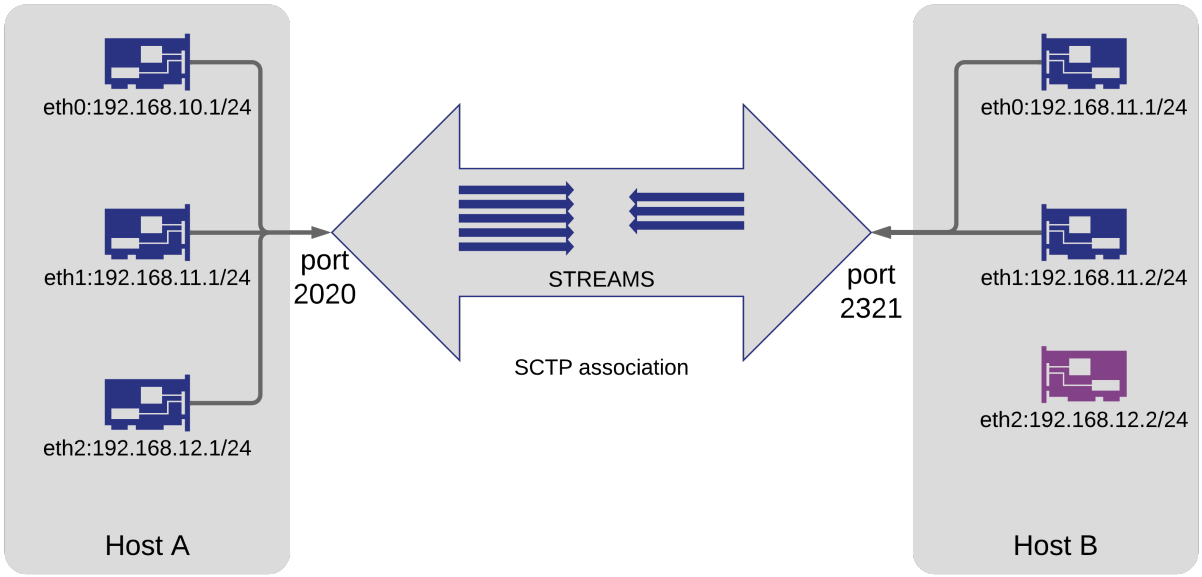


**figure 1: SCTP multi-homed association**

**Chunk**. This is a unit of information within an SCTP packet. Think about it as a specific protocol message, which can represent a request/response from the peers, state update or user data transfer. The structure of the chunk will be discussed later in this chapter.

## Time to check the specification

Section 1.3[5] and Section 1.4[6] describe all SCTP key terms and abbreviations.

Section 1.5[7] provides functional overview of the protocol. It describes how streams are used, how in-order delivery is achieved within a stream, how data messages are acknowledged and so on.

# Sample PCAP files

In the first five chapters we will work a lot with captures of network traffic saved in PCAP files. Throughout the book there are Wireshark screenshots showing the relevant information that we need, but I think it is also useful to have the PCAP files itself and view it by yourself. The whole book uses two PCAP files and you can find them in the sample code, in directory named **PCAPs**. There are multiple files there containing different SCTP scenarios that we will study. I encourage you to use the screenshots as a guidance and look at the actual PCAP files while reading the book. This way you will also gain some hands on experience with analysing SCTP associations with Wireshark.

[5] http://tools.ietf.org/html/rfc4960#section-1.3
[6] http://tools.ietf.org/html/rfc4960#section-1.4
[7] http://tools.ietf.org/html/rfc4960#section-1.5

Each time when I use a Wireshark screenshot I will also explicitly note the filename and packet number (the first column in Wireshark) of the SCTP chunk I am referring to. This way you can open the file by yourself and see some more context about the topic we are discussing.

# SCTP packet structure

Each SCTP packet contains a common header and one or more chunks. The header contains information used to identify the association. It has got fixed size and contains the following fields:

- source port number (16 bits)
- destination port number (16 bits)
- verification tag (32 bits)
- checksum (32 bits)

On fig. 2 there is a common header and INIT chunk, which is collapsed for better readability. This chunk in packet no. 1 from association.pcapng.

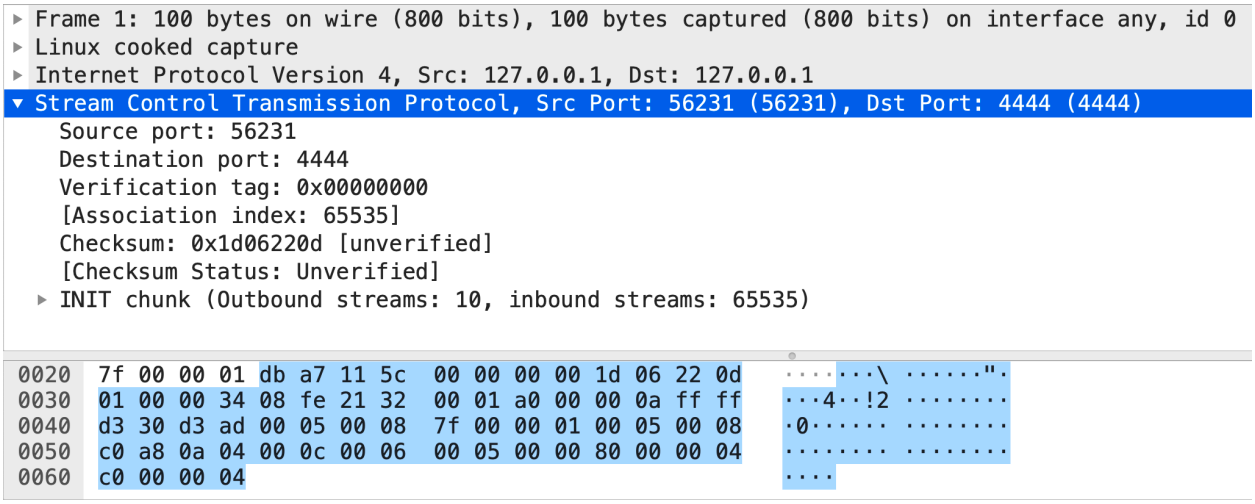


**figure 2: INIT chunk**

An SCTP chunk represents a protocol message, which can be used by the protocol itself (e.g. INIT, which is the first step in association establishment), or can contain user data (DATA chunk). It has got three fixed-length parameters and a variable length value. Each chunk is 4 byte aligned, which means that the whole chunk size have to be multiple of 4 bytes. If it is not, zero byte padding is added. The padding shouldn't be more than 3 bytes (4 byte padding doesn't make sense). Each chunk has got the following fields:

- chunk type - 8 bits, can be considered as and identification of the SCTP message. E.g. INIT, DATA, HEARTBEAT, etc.
- chunk flags - 8 bits, specific for each chunk type.

- chunk length - 16 bits, the length of the chunk in bytes, including all header fields (type, flags, length) plus the length of the value, excluding the padding.
- chunk value - variable length parameter. Contains various chunk parameters, which depend on the chunk type.

Fig. 3 shows SACK message, which contains common packet header and SACK chunk. The chunk's size is 16 bytes, which means that there is no padding. The chunk is in packet no. 6 from association.pcapng.

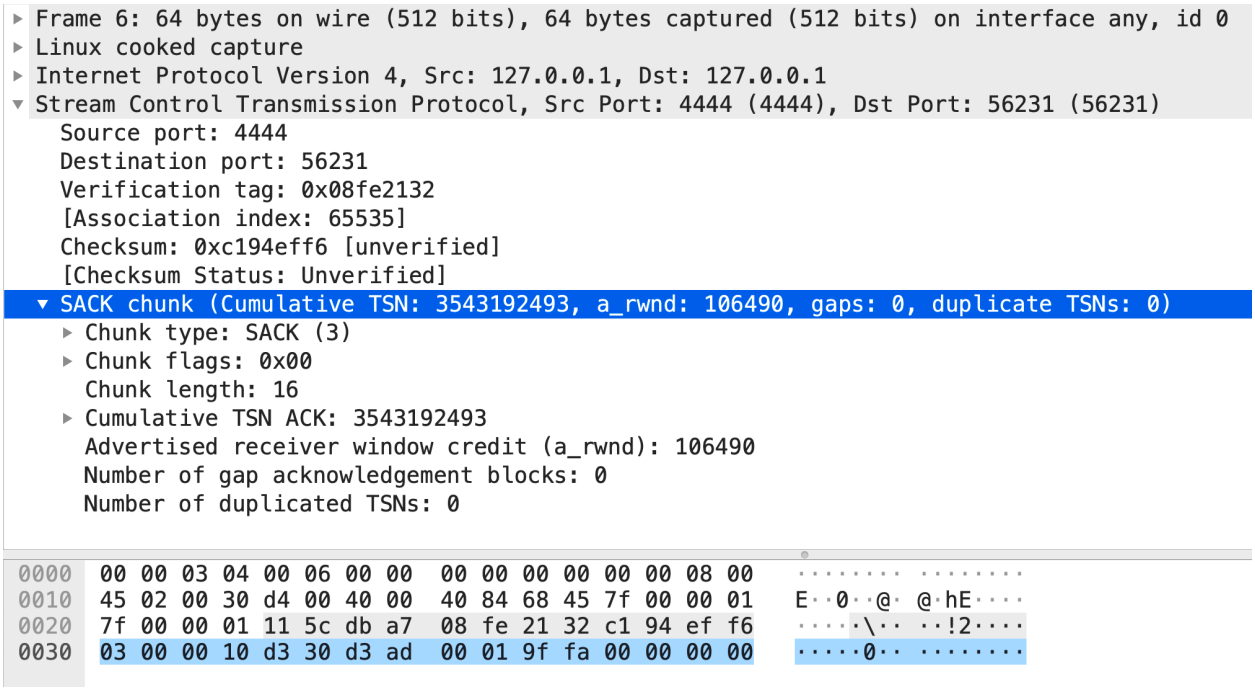


figure 3: SACK chunk

## Chunk parameters

The value of each chunk has specific format too. It contains two fixed-sized parameters and variable length value. Each parameter in the chunk must be 4 byte aligned, so the chunk parameter itself may also contain padding. The logic is the same as for the chunk value. Here is the structure of each parameter:

- chunk parameter type - 16 bits, the type of the parameter in the chunk.
- chunk parameter length - 16 bits, the length of the parameter, without the padding.
- chunk parameter value - the actual value for the specified parameter type.

## Time to check the specification

The common header of the SCTP packet is described in Section 3.1[8].

Chunk structure is described in Section 3.2[9].

All chunk definitions can be found in Section 3.3[10].

# An example of common header and chunks encoding

Wireshark makes packet decoding easy, but it's a good exercise to 'decode' one chunk manually and see how it's works. The packet on fig. 4 contains a SACK chunk and has got 4 parameters. The whole SCTP packet is selected and Wireshark has highlighted its hex representation at the bottom of the window. We will work with this data. This single chunk is extracted in decoding.pcapng from the sample code, so if you prefer you can open it directly with Wireshark.

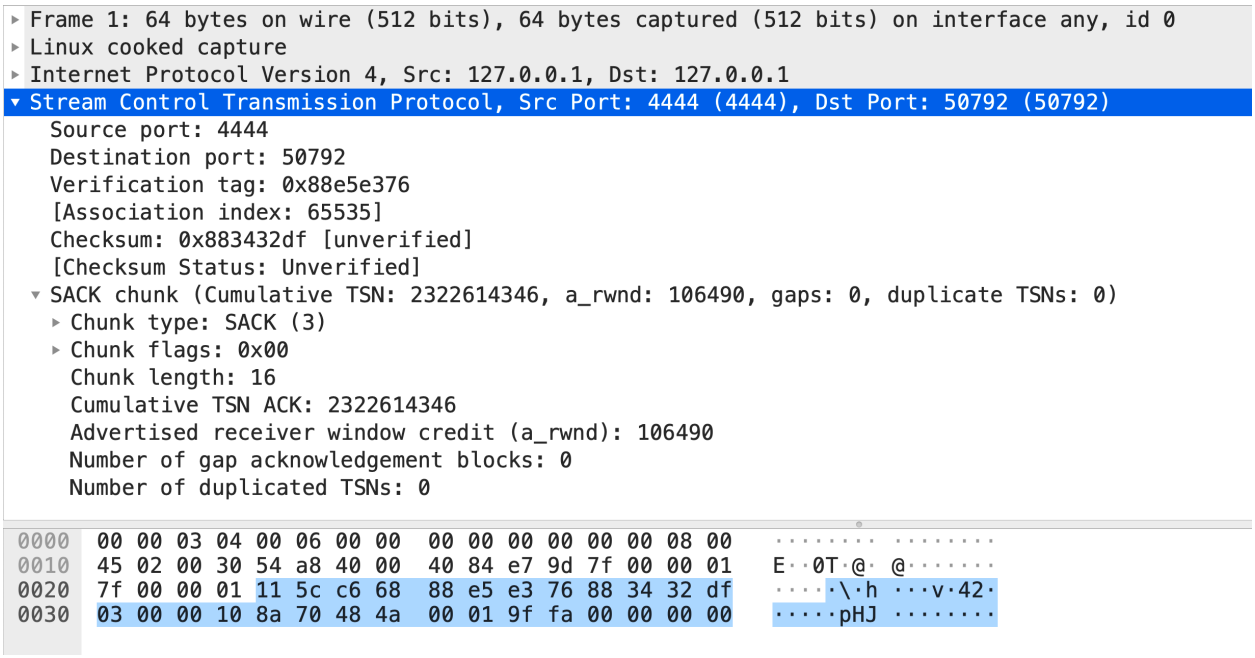


figure 4: SACK decoding example

## A few words about hex dumps

Wireshark and also other packet analysers use hex dumps to show the binary representation of the packet. Here is a short hex dump that we will use later:

[8] http://tools.ietf.org/html/rfc4960#section-3.1
[9] http://tools.ietf.org/html/rfc4960#section-3.2
[10] http://tools.ietf.org/html/rfc4960#section-3.3

0000 11 5c c6 68 88 e5 e3 76 88 34 32 df 03 00 00 10
0010 8a 70 48 4a 00 01 9f fa 00 00 00 00

We see two lines of hex digits. Each line starts with four digits, followed by 16 groups of two digits, separated by spaces.

A single hex digit represents 4 bits, so a group of two digits is 8 bits, which is 1 byte. Each line is a fixed stream of bytes. The four digits in the beginning of each line is offset, which is also in hex. 0x10 is 16 in hex, which exactly corresponds to the 16 bytes (16 groups of two digits) we have got on each line.

## The common header

And back to our packet. As we now know, each SCTP packet starts with a common header, which is 96 bits (12 bytes):

11 5c c6 68 88 e5 e3 76 88 34 32 df

We know that the common header has got fixed size and structure. Let's decode it:

- source port number (16 bits): 11 5c
- destination port number (16 bits): c6 68
- verification tag (32 bits): 88 e5 e3 76
- checksum (32 bits): 88 34 32 df

Convert these numbers to decimal and you will get the values from Wireshark.

## The chunk header

Next we should have at least one SCTP chunk. It's header is fixed length too (32 bits). We get the 4 bytes just after the common header - **03 00 00 10**. Let's decode them:

- chunk type (8 bits): 03. A quick glance in Section 3.2[11] shows that this is a SACK chunk.
- chunk flags (8 bits): 00
- chunk length (16 bits): 00 10. This is the length of the chunk - 16 bytes or 128 bits. 128 is divisible by 4 so there is no padding in this chunk. Don't forget that each chunk should be aligned on 4 bits.

[11] http://tools.ietf.org/html/rfc4960#section-3.2

## The chunk payload

Now let's parse the chunk itself. From the common header we know that the whole chunk is 16 (0x0010) bytes, or 128 bits. We have already parsed the common header, which is included in these 128 bits, so there is 128 - 32 = 96 bits left in the chunk. This is the chunk payload - **8a 70 48 4a 00 01 9f fa 00 00 00 00**.

To parse the payload we need the chunk definition. It is in Section 3.3.4[12]. It starts with a description of the message, which shows the name of its parameters and their sizes. A copy of it is show on fig. 5. This bitmap uses the same idea as the hex dump we talked about earlier. The only difference is that there are no hex values, but parameter names. This is a convenient way to see how parameters are transmitted over the wire and learn their sizes. Let's start parsing the parameters.

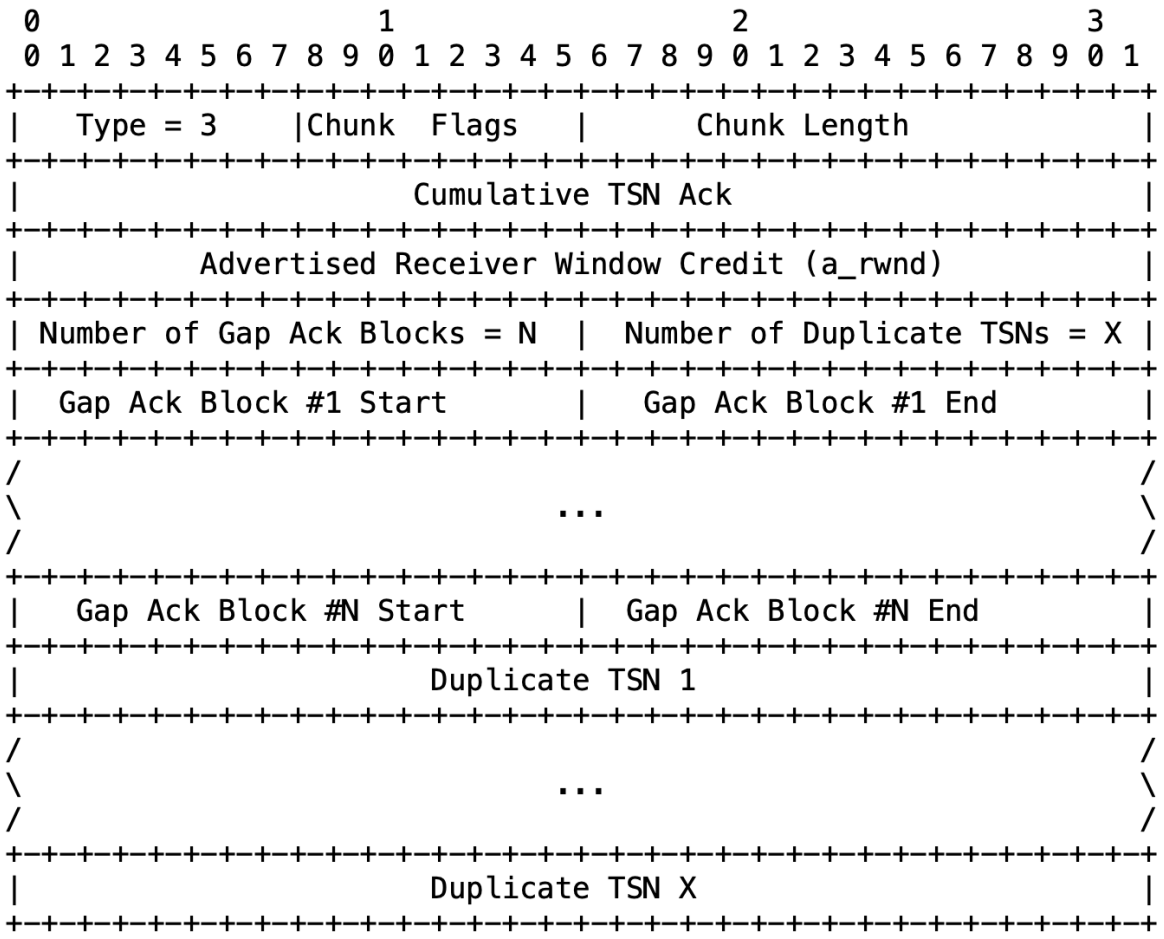


**figure 5: SACK chunk structure**

We already parsed Chunk type, flags and length. The flags value is 0x00. The flags description in

[12] http://tools.ietf.org/html/rfc4960#section-3.3.4

Section 3.3.4[13] says that the flags for SACK are always 0.

### The chunk parameters

The next one is **Cumulative TSN Ack**, which is 32 bits. The first 4 bytes from our buffer are **8a 70 48 4a**. This is 2322614346 in decimal, which matches the value we see in Wireshark. The payload which remains unparsed is **00 01 9f fa 00 00 00 00**.

At this point you shouldn't worry about the meanings of the parameters. We will talk about in the following chapters.

Let's move on the next parameter, which is **Advertised Receiver Window Credit (a_rwnd)**. It's length is also 32 bits, so we fetch the next 4 bytes from the unparsed payload: **00 01 9f fa**. This is 106490 in decimal, which matches the value of a_rwnd from Wireshark.

The unparsed part of the payload now is **00 00 00 00**. This is two bytes and we see in the message description that they correspond to **Number of Gap Ack Blocks** and **Number of Duplicate TSNs**. They are both zero, which means there are no Gap Ack Blocks and Duplicate TSNs.

And we are done. I hope this example convinced you that SCTP packet parsing is not a complicated task. Start with the fixed size headers and follow the descriptions in the specification from there. The same approach can be applied for decoding any chunk in Section 3.2[14].

## Key take-aways

SCTP is connection oriented and message oriented transport protocol, which uses IPv4/IPv6 as network layer. The protocol is defined in RFC 4960[15].

Each SCTP packet contains a common header (describing the association) and one or more chunks. Each chunk contains its own common header and zero or more chunk parameters.

[13] http://tools.ietf.org/html/rfc4960#section-3.3.4
[14] http://tools.ietf.org/html/rfc4960#section-3.2
[15] https://tools.ietf.org/html/rfc4960

# Chapter 2: Association initialisation

In this chapter we will see how SCTP association initialisation happens, what chunks are used and what parameters they have got.

Section 4[16] has got a detailed state diagram for a SCTP association. There are two main states - **CLOSED** and **ESTABLISHED**. Two endpoints can exchange data only when the association is in **ESTABLISHED** state. The transition between these two states is made by the association initialisation procedure.

## Association initialisation

Similarly to TCP, the peer initiating the association is called **client** and it is connecting to a **server**. Figure 1 shows the flow which leads to an initialised association (**ESTABLISHED** state). The chunk names are in bold, below them are their most important parameters. Xa/Xb, Ya/Yb are numeric values for each parameter respectively on the client and the server. The exact values are not important for now. Remember that a single SCTP packet can contain more than one chunk. Let's review each chunk in detail.

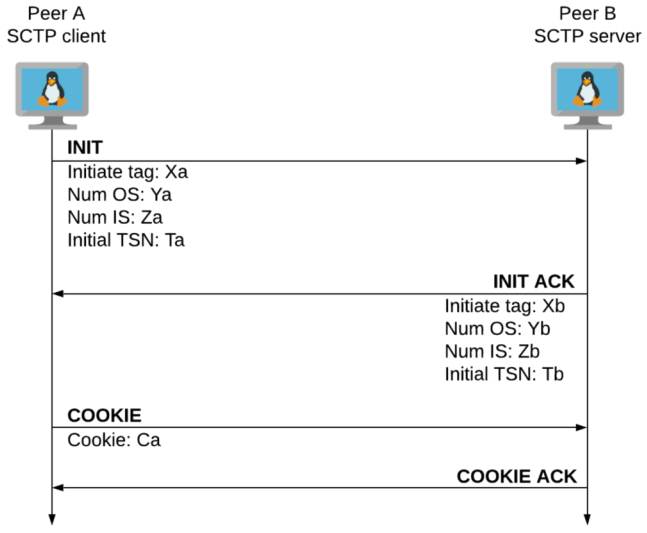


**figure 1: SCTP association initialisation**

[16] http://tools.ietf.org/html/rfc4960#section-4

## Time to check the specification

Section 5[17] describes association initialisation procedure.

# INIT chunk

The association initialisation starts with the client sending an **INIT** chunk to the server. On fig. 2 you can see a screenshot from Wireshark, showing such a chunk. This is packet no. 1 from association.pcapng. It has the following parameters:

- **Initiate tag**. This value should be saved by the receiver and set in **Verification tag** field in the SCTP common header for each message from the server to the client. Note that the **Verification tag** in the message, containing the **INIT** chunk, is 0 because it is the first one in the communication.
- **Advertised Receiver Window Credit (a_rwnd)** is the buffer size of the sender.
- **Number of outbound streams**. The requested number of streams from the client to the server. The server may not accept this number. The procedure for stream count negotiation is described later in the chapter. This value can't be zero.
- **Number of inbound streams**. The maximum number of inbound streams that the client supports. The server is obligated to respect this value, or error will occur. Also can't be zero.
- **Initial TSN**. A random number between 0 and 4294967295. TSN stands for Transmission Sequence Number. Each DATA chunk (user data) has attached a unique TSN which is used for acknowledgement and duplicated chunk detection. This parameters states the first TSN that will be used by the sender (the client in this case). For each new DATA chunk, the TSN is incremented by one.
- **Supported address types** and **IPv4 address parameters**. They contain the IP addresses which can be used to reach the client. They can be more than one, because of the SCTP's multi-homing feature.
- **ECN parameter**. ECN stands for *Explicit congestion notification* it is an extension, which is not part of the base protocol and hasn't got any relation to association establishment. It is out of the scope of this book, so I will skip it. Check the *Time to check the specification* section for more details.
- **Forward TSN supported parameter**. This parameter is from SCTP Partial Reliability Extension. It is also out of the scope of the book.

[17] http://tools.ietf.org/html/rfc4960#section-5

```
▸ Frame 1: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface any, id 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▾ Stream Control Transmission Protocol, Src Port: 56231 (56231), Dst Port: 4444 (4444)
    Source port: 56231
    Destination port: 4444
    Verification tag: 0x00000000
    [Association index: 65535]
    Checksum: 0x1d06220d [unverified]
    [Checksum Status: Unverified]
  ▾ INIT chunk (Outbound streams: 10, inbound streams: 65535)
    ▸ Chunk type: INIT (1)
      Chunk flags: 0x00
      Chunk length: 52
      Initiate tag: 0x08fe2132
      Advertised receiver window credit (a_rwnd): 106496
      Number of outbound streams: 10
      Number of inbound streams: 65535
      Initial TSN: 3543192493
    ▸ IPv4 address parameter (Address: 127.0.0.1)
    ▸ IPv4 address parameter (Address: 192.168.10.4)
    ▸ Supported address types parameter (Supported types: IPv4)
    ▸ ECN parameter
    ▸ Forward TSN supported parameter
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ........ ........
0010  45 02 00 54 00 00 40 00  40 84 3c 22 7f 00 00 01   E··T··@· @·<"····
0020  7f 00 00 01 db a7 11 5c  00 00 00 00 1d 06 22 0d   ·······\ ......".
0030  01 00 00 34 08 fe 21 32  00 01 a0 00 00 0a ff ff   ···4··!2 ........
0040  d3 30 d3 ad 00 05 00 08  7f 00 00 01 00 05 00 08   ·0······ ........
0050  c0 a8 0a 04 00 0c 00 06  00 05 00 00 80 00 00 04   ........ ........
0060  c0 00 00 04                                       ····
```

figure 2: INIT chunk

## Time to check the specification

Section 3.3.2[18] describes the INIT chunk and its parameters.
Section 3.3.2[19] is about Advertised Receiver Window Credit (a_rwnd).
Appendix A[20] from RFC 4960[21] further describes the ECN parameters.
RFC 3758[22] RFC 3758 is about the partial reliability extension, which is outside the topic of this book. Section 3.2 from this specification is about Forward TSN support parameter.

# INIT ACK chunk

The server confirms the association establishment by sending an **INIT ACK** chunk. The one shown on fig. 3 is the response of the **INIT** chunk from fig. 2. This is packet no. 2 from association.pcapng.

[18] http://tools.ietf.org/html/rfc4960#section-3.3.2
[19] http://tools.ietf.org/html/rfc4960#section-3.3.2
[20] http://tools.ietf.org/html/rfc4960#appendix-A
[21] http://tools.ietf.org/html/rfc4960
[22] https://tools.ietf.org/html/rfc3758#section-3.2

```
▶ Frame 1: 340 bytes on wire (2720 bits), 340 bytes captured (2720 bits) on interface any, id 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▼ Stream Control Transmission Protocol, Src Port: 4444 (4444), Dst Port: 56231 (56231)
    Source port: 4444
    Destination port: 56231
    Verification tag: 0x08fe2132
    [Association index: 65535]
    Checksum: 0x04cbd3d9 [unverified]
    [Checksum Status: Unverified]
  ▼ INIT_ACK chunk (Outbound streams: 10, inbound streams: 10)
    ▶ Chunk type: INIT_ACK (2)
      Chunk flags: 0x00
      Chunk length: 292
      Initiate tag: 0xa556dfe4
      Advertised receiver window credit (a_rwnd): 106496
      Number of outbound streams: 10
      Number of inbound streams: 10
      Initial TSN: 1569172118
    ▶ State cookie parameter (Cookie length: 260 bytes)
    ▶ ECN parameter
    ▶ Forward TSN supported parameter
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ........ ........
0010  45 02 01 44 00 00 40 00  40 84 3b 32 7f 00 00 01   E··D··@· @·;2····
0020  7f 00 00 01 11 5c db a7  08 fe 21 32 04 cb d3 d9   ·····\·· ··!2····
0030  02 00 01 24 a5 56 df e4  00 01 a0 00 00 0a 00 0a   ···$·V·· ········
0040  5d 87 aa 96 00 07 01 08  cc 70 a7 7e f9 c1 3c 8f   ]····... ·p·~··<·
0050  ef 94 f8 4c ca 12 16 fa  de d3 49 9e 00 00 00 00   ···L···· ··I·····
0060  00 00 00 00 00 00 00 00  00 00 00 00 e4 df 56 a5   ........ ......V·
0070  32 21 fe 08 00 00 00 00  00 00 00 00 4f 66 ac 31   2!...... ····Of·1
0080  9f 82 00 00 0a 00 0a 00  96 aa 87 5d 02 00 db a7   ........ ···]····
0090  7f 00 00 01 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00a0  00 00 00 00 00 00 00 00  5c 11 01 00 00 00 00 00   ........ \·......
00b0  80 02 00 24 b2 9a bf 9e  1f 0c 01 0c 02 66 3c 46   ···$···· ·····f<F
00c0  95 e8 35 c9 42 76 db 51  20 8f f6 16 23 f7 99 45   ··5·Bv·Q  ···#··E
00d0  39 e0 d3 70 00 00 00 00  00 00 00 00 00 00 00 00   9··p···· ........
00e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00f0  00 00 00 00 00 00 00 00  01 00 00 34 08 fe 21 32   ........ ···4··!2
0100  00 01 a0 00 00 0a ff ff  d3 30 d3 ad 00 05 00 08   ........ ·0·····
0110  7f 00 00 01 00 05 00 08  c0 a8 0a 04 00 0c 00 06   ........ ........
0120  00 05 00 00 80 00 00 04  c0 00 00 04 00 00 00 00   ........ ........
0130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0140  00 00 00 00 00 00 00 00  00 00 00 00 80 00 00 04   ........ ........
0150  c0 00 00 04                                       ····
```

figure 3: INIT ACK

The **Verification tag** in the common header is set to *0x08fe2132* - the **Initiate tag** from the **INIT**. Remember that only the **INIT** has got **Verification tag** set to 0. Each subsequent message will have the **Verification tag** set to the **Initiate tag** of the receiver. The parameters of the **INIT ACK** chunk are similar to the **INIT** ones on first sight, but they carry different meaning:

- **Initiate tag**. The server announces its Verification tag.
- **Advertised Receiver Window Credit (a_rwnd)**. The buffer size of the server.
- **Number of outbound streams**. The number of streams from the server to the client. This value should be equal or less than **Number of inbound streams** parameter in the **INIT** chunk. If the value is bigger the client can abort the association establishment procedures.

- **Number of inbound streams**. The maximum number of inbound streams, supported of the server. If the **Number of outbound stream** parameters in the INIT chunk of the client is bigger than this value, the client must adjust its maximum output streams number.
- **Initial TSN**. The first TSN that will be used by the server. What we discussed about this parameter in **INIT** is valid for **INIT ACK** too.
- **ECN parameter** and **Forward TSN supported parameter**. Again they are not related to the association establishment and we will skip them.

At this point both the client and the server know the number of input and output streams supported by each other. This procedure is used to negotiate an acceptable number for both peers in each direction. The protocol negotiates the maximum, not the actual number of streams in each direction. This doesn't mean that the SCTP user is obligated to use all of them.

There is one more parameter in the **INIT ACK** chunk, which was not present in the INIT - **State cookie**. When INIT is received, the server generates TCB, which stands for Transmission Control Block. It contains all the information about the association which is required by the server. The **State cookie** parameter contains the TCB with MAC (Message Authentication Code). It is used for integrity check and authentication of the TCB. After the state cookie is generated, the server deletes all the information about the association. This means that the server doesn't allocate any resources for the association at this point. Later, the client responds with **COOKIE ECHO** (discussed next), which contains the TCB with all the required information which the server needs to create the association. The MAC guarantees that the TCB is not modified by the client so the server can use it safely.

This approach is very clever from a security point of view. TCP is vulerable to DoS attacks, called SYN flood. In nutshell TCP connection establishing is a three step process. The client sends SYN, the server allocates resources for the connection, sends ACK and waits for SYN-ACK. This makes him vulnerable to a malicious clients, which send lots of SYN packets and doesn't establish or tear down the connection. SCTP's approach to association establishment mitigates this issue. If a malicious client tries to flood the server with INIT messages the SCTP stack will only calculate state cookies and send INIT ACKs, without keeping any state information. This way no resources will be wasted and the DoS attack is way harder to perform.

## Time to check the specification

Key terms[23] section contains definition for TCB.
Section 3.3.3[24] describes the INIT ACK chunk.
Section 5.1.3[25] is about state cookie generation.
Section 13[26] describes how TCBs are generated.

MAC is way beyond the scope of this book, but if you need more information check:

[23] http://tools.ietf.org/html/rfc4960#section-1.3
[24] https://tools.ietf.org/html/rfc3758#section-3.3.3
[25] http://tools.ietf.org/html/rfc4960#section-13
[26] http://tools.ietf.org/html/rfc4960#section-13

- RFC 2104[27]
- Hash-based message authentication code[28] article on Wikipedia

# COOKIE ECHO chunk

When the client receives **INIT ACK** chunk with state cookie, it should immediately respond with **COOKIE ECHO**. This chunk is shown on fig. 4 (packet no. 3 from association.pcapng). It has only one parameter **Cookie**, which contains the state cookie received.

```
▶ Frame 3: 312 bytes on wire (2496 bits), 312 bytes captured (2496 bits) on interface any,
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▼ Stream Control Transmission Protocol, Src Port: 56231 (56231), Dst Port: 4444 (4444)
    Source port: 56231
    Destination port: 4444
    Verification tag: 0xa556dfe4
    [Association index: 65535]
    Checksum: 0xaa256669 [unverified]
    [Checksum Status: Unverified]
  ▼ COOKIE_ECHO chunk (Cookie length: 260 bytes)
    ▶ Chunk type: COOKIE_ECHO (10)
      Chunk flags: 0x00
      Chunk length: 264
      Cookie: cc70a77ef9c13c8fef94f84cca1216faded3499e00000000…
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ........ ........
0010  45 02 01 28 00 00 40 00  40 84 3b 4e 7f 00 00 01   E··(··@· @·;N····
0020  7f 00 00 01 db a7 11 5c  a5 56 df e4 aa 25 66 69   ·····\ ·V···%fi
0030  0a 00 01 08 cc 70 a7 7e  f9 c1 3c 8f ef 94 f8 4c   ·····p·~ ··<····L
0040  ca 12 16 fa de d3 49 9e  00 00 00 00 00 00 00 00   ·····I· ........
0050  00 00 00 00 00 00 00 00  e4 df 56 a5 32 21 fe 08   ........ ··V·2!··
0060  00 00 00 00 00 00 00 00  4f 66 ac 31 9f 82 00 00   ........ Of·1····
0070  0a 00 0a 00 96 aa 87 5d  02 00 db a7 7f 00 00 01   .......] ........
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0090  00 00 00 00 5c 11 01 00  00 00 00 00 80 02 00 24   ····\··· .......$
00a0  b2 9a bf 9e 1f 0c 01 0c  02 66 3c 46 95 e8 35 c9   ........ ·f<F··5·
00b0  42 76 db 51 20 8f f6 16  23 f7 99 45 39 e0 d3 70   Bv·Q ··· #··E9··p
00c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00e0  00 00 00 00 01 00 00 34  08 fe 21 32 00 01 a0 00   .......4 ··!2····
00f0  00 0a ff ff d3 30 d3 ad  00 05 00 08 7f 00 00 01   ·····0·· ........
0100  00 05 00 08 c0 a8 0a 04  00 0c 00 06 00 05 00 00   ........ ........
0110  80 00 00 04 c0 00 00 04  00 00 00 00 00 00 00 00   ........ ........
0120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0130  00 00 00 00 00 00 00 00                           ........
```

figure 4: COOKIE ECHO

[27] http://tools.ietf.org/html/rfc2104
[28] http://en.wikipedia.org/wiki/Hash-based_message_authentication_code

## Time to check the specification

Section 3.3.11[29] describes COOKIE ECHO chunk.

# COOKIE ACK chunk

The server receives the **COOKIE ECHO** chunk and extracts the state cookie. Then it reads the TCB and computes its MAC. If it matches with the one in the message, the TCB is considered valid and authenticated. Then the server extracts from it all the required information about the association establishment. On success the server returns **COOKIE ACK** chunk and the association is considered established. There aren't any parameters in this chunk. Figure 5 shows an example. This is packet no. 4 from association.pcapng.

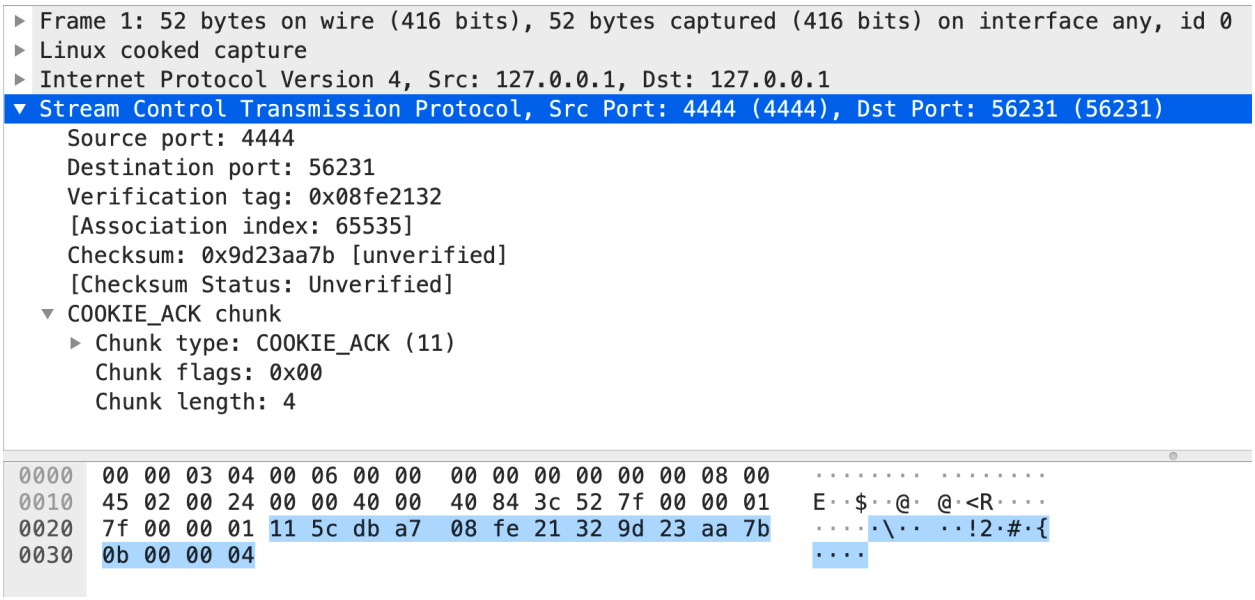


figure 5: COOKIE ECHO

## Time to check the specification

Section 5.1.5[30] explains how state cookie authentication works.

Section 3.3.12[31] describes COOKIE ACK chunk.

# Key take-aways

SCTP association establishment is a four step process. Four SCTP chunks are exchanged between the client and the server - INIT, INIT ACK, COOKIE ECHO, COOKIE ACK. COOKIE ECHO contains

[29] http://tools.ietf.org/html/rfc4960#section-3.3.11
[30] http://tools.ietf.org/html/rfc4960#section-5.1.5
[31] http://tools.ietf.org/html/rfc4960#section-3.3.12

TCB, which is all the information the server needs to establish the association. The client echoes back the TCB with the COOKIE ACK chunk. This approach makes DoS attacks against the server harder to carry on. Section 5[32] describes the whole association establishment procedure.

[32] http://tools.ietf.org/html/rfc4960#section-5

# Chapter 3: Data transfer in SCTP

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## DATA chunk

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### Time to check the specification

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## SACK chunk

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### Time to check the specification

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Data transfer procedures

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### How are gaps reported

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## How duplicate TSNs are reported

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Ordered and unordered data delivery

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Fragmentation and reassembly of user data

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Key take-aways

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Chapter 4: Association teardown

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Failure detection

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Endpoint Failure Detection

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### Time to check the specification

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Path Failure Detection

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### Time to check the specification

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Path Heartbeat

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### Time to check the specification

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## HEARTBEAT chunk

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### Time to check the specification

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## HEARTBEAT ACK chunk

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### Time to check the specification

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Association teardown

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### ABORT chunk

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### Time to check the specification

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Association shutdown

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Time to check the specification

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## SHUTDOWN chunk

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Time to check the specification

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## SHUTDOWN ACK and SHUTDOWN COMPLETE chunks

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Time to check the specification

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Key take-aways

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Chapter 5: Multi-homing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## How multi-homing works

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### No Host Name Address, IPv4 Address or IPv6 Address parameters present

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### Host Name Address parameter present

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### IPv4 Address or IPv6 Address parameters are present

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Time to check the specification

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Path verification

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Primary path

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Time to check the specification

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Association termination

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Example

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Normal operation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Primary path switching

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Key take-aways

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Chapter 6: Sockets introduction

In Linux, you deal with the network stack via sockets. I assume you are familiar how they work and you have got basic network programming knowledge in C. This includes what functions like socket(), bind(), listen() do, how you should call them and how to implement a simple client-server application. If you need a refreshment or you are new to Linux socket programming read some tutorials on this topic. There are a lot of good ones out there, but my favourite is Beej's Guide to Network Programming[33].

The sample code for this and the following chapters is in the book extras. Each directory in the project contains source code for a specific chapter and one separate folder with common boilerplate code. For a start extract the project and go to **ch6_one-to-one** directory. Spend some time to familiarise yourself with the code. It is a client-server application exchanging few messages over SCTP, but it is not that different from any client server application written in C.

## About the code

Please note that the code is not intended for production use. It is for educational purposes and demonstrates how SCTP is used in Linux. It has got some basic checks, but it is far from production ready. If you plan to use it in your project please take some time and add proper error handling.

Also for better readability in a book, I tried to to keep the client/server source files for each chapter relatively small. For this reason a lot of boilerplate code is extracted in the common directory.

## SCTP sockets in Linux

The SCTP API in Linux is specified in RFC 6458 - Sockets API Extensions for the Stream Control Transmission Protocol (SCTP)[34]. There are two interfaces available to the programmer - **one-to-one** and **one-to-many** socket style interface. The names come from the relationship between the association(s) and the socket. In one-to-one interface each socket represents a single association. This API is quite similar to the TCP sockets interface.

A typical one-to-one style server issues the following calls:

- socket()
- bind()

[33] http://beej.us/guide/bgnet/
[34] https://tools.ietf.org/html/rfc6458

- listen()
- accept()
- send()/receive()
- close()

And the client:

- socket()
- connect()
- send()/recv()
- close()

The socket type for one-to-one interface is **SOCK_STREAM**.

The one-to-many interface works in terms of messages. There one socket can handle a lot of associations and all the messages are sent/received via this socket. This approach is similar to the UDP sockets.

A one-to-many style server makes these calls:

- socket()
- bind()
- listen()
- recvmsg()/sendmsg()
- close()

It's worth mentioning that one-to-many style server doesn't need to call accept() in order to create an association. It is created on the fly for each new client.

And for a client:

- socket()
- sendmsg()/recvmsg()
- close()

Note that the client doesn't need to issue a connect() in order to communicate with a server. Client associations are also initiated on the fly. The socket type for one-to-many is **SOCK_SEQPACKET**

In this chapter we will see how one-to-one client/server application is implemented.

# Building the code

In this section I will explain how to build and run the sample code. It is very important for the second part of the book so I encourage you to get it now and look at it while reading the book.
The packages you need to build the samples are git, cmake, C compiler, the POSIX thread library and the SCTP tools for your distribution (libsctp-dev for debian and lksctp-tools-devel for redhat). I don't want to get into much details about Git and CMake so I'll just provide the required commands to get the code working.

First extract the sample code somewhere and go to the source code for the desired chapter:

```
cd sctp-sample-code/ch6_one-to-one/
```

CMake creates out of source build, which is a fancy name for "CMake uses separate build direcotry". Let's create it:

```
mkdir build
cd build
```

And run cmake. The argument is the path to the source code, which in our case is the parent directory:

```
cmake ..
```

You should see something like:

```
-- The C compiler identification is GNU 9.2.1
-- The CXX compiler identification is GNU 9.2.1
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for pthread.h
-- Looking for pthread.h - found
```

```
17      -- Looking for pthread_create
18      -- Looking for pthread_create - not found
19      -- Looking for pthread_create in pthreads
20      -- Looking for pthread_create in pthreads - not found
21      -- Looking for pthread_create in pthread
22      -- Looking for pthread_create in pthread - found
23      -- Found Threads: TRUE
24      -- Configuring done
25      -- Generating done
26      -- Build files have been written to: /tmp/sctp-sample-code/ch6_one-to-one_basic/\
27  build
```

Now you are ready to build the code:

```
1       make
```

The output on my machine is:

```
1       Scanning dependencies of target client
2       [ 50%] Building C object CMakeFiles/client.dir/client.c.o
3       Linking C executable client
4       [ 50%] Built target client
5       Scanning dependencies of target server
6       [100%] Building C object CMakeFiles/server.dir/server.c.o
7       Linking C executable server
8       [100%] Built target server
```

That's all. If everything is correct you should have got two binaries - server and client. Now start wireshark and run the binaries.

Open two terminals and start the server in the first one:

```
1       ./server
```

You should see:

```
1       Listening on port 4444
```

Then run the client in the other terminal:

```
./client 127.0.0.1
```

You should see:

```
Connecting...
OK
Sending message 1 of 5. Result: OK
Sending message 2 of 5. Result: OK
Sending message 3 of 5. Result: OK
Sending message 4 of 5. Result: OK
Sending message 5 of 5. Result: OK
Closing...
```

Now get back to the server terminal and check its output. You should see two new lines:

```
Got connection from 127.0.0.1
Connection from 127.0.0.1 closed by remote peer.
```

And finally check your PCAP trace to see the establishment of the SCTP association, the transfer of the messages and the association tear down. If you don't see anything, make sure you are listening on the correct interface. By default the client and the server communicate via localhost.

# The code

There is a sample code for each chapter located in separate directories. Each sample project contains similar set of files:

- defaults.h - contains various constants for the application. This file is almost identical for each chapter.
- server.c - generic implementation for the server. It contains all function calls required to establish one-to-one or one-to-many server.
- chapter specific source/header file. They contain function specific for the chapter. The are called from the server and possibly from the client.
- client.c - this chapter and the next one contain respectively one-to-one and one-to-many style clients. The following chapters omit the client unless it is necessary. You can use any client with any server. The reason to skip the client code is that it's usually the same.

In the root directory of the project there is a folder called common. It contains bolierplate code used from all the servers/clients. This code is not SCTP related but required so I will not review it in the book.

The code for this chapter looks like a typical TCP client/server implementation and this is intentional. I want to show you how easy it is to convert a TCP client/server program to a SCTP one. As a proof, let's try to rework this program to use TCP instead SCTP. Open defaults.h and change the PROTO constant. Its current value should be set to **IPPROTO_SCTP**, change it to **IPPROTO_TCP**. Now rebuild everything (run make clean && make), run the binaries again and check the PCAP. You should see that the whole communication is now over TCP. Don't forget to revert the change before you go on.

Let's have a look at the code. I have intentionally skipped some error handling, so don't be surprised if it crashes with bad input data. What I want is to focus on the socket API and keep the code as short and readable as possible. There are two C files for each binary - the client and the server. There is also a header (defaults.h) with some common constants for both binaries.

## Constants

In **defaults.h** there are some application wide parameters like default port number, address family, socket type, protocol etc. You can easily adjust the whole behaviour of the application from this header.

```
1  #ifndef DEFAULTS_H_
2  #define DEFAULTS_H_
3
4  #include <sys/socket.h>
5  #include <netinet/in.h>
6  #include <netinet/sctp.h>
7
8  enum {
9      SERVER_PORT = 4444,
10     ADDR_FAMILY = AF_INET,
11     SOCK_TYPE = SOCK_STREAM,
12     PROTO = IPPROTO_SCTP,
13     CLIENT_SEND_COUNT = 5,
14     SERVER_LISTEN_QUEUE_SIZE = 10,
15 };
16
17 #endif /* DEFAULTS_H_ */
```

## The server

main() reads the parameters passed to the binary (if any). You can pass port number different from the default one (4444). Then a SCTP socket is created (protocol is IPPROTO_SCTP). On success the server binds on all possible IP addresses and listens for new connections.

New thread is spawned for each new connection. This approach is not scalable, but will do the job for a demo program. The server waits for new connections in infinite loop. The only way to stop the server is to kill it.

Let's review server.c block by block. We will start with main(). Each important line is marked with a number. All explanations are after the code listing.

```
1  #include "defaults.h"
2  #include "server_helpers.h"
3  #include "inc.h"
4
5
6  void* handle_connection(void *thread_data);
7
8  int main(int argc, char* argv[])
9  {
10     int server_port = handle_server_cmd_arg(argc, argv, SERVER_PORT);  // 1
11     if(server_port == -1) {
12         return 1;
13     }
14
15     struct sockaddr_in bind_addr = get_server_bind_sockaddr(ADDR_FAMILY,
16                                                             server_port);  // 2
17
18     int server_fd = 0;
19     if((server_fd = socket(ADDR_FAMILY, SOCK_TYPE, PROTO)) == -1) {     // 3
20         perror("socket");
21         return 1;
22     }
23
24     if(bind(server_fd, (struct sockaddr*)&bind_addr, sizeof(bind_addr)) == -1) { // 4
25         perror("bind");
26         return 1;
27     }
28
29     if(listen(server_fd, SERVER_LISTEN_QUEUE_SIZE) != 0) {  // 5
30         perror("listen");
31         return 1;
32     }
33
34     printf("Listening on port %d\n", server_port);
35
36     while(1) {
```

```
37          int* client_fd = malloc(sizeof(int));       // 6
38          if(client_fd == NULL) {
39              printf("malloc error\n");
40              return 1;
41          }
42
43          struct sockaddr client_addr;
44          unsigned int client_addr_len = sizeof(client_addr);
45
46          if((*client_fd = accept(server_fd, &client_addr,
47                                  &client_addr_len)) == -1) {    // 7
48              perror("accept");
49              free(client_fd);
50              return 1;
51          }
52
53          log_address("Got connection from", (struct sockaddr_in*)&client_addr);
54
55          if(start_server_thread(&handle_connection, (void*)client_fd)) {   // 8
56              free(client_fd);
57          }
58      }
59
60      return 0;
61  }
62
63  void* handle_connection(void *thread_data) {
64      int socket = *(int*)thread_data;
65      free(thread_data);
66
67      while(1) {
68          char buf[1024];
69          int recv_len = 0;
70
71          if((recv_len = recv(socket, &buf, sizeof(buf), 0)) == -1) {     // 9
72              perror("recv");
73              return NULL;
74          }
75
76          if(recv_len == 0) {
77              printf("Connection from closed by remote peer.\n");
78              close(socket);
79              return NULL;
```

```
80          }
81
82          printf("Message received: %s\n", buf);
83
84          if(send(socket, "OK", strlen("OK"), 0) == -1) {      // 10
85              perror("send");
86              return NULL;
87          }
88      }
89  }
```

I want to say a few words about the lines with comments:

1. **handle_server_cmd_arg()** parses the command line parameters for the binary and returns the port number on which the server should bind. This function is from common.
2. **get_server_bind_sockaddr()** prepares a **sockaddr_in** structure for our server. Also from common.
3. A SCTP socket is created. The constants are defined in defaults.h. ADDR_FAMILY is AF_INET, SOCK_TYPE - SOCK_STREAM and PROTO is IPPROTO_SCTP.
4. The socket is bound to the server port on all local addresses.
5. The listen call waits for new connections.
6. Now we are in the server loop. We need to pass the file desriptor for the new assocation to the handling thread, so we allocate it dynamically.
7. **accept()** is blocking. The function returns when a client is connected.
8. **start_server_thread()** is a helper function, which executes **handle_connection()** in a new thread. It is from common.
9. A message from the client is read.
10. And a response is sent.

## The client

The client reads an IP address and optionally a port number from the command line. Then sends a number of messages to the server (specified by the **CLIENT_SEND_COUNT** parameter), reads the responses and exits. The code is quite similar to the server. Let's see it:

```c
#include "defaults.h"
#include "inc.h"
#include "client_helpers.h"


int main(int argc, char* argv[]) {
    server_endpoint_t params;
    if(handle_client_cmd_arg(argc, argv, SERVER_PORT, &params)) {       // 1
        return 1;
    }

    int client_fd = 0;
    if((client_fd = socket(ADDR_FAMILY, SOCK_TYPE, PROTO)) == -1) {     // 2
        perror("socket");
        return 1;
    }

    struct sockaddr_in peer_addr;
    if(get_peer_sockaddr(&peer_addr, ADDR_FAMILY, params.address,
                         params.port)) {   // 3
        return 1;
    }

    printf("Connecting...\n");

    if(connect(client_fd, (struct sockaddr*)&peer_addr,
                         sizeof(peer_addr)) == -1) {     // 4
        perror("connect");
        return 1;
    }

    char buf[1024];
    for(int i = 0; i < CLIENT_SEND_COUNT; ++i) {
        printf("Sending message %d of %d. Result: ", i+1, CLIENT_SEND_COUNT);

        memset(buf, 0, sizeof(buf));
        snprintf(buf, sizeof(buf)-1, "DATA %d", i);

        if(send(client_fd, &buf, strlen(buf), 0) == -1) {   // 5
            perror("send");
            return 1;
        }

```

```
44          memset(buf, 0, sizeof(buf));
45
46          if(recv(client_fd, &buf, sizeof(buf), 0) == -1) {   // 6
47              perror("recv");
48              return 1;
49          }
50
51          printf("%s\n", buf);
52      }
53
54      printf("Closing...\n");
55      if(close(client_fd) == -1) {    // 7
56          perror("close");
57          return 1;
58      }
59
60      return 0;
61  }
```

Again let's have a look at the lines with the comments:

1. **handle_client_params()** reads the command line parameters for the client. Function from common.
2. Creates a SCTP socket.
3. **get_peer_sockaddr()** creates a **sockaddr_in** structure. Function from common.
4. Connects to the server.
5. Sends a message.
6. Reads back the response.
7. The association is closed.

# Key take-aways

SCTP one-to-one interface represents one to one relationship between a socket and an association. This interface is quite similar to the TCP one. By changing some constants and minor code adjustments you can easily convert a TCP application to SCTP one.

# Chapter 7: SCTP Linux API: One-to-many style interface

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## One-to-many in a nutshell

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## The code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### Constants

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### The server

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

#### Receiving messages with recvmsg()

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### Sending messages with sendmsg()

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## onetomany.c

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## The client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Key take-aways

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Chapter 8: Working with SCTP ancillary data

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Introduction

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## How ancillary data works

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Code snippets showing how to accomplish a few common tasks

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Ancillary data in SCTP protocol

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### Ancillary data that can be received with recvmsg()

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### SCTP Receive Information Structure (SCTP_RCVINFO)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### SCTP Next Receive Information Structure (SCTP_NXTINFO)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Ancillary data that can be passed to sendmsg()

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### SCTP Initiation Structure (SCTP_INIT)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### SCTP Send Information Structure (SCTP_SNDINFO)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### SCTP PR-SCTP Information Structure (SCTP_PRINFO)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### SCTP AUTH Information Structure (SCTP_AUTHINFO)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### SCTP Destination IPv4 Address Structure (SCTP_DSTADDRV4) and SCTP Destination IPv6 Address Structure (SCTP_DSTADDRV6)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# The code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## The server

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## anc.c

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## The client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Key take-aways

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Chapter 9: SCTP notifications in Linux

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Introduction

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## How notifications work

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Enabling SCTP notifications

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Notification structure

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Events

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## SCTP_ASSOC_CHANGE

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## SCTP_PEER_ADDR_CHANGE

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## SCTP_REMOTE_ERROR

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## SCTP_SEND_FAILED

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## SCTP_SHUTDOWN_EVENT

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## SCTP_ADAPTATION_INDICATION

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## SCTP_PARTIAL_DELIVERY_EVENT

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## SCTP_AUTHENTICATION_EVENT

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### SCTP_SENDER_DRY_EVENT

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### SCTP_NOTIFICATIONS_STOPPED_EVENT

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### SCTP_SEND_FAILED_EVENT

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## The code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### The server

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

### notif.c

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Key take-aways

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Chapter 10: SCTP specific socket functions in Linux

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Obtaining association id

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Getting local and remote association addresses

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Sending and receiving data

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Peel off

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## One-to-one server

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## One-to-many server

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Peel-off server

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

# Chapter 11: Multi-homing in Linux

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Binding and connecting

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## How to implement multi-homing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## The server

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## The client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Running the code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Few words about one-to-one style sockets

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.

## Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/sctp.