SCTPTrace

An extension of tcptrace for SCTP

Christos Christoforidis
Niklas Ivarsson
Henrik Johansson
Jonas Nilsson

Date:	21-01-16
Instructor:	Karl-Johan Grinnemo

Table of contents

Introduction

When it comes to analyzing TCP [4] data and extracting the information in such a way that it becomes viewable, there are a couple of tools, of which one of them is tcptrace. Tcptrace is an analyzing tool that is used to analyze special dump files created from programs such as tcpdump, snoop and WinDump. Tcptrace became published for a broader public in the late 1996 by Shawn Ostermann. Since then functionalities, changes and fixes has been implemented for example the extension to create graphs and trace UDP packets [2]. From the dump files a trace will be done, and depending on the input from the user, tcptrace can present this information in a number of ways such as plain text, trace files, graphs etc. all depending on the amount of information the user is looking for. The extensive information traced will be viewed and divided for each connection found. For each connection, information such as retransmits, throughput, round trip times, bytes and packets sent and received etc. can be presented [2].

This project came to be, since there has been a desire to see a tool for SCTP that provides the same functionalities as tcptrace. The project, called sctptrace, aimed to implement as much of the previous TCP functionalities as possible for the SCTP protocol.

Overview of SCTP

SCTP is a protocol which is designed to transport PSTN (Public Switched Telephone Network) signal messages over IP networks, but with a little imagination it has additional uses as well. SCTP is a protocol that works on top of a connectionless packet network, and is designed in such a way that includes congestion avoidance and resistance to flooding and masquerade attacks. Some of the services that it provides is:

- acknowledged error-free, non-duplicated transfer of user data.
- sequenced delivery of user messages within multiple streams, with an option for order-of-arrival delivery of individual user messages.
- network-level fault tolerance through supporting of multihoming at either or both ends of an association.

SCTPTrace have a lot of functions. Some of them which have been developed are:

Association:
This is a relationship between different SCTP endpoints, it contains the necessary information in order to maintain a connection such as the protocol state information which includes the verification tags and the transmission sequence numbers that is currently active. Association provides the possibility to transfer information over multiple paths and in different logical streams [1].

Multihoming:
A more formal approach of multihoming is that it allows for multiple paths between two SCTP endpoints. A path is the route taken from the sender to one of the receiver end ports. It should be noted that by sending a packet to another endpoint doesn't necessary imply that another path is used [1]. An example of multihoming could be say that each user has a wifi and an ethernet connection available, then there are four possibilities for paths, wifi-ethernet, ethernet-wifi, ethernet-ethernet and wifi-wifi.

Streams:
A stream is a channel between two SCTP endpoints. All messages are transmitted in sequence by the stream [1]. The relation behind streams can be split up between different logical streams that are independent, which gives us the effect that a delay on one stream does not inflict any delay on other streams. One stream could be used to carry the control data, another stream could be used to transport smaller pieces of data, and a third stream could be used to carry larger pieces of data. These are for example divided in order to make sure that the large files don't delay the control information.

RTT:
RTT (round-trip time) is the time elapsed between the transmission of a packet and its acknowledgement.

Design and implementation

The connection between an association and streams

The *ttp* array, described in section under "*Important Data Structures*", may consist of several associations. Each association has a list of paths, creating a linked list of paths. Each element in this list contains two *tcbs*, one for each direction. Each *tcb* got a linked list containing stream specific information. Figure 1 illustrates the association structure.

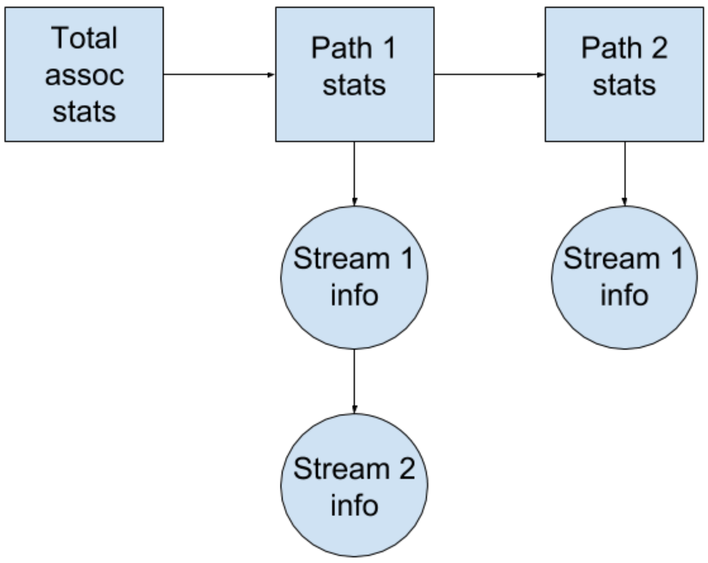


Figure 1: The association structure.

Important Data Structures

tcb (declared in tcptrace.h)

Tcb is the largest data structure in the program. It contains both SCTP-specific information (e.g., total chunks [5]) and general information about packets (such as the total number of packets). The structure also contains a pointer to both *sctp_pair* and *stream_info;* the *sctp_pair* is the start of the list of paths, the *sctp_pair* is the start of the list of streams.

stream_info (declared in tcptrace.h)

As the name implies this structure contains all the information needed for a stream. The structure has a number to identify the stream, a pointer to *data_info,* and a pointer to the next *stream_info* in the list.

data_info (declared in tcptrace.h)

The data structure contains specific information for data packets. It is used to save stream-specific information such as the amount of data bytes sent on a stream.

sctp_pair (declared in tcptrace.h)

The data structure includes information for TCP connections and SCTP associations. It includes a *tcp_pair_addrblock* (defined page 4) as well as names. Furthermore, it contains *tcbs* for both directions. It also has a pointer to another *sctp_pair* to support multihoming. When you go to the next *sctp_pair,* you get the information for another path in the association. Note that paths will exist even if no traffic has been recorded on them. Thus, when you encounter a path in the list without traffic, there might still exist more paths further down the list that contain traffic.

tcp_pair_addrblock (declared in tcptrace.h)

The data structure contains multiple IP addresses and port numbers for a TCP connection. A pointer to another *addr* block has been added in order to support multihoming in SCTP.

ptp_snap (declared in tcptrace.h)

This is a snapshot of a *sctp_pair*, it contains the address block of a *sctp_pair* and a pointer to that *sctp_pair*. It also has pointers to two other snapshots called *left* and *right*. Every snapshot is part of an AVL tree [3]. The AVL tree is used to find the correct *sctp_pair* by comparing the IP addresses and port numbers with the snapshots placed in the AVL tree.

tcp_pair ttp[] (declared in tcptrace.h)

This array used to contain TCP connections, but the *tcp_pair* data structure has been altered so that SCTP specific information is also contained in the *tcp_pair* structure. The *tcp_pair* structure has been renamed to *sctp_pair*. In the case of TCP traffic, it contains TCP connection however if instead there is SCTP traffic, it contains SCTP associations.

Important functions

trace_done (declared in trace.c)

The function calls the main printing function something which is done once the trace has been completed. It starts by printing the brief information about the trace. If the appropriate flag has been set, the trace information is followed by debugging information. The final call is to print the actual information, if it's an SCTP connection then it will print the information about each association and call *SctpPrintTrace*. If the trace has been done and it doesn't contain any SCTP packets, then the TCP information is being printed.

ProcessFile (declared in tcptrace.c)

The function reads in the file and divides it into packets. It checks whether the packet is a TCP, UDP, or SCTP packet by using the *getsctpheader*, *gettcpheader* and *getudpheader* functions; sets the variables, and moves on to the packet trace.

CopyAddr (declared in trace.c)

The function copies an IPv4 or an IPv6 address and ports numbers into an *addrblock* structure. It also creates a hash value which is based on both IP addresses and port numbers.

WhichDir (declared in trace.c)

Whichdir verifyies the IP addresses and port numbers and return the packet direction.

findStream (declared in output.c)

The function uses a *tcb* packet and an ID to search through the *tcbs* stream list in order to find the correct stream that corresponds to the ID. It is used in the *dosctptrace* to add all streams from the different paths to the pertinent association.

SctpPrintTrace (declared in trace.c)

The function is the primary function that prints the information that has been gathered from the input file. It starts by collecting the information from each path and add it to the corresponding association. This is followed by collecting the time information from the first and last packet sent. It prints the gathered information for the given association depending on what flags have been set. Next, it calculates the stream length, prints the information, followed by some throughput and RTT information.

GetDataInfo (declared in output.c)

It copies the stream list from the packet into a local variable, and then checks if this is empty or not. If not empty, it uses the given ID and searches the stream list for the corresponding ID. If found, it returns the stream object with the corresponding ID, otherwise it returns a false.

get_assoc_ip (declared in trace.c)

The function puts all the IP pairs in an association into a list. This function enables support for multihoming in SCTP. This function is invoked when an INIT and an INIT_ACK chunk have been found. It is important to acknowledge the difference between INIT and INIT_ACK IP: s, which is made by separating them into *a_address* and *b_address*. At the moment, the support for IPv6 is not implemented.

ParseArgs (declared in tcptrace.c)

The function checks arguments that have been set. When there are incomplete arguments, the function *BadArg* is called which prints the string supplied as argument. When there are correct arguments, certain global variables are set.

UniqueStreamCounter (declared in output.c)

It creates a list of unique stream IDs. This list is used when printing the stream information.

get_stream_tcb (declared in trace.c)

Gets the specific stream information pertinent to a specific stream. The function is used when deciding on which stream to add information to.

dosctptrace (declared in trace.c)

The function was created from *dotrace*. Modifications have been done in order to support SCTP. Some TCP-specific code may still be present. The TCP specific code has as far as we can tell no effect on our results.

When entering this function, there is a pointer to a SCTP header declared and initiated. The function extracts the common header to process it for information. After the common header has been processed, the function extracts SCTP chunks and processes them one at a time. Every chunk header got the same data fields (*chunktype*, *chunklength*). Macros are used to check the *chunktype* of a chunk. If we want to extract information unique to a INIT chunk, we use the INIT_SET macro to ascertain that we got the correct type before we continue extracting the information, same goes for any other chunk type.

An association is created when the INIT_ACK chunk is received, or more accurately, the multihoming which is supported in the SCTP protocol may give additional IP addresses on either side, these IPs will be added to the existing structure, forming an association. For all the IPs, pairs of IP addresses which are possible paths are created; these paths are later added to the association list.

NewTTPsctp (declared in trace.c)

A function that was altered from a function that is used to create connections in TCP. The function is called to create associations as well as individual paths in SCTP. When used to create individual paths, some modifications, such as correcting the IP addresses, need to be done afterwards.

FindTTPsctp (declared in trace.c)

A function that is basically a copy of a similar function for TCP that is used to find connections. It is used to correctly identify a packet with its association or path. It uses an AVL tree to find the path.

getsctp (declared in tcptrace.c)

The function returns a pointer to the next SCTP header. This implies that we have read in the packet into memory, and can start processing it for information.

sack_in (declared in rexmit.c)

The function is checking the packets, the time values, and collects the proper RTT samples.

rtt_ackin (declared in rexmit.c)

rtt_ackin calculates the RTT values. It gets called from the *sack_in* function.

The implementation for RTT calculation has been copied from *tcptracer* and reconstructed for the *sctptracer*. The main calculation is that the program takes the time for the data packet and tries to find the time for the acknowledgment of the data packet. Next, the program creates a RTT sample with the help of the previous values. When the user executes the graph command, the program takes all the values of rtt samples and plots them like figure 2.

User manual

When using the SCTPtrace, combinations of different flags can be used in order to view the trace result in different ways. Some of the combinations are demonstrated though examples.

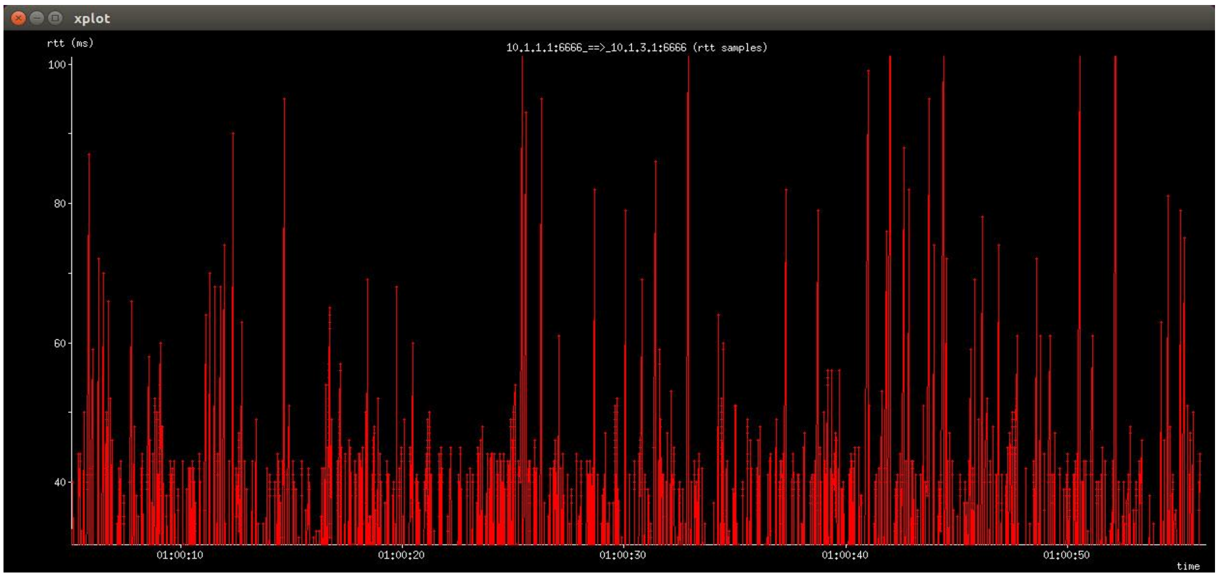


Figure 2: RTT graph for a trace file in xplot.

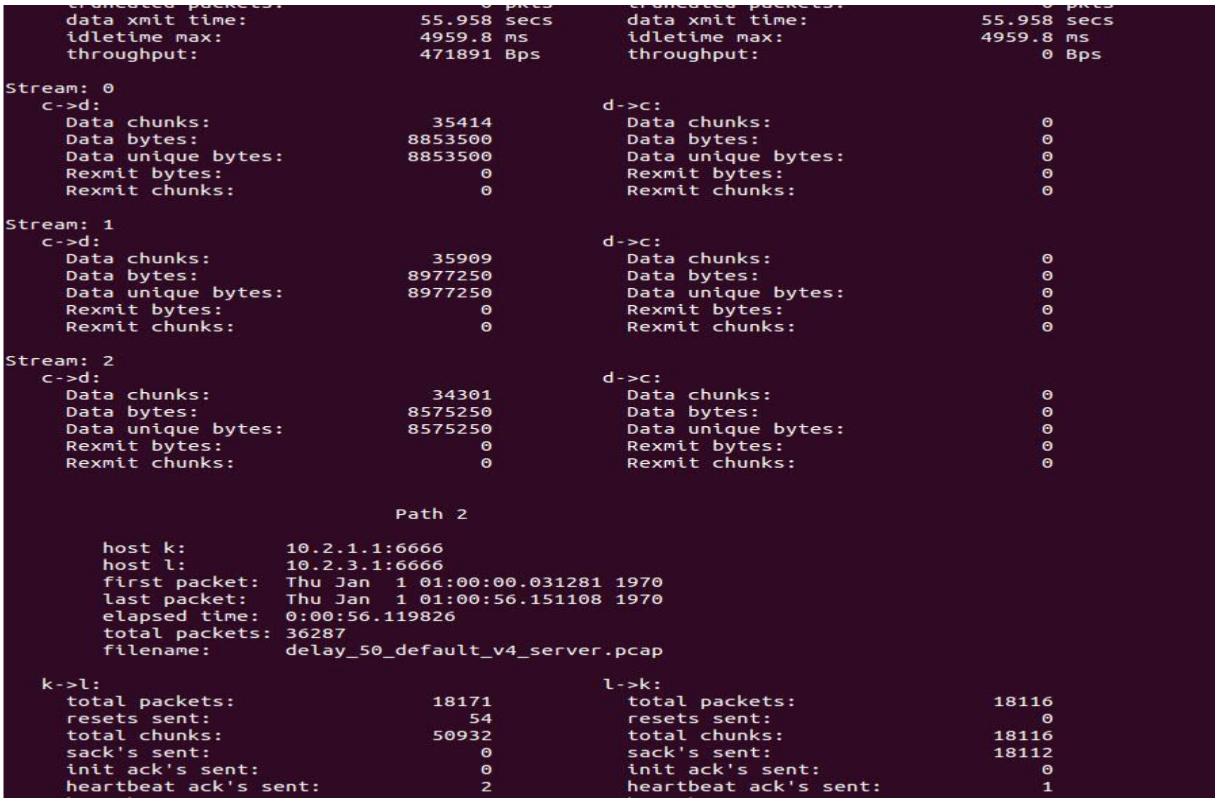
Figure 2, shows the plotted RTT values for a large trace file, in *xplot* [6] it's possible to zoom in in a specific part of the graph to get a more accurate view.

The command that has been used is:

./tcptrace -l -G < file.txt,

where *file.txt* is the name of the dump file.This will create plotfiles that has the ending ".xpl" which can be opened using the *xplot.org* program with the command:

./xplot.org < a2b_rtt.xpl



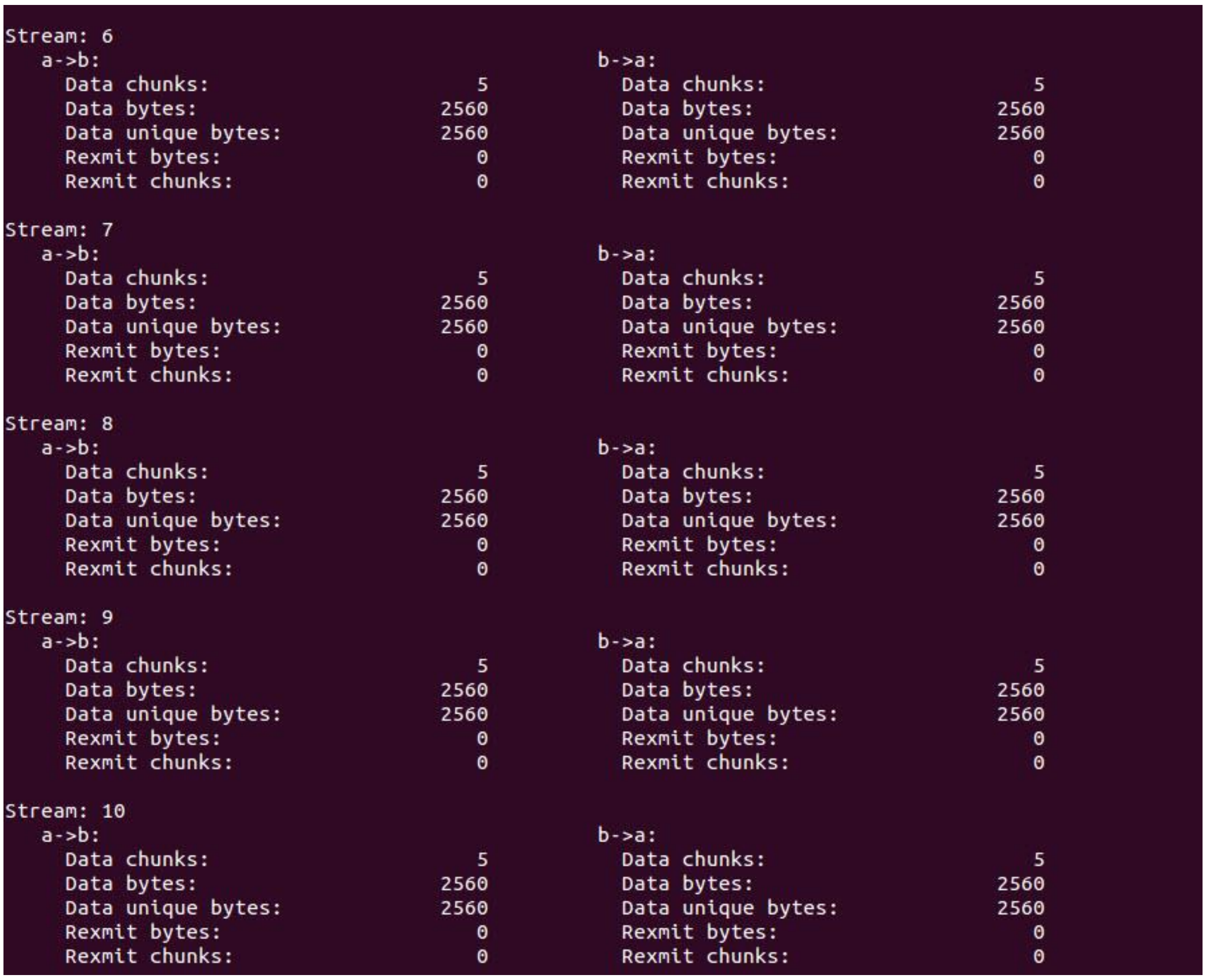
```
     data xmit time:              55.958 secs      data xmit time:              55.958 secs
     idletime max:                4959.8 ms        idletime max:                4959.8 ms
     throughput:                  471891 Bps       throughput:                       0 Bps

Stream: 0
   c->d:                                         d->c:
     Data chunks:                  35414           Data chunks:                      0
     Data bytes:                 8853500           Data bytes:                       0
     Data unique bytes:          8853500           Data unique bytes:                0
     Rexmit bytes:                     0           Rexmit bytes:                     0
     Rexmit chunks:                    0           Rexmit chunks:                    0

Stream: 1
   c->d:                                         d->c:
     Data chunks:                  35909           Data chunks:                      0
     Data bytes:                 8977250           Data bytes:                       0
     Data unique bytes:          8977250           Data unique bytes:                0
     Rexmit bytes:                     0           Rexmit bytes:                     0
     Rexmit chunks:                    0           Rexmit chunks:                    0

Stream: 2
   c->d:                                         d->c:
     Data chunks:                  34301           Data chunks:                      0
     Data bytes:                 8575250           Data bytes:                       0
     Data unique bytes:          8575250           Data unique bytes:                0
     Rexmit bytes:                     0           Rexmit bytes:                     0
     Rexmit chunks:                    0           Rexmit chunks:                    0


                                Path 2

        host k:         10.2.1.1:6666
        host l:         10.2.3.1:6666
        first packet:   Thu Jan  1 01:00:00.031281 1970
        last packet:    Thu Jan  1 01:00:56.151108 1970
        elapsed time:   0:00:56.119826
        total packets:  36287
        filename:       delay_50_default_v4_server.pcap

   k->l:                                         l->k:
     total packets:                18171           total packets:                18116
     resets sent:                     54           resets sent:                      0
     total chunks:                 50932           total chunks:                 18116
     sack's sent:                      0           sack's sent:                  18112
     init ack's sent:                  0           init ack's sent:                  0
     heartbeat ack's sent:             2           heartbeat ack's sent:             1
```

Figure 3: Information about streams and paths.

Figure 3, shows the presented information about streams and paths when both the *-path* and the *-stream* flags are used. First, the information about the path is presented followed by information about the different streams on that specific path. The command that has been used is:

./tcptrace -l -path -stream < file.txt

where *file.txt* is the name of the dump file. By using the *-l* flag you get the extended view of the information. And, by using the *-stream* and the *-path* flags together, the information is presented in such a way that for each association each path is viewed, and for each path each stream is viewed.



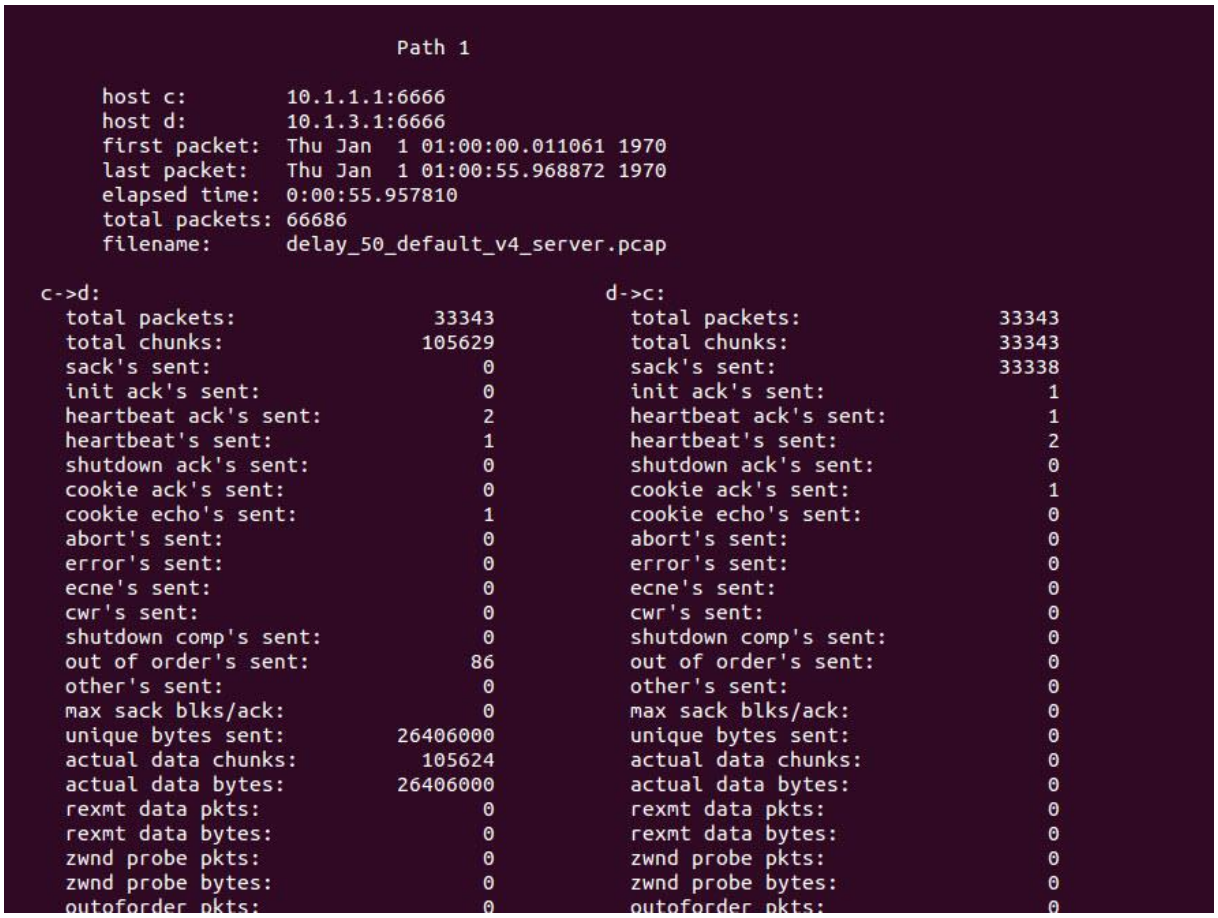
```
Stream: 6
   a->b:                                         b->a:
     Data chunks:                    5             Data chunks:                    5
     Data bytes:                  2560             Data bytes:                  2560
     Data unique bytes:           2560             Data unique bytes:           2560
     Rexmit bytes:                   0             Rexmit bytes:                   0
     Rexmit chunks:                  0             Rexmit chunks:                  0

Stream: 7
   a->b:                                         b->a:
     Data chunks:                    5             Data chunks:                    5
     Data bytes:                  2560             Data bytes:                  2560
     Data unique bytes:           2560             Data unique bytes:           2560
     Rexmit bytes:                   0             Rexmit bytes:                   0
     Rexmit chunks:                  0             Rexmit chunks:                  0

Stream: 8
   a->b:                                         b->a:
     Data chunks:                    5             Data chunks:                    5
     Data bytes:                  2560             Data bytes:                  2560
     Data unique bytes:           2560             Data unique bytes:           2560
     Rexmit bytes:                   0             Rexmit bytes:                   0
     Rexmit chunks:                  0             Rexmit chunks:                  0

Stream: 9
   a->b:                                         b->a:
     Data chunks:                    5             Data chunks:                    5
     Data bytes:                  2560             Data bytes:                  2560
     Data unique bytes:           2560             Data unique bytes:           2560
     Rexmit bytes:                   0             Rexmit bytes:                   0
     Rexmit chunks:                  0             Rexmit chunks:                  0

Stream: 10
   a->b:                                         b->a:
     Data chunks:                    5             Data chunks:                    5
     Data bytes:                  2560             Data bytes:                  2560
     Data unique bytes:           2560             Data unique bytes:           2560
     Rexmit bytes:                   0             Rexmit bytes:                   0
     Rexmit chunks:                  0             Rexmit chunks:                  0
```

Figure 4: Stream specific information.

Figure 4, shows the presented information when only the *-stream* flag is used. First, at the top is the total information about the whole association shown. Next is shown the information of each stream in the association. The command that has been used is:

./tcptrace -l -stream < file.txt

where *file.txt* is the name of the dump file. By using the *-l* flag, you get the extended view of the information, and by using the *-stream* flag the collected information of the streams will be printed out.



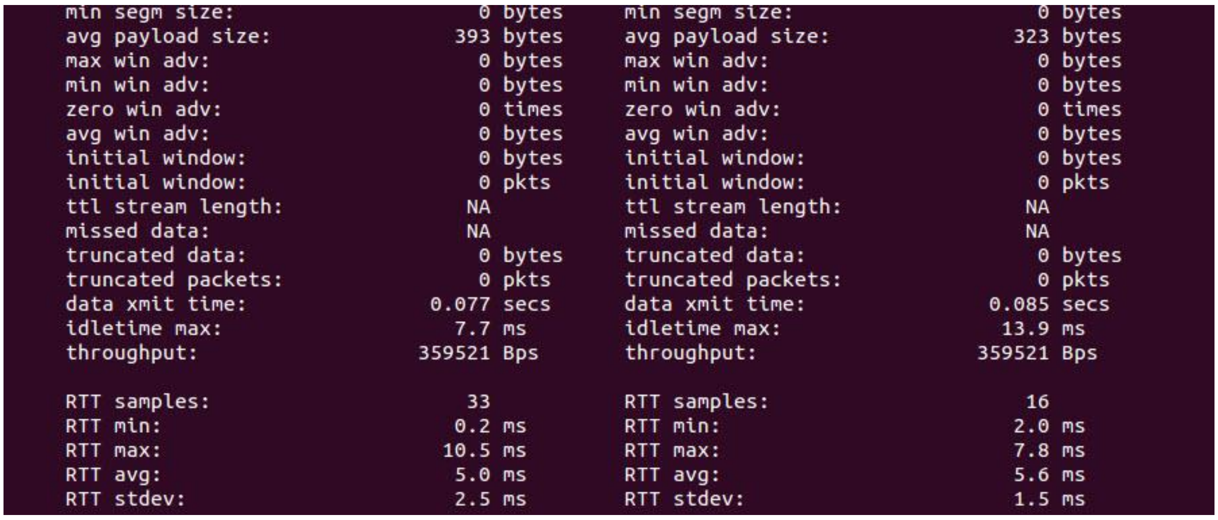
```
                         Path 1

     host c:        10.1.1.1:6666
     host d:        10.1.3.1:6666
     first packet:  Thu Jan  1 01:00:00.011061 1970
     last packet:   Thu Jan  1 01:00:55.968872 1970
     elapsed time:  0:00:55.957810
     total packets: 66686
     filename:      delay_50_default_v4_server.pcap

  c->d:                                       d->c:
    total packets:              33343           total packets:              33343
    total chunks:              105629           total chunks:               33343
    sack's sent:                    0           sack's sent:                33338
    init ack's sent:                0           init ack's sent:                1
    heartbeat ack's sent:           2           heartbeat ack's sent:           1
    heartbeat's sent:               1           heartbeat's sent:               2
    shutdown ack's sent:            0           shutdown ack's sent:            0
    cookie ack's sent:              0           cookie ack's sent:              1
    cookie echo's sent:             1           cookie echo's sent:             0
    abort's sent:                   0           abort's sent:                   0
    error's sent:                   0           error's sent:                   0
    ecne's sent:                    0           ecne's sent:                    0
    cwr's sent:                     0           cwr's sent:                     0
    shutdown comp's sent:           0           shutdown comp's sent:           0
    out of order's sent:           86           out of order's sent:            0
    other's sent:                   0           other's sent:                   0
    max sack blks/ack:              0           max sack blks/ack:              0
    unique bytes sent:       26406000           unique bytes sent:              0
    actual data chunks:        105624           actual data chunks:             0
    actual data bytes:       26406000           actual data bytes:              0
    rexmt data pkts:                0           rexmt data pkts:                0
    rexmt data bytes:               0           rexmt data bytes:               0
    zwnd probe pkts:                0           zwnd probe pkts:                0
    zwnd probe bytes:               0           zwnd probe bytes:               0
    outoforder pkts:                0           outoforder pkts:                0
```

Figure 5: Path information.

Figure 5, shows the information about a specific path. The command that has been used is:

./tcptrace -l -path < file.txt

where *file.txt* is the name of the dump file. By using the *-l* flag you get the extended view of the information, and by using the *-path* flag the collected information of the different paths for each association will be printed out.



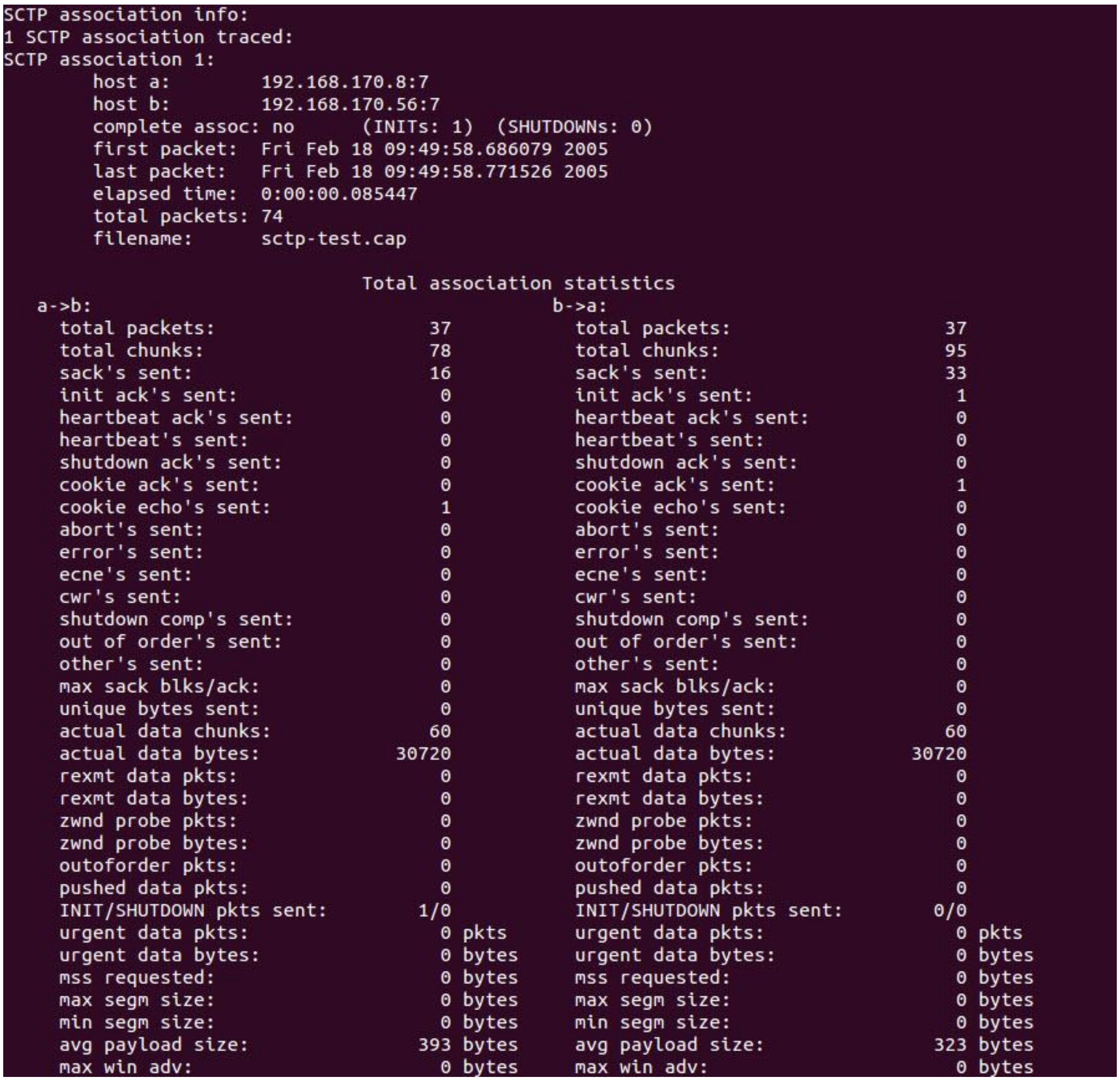
```
    min segm size:                   0 bytes     min segm size:                   0 bytes
    avg payload size:              393 bytes     avg payload size:              323 bytes
    max win adv:                     0 bytes     max win adv:                     0 bytes
    min win adv:                     0 bytes     min win adv:                     0 bytes
    zero win adv:                    0 times     zero win adv:                    0 times
    avg win adv:                     0 bytes     avg win adv:                     0 bytes
    initial window:                  0 bytes     initial window:                  0 bytes
    initial window:                  0 pkts      initial window:                  0 pkts
    ttl stream length:              NA           ttl stream length:              NA
    missed data:                    NA           missed data:                    NA
    truncated data:                  0 bytes     truncated data:                  0 bytes
    truncated packets:               0 pkts      truncated packets:               0 pkts
    data xmit time:              0.077 secs      data xmit time:              0.085 secs
    idletime max:                  7.7 ms        idletime max:                 13.9 ms
    throughput:                 359521 Bps       throughput:                 359521 Bps

    RTT samples:                    33           RTT samples:                    16
    RTT min:                       0.2 ms        RTT min:                       2.0 ms
    RTT max:                      10.5 ms        RTT max:                       7.8 ms
    RTT avg:                       5.0 ms        RTT avg:                       5.6 ms
    RTT stdev:                     2.5 ms        RTT stdev:                     1.5 ms
```

Figure 6: RTT information

Figure 6, shows the presented information when the flag *-r* is used. The new information presented with this flag includes the round trip times, which comes after the original information about the association. The command that has been used is:

./tcptrace -l -path < file.txt

where *file.txt* is the name of the dump file. By using the *-l* flag, you get the extended view of the information, and by using the *-r* flag the RTT calculations gets made during the trace.



```
SCTP association info:
1 SCTP association traced:
SCTP association 1:
        host a:        192.168.170.8:7
        host b:        192.168.170.56:7
        complete assoc: no      (INITs: 1)  (SHUTDOWNs: 0)
        first packet:  Fri Feb 18 09:49:58.686079 2005
        last packet:   Fri Feb 18 09:49:58.771526 2005
        elapsed time:  0:00:00.085447
        total packets: 74
        filename:      sctp-test.cap

                                 Total association statistics
   a->b:                                      b->a:
     total packets:                   37        total packets:                   37
     total chunks:                    78        total chunks:                    95
     sack's sent:                     16        sack's sent:                     33
     init ack's sent:                  0        init ack's sent:                  1
     heartbeat ack's sent:             0        heartbeat ack's sent:             0
     heartbeat's sent:                 0        heartbeat's sent:                 0
     shutdown ack's sent:              0        shutdown ack's sent:              0
     cookie ack's sent:                0        cookie ack's sent:                1
     cookie echo's sent:               1        cookie echo's sent:               0
     abort's sent:                     0        abort's sent:                     0
     error's sent:                     0        error's sent:                     0
     ecne's sent:                      0        ecne's sent:                      0
     cwr's sent:                       0        cwr's sent:                       0
     shutdown comp's sent:             0        shutdown comp's sent:             0
     out of order's sent:              0        out of order's sent:              0
     other's sent:                     0        other's sent:                     0
     max sack blks/ack:                0        max sack blks/ack:                0
     unique bytes sent:                0        unique bytes sent:                0
     actual data chunks:              60        actual data chunks:              60
     actual data bytes:            30720        actual data bytes:            30720
     rexmt data pkts:                  0        rexmt data pkts:                  0
     rexmt data bytes:                 0        rexmt data bytes:                 0
     zwnd probe pkts:                  0        zwnd probe pkts:                  0
     zwnd probe bytes:                 0        zwnd probe bytes:                 0
     outoforder pkts:                  0        outoforder pkts:                  0
     pushed data pkts:                 0        pushed data pkts:                 0
     INIT/SHUTDOWN pkts sent:        1/0        INIT/SHUTDOWN pkts sent:        0/0
     urgent data pkts:                 0 pkts   urgent data pkts:                 0 pkts
     urgent data bytes:                0 bytes  urgent data bytes:                0 bytes
     mss requested:                    0 bytes  mss requested:                    0 bytes
     max segm size:                    0 bytes  max segm size:                    0 bytes
     min segm size:                    0 bytes  min segm size:                    0 bytes
     avg payload size:               393 bytes  avg payload size:               323 bytes
     max win adv:                      0 bytes  max win adv:                      0 bytes
```

Figure 7: Information about an association.

Figure 7, shows the information when no extra flags are used, which gives us the information about each association that could occur in a trace file. The command that has been used is:

./tcptrace -l < file.txt

where *file.txt* is the name of the dump file. By using the *-l* flag, you get the extended view of the information.

Concluding Remarks

We have added SCTP functionalities to tcptrace. The program can show statistics based on associations and streams. We have added support for showing RTT information, both in text and in graph form. The flags that were added is the *-path* and *-stream* flag, and they work in combination as well. A limitation of the program is that its unable to handle trace-files that include both TCP and SCTP traffic, in that case only SCTP statistics will be shown.

The development of this extension has been more result-oriented, therefore the program has not been fully tested. A lot of the code has been repurposed from the original program code in order to fit in for the SCTP functionalities and therefore not written from scratch. This has made the overall understanding a little worse than normal and unnoticed errors may exist which are yet to be fixed properly. Further development of the program should start with separating the code that is used in the SCTP version of the program, thus making it a separate module that can more easily be changed, and making it easier to get a good overview of the code. Once this major refactoring has been done, the next step should be to implement and make the program work for IPv6, and also implement the remaining graphs such as the time sequence graph. Also, the presentation of the statistics may be organized in a better fashion in order to increase readability.

References

[1] RFC4960 Stream Control Transmission Protocol

[2] TCPtrace, www.tcptrace.org/, 14-12-2015

[3] AVL tree, en.wikipedia.org/wiki/AVL_tree/, 21-12-2015

[4] RFC 793 Transmission Control Protocol

[5] Chunk, en.wikipedia.org/wiki/SCTP_packet_structure#DATA_chunk/, 21-01-2016

[6] xplot, http://www.xplot.org/, 14-01-2016