

**Mobile Computing Project, WS17/18**

**Documentation**

**on**

**“Protocol Emulation for SCTP”**

M. Engg. in Information Technology

Faculty of Computer Science and Engineering

Frankfurt University of Applied Sciences

**Supervised By:**

**M.Sc. Michael Steinheimer**

**M.Eng. Gregor Frick**

**Prof. Dr. Ulrich Trick**

**Submitted By:**

Name: Mahnaz Rahman

Matriculation Number: 1189920

Abstract

Stream Control Transmission Protocol(SCTP) is a reliable, message-oriented transport layer protocol which operates on top of IP. SCTP protocol offers sequenced delivery of messages within multiple streams, with an option for order-of-arrival delivery of individual user messages. Results from theoretical analysis, simulation and experiments show that SCTP provides high availability, increased reliability and improved security for socket initiation. This document will give an overview of protocol and the implementation structure.

Table of Contents

1.0 [Principle of Projects and Protocol Functionality 5](#_Toc507268827)

[1.1 Overview 5](#_Toc507268828)

[1.2 Principle of Project 5](#_Toc507268829)

[1.3 Protocol Functionality 5](#_Toc507268830)

[1.3.1 SCTP Common Header 6](#_Toc507268831)

2.0 [Principles of Designed Architecture 7](#_Toc507268832)

[2.1 Initiation (INIT) 8](#_Toc507268833)

[2.2 Initiation Acknowledgment (INIT ACK) 8](#_Toc507268834)

[2.2.1 State Cookie 8](#_Toc507268835)

[2.3 Cookie Echo (COOKIE ECHO) 9](#_Toc507268836)

[2.4 Cookie Acknowledgement (COOKIE ACK) 9](#_Toc507268837)

[2.5 User Data Transfer 9](#_Toc507268838)

[2.5.1 Payload Data (DATA) 10](#_Toc507268839)

[2.5.2 Selected Acknowledgement (SACK) 10](#_Toc507268840)

[2.6 Shutdown Association (SHUTDOWN) 10](#_Toc507268841)

[2.7 Shutdown Acknowledgement (SHUTDOWN ACK) 11](#_Toc507268842)

[2.8 Shutdown Complete 11](#_Toc507268843)

[3.0 Application Logic 12](#_Toc507268844)

[3.1 Class and Method used for SCTP Protocol 12](#_Toc507268845)

[4.0 A Manual of the Application 17](#_Toc507268846)

[5.0 Wireshark Captures 17](#_Toc507268847)

[6.0 Reference 20](#_Toc507268848)

**List of Figures**

[Figure 1: Simplified OSI Layer Stack 5](file:///C:\Users\mrahm\Downloads\3rd%20Sem_IT\Mobile%20Computing\SCTP_Report.docx#_Toc507268929)

[Figure 2: SCTP Common Header Format 6](file:///C:\Users\mrahm\Downloads\3rd%20Sem_IT\Mobile%20Computing\SCTP_Report.docx#_Toc507268930)

[Figure 3: A Diagram of SCTP Association and Shutdown 7](file:///C:\Users\mrahm\Downloads\3rd%20Sem_IT\Mobile%20Computing\SCTP_Report.docx#_Toc507268931)

[Figure 4 The Format of INIT Chunk 8](file:///C:\Users\mrahm\Downloads\3rd%20Sem_IT\Mobile%20Computing\SCTP_Report.docx#_Toc507268932)

[Figure 5: Cookie Echo Packet Format 9](file:///C:\Users\mrahm\Downloads\3rd%20Sem_IT\Mobile%20Computing\SCTP_Report.docx#_Toc507268933)

[Figure 6: Data Packet Format 10](file:///C:\Users\mrahm\Downloads\3rd%20Sem_IT\Mobile%20Computing\SCTP_Report.docx#_Toc507268934)

[Figure 7: Wireshark Capture of Cookie Echo and Cookie Ack Chunk 17](#_Toc507268935)

[Figure 8: Wireshark Capture of Data Chunk 18](#_Toc507268936)

[Figure 9: Wireshark Capture of Sack Chunk 18](#_Toc507268937)

[Figure 10: Wireshark Capture of Shutdown and Shutdown Ack Chunk 19](#_Toc507268938)

[Figure 11: Wireshark Capture of Shutdown Complete Chunk 19](#_Toc507268939)

# **1.0 Principle of Projects and Protocol Functionality**

## **1.1 Overview**

Stream Control Transmission Protocol (SCTP) is a Transport layer protocol which is used to transmit multiple streams of data at the same time between two end-points which has established a connection in a network.

SCTP User Application

SCTP User Application

Logical Communication between SCTP Layers

SCTP Transport Service

SCTP Transport Service

Network Layer (IP)

Network Layer (IP)

Data Link Layer

Data Link Layer

Physical Layer

Physical Layer

Figure 1: Simplified OSI Layer Stack

## **1.2 Principle of Project**

The principle of this project is to emulate the behavior of the SCTP protocol.

## **1.3 Protocol Functionality**

The SCTP transport service can be decomposed into a number of functions but the main functionality is the association startup and takedown.

An association is initiated by a request from a SCTP user. It requires a four-way handshake to establish a connection. Firstly, the client sends an INIT chunk to the server to start an association. On receipt of the first packet the server sends an INIT-ACK chunk as a response. This INIT-ACK chunk contains a state cookie. The state cookie must contain a Message Authentication Code (MAC), along with a time stamp corresponding to the creation of the cookie, the life span of the state cookie, and the information necessary to establish the association. After getting the INIT-ACK signal, the client sends a COOKIE-ECHO which just echoes the state cookie. Then finally the server responses by sending COOKIE-ACK which acknowledges the COOKIE-ECHO and moves the association to established state.

SCTP also supports graceful close of an active association on request from the SCTP user. SCTP does not allow a half close situation. If one endpoint wants to close an association then the other endpoint must stop sending new data. If any data are remained in the queue, those are sent and then the association is closed.

### **1.3.1 SCTP Common Header**

According to the RFC 4960, SCTP packet format is composed of a common header and chunks. A chunk contains either control information or user data. The SCTP common header field contains Source Port Number, Destination Port number, Verification Tag and Checksum.

Destination Port Number (16 bits)

Source Port Number (16 bits)

Verification Tag (32 bits)

Checksum (32 bits)

Figure 2: SCTP Common Header Format

Source Port Number:

This is the SCTP sender’s port number. It can be used by the receiver in combination with the source IP address, the SCTP destination port, and possibly the destination IP address to identify the association to which this packet belongs. The port number 0 must not be used.

Destination Port Number:

This is the SCTP port number to which this packet is destined. The receiving host will use this port number to de-multiplex the SCTP packet to the correct receiving endpoint/application. The port number 0 must not be used.

Verification Tag:

The receiver of this packet uses the Verification Tag to validate the sender of this SCTP packet. On transmit, the value of this Verification Tag must be set to the value of the Initiate Tag received from the peer endpoint during the association initialization, with the following exceptions:

* A packet contains an INIT chunk must have a zero verification chunk.
* An INIT chunk must be the only chunk in the SCTP packet carrying it.

Checksum:

This field contains the checksum of this SCTP packet.

# **2.0 Principles of Designed Architecture**

To implement the behavior of the network elements, here java programming language is used. It is a terminal-based management menu which does not require any graphical user interface for configuring and controlling the scenario of the protocol communication and respective nodes. The architecture of the SCTP protocol is provided below:

Socket, bind, listen  
Accept(blocks)

Server

Socket  
Connect(blocks)

SHUTDOWN-COMPLETE

S

SHUTDOWN-ACK

SHUTDOWN

INIT-ACK

SACK

DATA

COOKIE-ACK

COOKIE-ECHO

INIT

INIT-ACK

Client

Figure 3: A Diagram of SCTP Association and Shutdown

## **2.1 Initiation (INIT)**

This chunk is used to initiate an association between two endpoints. The format of the INIT chunk is depicted below:

Optional/ Variable Length Parameters

Number of Inbound Streams

Number of Outbound Streams

Initial TSN

Advertised Receiver Window Credit (a\_rwnd)

Initiate Tag

Chunk Length

Chunk Flags

Type = 1

Figure 4 The Format of INIT Chunk

The INIT chunk contains some parameters which is known as mandatory. Those are:

* Initiate Tag
* Advertised Receiver Window Credit (a\_rwnd)
* Number of Outbound Streams
* Number of Inbound Streams
* Initial TSN

We have set the parameters for the INIT chunk and then sent it to the server from a client. If an INIT chunk is received with known parameters which are not optional parameter of the chunk, then the server should process the INIT chunk and send back an INIT ACK.

## **2.2 Initiation Acknowledgment (INIT ACK)**

This chunk is used to acknowledge the initiation of an SCTP association. The parameter part of the INIT ACK is formatted similarly to the INIT chunk. One difference is, it uses a State- Cookie.

### **2.2.1 State Cookie**

When sending an INIT ACK as a response to an INIT chunk. The sender of INIT ACK chunk creates a State Cookie and sends it in the State Cookie parameter of the INIT ACK. Inside this cookie, the sender should include a MAC when the state cookie is created.

## **2.3 Cookie Echo (COOKIE ECHO)**

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

Length

Chunk Flags

Type = 10

Cookie

Figure 5: Cookie Echo Packet Format

Cookie is a variable size. This field must contain the exact cookie received in the state cookie parameter from the previous INIT ACK. A Cookie Echo does NOT contain a State Cookie parameter; instead, the data within the State Cookie’s Parameter Value becomes the data within the Cookie Echo’s Chunk Value. This allows an implementation to change only the first 2 bytes of the State Cookie parameter to become a COOKIE ECHO chunk.

## **2.4 Cookie Acknowledgement (COOKIE ACK)**

This chunk is used only during the initialization of an association. It is used to acknowledge the receipt of a COOKIE ECHO chunk.

Length 

Chunk Flags

Type = 11

This chunk must precede any DATA or SACK chunk sent within the association but may be bundled with one or more DATA chunks or SACK chunk’s in the same SCTP packet. When a server sends Cookie Acknowledgement, it means the association is established.

## **2.5 User Data Transfer**

Data transmission only happens in the established state. An SCTP receiver must be able to receive a minimum of 1500 byte in one SCTP packet. This means that an SCTP endpoint must not indicate less than 1500 bytes in its initial a\_rwnd sent in the INIT or INIT ACK. For transmission efficiency, SCTP defines mechanisms for bundling of small user messages and fragmentation of large user messages.

### **2.5.1 Payload Data (DATA)**

The DATA chunk uses the following format:

U

E

B

Reserved

Length

Type = 0

TSN

Stream Sequence Number n

Stream Identifier S

Payload Protocol Identifier

User Data

Figure 6: Data Packet Format

This DATA chunk is sent to the server by client. Multiple message can be sent in one association.

### **2.5.2 Selected Acknowledgement (SACK)**

This chunk is sent to the peer endpoint to acknowledge received DATA chunks and to inform the peer endpoint of gaps in the received subsequences of DATA chunks as represented by their TSNs. The SACK MUST contain the Cumulative TSN Ack, Advertised Receiver Window Credit (a\_rwnd), Number of Gap Ack Blocks, and Number of Duplicate TSNs fields. By definition, the value of the Cumulative TSN Ack parameter is the last TSN received before a break in the sequence of received TSNs occurs; the next TSN value following this one has not yet been received at the endpoint sending the SACK. This parameter therefore acknowledges receipt of all TSNs less than or equal to its value.

## **2.6 Shutdown Association (SHUTDOWN)**

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

Chunk Flags

Length

Type = 7

Cumulative TSN Ack

Since the SHUTDOWN message does not contain Gap Ack Blocks, it cannot be used to acknowledge TSNs received out of order.

## **2.7 Shutdown Acknowledgement (SHUTDOWN ACK)**

This chunk must be sent by server to the client to acknowledge the receipt of the SHUTDOWN chunk at the completion of the shutdown process.

Length

Chunk Flags

Type = 8

## **2.8 Shutdown Complete**

This chunk must be used to acknowledge the receipt of the SHUTDOWN ACK chunk at the completion of the shutdown process

Length

T

Reserved

Type = 14

So, this is how the termination of association is happened. A shutdown of an association is considered a graceful close where all data in queue by either endpoint is delivered to the respective peers. However, in the case of a shutdown, SCTP does not support a half-open state wherein one side may continue sending data while the other end is closed. When a client performs shutdown, the association on each peer will stop accepting new data and only deliver data in queue at the time of sending or receiving chunk.

# **3.0 Application Logic**

## **3.1 Class and Method used for SCTP Protocol**

In this section we will describe about the classes and methods have been used.

A screenshot of a cell phone

Description generated with very high confidence

To create a communication between a client and a server, TCP socket is used as TCP is a connection-oriented protocol and useful for implementing network services. So, in the Client class at first-

* we have opened a socket and have given a port number on which the server will try to connect.
* Then, we have created Input and Output Stream to the socket so that we can read and write to the socket according to the SCTP protocol.
* Clean up

Now, in the Server class it will receive text from the client and then sends the exact text back to the client. This is how the socket will be created between the server and the client.

Then, we have used another class which is called XmlDataUtils to generate XML messages. Under this class, each method generates particular XML for each request such as- INIT, DATA, COOKIE etc. Methods will take some dynamic values as parameter and then will concatenate with string to generate as XML string message.

**Code Snippet for INIT Request**

/\*\*

\* Writes a String to a Output Stream As a Modified UTF-8 Encoding

\*/

**if** (sctpSocket != **null** && os != **null** && is != **null**) {

**try** {

System.***out***.println("Sending INIT request with verification tag: " + verficationTag);

os.writeUTF(XmlDataUtils.*GetInitXml*(srcPort, destPort, verficationTag));

/\*\*

\* Force Data to the Output Stream

\*/

os.flush();

After getting the initiation request from the client, the server will reply with an acknowledgement message.

Code Snippet for INIT ACK Response

/\*\*

\* Sending Response to the Client

\*/

**if** (Line.contains("<INIT>")) {

String start = "<sctp.verification\_tag>";

String end = "</sctp.verification\_tag>";

verificationTag = Line.substring(Line.indexOf(start) + start.length(), Line.indexOf(end));

System.***out***.println(">>>> INIT request received from client with verification tag: " + verificationTag);

System.***out***.println(">>>> Sending INIT\_ACK request with verification tag: " + verificationTag);

os.writeUTF(XmlDataUtils.*GetInitAckXml*(srcPort, destPort, verificationTag));

os.flush();

}

When Client will get INIT ACK, it will again send Cookie

Code Snippet for COOKIE ECHO Request

/\*\*

\* Sending Cookie Echo Request

\*/

**if** (responseLine.contains("INIT\_ACK")) {

System.***out***.println(">>>>INIT acknowledged by server with verification tag: " + verficationTag);

System.***out***.println(">>>>Sending COOKIE ECHO request.....");

os.writeUTF(XmlDataUtils.*GetCookieEchoXml*(srcPort, destPort, verficationTag));

os.flush();

}

After getting the Cookie Echo Request, the Server will reply by giving Cookie Acknowledgement.

Code Snippet for COOKIE ACK Response

/\*\*

\* Sending Cookie Acknowledgement Response

\*/

**else** **if** (Line.contains("<COOKIE\_ECHO>")) {

System.***out***.println(">>>> Received COOKIE\_ECHO from client");

System.***out***.println(">>>> Sending COOKIE\_ACK request.....");

os.writeUTF(XmlDataUtils.*GetCookieAckXml*(srcPort, destPort, verificationTag));

os.flush();

}

With the Cookie Acknowledgement message, the association will be established. Now the client can send data to the server.

Code Snippet for Data Chunk

/\*\*

\* Sending Data

\*/

**else** **if** (responseLine.contains("COOKIE\_ACK")) {

System.***out***.println(">>>>Cookie is acknowledged by server");

System.***out***.println("Enter a message:");

Scanner scanner = **new** Scanner(System.***in***);

String msg = scanner.nextLine();

scanner.close();

System.***out***.println(">>>>Sending data with message....");

os.writeUTF(XmlDataUtils.*GetDataXml*(srcPort, destPort, verficationTag, msg));

os.flush();

}

If the Client sends data chunk to the server, the server will respond with SACK chunk.

Code Snippet for SACK Chunk

/\*\*

\* Sending SACK Response

\*/

**else** **if** (Line.contains("<DATA>")) {

System.***out***.println(">>>> DATA is received from client");

String start = "<data.data>";

String end = "</data.data>";

String msg = Line.substring(Line.indexOf(start) + start.length(), Line.indexOf(end));

msg = msg.replaceAll(":", "");

System.***out***.println("\*\*\*\* Client message in DATA:");

System.***out***.println("----------------------------");

System.***out***.println(XmlDataUtils.*toString*(msg));

System.***out***.println();

System.***out***.println(">>>> Sending SACK request......");

os.writeUTF(XmlDataUtils.*GetSackAckXml*(srcPort, destPort, verificationTag));

os.flush();

}

If a client wants to terminate the association, it can send a Shutdown request to the server.

Code Snippet for Shutdown Chunk

/\*\*

\* Sending Shutdown Request

\*/

**else** **if** (choice == 2)

{

System.***out***.println();

System.***out***.println(">>>> Sending shutdown request.....");

os.writeUTF(XmlDataUtils.*GetShutDownXml*(srcPort, destPort, verficationTag));

os.flush();

}

Shutdown process needs three-way handshake. So, when a client sends Shutdown chunk, the server will send back Shutdown Acknowledgment message.

Code Snippet for Shutdown Ack Chunk

/\*\*

\* Sending Shutdown Acknowledge

\*/

**else** **if** (Line.contains("<SHUTDOWN>")) {

System.***out***.println(">>>> SHUTDOWN request is received from client");

System.***out***.println(">>>> Sending SHUTDOWN\_ACK request.....");

os.writeUTF(XmlDataUtils.*GetShudownAckXml*(srcPort, destPort, verificationTag));

os.flush();

}

And finally the client will send Shutdown Complete to terminate the association.

Code Snippet for Shutdown Complete Chunk

/\*\*

\* Sending Shutdown Complete Request

\*/

**else** **if** (responseLine.contains("SHUTDOWN\_ACK")) {

System.***out***.println();

System.***out***.println(">>>> SHUTDOWN is acknowledged by server");

System.***out***.println(">>>> Sending SHUTDOWN\_COMPLETE request....");

os.writeUTF(XmlDataUtils.*GetShutDownCompleteXml*(srcPort, destPort, verficationTag));

os.flush();

**break**;

}

# **4.0 A Manual of the Application**

In this chapter, we will describe how to execute the application. To observe the emulation of the SCTP protocol, a manual is given below:

* Run the jar file of Server application in a Linux Operating System.

Command: java -jar filename.jar

* Open the Wireshark and start taking capture on Loopback Interface.
* Run the jar file of Client application

Command: java -jar filename.jar

* Stop taking captures on Wireshark by pressing the stop button.

# **5.0 Wireshark Captures**

In this chapter, we have included some Wireshark captures of the SCTP protocol which is presented below:

A screenshot of a social media post

Description generated with very high confidence

Figure 7: Wireshark Capture of Cookie Echo and Cookie Ack Chunk

A close up of text on a white background

Description generated with very high confidence

Figure 8: Wireshark Capture of Data Chunk

A screenshot of a cell phone

Description generated with very high confidence

Figure 9: Wireshark Capture of Sack Chunk

A screenshot of a cell phone

Description generated with very high confidence

Figure 10: Wireshark Capture of Shutdown and Shutdown Ack Chunk

A screenshot of a social media post

Description generated with very high confidence

Figure 11: Wireshark Capture of Shutdown Complete Chunk

# **6.0 Reference**

IETF RFC 4960(2007): Stream Control Transmission Protocol(SCTP), IETF