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

List of Figures

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.

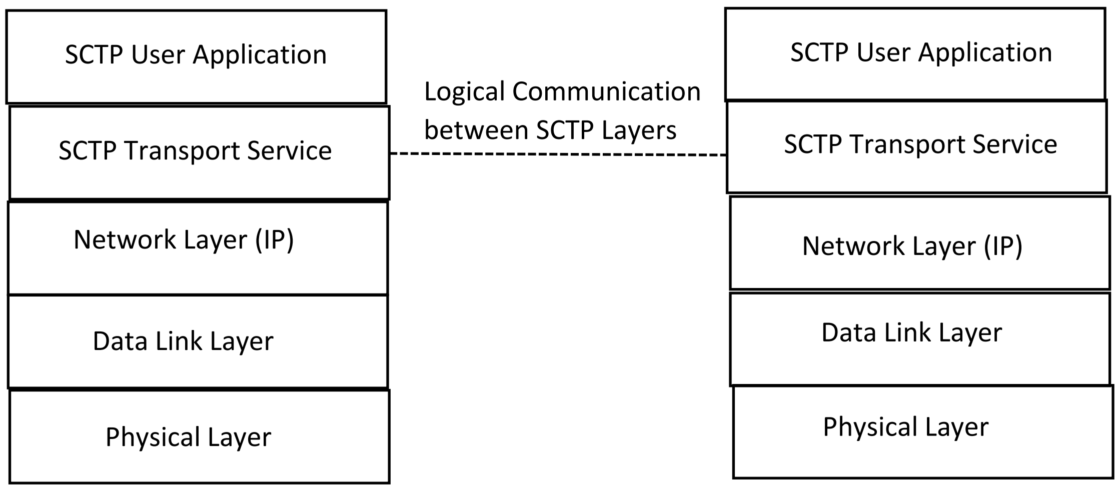


Figure 1: Simplified OSI Layer Stack

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.

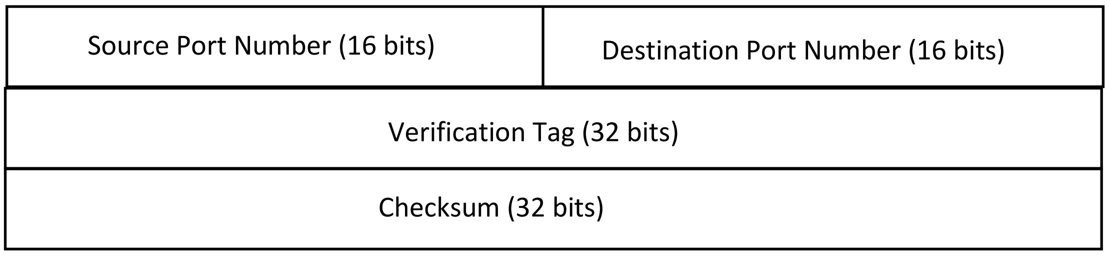

Source Port Number (16 bits)
Destination Port Number (16 bits)
Verification Tag (32 bits)
Checksum (32 bits)

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:

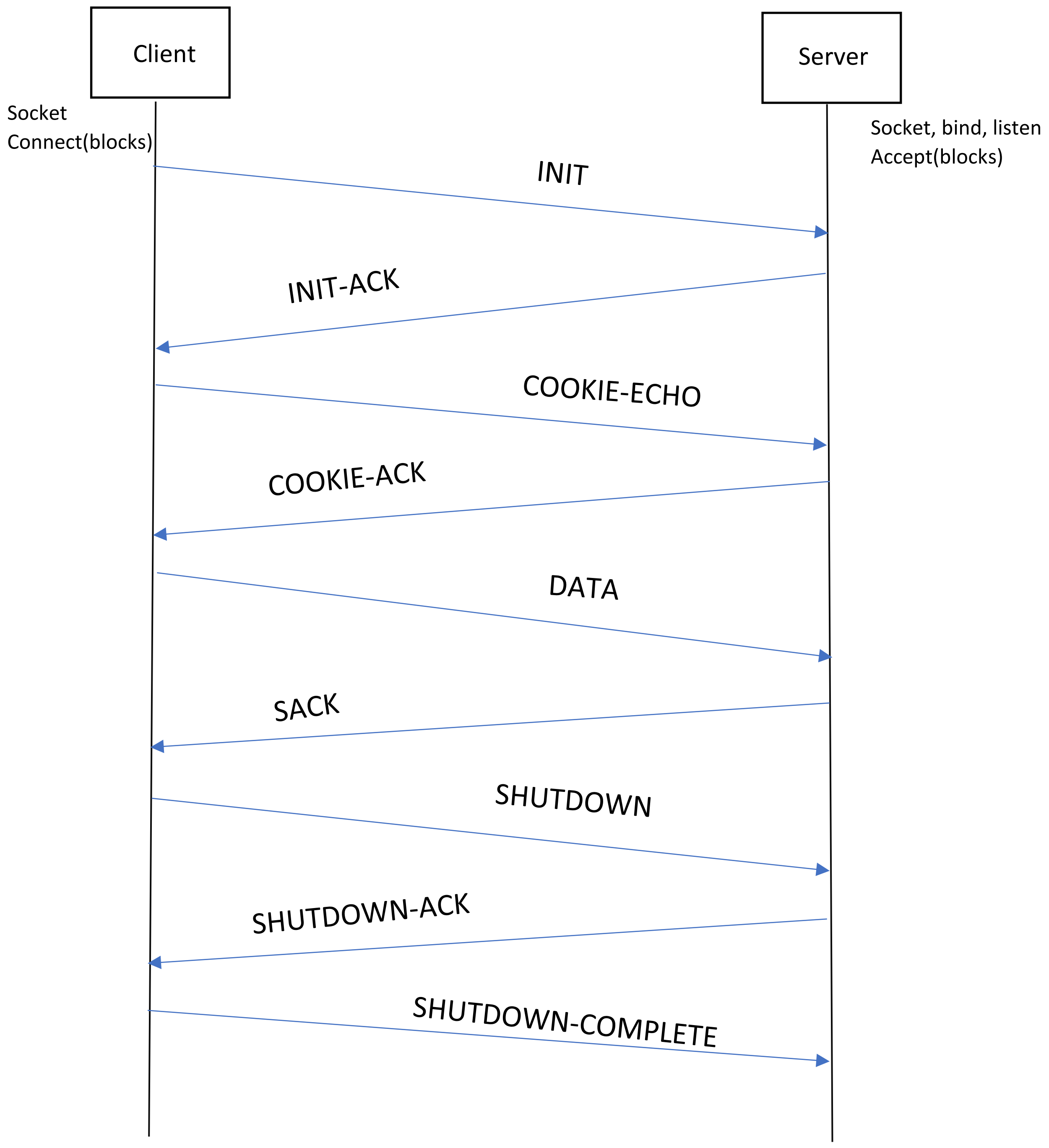


Figure 3: A Diagram of SCTP Association and Shutdown

2.1 Initiation (INIT)

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

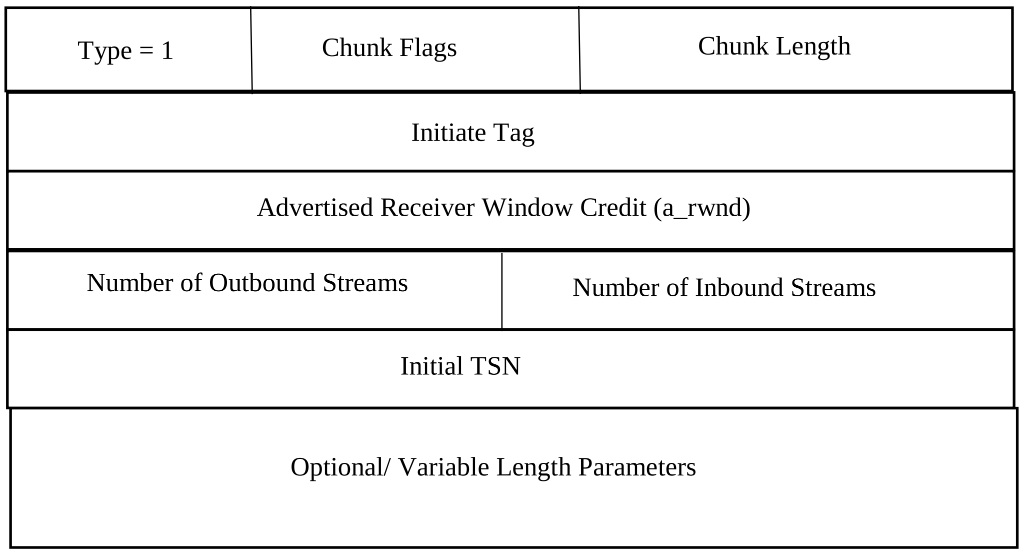

Type = 1
Chunk Flags
Chunk Length
Initiate Tag
Advertised Receiver Window Credit (a_rwnd)
Number of Outbound Streams
Number of Inbound Streams
Initial TSN
Optional/ Variable Length Parameters

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.

Type = 10	Chunk Flags	Length
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.

Type = 11	Chunk Flags	Length

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:

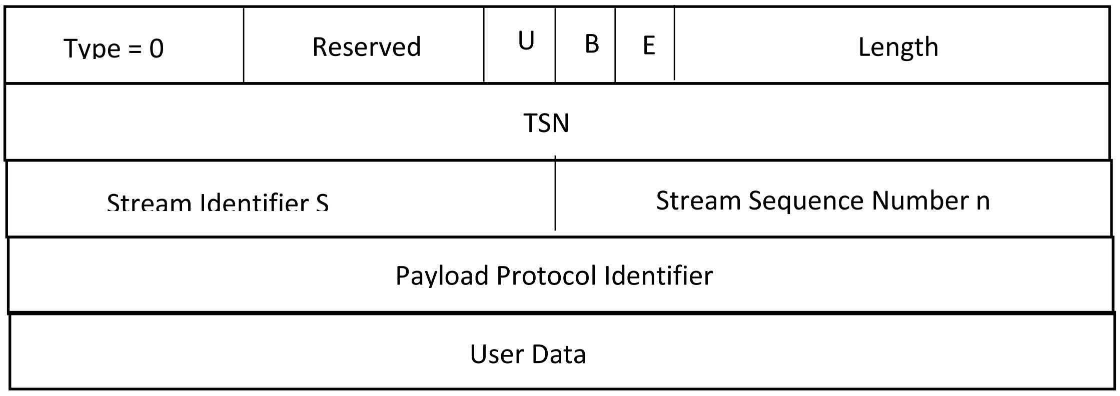


Figure 6: Data Packet Format

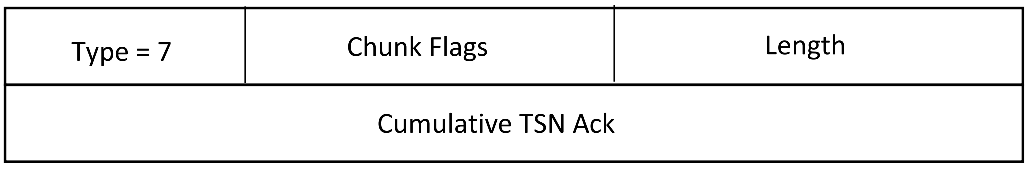
This DATA chunk is sent to the server by client. Multiple message can be sent in one association.

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.



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.

Type = 8	Chunk Flags	Length

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

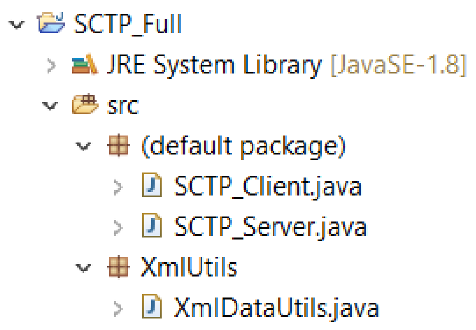
Type = 14	Reserved	T	Length

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.



```
SCTP_Full
    JRE System Library [JavaSE-1.8]
    src
        (default package)
            SCTP_Client.java
            SCTP_Server.java
        XmlUtils
            XmlDataUtils.java
```

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.

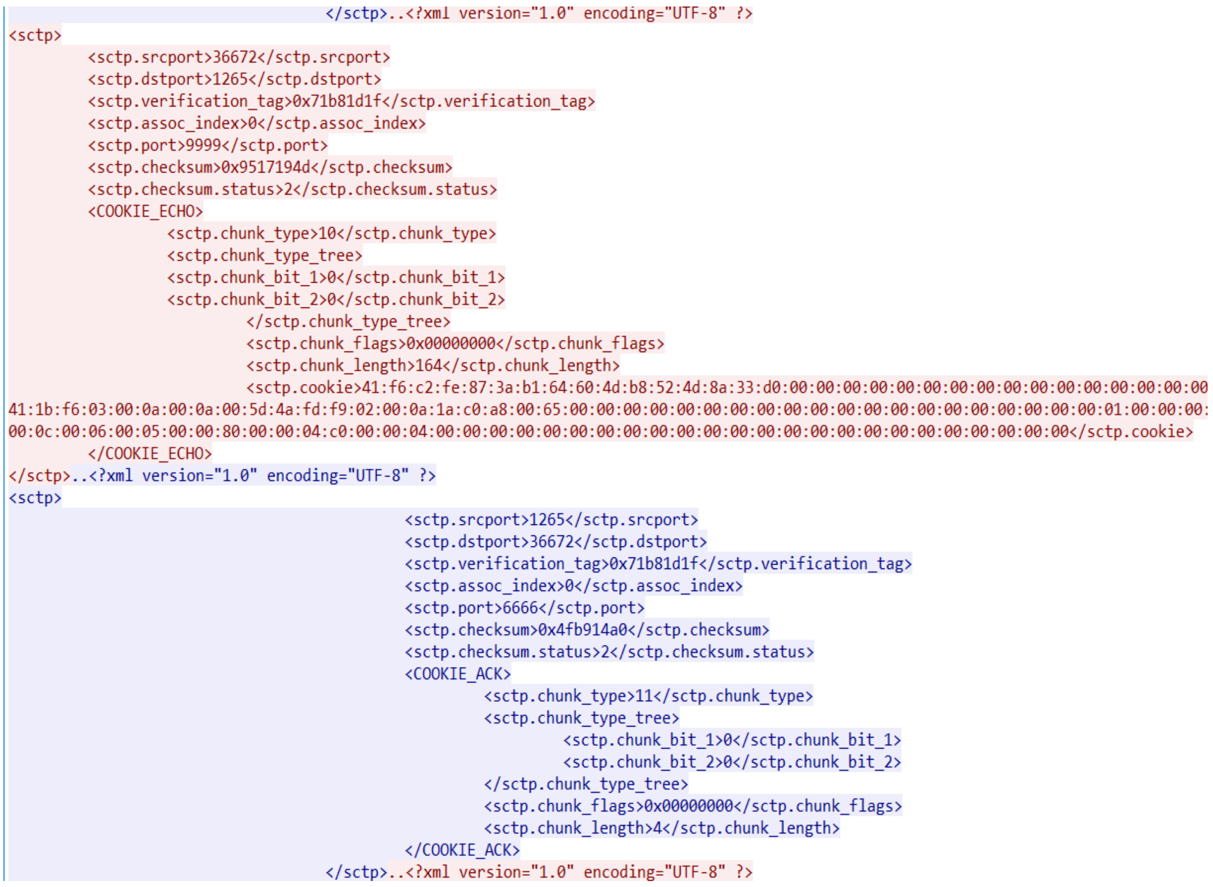
- Run the jar file of Client application

  Command: java -jar filename.jar

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

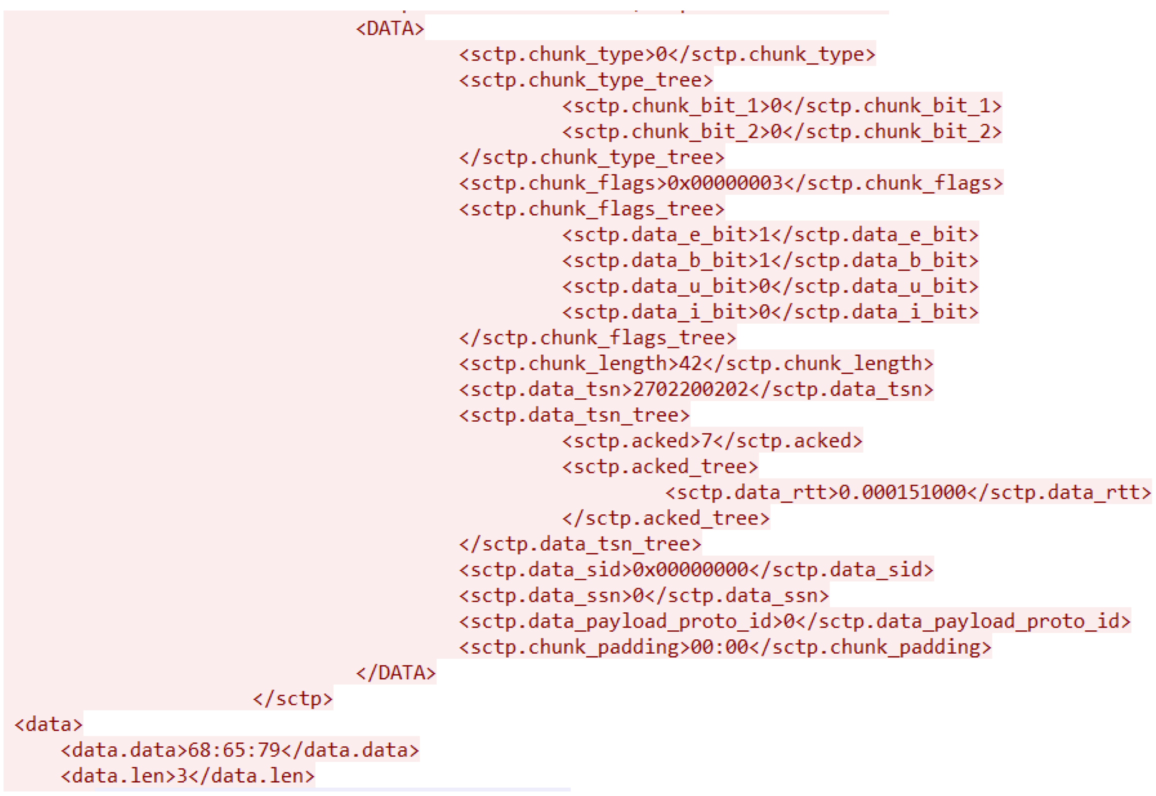
5.0 Wireshark Captures

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



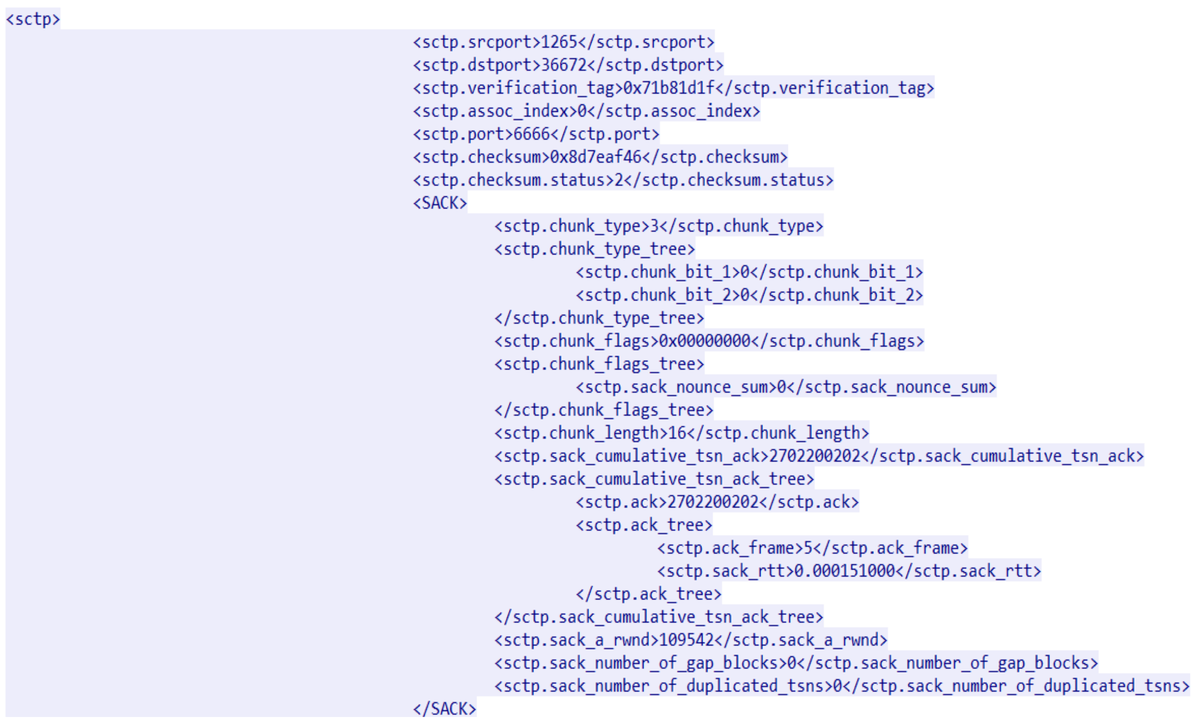
```
                                        </sctp>..<?xml version="1.0" encoding="UTF-8" ?>
<sctp>
          <sctp.srcport>36672</sctp.srcport>
          <sctp.dstport>1265</sctp.dstport>
          <sctp.verification_tag>0x71b81d1f</sctp.verification_tag>
          <sctp.assoc_index>0</sctp.assoc_index>
          <sctp.port>9999</sctp.port>
          <sctp.checksum>0x9517194d</sctp.checksum>
          <sctp.checksum.status>2</sctp.checksum.status>
          <COOKIE_ECHO>
                    <sctp.chunk_type>10</sctp.chunk_type>
                    <sctp.chunk_type_tree>
                    <sctp.chunk_bit_1>0</sctp.chunk_bit_1>
                    <sctp.chunk_bit_2>0</sctp.chunk_bit_2>
                              </sctp.chunk_type_tree>
                              <sctp.chunk_flags>0x00000000</sctp.chunk_flags>
                              <sctp.chunk_length>164</sctp.chunk_length>
                              <sctp.cookie>41:f6:c2:fe:87:3a:b1:64:60:4d:b8:52:4d:8a:33:d0:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
41:1b:f6:03:00:0a:00:0a:00:5d:4a:fd:f9:02:00:0a:1a:c0:a8:00:65:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:01:00:00:00:
00:0c:00:06:00:05:00:00:80:00:00:04:c0:00:00:04:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00</sctp.cookie>
          </COOKIE_ECHO>
</sctp>..<?xml version="1.0" encoding="UTF-8" ?>
<sctp>
                                                  <sctp.srcport>1265</sctp.srcport>
                                                  <sctp.dstport>36672</sctp.dstport>
                                                  <sctp.verification_tag>0x71b81d1f</sctp.verification_tag>
                                                  <sctp.assoc_index>0</sctp.assoc_index>
                                                  <sctp.port>6666</sctp.port>
                                                  <sctp.checksum>0x4fb914a0</sctp.checksum>
                                                  <sctp.checksum.status>2</sctp.checksum.status>
                                                  <COOKIE_ACK>
                                                            <sctp.chunk_type>11</sctp.chunk_type>
                                                            <sctp.chunk_type_tree>
                                                                      <sctp.chunk_bit_1>0</sctp.chunk_bit_1>
                                                                      <sctp.chunk_bit_2>0</sctp.chunk_bit_2>
                                                            </sctp.chunk_type_tree>
                                                            <sctp.chunk_flags>0x00000000</sctp.chunk_flags>
                                                            <sctp.chunk_length>4</sctp.chunk_length>
                                                  </COOKIE_ACK>
                                        </sctp>..<?xml version="1.0" encoding="UTF-8" ?>
```

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



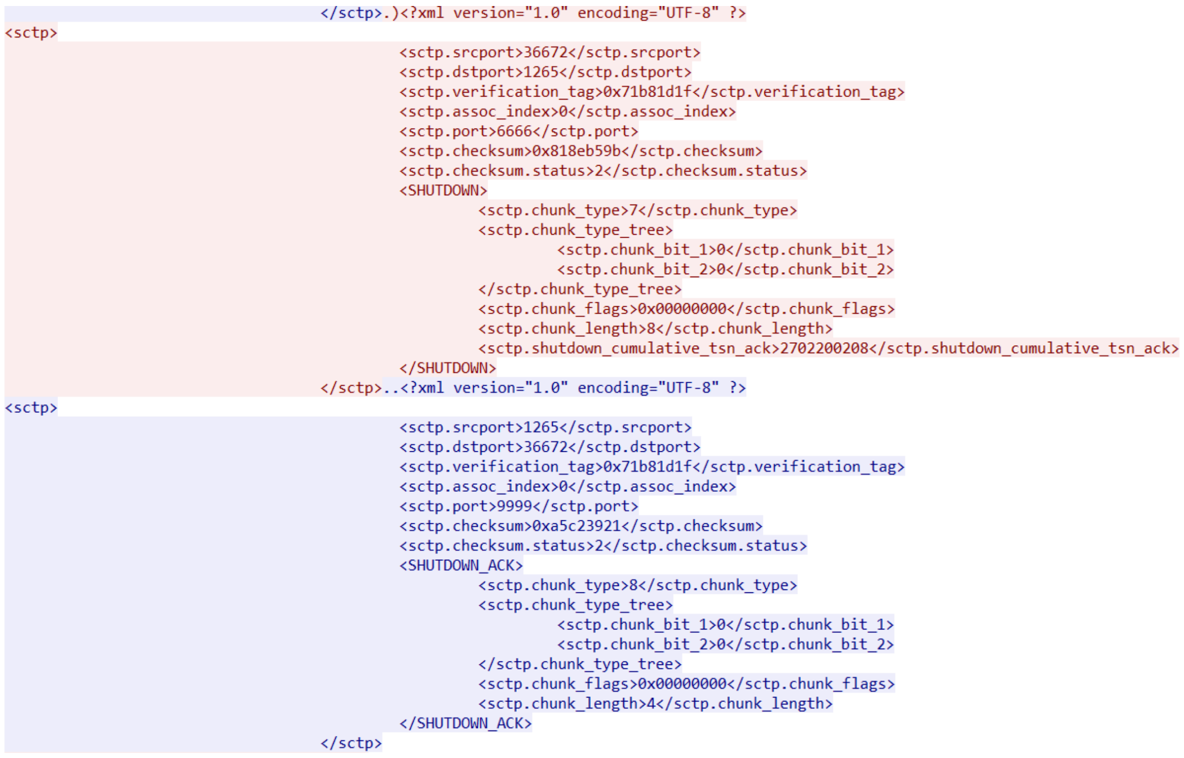
```
                        <DATA>
                                <sctp.chunk_type>0</sctp.chunk_type>
                                <sctp.chunk_type_tree>
                                        <sctp.chunk_bit_1>0</sctp.chunk_bit_1>
                                        <sctp.chunk_bit_2>0</sctp.chunk_bit_2>
                                </sctp.chunk_type_tree>
                                <sctp.chunk_flags>0x00000003</sctp.chunk_flags>
                                <sctp.chunk_flags_tree>
                                        <sctp.data_e_bit>1</sctp.data_e_bit>
                                        <sctp.data_b_bit>1</sctp.data_b_bit>
                                        <sctp.data_u_bit>0</sctp.data_u_bit>
                                        <sctp.data_i_bit>0</sctp.data_i_bit>
                                </sctp.chunk_flags_tree>
                                <sctp.chunk_length>42</sctp.chunk_length>
                                <sctp.data_tsn>2702200202</sctp.data_tsn>
                                <sctp.data_tsn_tree>
                                        <sctp.acked>7</sctp.acked>
                                        <sctp.acked_tree>
                                                <sctp.data_rtt>0.000151000</sctp.data_rtt>
                                        </sctp.acked_tree>
                                </sctp.data_tsn_tree>
                                <sctp.data_sid>0x00000000</sctp.data_sid>
                                <sctp.data_ssn>0</sctp.data_ssn>
                                <sctp.data_payload_proto_id>0</sctp.data_payload_proto_id>
                                <sctp.chunk_padding>00:00</sctp.chunk_padding>
                        </DATA>
                </sctp>
<data>
    <data.data>68:65:79</data.data>
    <data.len>3</data.len>
```

Figure 8: Wireshark Capture of Data Chunk



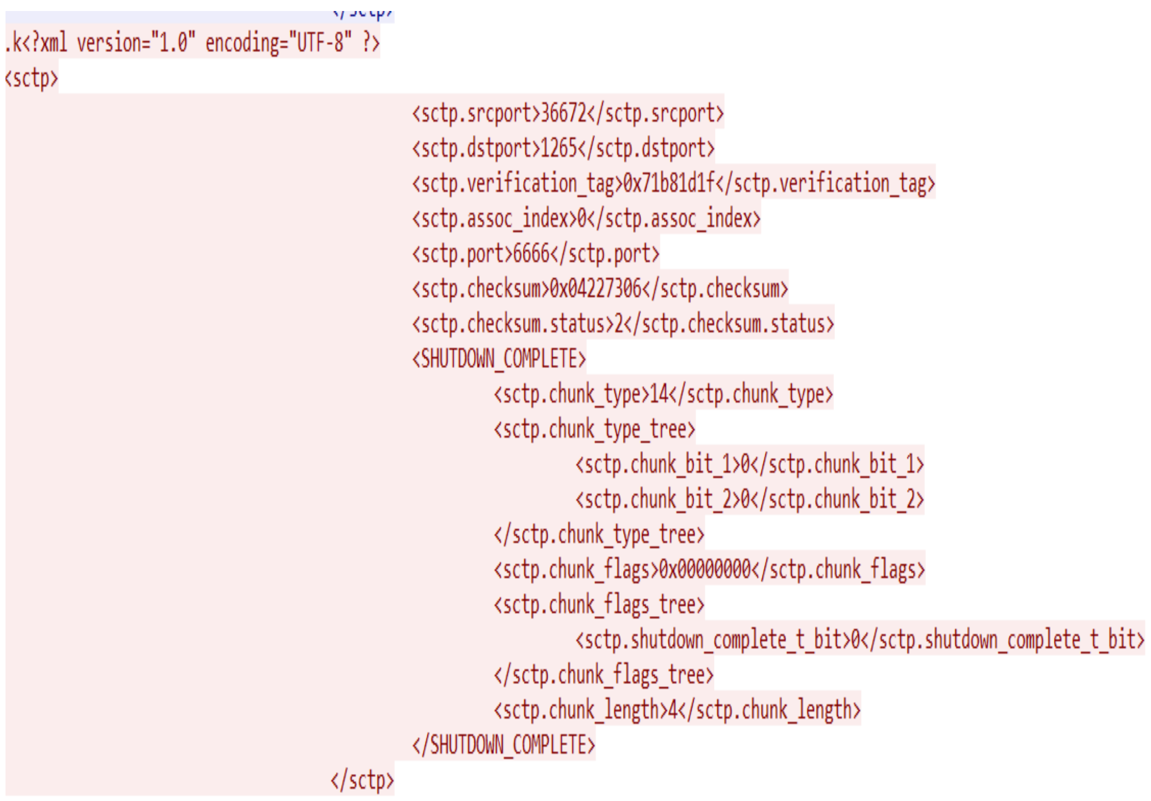
```
<sctp>
                        <sctp.srcport>1265</sctp.srcport>
                        <sctp.dstport>36672</sctp.dstport>
                        <sctp.verification_tag>0x71b81d1f</sctp.verification_tag>
                        <sctp.assoc_index>0</sctp.assoc_index>
                        <sctp.port>6666</sctp.port>
                        <sctp.checksum>0x8d7eaf46</sctp.checksum>
                        <sctp.checksum.status>2</sctp.checksum.status>
                        <SACK>
                                <sctp.chunk_type>3</sctp.chunk_type>
                                <sctp.chunk_type_tree>
                                        <sctp.chunk_bit_1>0</sctp.chunk_bit_1>
                                        <sctp.chunk_bit_2>0</sctp.chunk_bit_2>
                                </sctp.chunk_type_tree>
                                <sctp.chunk_flags>0x00000000</sctp.chunk_flags>
                                <sctp.chunk_flags_tree>
                                        <sctp.sack_nounce_sum>0</sctp.sack_nounce_sum>
                                </sctp.chunk_flags_tree>
                                <sctp.chunk_length>16</sctp.chunk_length>
                                <sctp.sack_cumulative_tsn_ack>2702200202</sctp.sack_cumulative_tsn_ack>
                                <sctp.sack_cumulative_tsn_ack_tree>
                                        <sctp.ack>2702200202</sctp.ack>
                                        <sctp.ack_tree>
                                                <sctp.ack_frame>5</sctp.ack_frame>
                                                <sctp.sack_rtt>0.000151000</sctp.sack_rtt>
                                        </sctp.ack_tree>
                                </sctp.sack_cumulative_tsn_ack_tree>
                                <sctp.sack_a_rwnd>109542</sctp.sack_a_rwnd>
                                <sctp.sack_number_of_gap_blocks>0</sctp.sack_number_of_gap_blocks>
                                <sctp.sack_number_of_duplicated_tsns>0</sctp.sack_number_of_duplicated_tsns>
                        </SACK>
```

Figure 9: Wireshark Capture of Sack Chunk



```
                                    </sctp>.)<?xml version="1.0" encoding="UTF-8" ?>
<sctp>
                                             <sctp.srcport>36672</sctp.srcport>
                                             <sctp.dstport>1265</sctp.dstport>
                                             <sctp.verification_tag>0x71b81d1f</sctp.verification_tag>
                                             <sctp.assoc_index>0</sctp.assoc_index>
                                             <sctp.port>6666</sctp.port>
                                             <sctp.checksum>0x818eb59b</sctp.checksum>
                                             <sctp.checksum.status>2</sctp.checksum.status>
                                             <SHUTDOWN>
                                                      <sctp.chunk_type>7</sctp.chunk_type>
                                                      <sctp.chunk_type_tree>
                                                               <sctp.chunk_bit_1>0</sctp.chunk_bit_1>
                                                               <sctp.chunk_bit_2>0</sctp.chunk_bit_2>
                                                      </sctp.chunk_type_tree>
                                                      <sctp.chunk_flags>0x00000000</sctp.chunk_flags>
                                                      <sctp.chunk_length>8</sctp.chunk_length>
                                                      <sctp.shutdown_cumulative_tsn_ack>2702200208</sctp.shutdown_cumulative_tsn_ack>
                                             </SHUTDOWN>
                                    </sctp>..<?xml version="1.0" encoding="UTF-8" ?>
<sctp>
                                             <sctp.srcport>1265</sctp.srcport>
                                             <sctp.dstport>36672</sctp.dstport>
                                             <sctp.verification_tag>0x71b81d1f</sctp.verification_tag>
                                             <sctp.assoc_index>0</sctp.assoc_index>
                                             <sctp.port>9999</sctp.port>
                                             <sctp.checksum>0xa5c23921</sctp.checksum>
                                             <sctp.checksum.status>2</sctp.checksum.status>
                                             <SHUTDOWN_ACK>
                                                      <sctp.chunk_type>8</sctp.chunk_type>
                                                      <sctp.chunk_type_tree>
                                                               <sctp.chunk_bit_1>0</sctp.chunk_bit_1>
                                                               <sctp.chunk_bit_2>0</sctp.chunk_bit_2>
                                                      </sctp.chunk_type_tree>
                                                      <sctp.chunk_flags>0x00000000</sctp.chunk_flags>
                                                      <sctp.chunk_length>4</sctp.chunk_length>
                                             </SHUTDOWN_ACK>
                                    </sctp>
```

Figure 10: Wireshark Capture of Shutdown and Shutdown Ack Chunk



```
.k<?xml version="1.0" encoding="UTF-8" ?>
<sctp>
                                             <sctp.srcport>36672</sctp.srcport>
                                             <sctp.dstport>1265</sctp.dstport>
                                             <sctp.verification_tag>0x71b81d1f</sctp.verification_tag>
                                             <sctp.assoc_index>0</sctp.assoc_index>
                                             <sctp.port>6666</sctp.port>
                                             <sctp.checksum>0x04227306</sctp.checksum>
                                             <sctp.checksum.status>2</sctp.checksum.status>
                                             <SHUTDOWN_COMPLETE>
                                                      <sctp.chunk_type>14</sctp.chunk_type>
                                                      <sctp.chunk_type_tree>
                                                               <sctp.chunk_bit_1>0</sctp.chunk_bit_1>
                                                               <sctp.chunk_bit_2>0</sctp.chunk_bit_2>
                                                      </sctp.chunk_type_tree>
                                                      <sctp.chunk_flags>0x00000000</sctp.chunk_flags>
                                                      <sctp.chunk_flags_tree>
                                                               <sctp.shutdown_complete_t_bit>0</sctp.shutdown_complete_t_bit>
                                                      </sctp.chunk_flags_tree>
                                                      <sctp.chunk_length>4</sctp.chunk_length>
                                             </SHUTDOWN_COMPLETE>
                                    </sctp>
```

Figure 11: Wireshark Capture of Shutdown Complete Chunk

6.0 Reference

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