

MEE 10:02

Real-Time Live RTT Analyzer

Venkata Santosh Pemmaraju

KARLSKRONA, JANUARY 2010
DEPARTMENT OF TELECOMMUNICATION SYSTEMS
SCHOOL OF ENGINEERING
BLEKINGE INSTITUTE OF TECHNOLOGY
371 79 KARLSKRONA, SWEDEN

Abstract

Due to rapid increasing in complexity of Internet, quantifying the performance of protocol helps in assessing the application behavior with respect to network performance. TCP is an important protocol that is used by some important applications on Internet such as HTTP, FTP and soon. To analyze TCP, Round trip times is one of the metric used. As it is a internal metric of TCP used to find the retransmission timeout of sent packet. Round trip times means measuring elapsed time between the sent packet and receiving its acknowledgment that covers the sequence number of the sent packet(i.e., from source to destination and vice-a-versa). Round trip times is a metric that is recognised by IETF as Quality of Service parameter. We design a tool here that will calculate the round trip times for each stream. This round trip times helps us to understand the protocol behavior. In this document, we discuss the design, implementation issues carried out while developing the tool. This tool is capable of reading offline as well as online streams and helps us to analyze the statistics obtained from collected round trip times of each stream.

Keywords

Computer Network Measurements, Distributed Passive Measurement Infrastructure, Transport Control Protocol, Round Trip Times, Real-time Measurements, Statistics.

Acknowledgements

I greatly thankful to God, , my Guru Sri Saibaba, my Parents and Dr. Patrik Arlos. If I reckon, I have not made any bad decision to study in Sweden. Sweden is a wonderful which I never forget, it helped to build my character and personality. I personally feel Dr. Patrik Arlos as a man of integrity and a visionary, he had inspired us with his lectures and innovative subjects. If I travel back to days of implementing the project, I find there are innumerable reminiscences with humourous and thoughtful experiences. He usually shares tit bits of information which makes a lot sense in my work. His endurance is unlimited to bear our ignorance throughout this work. I convey my gratitude to him, my supervisor, mentor and well wisher, for giving me this opportunity and encouragement. My special thanks to my dearest Mother P. Bharathi and Father P. Syamala Rao, who made me to reach this level without whom it is impossible to attain.

Pemmaraju Venkata Santosh
Karlskrona, December 2009

Contents

Abstract	iii
Acknowledgements	v
1 Introduction	1
1.1 Related Work	2
1.2 Background	2
2 Technical Background	5
2.1 Distributed Passive Measurement Infrastructure	5
2.2 Internet Protocol	6
2.3 Transmission Control Protocol	7
2.4 Round trip times	7
2.5 Statistics	9
2.6 Resource Investigation	9
3 Design	11
3.1 Design and Implementation	11
4 Verification And Validation	21
4.1 Verification	21
4.2 Validation	21
5 Conclusions and Future Work	25
5.1 Conclusions	25
5.2 Future work	25
A How To Use	27
B PacketProcessor Thread	29
C DataThread Thread	37
D ControlThread Thread	47
E Globals Of PacketProcessor Package	55
E.1 Globals	55
E.2 Global Variables	57
E.3 Global Functions	58

E.4 Global Structures	59
F ServerThreaded Thread	65
G DataConnection Thread	71
H ControlConnection Thread	75
I DataHandler Thread	81
J Calculator Thread	85
K StreamDistinguisher Class	95
L StreamDistinguisher Implemenatation	97
M RoundTripNode Class	103
N RoundTripNode Implementation	105
O Globals Of Dataevaluator	113
O.1 Globals	113
O.2 Global Variables	117
O.3 Global Functions	119
O.4 Global Structures	119
Bibliography	125

List of Figures

2.1	DPMI packet header.	5
2.2	IP Header	6
2.3	TCP Header	7
3.1	High level design diagram	11
3.2	Low level design diagram	12
3.3	Block diagram	13
3.4	DataThread Flowchart.	14
3.5	Calculator Flowchart	17
3.6	Cmp4same() Flowchart	18
3.7	Cmp4opp() Flowchart	19
3.8	CmpRTT() Flowchart.	19
3.9	ComputePDU() Flowchart.	20
3.10	deleteNode() Flowchart.	20
4.1	Graphical representation of Tcptrace.	22
4.2	Graphical representation of Real-time live RTT analyzer tool.	23
4.3	Comparison of Tcptrace and Real-time live RTT analyzer.	23

List of Tables

4.1	Output Demo of Real-Time Live RTT Analyzer Tool.	24
-----	--	----

Chapter 1

Introduction

Internet technology allows information sharing across the world, it is almost based upon TCP/IP architecture. The most prominent forms of information that are carried by the TCP/IP are mail, file transfer, entertainment and voice. TCP/IP transport layer is shared between connectionless and connection oriented service protocols. User Datagram Protocol (UDP) is a connectionless service protocol, examples for UDP are Voice Over Internet Protocol (VoIP) and online gaming. Transmission Control Protocol (TCP) comes under connection oriented service protocol, examples of TCP are Hyper-Text Transport Protocol (HTTP), File Transfer Protocol (FTP) [6]. TCP is used by trivial application on Internet. To analyze TCP, Round trip time is one of metric used. Round Trip Times (RTT) is defined as the elapsed time between the sent packet and receiving acknowledgement that cover the sequence number of sent packet. RTT is one of Quality of Service parameter accepted by IETF.

TCP utilizes RTT calculations for retransmitting a packet if a packet loss occur and avoids duplication in Internet by adjusting the retransmission timeout if delay varies[7]. RTT can be useful for identifying the reasons such as performance of unstable interactive real-time applications due to delays variations, minimum delays indicates that the propagation and transmission delays are the causes, minimum values of RTT indicates the bandwidth utilization is less and delays above minimum indicates congestions in the path. Unlike One Way Transit Time (OWTT) which is also a delay metric accepted by IETF, RTT is easy to interpret and easy to gather data[4]. RTT is a cumulative delay of events that occur while communicating for one interaction between the two nodes with respect to source [3].RTT delays also helps in understanding the interior sections of the network behavior. This motivates the measurement of the RTT.

My contribution in the thesis is to design and implement a tool that evaluates the RTT. The RTT output obtained is further sampled by the tool for evaluating mean, variance minimum and maximum of the RTT values obtained for a particular interval time.

This document details the design, implementation of the tool for analyzing RTT of TCP streams. Statistics such as mean, minimum, maximum and variance are evaluated for analysis. The tool uses DPMI network to obtain data, as

such, it can work on the online and offline data. This chapter further explains contribution, related work and background. Next chapter details the technical background needed to understand the protocols and the intended tool.

1.1 Related Work

There are many Internetworking measurement groups which are interested in Internet measurements. Lot of work is being carried on in this field of research to further enhance the performance of the network by analyzing the application and protocol behavior. Some of the related work is mentioned here, as follows. Tstat is one of the related works, which is implemented with Round Robin Database (RRD) tool. It can work on offline as well as online data. In order to work with online data, configuration file has to be updated to understand the relevant network. It works depending up on the configuration file and allows only those specified data streams or you can specify streams which you are interested in. It can track maximum of 3500 TCP streams with standard deviation, max and min observed during connection lifetime. Tstat represent standard deviation, minimum and maximum in milliseconds which is a disadvantage to it. And also, it can only represent standard deviation, mean, maximum and minimum in xml format using rrdump function and it does not include any further data [1]. Tcptrace is another related work, which generates graphs using xplot or jPlot applications. It can work only on offline files having pcap as extension. It generate RTT values dump files with respect to streams. Its statistics includes number of rtt samples, minimum, maximum and average standard deviation values of the whole stream rather than for sampling intervals. The rtt values that are dumped in to the respective files are represented in milliseconds and does not include the time of acknowledgement received, which are short comings to the tool [5].

1.2 Background

Internet network measurements are generally based upon either passive or active measurement techniques. Passive measurements are based on already collected data where as active measurements are based on collecting and measuring live data instantly. There are number of tools that can be used to perform passive measurement technique. Active measurement technique is little expensive as it needs external hardware. TCP is an essential protocol and its analysis is being done frequently by many. Internet is expanding rapidly in all spheres. Initially HTTP traffic which has become basic form of using Internet, was major bandwidth utilizing protocol. Then Peer-to-Peer Protocol (P2PP) came into existance with growing presence. Although shift is apparent, still the HTTP traffic is constantly growing [9]. HTTP is not the only protocol that uses TCP depending upon application usage requirements many application use TCP as its transport layer protocol.

Now-a-days Internet connection is not just restricted to broadband modem connection rather GSM broadband and GPS Internet connections services are also available. New avenues are opened to analyze the protocols in different sce-

1.2. Background

narios. TCP is a conventional protocol which is a reliable and connection oriented service [9].

TCP adapts to the network it is connected and reduce overloading in the network. TCP network adaptation is done by slow start method, in which it sends only one packet initially and waits for the acknowledgement, depending upon acknowledgement received, it can change its packet size and congestion window size. If the congestion window size is increased beyond the network capacity, packet loss take place. Hence, the RTTs are used for calculating retransmission timeout. It reduces the data loss in the network by calculating RTT with help of its internal timer. In order to analyze TCP, RTT is used to understand the delay variations that helps to comprehend the application behavior. RTT comes under critical performance metrics of Quality of Service (QoS) under connection oriented services that will enable us to identify different characteristics of the networks, such as route stability, delays on shared segment and the factors that influence the RTT[4].

Chapter 2

Technical Background

In this chapter, we discuss the details regarding the information that is used in the implementing the tool. The information is initially in raw data, which is processed to evaluate the round trip times. Firstly the information is divided into raw packets which are captured through the dag interfaces from Distributed Passive Measurement Infrastructure (DPMI). This captured data is read by an application interface to further processing to evaluate RTT. The captured data can be of any type, for example it can be of IPv4 or IPv6. Based upon the IP it can be TCP or UDP, here, we are dealing with only TCP, as TCP is a connection oriented service, RTT is the time difference between the sent packet and its acknowledgement. This chapter will discuss about DPMI, IP, TCP format structures and the information that is been in use for the tool to evaluate RTT.

2.1 Distributed Passive Measurement Infrastructure

Distributed Passive Measurement Infrastructure (DPMI) is used by academicians and researchers in order to analyze the behavior of networks. This Infrastructure provides ease of use, modularity, security and cost effectiveness. It consists of Measuring Points (MP) and Measurement Area Controller (MArC), each MP located in the network can be controlled by MArC. The data capturing is done through the dag interface cards, which are wiretaps, connected to one MP. Each captured packet is encapsulated with DPMI packet format, which consists of Measurement Point ID (MPID), Capture Interface (CI), Frame Length (FL), Capture Length (CL), Time Stamp, actual packet. The timestamp that is used

Measuring Point ID (MPID)	Capture Interface (CI)	Frame length (FL)	Capture Length (CL)	Time Stamp	Actual Packet
---------------------------	------------------------	-------------------	---------------------	------------	---------------

Figure 2.1: DPMI packet header.

in DPMI is an absolute precision timestamp, which gives the actual time at which the packet has been captured. The above information is used to differentiate the packets based upon their MPs and CIs. De-capsulation of the frame will result

in actual packet, this packet is a raw data which is in big-endian or little-endian format. The following figure depicts the structure of DPMI packet[10].

2.2 Internet Protocol

In this section we will describe the structure of IP header structure. In order to evaluate round trip times of TCP streams, stream identification is necessary. In this context, it is assumed that a stream is defined as the data flow between one source and one destination for a particular application i.e., the Connection established between two end points for a particular application. A combination of source address and source port is called socket vice-versa. A combination of socket that gives the information about the connection that is established is called socket pair. In order to derive the socket pair information IP header is necessary in which we find IP source address and IP destination address. In IP header we find two important data, to which protocol does this Packet Data Unit (PDU) belongs and to between which source and destination the data is transit taking place. The following is figure and description of IP header with interest in field within the context. Figure: 2.2 shows the IP header, the fields in interest

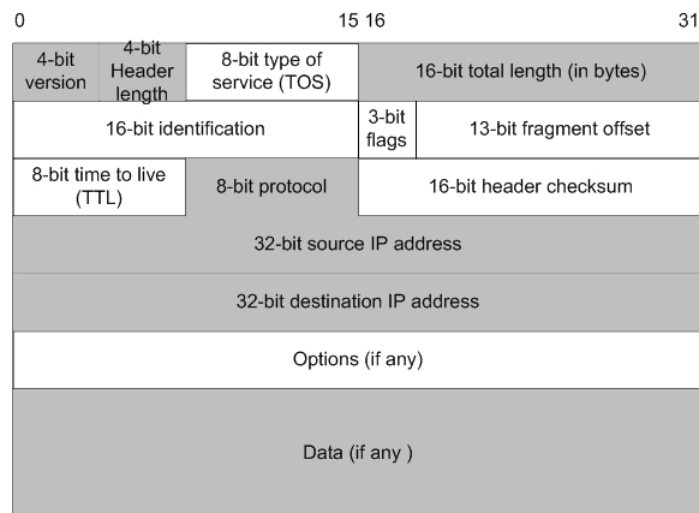


Figure 2.2: IP Header

are highlighted are explained below: The protocol version in interest is version number 4, this version of IP is called as IPv4. The header length is the number of 32 bit words in the header, including any options. Since this is a 4-bit field, it limits the header to 60 bytes. The total length field is the total length of the IP datagrams in bytes. Using this field and the header length field, we know where the data portion of the IP datagram starts and its length. Since this is a 16 bit field, the maximum size of an IP datagram is 65535 bytes. The protocol field is used to identify to which higher layer protocol the IP packet belongs to. Each IP datagram contains the source IP address and the destination IP address. These are the 32 bit values. Data field is the final field [11].

2.3 Transmission Control Protocol

TCP is a reliable and stream oriented protocol which compensates data loss, delay and duplication. In order to collect the data and evaluate the round trip times for each streams in the network, we need to know about the TCP, it is accountable to both its upper layer and the lower layer, it takes the data from the upper layer while sending the data to a destination, attaches header then sends it to lower layer and while receiving the data from a destination the data is checked then removing the header sends it to the upper layers. The following figure depicts the TCP header and the highlighted fields represent interest in data with respect to the context. Figure:2.3 shows the TCP header, the fields

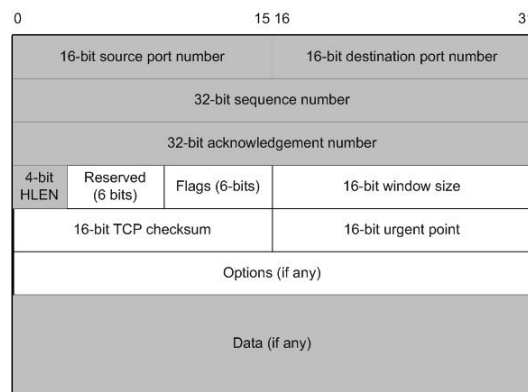


Figure 2.3: TCP Header

in interest are highlighted are explained below: TCP connection contains source and destination port numbers to identify the sending and receiving application. These two values along with source and destination IP addresses in IP header uniquely identify each connection. A source IP address and source port number are when combined called as socket. Source and destination sockets, when combined define a particular stream in our context. Sequence number is 4 byte/32 bit number represents ordering the sequence in which the data sent in order to ensure reliable data transfer. The sequence number is a 32 bit unsigned number which wraps around 0 after it reach to $2^{32}-1$. When a new connection is being established, the SYN flag is turned on. The sequence number field contains the initial sequence number (ISN) chosen by this host for this connection. The sequence number of the first byte of data sent by this host will be the ISN plus one. The acknowledgement number contains the next sequence number that the receiver expects to receive from sender. Once the connection establishes the ack flag is kept always on. Header length describes the length of the header in 32 bit words. It is important because the option field is variable The tail ending portion of the TCP segment is the data portion with padding [7].

2.4 Round trip times

Round trip times is defined as the elapsed time between the sent packet with a particular sequence number and receiving acknowledgement that covers the se-

quence number of that sent packet. TCP uses RTT internally to evaluate the Retransmission Time Out (*RTO*) to decide whether to resend the sent packet or not, if there is delay in receiving acknowledgement for that packet.

When a connection is established between two terminal nodes, the nodes exchange their respective Initial Sequence Number (*ISN*). The sequence number is the combined data first byte address and *ISN*, the number is termed as absolute sequence number. At any instance, amount of data that is been transferred between the nodes can be calculated by subtracting the *ISN*, this number is termed as relative sequence number. Sequence number represents one more than number of bytes that it previously sent i.e., it represent the first byte in the present payload. TCP acknowledgment field number tells us that, it is the response from destination representing the number of bytes that it received and request for the next byte for which it is expecting. In order to match the sent packet and its acknowledgement, the sent packet sequence number, length of sent packet payload and received packet acknowledgement number is needed. Following equation 2.1 shows the relation between *sSEQ*, *sPLEN* and *dACK*, which are used to find the matching packets.

$$sSEQ + sPLEN == dACK \quad (2.1)$$

where *sSEQ* is sent sequence number, *sPLEN* is sent packet payload length and *dACK* acknowledgement number in the packet received from destination. The receiving packet acknowledgement (*dACK*) number prediction can also be done through, the addition of *sSEQ* and *sPLEN*.

$$RTT = dPkt_Time - sPkt_Time \quad (2.2)$$

Equation 2.2 is the relation that shows the RTT equation, where *sPkt_Time* is the time when sent packet is sent and *dPkt_Time* is the time when destination packet is received. If a match is found, sent packet time is subtracted from received packet time, which is desired RTT (i.e., shown in equation 2.2). The sequence and acknowledgement match is done on unique socket pair information. In a high speed network, when a connection is established between two nodes window size is exchanged between them. The window size is the buffer space which is used to queue the incoming packet before processing them. This window size will also be useful for bulk transmissions of data and avoid unnecessary acknowledgements improving network utilization, as this window size avoid congestion in the network, it is called as congestion window (cnwd). If receivers cnwd is big enough to accommodate more packets, then the source transmits according to the cnwd of receiver that is being advertised. The window at the sender side will enable the sender to keep track of the data that reached the destination. TCP uses RTT to find the suitable time for Retransmission Timeout (*RTO*). *RTO* is the timing trigger that occurs when the minimum expectation time of receiving acknowledgement is elapsed. *RTO* is used to avoid data loss and RTT helps in overcome from the unnecessary overloading of the network. To evaluate *RTO*, Smoothed Round Trip Time (*SRTT*) is evaluated first as follows:

$$SRTT = (\alpha * SRTT) + ((1 - \alpha) * RTT) \quad (2.3)$$

then *RTO* is computed as:

$$RTO = \min[UBOUND, \max[LBOUND, (\beta * SRTT)]] \quad (2.4)$$

Equations 2.3 and 2.4 are relations used by TCP for its internal use. Here *UBOUND* in equation 2.4 is an upper timeout threshold value and *LBOUND* is a lower timeout threshold value. α is a smoothing factor (for eg: .8 to .9) and β is a delay variance factor (eg: 1.2 to 2.0). It exponentially increases the time between the retransmission after elapsing the expected acknowledgement reception time. This RTT analysis to a certain extent will enable to view network behavior on whole.[7] Selective acknowledgements (SACK) are introduced in TCP to avoid packet losses due to cumulative acknowledgements scheme. SACKs are seen in TCP header options as discussed earlier. Rather than sending acknowledgements to each every packet the SACKs send acknowledgement to the most recently received packet [12].

2.5 Statistics

RTT statistics in this tool is considered to be evaluated through Time-based method. In this tool we are interested to find mean, variance, maximum and minimum of RTT values obtained for given time interval for each stream. Mean (μ_{RTT}) is obtained by the product of sum of RTT values ($\sum RTT$) obtained for a given time interval and number of RTT values (N) obtained for that particular interval. Sum of RTT values for a given interval is shown in the equation 2.5.

$$\sum_{k=1}^N RTT = RTT_1 + RTT_2 + \dots + RTT_N \quad (2.5)$$

RTT mean is shown in the following equation 2.6.

$$\mu_{RTT} = \frac{\sum_{k=1}^N RTT}{N} \quad (2.6)$$

Variance of RTT ($Var(RTT)$) is shown in the following equation 2.7.

$$Var(RTT) = \frac{(\sum_{k=1}^N RTT)^2}{N - (\mu_{RTT})^2} \quad (2.7)$$

2.6 Resource Investigation

The task to design and implement a tool that evaluates RTT, is more component based with available resources rather than building it from scratch. The resources we obtained are a C++ component built upon GCC called packetprocessor, using this a new component is integrated to evaluate the RTT for each stream. In order to build this component, the packetprocessor design has to be investigated initially. Packetprocessor is a TCP client tool that is responsible for capturing packets and sending to a TCP server tool that accepts these packets. It is responsible for capturing packets from offline files as well as live data from ethernet interface that are sent by DPML. Packet processor analysis follows

1. It has 1 process thread and 2 child threads. The datathread and controlthread are the two child threads and packetprocessorthreaded is the process thread.

2. Datathread class is responsible for listening the interface, analyzing which type of packet it is, reading the packet information and sending the info gathered in the packet to the TCP server tool which accepts these packets.
3. Controlthread class is responsible for exchanging control information between server tool and packetprocessor client tool.
4. Packetprocessorthreaded class is a process thread which is responsible for establishing connection between server tool and packetprocessor and controlling the child threads i.e., controlthread and datathread.
5. The information that is exchanged in between server tool and packetprocessor is as follows
 - MPID measuring point information data.
 - NIC network interface card.
 - CAPLEN length of the captured packet.
 - IPSRC source ip address.
 - IPDST destination ip address.
 - PROTO type of protocol in decimal representation.
 - PORTSRC source port number.
 - PORTDST destination port number.
 - TIME timestamp at which the packet is captured.
 - IDENTIFICATION hash code of captured packet.

Dataevaluator (TCP server tool) analysis follows

1. It includes a process thread called serverthreaded and child threads are dataConnection, dataHandler, controlConnection and calculator.
2. Serverthreaded process thread is responsible to handle and manage the child threads.
3. DataConnection thread is responsible for listening connection requests from clients.
4. DataHandler is responsible for accepting data from the client and storing them.
5. ControlConnection thread is responsible for exchanging control information between the server and clients tools.
6. Calculator is responsible for reading the packets from dataHandler storage, creating data structures and deleting the datastructures.

Chapter 3

Design

3.1 Design and Implementation

A design for a tool that evaluates RTT is done in this section with retrospect to the information gathered in the previous chapters and investigating the resources available. RTT can be evaluated with help of sequence number, acknowledge number and the packet length information, that is held in TCP packet. Implementation of the above design is done using C++ programming using GCC compiler in UNIX environment. Design of the tool is done by using the equations 2.1 and 2.2 as follows. Figure 3.1 shows an initial high level design diagram. Based upon

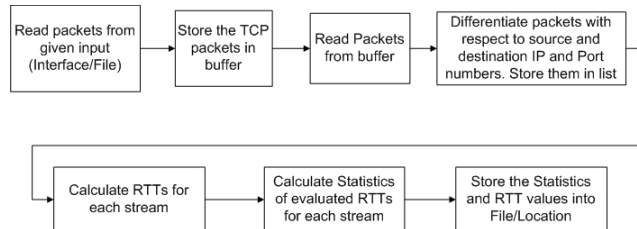


Figure 3.1: High level design diagram

the job responsibility blocks or modules are created. This follows a figure 3.2 a medium level design diagram which explains the data structures and association with each other. From both figures 3.1 and 3.2 we can understand as follows. Data is read from an interface or file. This packet is read and sent to the buffer, if the packet belongs TCP. Buffer is a FIFO (First In First Out) buffer and this buffered data is then again read from the buffer to sort them according to their respective streams. As shown figure 3.2 stream list pointer is pointed to the end of the packet list. With this we can understand that each acknowledgement received will give the acknowledgement to received number of bytes (i.e., acknowledgement to most recently sent packet). So, it is unnecessary to start checking from the first packet. Figure 3.2 two data structures are implemented one is the main list in which a new node is created to each new stream that is not in the list and other the other is the child list. Each node in the main list has pointer that points to its child list. Child list is used to store the packet information where as the main list stores only stream information. Here the stream from source is

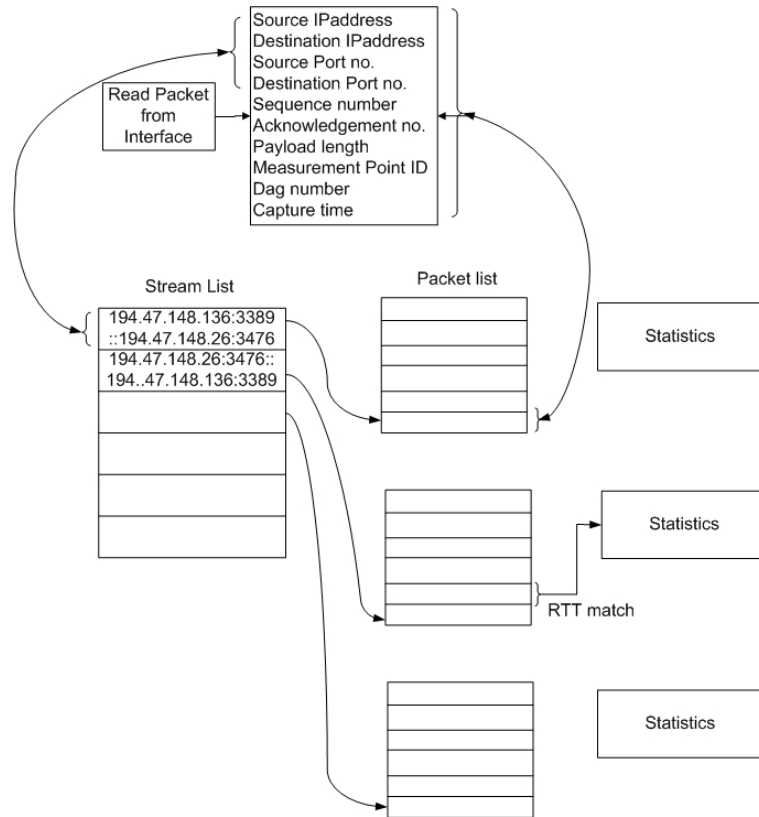


Figure 3.2: Low level design diagram

different from the destination even though they belong to same connection, as the RTT for particular stream avoids obfuscation and sorting at the end. Source acknowledgements are identified from the packets of destination and vice versa. In the next stage, RTT is calculated by comparing the each packet of the corresponding opposite stream, if there is any match between the acknowledgement and sequence as described earlier. Then RTT is calculated, this data is fed to next stage to calculate the statistics of that respective stream. These statistics are then can be used to analyze the behavior of the network. Designing with respect to the assessment of data requirement constraints.

Based upon the figure 3.3 each block is explained and significant blocks are explained with flowcharts. For more information about each thread see respective appendix. Figure 3.3 shows that the rectangle shape blocks represent threads and oval shaped blocks represents objects. This design is considered while implementation to obtain the desired result. Except the late arrived packets, the most recent packet sequence number of the corresponding stream is one more than length of the bytes that is transferred in the previous packet. Each packet is acknowledged from the opposite stream as described above. As shown in the figure PacketProcessorThreaded, dataThread and controlThread threads belongs to Packetprocessor package. ServerThreaded, dataConnection,

3.1. Design and Implementation

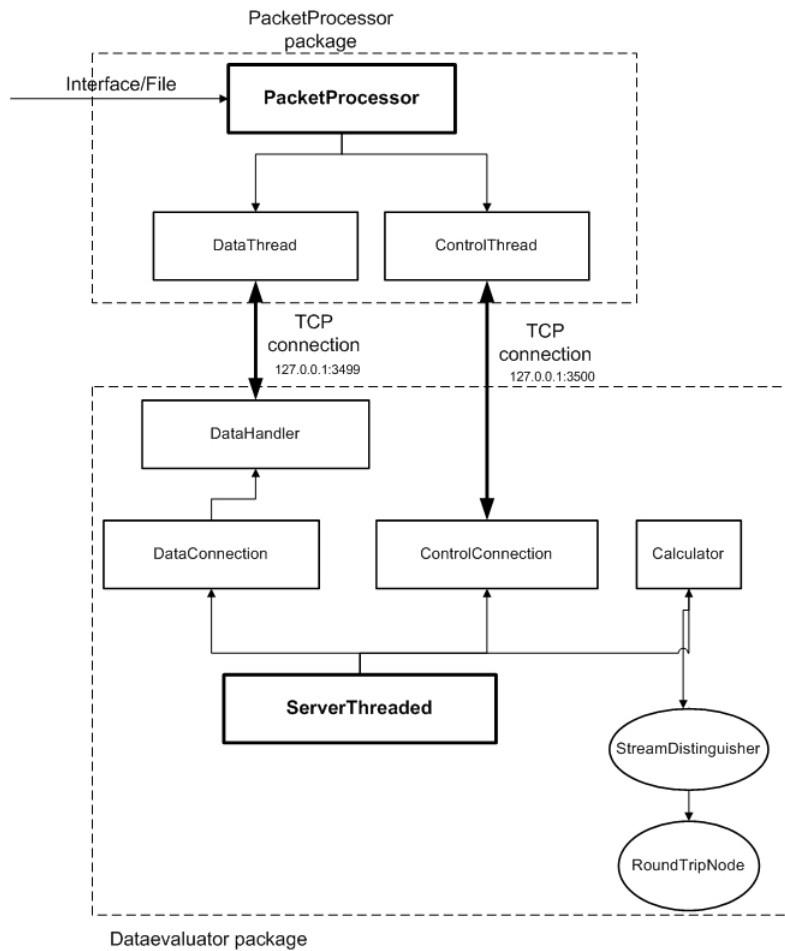


Figure 3.3: Block diagram

dataHandler, controlConnection, calculator, streamDistinguisher and RoundTripNode belongs to Dataevaluator package. Each block in the figure is responsible for a specific job. As shown in the figure 3.3 two TCP connection are made between the packetprocessor and Dataevaluator. One connection is in between the dataThread of packetprocessor and dataHandler of Dataevaluator. Second connection is in between the controlThread of packetprocessor and controlConnection of Dataevaluator. Each thread is described briefly, flowcharts for the threads are shown based upon their significance and responsibility as follows.

PacketProcessorThreaded is a parent thread which is responsible for handling the controlThread and dataThread, it responsible for connection between Dataevaluator and itself, and it is also responsible for taking input file/interface details from command line for reading data. It acts as a client to Dataevaluator.

ControlThread is a thread which is used to send and receive control information for the TCP connection between packetprocessor and Dataevaluator.

DataThread is a thread which is responsible to read data from interface or given input file, which is given while executing the packetprocessor application.

The data that is read from interface or file location is sent to the Dataevaluator only if, the packets belongs to TCP protocol. It extracts the IP and TCP header details to copy into a new structure and this data structure is sent as a packet via TCP connection to Dataevaluator. The flowchart of dataThread is shown in figure 3.4.

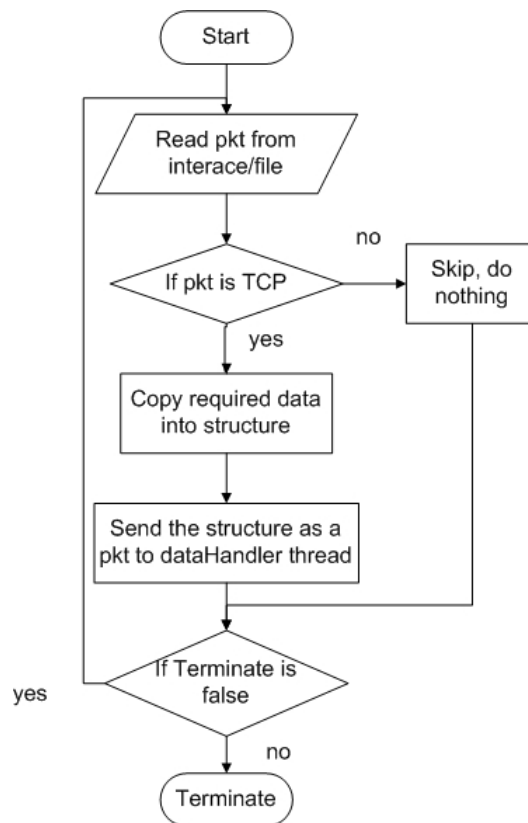


Figure 3.4: DataThread Flowchart.

Dataevaluator is a package which consists of serverThreaded, dataConnection, dataHandler, controlConnection and calculator threads and streamDisitnguisher, RoundTripNode as objects.

ServerThreaded is a parent thread which invokes and handles the dataCon-
nection, controlConnection and calculator threads.

DataConnection thread is again parent thread to dataHandler thread. Dat-
aConnection thread is responsible for accepting the connection from the clients.
It creates a structure which contains the client's connection information and this
sturcture is sent to dataHandler thread. ControlConnection is responsible for
sending the control information to the packetprocessor. It monitors the connec-

tion status and updates this information to packetprocessor.

DataHandler is a child thread of dataConnection, it is initiated by dataConnection thread when the packetprocessor thread requests for a connection. It is responsible to receive the data from the packetprocessor and store it in the buffer space. It is responsible for tracking the buffer locations and updating the counters.

Calculator is a thread, responsible for reading the packets from buffer which are stored by the dataHandler thread. To read packet from buffer calculator thread consists of findOldest(), checkCleanPList(), getOldest() and cleanPList() functions, that are used for monitoring and manipulating the buffer. Function checkCleanPList() is used to increment a counter if there are unread newly arrived packets in the buffer. Function findOldest() is used to know which packet is arrived first, and that packet is read using the function getOldest(), the buffer is used in FIFO order. Function cleanPList() is used to free the memory of the packets that are read from the buffer. The remaining responsibility is to create, invoke and destroy the streamDistinguisher. Figure 3.5 explains the overall functionality of the thread.

StreamDistinguisher is a linked list object which differentiates the streams with respect to the socket pair information. First streamDistinguisher object is created by calculator when first packet is read. StreamDistinguisher object when created, it creates a file using socket pair information and RoundTripNode object. When next packet is read it is compared with the first object using function cmp4same() by the calculator. Figure 3.6 shows the cmp4same() functionality in a flowchart. If a match is found then this packet is sent to the RoundTripNode to create a node at the end of the list. If no match is found a new streamDistinguisher node is created with its file and RoundTripNode object. As we know from earlier chapter streams are different with respect to source and destination. Source socket pair is different from destination socket pair. After identifying the stream, function cmp4opp() is used to find the destination socket by conversing its socket pair information.

Figure 3.7 shows the cmp4opp() function functionality in a flowchart. If a match is found then RoundTripNode function cmpRTT() is invoked to find a match between this packet acknowledgement number and the sequence number in the list of packets.

RoundTripNode is the last stage where the sequence and acknowledgement numbers are compared with stored packets and newly arrived packet. RoundTripNode is a stream specific linked list, that means only one stream data are found. Each streamDistinguisher node has one RoundTripNode object associated. Comparison takes place between the MPID, CI, source sequence number, source packet payload length and destination packet acknowledgement number. This comparison is done only if the packet is not gone through RTT evaluation, it helps in reducing the redundancy. Comparison of CI is done by using only first 4 characters of sent and receive packets. This forms a configuration constraint that is to be obeyed while configuring the network. If a match is found between the above mentioned then RTT evaluation is done using the function computePDU(). Each streamDistinguisher node has a pointer to last node of RoundTripNode to reduce the computation of comparison and iterations.

Function computePDU() is RoundTripNode function, its functionality is shown

in figure 3.9. By using function `computePDU()`, RTT is evaluated by subtracting sent packet time from receive acknowledgement time. These RTT are then given to find mean, variance, maximum and minimum for given sampling time. Mean, variance of RTT are evaluated using the equation 2.5, 2.6 and 2.7. Timestamp that is received, is divided into two parts seconds and fractional seconds. Fractional seconds is represented with precision of pico seconds. As general operating systems are constrained to 10^{-6} – 10^{-7} fractional precision, for consistent and accurate results a micro seconds i.e., 10^{-6} seconds precision is opted. To evaluate RTT values and further calculations, a third party library used, as there is no built in data types that support to store a double value of 10^{-12} precision into it. A library called `quad_real` is used for storing seconds and fractional pico seconds into one variable. After evaluating the RTT and its statistics, recycling of the nodes are done using function `deleteNode()` of `RoundTripNode` object. `DeleteNode()` flowchart as shown in figure 3.10 will remove the nodes one less than the present node.

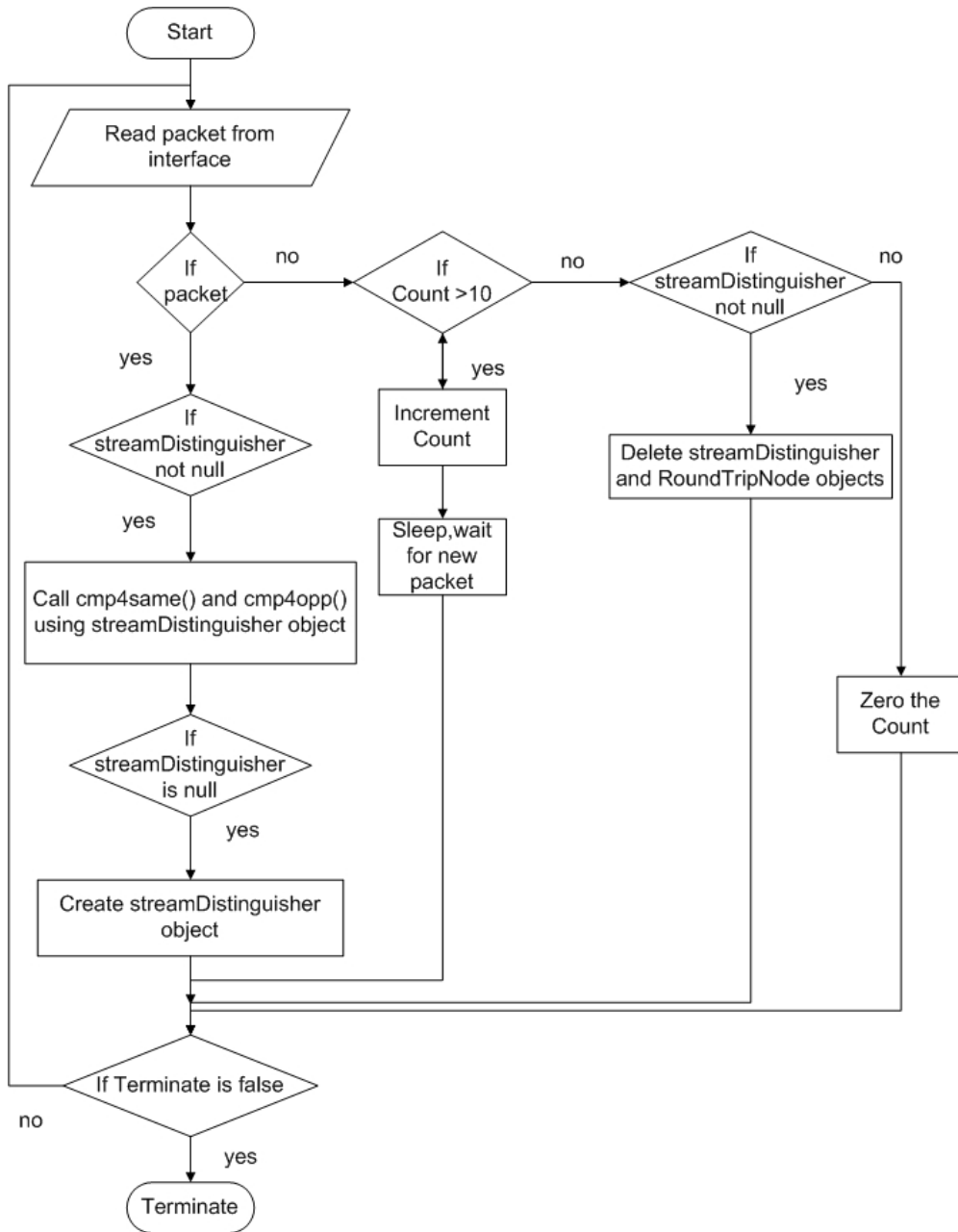


Figure 3.5: Calculator Flowchart

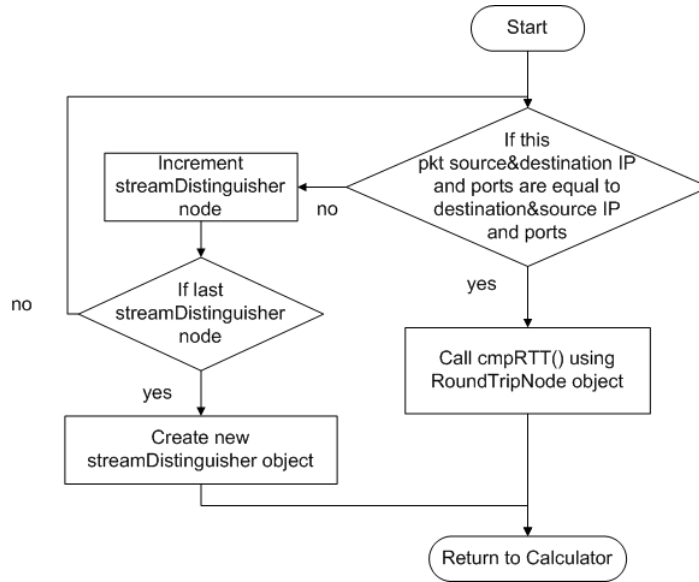


Figure 3.7: Cmp4opp() Flowchart

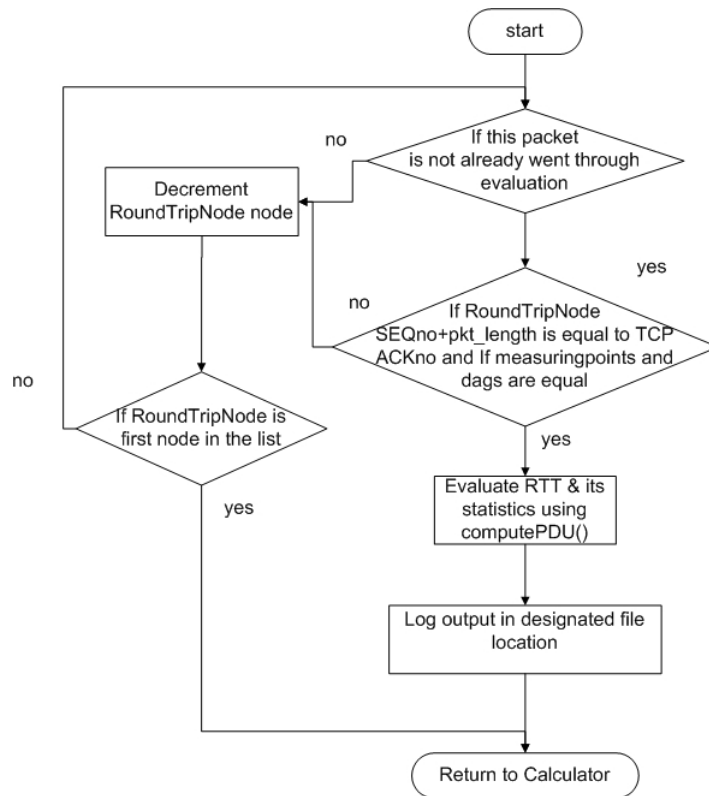


Figure 3.8: CmpRTT() Flowchart.

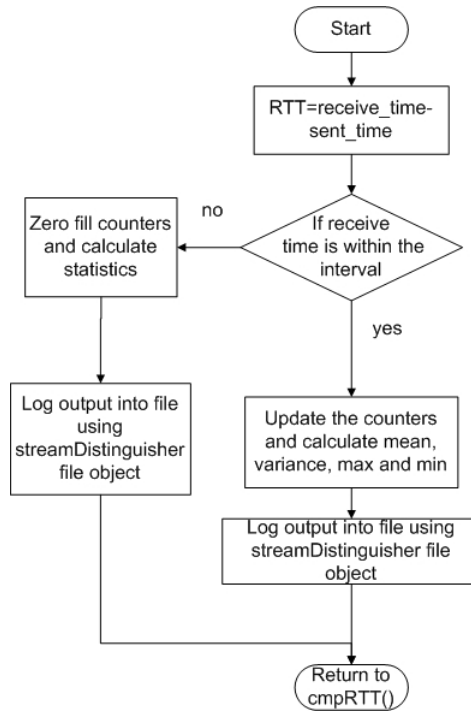


Figure 3.9: ComputePDU() Flowchart.

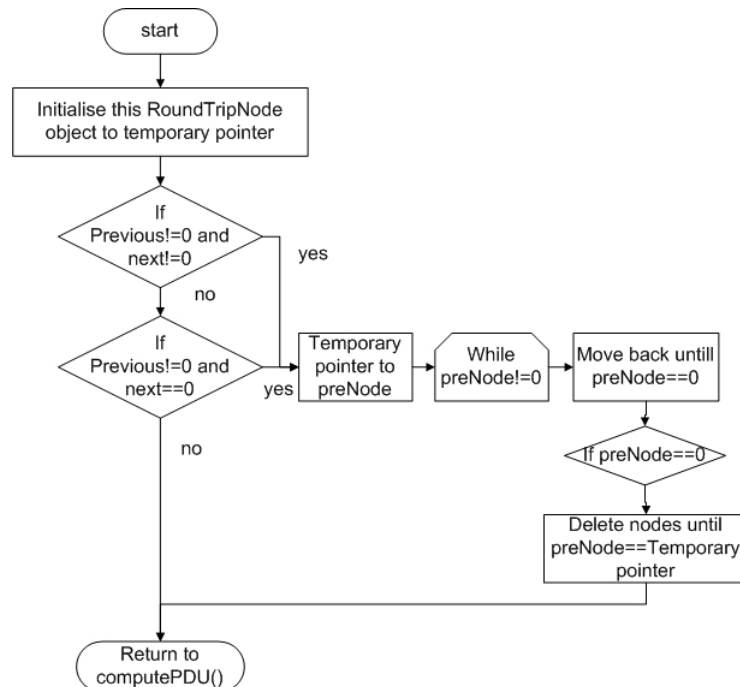


Figure 3.10: deleteNode() Flowchart.

Chapter 4

Verification And Validation

In this chapter the process that carried out while verification and validation of the tool is discussed here with detail. Firstly verification where the data accuracy and calculations are verified with expected results, then validation where the tool is compared with previously tested validated tool that is available in open source.

4.1 Verification

Verification is done while implementing the tool and after implementing it. The tool is tested with predefined streams to know the behaviour of linked lists. Nodes creation in the streamDistinguisher class are accordingly done with respect to the number of streams. Each Node of the streamDistinguisher is assigned a RoundTripNode class list, which is satisfied. Comparison between the nodes and incoming packets are checked using the logs generated from each object. While configuring the the internal network a predefined intervals are used between the streams, to replicate the actual network using DPMM infrastructure. It is known that a constant delay of 10 ms is applied between ht105–ht103 and a inconsistent delay of 10 μ s–100 μ s is introduced between ht104–ht103. Observation of results given by the tool is satisfactory. After calculating mean, variance, maximum and minimum for the RTT's, they are sent to another application for graphical representation.

4.2 Validation

Tool validation is done here by comparing the output of our tool and an established tool which has good reputation. The tool that is being used for validating the present tool is **tcptrace**. It is developed by Dr. Shawn Ostermann at Ohio University. Tcptrace works on offline data and takes pcap extension files as its input. Command line options used for obtaining the desired results are

```
#tcptrace -Znrl <inputfile.pcap>
```

This command includes -Z option for dumping rtt samples in respective files, -n option for not converting the IPaddress, -l option for creating long output and -r option for including number of rtt samples from each stream in long output. A file with good number of samples are taken into consideration from both the tools and are compared using graphs. A stream 194.47.148.136:3389 to

194.47.148.26:3476 (source and destination IP:port) is taken for comparing the RTT values in graphical notation is shown in the following figures 4.1, 4.2 and 4.3. As we can see the figure 4.3 a comparison is made between the figures

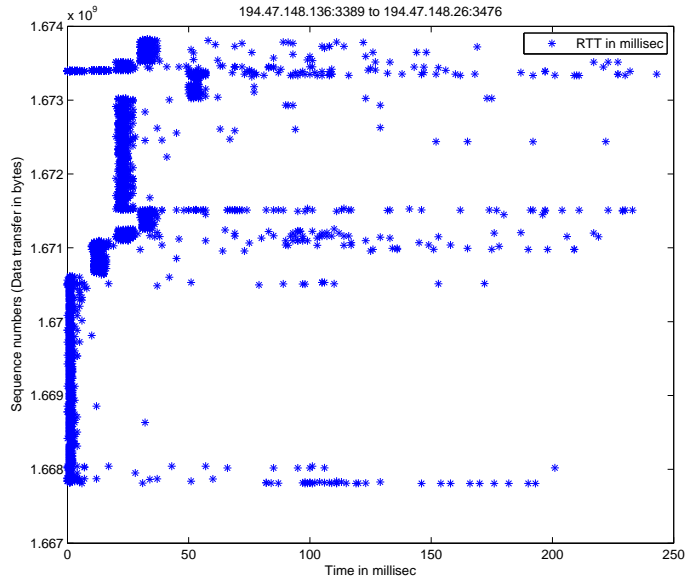


Figure 4.1: Graphical representation of Tcptrace.

4.1 and 4.2. We can say from figure 4.3 that no data of Tcptrace is missing in Real-time live RTT analyzer, instead some excess data is seen. This excess data after assessing we found that they are post-loss acknowledgments. The reasons for acknowledgement packets arriving late can be many, and these post-loss acknowledgements are not identified by the Real-time live RTT analyzer. Other than these no other differences are found between the Tcptrace and Real-time live RTT analyzer data.

Demo of Real-time live RTT analyzer tool output is shown here.

4.2. Validation

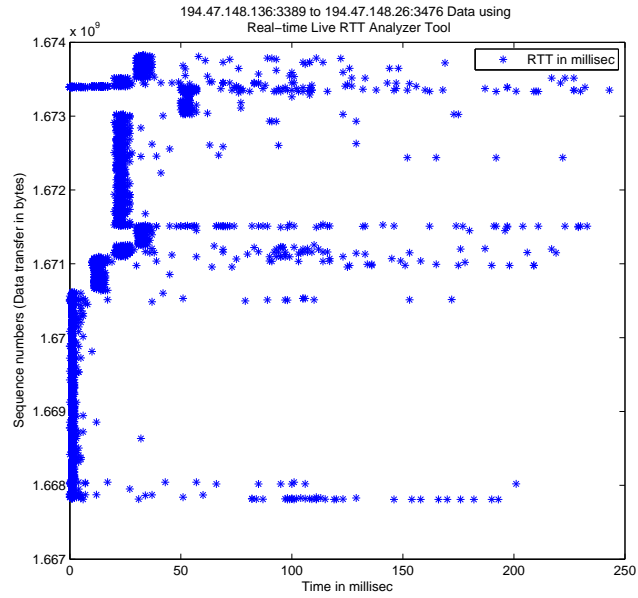


Figure 4.2: Graphical representation of Real-time live RTT analyzer tool.

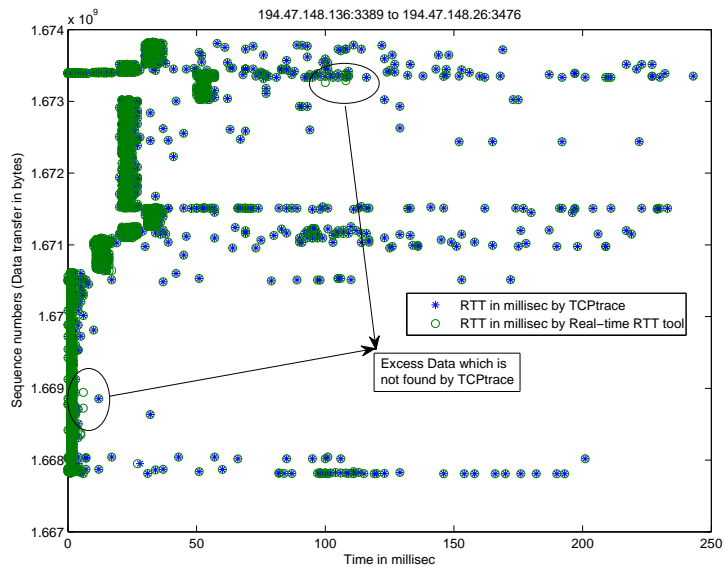


Figure 4.3: Comparison of Tcptrace and Real-time live RTT analyzer.

Timestamp	Source	Destination	Src.dag	dst.dag	RTT[in 10 ⁷ sec]	Seq. No.	RTT[in 10 ³ sec]
1251888452.742561	130.239.17.4:80	194.47.148.136:2706	dag11	dag10	0.0000905	790791318	0
1251888452.825293	130.239.17.4:80	194.47.148.136:2706	dag11	dag10	0.0010078	790792779	1
1251888452.899394	130.239.17.4:80	194.47.148.136:2706	dag11	dag10	0.0002311	790795699	0
1251888452.975303	130.239.17.4:80	194.47.148.136:2706	dag11	dag10	0.0002192	790798619	0
1251888452.976850	130.239.17.4:80	194.47.148.136:2706	dag11	dag10	0.0002272	790801539	0
1251888453.051414	130.239.17.4:80	194.47.148.136:2706	dag11	dag10	0.0001902	790804459	0
1251888453.051666	130.239.17.4:80	194.47.148.136:2706	dag11	dag10	0.0001660	790807379	0
1251888453.051912	130.239.17.4:80	194.47.148.136:2706	dag11	dag10	0.0002260	790810299	0
1251888453.127438	130.239.17.4:80	194.47.148.136:2706	dag11	dag10	0.0001855	790813219	0
1251888453.127684	130.239.17.4:80	194.47.148.136:2706	dag11	dag10	0.0002290	790816139	0
1251888453.127931	130.239.17.4:80	194.47.148.136:2706	dag11	dag10	0.0001700	790819059	0
1251888453.128177	130.239.17.4:80	194.47.148.136:2706	dag11	dag10	0.0001608	790821979	0
1251888453.203313	130.239.17.4:80	194.47.148.136:2706	dag11	dag10	0.0002220	790824899	0
1251888453.203585	130.239.17.4:80	194.47.148.136:2706	dag11	dag10	0.0001603	790827819	0
1251888453.203831	130.239.17.4:80	194.47.148.136:2706	dag11	dag10	0.0001625	790830739	0

Table 4.1: Output Demo of Real-Time Live RTT Analyzer Tool.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This thesis has discussed the *implementation details of tool Real-time Live RTT Analyzer*. During the testing phase our concentration of the tool was to check accuracy of data obtained. Accuracy of the data obtained by Real-time Live RTT Analyzer tool is verified and validated. At the beginning of the testing we have encountered duplicate acknowledgements, to remove them a constraint is introduced in function `cmp4same()` (see in figure 3.6) of `streamDistinguisher`, that checks if there are packets with same sequence already exists or not, before creating an object. This removes all the duplicate acknowledgements. But still post-loss acknowledgements are not being removed. So, a while loop is created which will duplicate the packets with newly arrived packet by using a constraint. This constraint is that if same sequence number already exists in previous packets which are not already evaluated RTT then duplicate the packets with newly arrived packet. This loop is unnecessary as it is nothing but to remove only some of the post-loss acknowledgements and also regrades the performance of the tool. The tool presently works good for offline as well as online, with an exception a few streams (i.e., <20) of online data will give a good performance than many streams (i.e., >20).

5.2 Future work

In future work, if while loop in function `cmp4same()` of `streamDistinguisher` object is removed then the performance of the tool is improved to a great extent. We can use more streams of online data evaluation for RTT. Moreover, if some scrutiny and trails are done on the function `deleteNode()` of `RoundTripNode` then the performance can be improved to a much extent. Even the post-lost acknowledgements can be avoided, by applying trail and error method on function `deleteNode()` (see figure 3.10). As security issues are not addressed, they can be applied. It will be easy and efficient to use this already implemented data structures for through-put and delay analysis also.

Appendix A

How To Use

A brief description for how to use the tool is given here. The tool system contains three packages dataevaluator and packetprocessor.

Dataevaluator acts as server tool which accepts connections from packetprocessor as well as matstuff. It is responsible for evaluating RTTs and their mean, variance, maximum and minimum. In order to start dataevaluator server tool

```
$cd dataevaluator
$make clean
$make
$./server
now the server tool is started.
```

Packetprocessor acts as client tool which reads the packets from DPPI framework and sends it to the dataevaluator. Packetprocessor client tool has number of options. Those options can be known by giving -h as first argument, used generally for help. Packetprocessor takes -a or -h for first argument, -a option specifies client tool the ipaddress of the server tool. Its second argument will either to read from already captured data file or from live data i.e., ethernet port address. The two combinations are show below

```
#cd packetprocessor
#make clean
#make
#./packetprocessor-1.0 -a <ipaddress> <already captured data file>
While specifying data file use full path, if it is not in current working directory.
#./packetprocessor-1.0 -a <ipaddress> -i <etherent port id> <ethernet port address>
```

Ethernet port id is either eth0,eth1 or eth2 depending upon requirement.

Output can be seen in the folder named output in the Dataevaluator package. Each file name is given by the each stream information. We can see as many files as many streams. In each file data is logged into as columns which are shown in the table 4.2. We can add more columns according to needs in the function computePDU() of RoundTripNode object. Now you can see the results of specified stream.

Appendix B

PacketProcessor Thread

```
#ifndef __cplusplus
extern "C" {
#endif
#include <cap_utils.h>
// get the interface
#ifdef __cplusplus
}
#endif
#include <pthread.h>
#include <unistd.h>
#include <getopt.h>
#include <time.h>
#include <iostream>
#include <iomanip>
#include <string.h>
#include <math.h>
#include <signal.h>
#include <netdb.h>
#include <string>
#include <stdio.h>
#include <errno.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <openssl/sha.h>
#include <stdio.h>
#include <sys/types.h>
#define APP_VERSION_MAJOR 0
#define APP_VERSION_MINOR 1
#define ETHERTYPE_VLAN 0x8100
#define SERVER_PORT 3499
#include "globals.functions.h"
#include "../packetprocessor/globals_structures.h"
```

```
    /* timeout counter */
int timeout;// Counter for the number of timeouts that has occurred.
int timeout_value; // Timeout time in seconds.
pthread_t dataThread, controlThread;
int terminate; /* Set to 1 to terminate all threads !*/
int SendSyncMsg; /* Send synch pulse on data */
int threadCount;
double pkts;
struct pduinfo myinfo;
struct pduinfo myinfoSync;
int srvsocket, srvctrl, ioctl_return;
struct hostent *he;
struct sockaddr_in server_addr,server_addrCtrl,myDataAddr, myDataAddrctrl;
char servermessage[1500];
int SOCKID,i, buffercontent, readbytes, sumofbytes, Serverport, verbose;
char* servername;
struct ctrlInitMsg CIM;
int dataPort, ctrlPort, sentbytes, sendCount;
struct pduinfo sendPDUS[512];
int sendSize;
struct timeval DESENDtime;

    /* Libcap 0.5 variables.. */
char *filename;
struct filter* myfilter;
struct stream inStream; // Stream to read from
int streamType; // Stream type
int l; // Temporary variable
char *nic; // Holder of NIC string
int portnumber; // Port to listen to
int rpStatus; // read post status
char d; // For dealing with packets
char* data;
char** dataPtr;
double pktCount;

    /* END of Libcap 0.5 variables ..*/
struct timeval lastDPMIread;
int DPMIactive;
int DPMItimeoutCount;
int sample;
int sampletype;
int identificationMethod;
int noPdus;
int batchSize;
int currPduCount;
int currBatchCount;
uint32_t starttime;
struct PPfilter myFilters[10];
int inputPDUs;
```

```
int filteredPDUs;
int sentPDUs;
int maxInputRate, maxFilterRate,maxSentRate;
int CinputPDUs;
int CfilteredPDUs;
int CsentPDUs;
int avgInput;
int avgFiltered;
int avgSent;
int silenceTime;
int historyInput[10];
int historyFilter[10];
int historySent[10];
int SAMPLETIME;
void statisticsSampler(int signo);
```

```
int main (int argc, char **argv){
signal(SIGINT,cleanup_main);
extern int optind;//, opterr, optopt;
register int op;
int this_option_optind;
int option_index;
avgInput=avgFiltered=avgSent=0;
SAMPLETIME=10;
nic=0;
portnumber=0x0810;
pktCount=0;
data=&d;
dataPtr=&data;
streamType=0;
sentbytes=0;
sendCount=0
; servername=
0; Serverport=SERVER_PORT;
verbose=0;
silenceTime=0;
lastDPMIread.tv_sec=0;
DPMItimeoutCount=0;
DPMIinactive=0;
sample=1;
sampletype=0;
identificationMethod=0;
noPdus=-1;
batchSize=-1;
currPduCount=0;
currBatchCount=0;
starttime=0;
SendSyncMsg=0;
int i;
```

```
    for(i=0;i<10;i++){
myFilters[i].index=0;
historyInput[i]=-1;
historyFilter[i]=-1;
historySent[i]=-1;
}

    inputPDUs=filteredPDUs=sentPDUs=0;
maxInputRate= maxFilterRate=maxSentRate=0;
static struct option long_options[] = {
{"pkts", 0, 0, 'p'},
{"evaluator",0,0,'a'},
{"eport",0,0,'b'},
{"help", 0, 0, 'h'},
{"if", 1,0,'i'},
{"tcp", 1,0,'t'},
{"udp", 1,0,'u'},
{"port", 1,0, 'v'},
{0, 0, 0, 0}
};
SHA_CTX ctx;
myfilter=createfilter(argc,argv);
SHA1_Init(&ctx);
pkts=-1;

    if(argc<2){
printf("use %s -h or -help for help",argv[0]);
exit(0);
}

    while (1) {
this_option_optind = optind ? optind : 1;
option_index = 0;
op = getopt_long (argc, argv, "a:b:hi:t:u:v:p:",long_options, &option_index);
if (op == -1)
break;

    switch (op) {
/* Database */
case 'a':
l=strlen(optarg);
servername=(char*)malloc(l+1);
bzero(servername, l+1);
strncpy(servername, optarg, l);
he=gethostbyname(servername);
if(he==NULL) {
printf("%s: unknown host '%s'",argv[0],servername);
exit(1);
}
break;
```

```

case 'b':
Serverport=atoi(optarg);
break;
case 'p':
pkts=atoi(optarg);
break;
case 'i':
l=strlen(optarg);
nic=(char*)malloc(l+1);
strncpy(nic,optarg,l);
streamType=1;
break;
case 'u':
streamType=2;
break;
case 't':
streamType=3;
break;
case 'v':
portnumber=atoi(optarg);
break;
case 'h':
printf("_____");
printf("Application Version %d.%d",APP_VERSION_MAJOR,APP_VERSION_MINOR);
printf("Application Options");
printf("-p or -pkts Number of pkts to show [default all]");
printf("-a or -evaluator IP address of evaluator.");
printf("-b or -eport Port number of evaluator, default 1500.");
printf("-h or -help this text");
printf("-i or -if <NIC> Listen to NIC for Ethernet multicast address,");
printf(" identified by <INPUT> (01:00:00:00:00:01).");
printf("-t or -tcp Listen to a TCP stream.");
printf(" identified by <INPUT> (192.168.0.10). ");
printf("-u or -udp Listen to a UDP multicast address.");
printf(" identified by <INPUT> (225.10.11.10).");
printf("-v or -port TCP/UDP port to listen to. Default 0x0810.");
printf("<INPUT> If n,t or u hasn't been declared, this ");
printf(" is interpreted as a filename.");
printf("Usage:");
printf("%s [filter options] [application options] <INPUT>", argv[0]);
exit(0);
break;
}
}
l=strlen(argv[argc-1]);
filename=(char*)malloc(l+1);
bzero(filename, l+1);
strncpy(filename, argv[argc-1], l);

```

```

/* Connecting to Evaluator */

```

```

if ( (he=gethostbyname(servername)) == NULL ) {
perror("Gethostby name");
exit(1);
}

    if( (srvsocket=socket(PF_INET, SOCK_STREAM,0)) == -1 ) {
perror("Socket.");
exit(1);
}

    if( (srvctrl=socket(PF_INET, SOCK_STREAM,0)) == -1 ) {
perror("Socket.");
exit(1);
}

    server_addr.sin_family= AF_INET;
server_addr.sin_port = htons(Serverport);
server_addr.sin_addr = *((struct in_addr *)he->h_addr);
memset(server_addr.sin_zero,'\0',sizeof(server_addr.sin_zero));
server_addrCtrl.sin_family= AF_INET;
server_addrCtrl.sin_port = htons(Serverport+1);
server_addrCtrl.sin_addr = *((struct in_addr *)he->h_addr);
memset(server_addrCtrl.sin_zero,'\0',sizeof(server_addrCtrl.sin_zero));

    /* Alarm test. */
timeout=0; // Make sure that the timeout is cleared.
timeout_value=2;

    if(signal(SIGALRM,alarmHandler)==SIG_ERR) {
perror("Problems setting up alarm handler.");
exit(0);
}

    /*Set alarm to 2 seconds */
if(alarm(timeout_value)!=0) {
perror("Cannot set alarm time.");
exit(0);
}

    if(connect(srvsocket,(struct sockaddr *)&server_addr,sizeof(server_addr)) ==
-1 ) {
perror("Cannot connect.");
exit(1);
}

    int leng=sizeof(myDataAddr);
getsockname(srvsocket, (struct sockaddr*)&myDataAddr, (socklen_t*)&leng);
dataPort=ntohs(myDataAddr.sin_port);
printf("MAIN:Starting dataHandlerThread ->");
if(pthread_create(&dataThread, NULL, dataHandlerThread, NULL)) {

```

```

perror("Cannot create data connection thread.");
abort();
}

    if(connect(srvctrl,(struct sockaddr *)&server_addrCtrl,sizeof(server_addrCtrl))
== -1 ) {
perror("Cannot connect.");
exit(1);
}

    getsockname(srvctrl, (struct sockaddr*)&myDataAddrctrl, (socklen_t*)&leng);
ctrlPort=ntohs(myDataAddrctrl.sin_port);
printf("MAIN:Starting controlConnection thread ->");
if(pthread_create(&controlThread, NULL, controlConnection, NULL)) {
perror("Cannot create control connection thread.");
abort();
}

    printf("Connections:");
printf("DATA %s:%d - %s:%d", inet_ntoa(myDataAddr.sin_addr), ntohs(myDataAddr.sin_port),
servername, Serverport);
printf("CTRL %s:%d - %s:%d", inet_ntoa(myDataAddrctrl.sin_addr), ntohs(myDataAddrctrl.sin_port),
servername, Serverport+1);

    printf("MAIN: WAITING FOR %d THREADS.", threadCount);
pthread_join(controlThread,NULL);
pthread_join(dataThread,NULL);
close(srvsocket);
close(srvctrl);
printf("Leaving.");
return 0;
}

void alarmHandler(int signo){
timeout++;
alarm(timeout_value);
return;
}

void cleanup_main(int signo){
printf("Cleanup_main, called %d times. There are %d threads.", terminate+1,
threadCount);
terminate++;
return;
}

```


Appendix C

DataThread Thread

```
#ifndef __cplusplus
extern "C" {
#endif
#include <cap_utils.h>
// get the interface
#ifdef __cplusplus
}
#endif

    #include <pthread.h>
#include <unistd.h>
#include <getopt.h>
#include <time.h>
#include <iostream>
#include <iomanip>
#include <string.h>
#include <math.h>
#include <signal.h>
#include <string>
#include <stdio.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <openssl/sha.h>
#include <stdio.h>
#include <sys/types.h>
#define APP_VERSION_MAJOR 0
#define APP_VERSION_MINOR 1
#define ETHERTYPE_VLAN 0x8100
#define SERVER_PORT 3490
#include "global_variables.h"
#include "globals_functions.h"
// THIS IS SHARED WITH DATAEVALUATOR!!! Possible a split needed, as
PP does not need all the internal
// Structures of the DE.
```

```
#include "../packetprocessor/globals_structures.h"
void* dataHandlerThread(void* ptr){
cap_head *caphead;
struct ethhdr *ether=0;
struct ether_vlan_header *vlan=0;
struct ip *ip_hdr=0;
struct tcphdr *tcp=0;
struct udphdr *udp=0;
struct icmphdr *icmp=0;
u_char* payload=0;
sendSize=1;
struct timeval tv1,tv2,tv3;
double dif;
int VLANPRESENT=0;
int vlan_id,netProto,srcport, dstport, seqno, ackno, window;
FILE* sentdata;
sentdata=fopen("sentDE.txt","w");
printf("->dataThread()");
threadCount++;

SOCKID=-1;
alarm(2);
buffercontent=readbytes=sumofbytes=0;
bzero(servermessage,1500);
int state=0;
while(srvsocket && state == 0){
ioctl_return=ioctl(srvsocket, FIONREAD, &buffercontent);
if((unsigned)buffercontent>=(unsigned)sizeof(int)) {
readbytes=recv(srvsocket, &SOCKID, sizeof(int),0);
if(readbytes == -1 ) {
if(errno==EINTR) {
printf("EINTR.");
} else {
printf("Some other error.");
}
} else if (readbytes == 0) {
printf("Other side has closed the connection.");
close(srvsocket);
} else {
sumofbytes+=readbytes;
SOCKID=ntohl(SOCKID);
printf("The identifier is %d.", SOCKID);
bzero(servermessage,1500);
timeout=0;
state++;
}
} else {
alarm(0);
printf("Waiting for server (sleeping 1 s) to provide the identifier.");
sleep(1);
```

```

alarm(2);
}
}
alarm(0);

    /* Opening Stream */
if(signal(SIGALRM,alarmHandler)==SIG_ERR){
perror("Problems setting alarm.");
close(srvctrl);
close(srvsocket);
exit(0);
}
timeout=0;
int streamstatus=0;
timeout_value=5;
alarm(timeout_value);
while(streamstatus==0 && timeout<10) {
streamstatus=openstream(&inStream,filename, streamType,nic,portnumber);
if(streamstatus==0 && timeout==0){
/* Problems opening stream, */
close(srvctrl);
close(srvsocket);
exit(0);
}
}
if(timeout==10 && streamstatus==0) {
/* Ten timeouts, lets get out of here.. */
close(srvctrl);
close(srvsocket);
perror("Cannot open stream.");
exit(0);
}
/* Clear alarm */
alarm(0);
printf("Successfully open the DPMI Stream.");
if(verbose) {
printf("Comment size: %d, ver: %d.%d id: %s comments: %s",inStream.FH.comment_size,
inStream.FH.version.major,
inStream.FH.version.minor, inStream.FH.mpid, inStream.comment);
printf("Capture version: %d.%d.", inStream.FH.version.major, inStream.FH.version.minor);
printf("Measurement Point: %s.", inStream.FH.mpid);
printf("Comment:%s", inStream.comment);
printf("—————");
}
int tSentbytes=0;
if((int)pkts== -1) {
printf("Reading infinite number of packets from the DPMI.");
} else {
printf("Reading %d packets, then terminating", (int)pkts);
}

```

```

/* Link capacity. */
double readPkts=1;
inputPDUs=1;
filteredPDUs=0;
sentPDUs=0;
rpStatus=1;
*dataPtr=(inStream.buffer+inStream.readPos);
int sampleThisPDU=0;
int myCounter;
int filterMatch=0;
int firstTime=0;
int addTime=0;
int Loops=0;
char *strptr, s1[17],s2[17];
int plsize;
printf("Waiting 1 seconds before starting the main data stream.");
printf("Were about enter..");
gettimeofday(&tv1,NULL);
while(rpStatus!=0 && terminate==0){
if(SendSyncMsg>0){
fprintf(sentdata,"here we are about to sync mesg data to dataevaluator %d",SendSyncMsg);
strncpy(myinfoSync.mpid,"1111111",8);
strncpy(myinfoSync.nic,"1111111",8);
myinfoSync.caplen=0;
myinfoSync.len=0;
myinfoSync.window=0;
myinfoSync.proto=0;
myinfoSync.portsrc=0;
myinfoSync.portdst=0;
myinfoSync.ts.tv_sec=0;
myinfoSync.ts.tv_nsec=0;

/* etc... */
gettimeofday(&tv3,NULL);
myinfoSync.ipsrc=htonl(tv3.tv_sec);
myinfoSync.ipdst=htonl(tv3.tv_nsec);
myinfoSync.tcpseq=htonl(DESENDtime.tv_sec);
myinfoSync.tcpack=htonl(DESENDtime.tv_nsec);
sentbytes=send(srvsocket, &myinfoSync, 1*sizeof(struct pduinfo),0);
tSentbytes+=sentbytes;
sentPDUs+=sendCount;
SendSyncMsg--;
fprintf(sentdata,"*****mpid,nic %s:%s DEtime %d:
%d",myinfoSync.mpid,myinfoSync.nic,myinfoSync.ipsrc,myinfoSync.ipdst);
fprintf(sentdata,"sentbytes %d",sentbytes);
}

data=*dataPtr;
caphead=(cap_head*)data;
/*TMP!!! */

```

```

if(readPkts==1) {
firstTime=caphead->ts.tv_sec;
Loops=0;
addTime=0;
}
printf("Parsing a PDU pdu at %p.", dataPtr);
caphead->ts.tv_sec=caphead->ts.tv_sec+Loops*addTime;
/* END TMP!!! */
ether=(struct ethhdr*)(data+sizeof(cap_head));
if(ntohs(ether->h_proto)==ETHERTYPE_VLAN){
printf("vlan!");
vlan=(struct ether_vlan_header*)(data+sizeof(cap_head));
vlan_id=0x0FFF&ntohs(vlan->vlan_tci);
VLANPRESENT=4;
netProto=ntohs(vlan->h_proto);
} else {printf("novlan ");
VLANPRESENT=0;
netProto=ntohs(ether->h_proto);
}
srcport=0;
dstport=0;
seqno=0;
ackno=0;
window=0
; tcp=0;
udp=0;
icmp=0;
payload=0;
printf("Packet contains a %d ", netProto);
switch(netProto) {
case ETHERTYPE_IP: /* Packet contains an IP, PASS TWO! */
ip_hdr=(struct ip*)(data+sizeof(cap_head)+sizeof(struct ethhdr)+VLANPRESENT);

    switch(ip_hdr->ip_p){
case IPPROTO_TCP:
printf("TCP packets here ");
tcp=(struct tcphdr*)(data+sizeof(cap_head)+sizeof(struct ethhdr)+VLANPRESENT+4*(ip_hdr->ip_hl));
srcport=ntohs(tcp->source);
dstport=ntohs(tcp->dest);
seqno=ntohl(tcp->seq);
ackno=ntohl(tcp->ack_seq);
window=ntohs(tcp->window);
if(tcp->syn){myinfo.flags.SYN=1;}else{myinfo.flags.SYN=0;}
if(tcp->fin){myinfo.flags.FIN=1;}else{myinfo.flags.FIN=0;}
if(tcp->ack){myinfo.flags.ACK=1;}else{myinfo.flags.ACK=0;}
if(tcp->rst){myinfo.flags.RST=1;}else{myinfo.flags.RST=0;}
if(tcp->psh){myinfo.flags.PUSH=1;}else{myinfo.flags.PUSH=0;}
if(tcp->urg){myinfo.flags.URG=1;}else{myinfo.flags.URG=0;}
payload=(u_char*)tcp+sizeof(struct tcphdr);

```

```

plsize=caphead->len-sizeof(struct ethhdr)-4*ip_hdr->ip_hl-4*(tcp->doff)-VLANPRESENT;
printf("srcport= %d",srcport);
printf("destport=%d",dstport);
printf("seqno=%d",seqno);
printf("ackno=%d",ackno);
break;
case IPPROTO_UDP:
udp=(struct udphdr*)(data+sizeof(cap_head)+sizeof(struct ethhdr)+VLANPRESENT+4*(ip_hdr->ip_hl));
srcport=ntohs(udp->source);
dstport=ntohs(udp->dest);
payload=(u_char*)tcp+sizeof(struct udphdr);
plsize=caphead->len-sizeof(struct ethhdr)-4*ip_hdr->ip_hl-sizeof(struct udphdr)-VLANPRESENT;
break;
case IPPROTO_ICMP:
icmp=(struct icmphdr*)(data+sizeof(cap_head)+sizeof(struct ethhdr)+VLANPRESENT+4*(ip_hdr->ip_hl));
srcport=(icmp->type);
dstport=(icmp->code);
payload=(u_char*)tcp+sizeof(struct icmphdr);
plsize=caphead->len-sizeof(struct ethhdr)-4*ip_hdr->ip_hl-sizeof(struct icmphdr)-VLANPRESENT;
break;
default:
break;
}
break;
default: /* Unknown link protocol */
break;
}
if(netProto!=ETHERTYPE_IP){
printf("Not a IP.");
} else {
printf("checking filters ");
/* Check filters */
filterMatch=0;
for(myCounter=0;myCounter<10&&filterMatch==-1;myCounter++){
if(myFilters[myCounter].index!=0) {
if(myFilters[myCounter].index&1) { // IP src.
if((((uint32_t)ip_hdr->ip_src.s_addr)&(myFilters[myCounter].IPsrcMask))!=(myFilters[myCounter].IPsrcMask))
continue;
}
}
if(myFilters[myCounter].index&2) { // IP dst.
if((((uint32_t)ip_hdr->ip_dst.s_addr)&(myFilters[myCounter].IPdstMask))!=(myFilters[myCounter].IPdstMask))
continue;
}
}
if(myFilters[myCounter].index&4) { // IP protocol

```

```

if(ip_hdr->ip_p!=htons(myFilters[myCounter].IP_PROTO)){
continue;
}
}
if(myFilters[myCounter].index&8) { // Transport Source
if((srcport&myFilters[myCounter].SrcPortMask)!=(myFilters[myCounter].SrcPort)){
continue;
}
}
if((myFilters[myCounter].index)&(16)) { // Transport Destination
if((dstport&myFilters[myCounter].DstPortMask)!=(myFilters[myCounter].DstPort)){
continue;
}
}
filterMatch=myCounter;
} //if(myFilters[myCounter].index>-1)
}
if(filterMatch==0 && myCounter==10) {
/* Checked all filters, no active once.
i.e. let everything pass!
If filter 0 would match then filterMatch=0 and myCounter=0
which is acceptable */
filterMatch=11;
}
if(ip_hdr->ip_p != IPPROTO_TCP){
filterMatch=-1;
printf("Not a TCP packet, should drop it.");
}
printf("Filter done? %d ",filterMatch);

/* Check do we sample? */
printf("checking whether we do sample");
sampleThisPDU=0;
if (sample==1) {
if (samplotype==0) {
sampleThisPDU=1;
}
if (samplotype==1){
if(currPduCount==noPdus){
sampleThisPDU=1;
currPduCount=0;
} else {
currPduCount++;
}
}
if (samplotype==2) {
if((uint32_t)caphead->ts.tv_sec >= starttime) {
if(currPduCount==noPdus){
sampleThisPDU=1;
currPduCount=0;
}
}
}
}

```

```
} else {
currPduCount++;
}
}
}
if (sampletype==3) {
if(currBatchCount<batchSize){
sampleThisPDU=1;
currBatchCount++;
}
}
if (sampletype==4) {
if((uint32_t)caphead->ts.tv_sec >= starttime) {
if(currBatchCount<batchSize){
sampleThisPDU=1;
currBatchCount++;
}
}
}
if (sampletype==5) {
if (currPduCount==noPdus) {
if(currBatchCount<batchSize){
sampleThisPDU=1;
currBatchCount++;
} else {
currBatchCount=0;
currPduCount=0;
}
}
}
if (sampletype==6) {
if((uint32_t)caphead->ts.tv_sec >= starttime) {
if (currPduCount==noPdus) {
if(currBatchCount<batchSize){
sampleThisPDU=1;
currBatchCount++;
} else {
currBatchCount=0;
currPduCount=0;
}
}
}
}
}
if( (sample == 1) && (sampleThisPDU==1) && (filterMatch>-1) ) {
filteredPDUs++;
printf(“filterMatch = %d , iphdr = %p , ipproto = %d ”,filterMatch, ip_hdr,ip_hdr-
>ip-p);
//bzero(theString,2000);
printf(“we are sampling ”);
```

```

/*modified-struct packinfo*/
printf(“%p ”, caphead);
memcpy(myinfo.mpid,caphead->mampid,8);
memcpy(myinfo.nic,caphead->nic,8);
printf(“we are sampling2 ”);

    myinfo.caplen=caphead->caplen;
myinfo.len=(uint16_t)plsize;
printf(“we are sampling3 IP header %p ”, ip_hdr);
myinfo.ipsrc=(uint32_t)(ip_hdr->ip_src.s_addr);
myinfo.ipdst=(uint32_t)(ip_hdr->ip_dst.s_addr);
printf(“we are sampling4 ”);

    myinfo.proto=(uint16_t)(ip_hdr->ip_p);
myinfo.portsrc=(uint16_t)(srcport);
myinfo.portdst=(uint16_t)(dstport);
printf(“we are sampling5 ”);

    myinfo.ts.tv_sec= caphead->ts.tv_sec;
myinfo.ts.tv_nsec=caphead->ts.tv_nsec;
myinfo.tcpseq=(seqno);
myinfo.tcpack=(ackno);
/*end pack info*/
printf(“till here there is no problem ”);
printf(“****myinfo.proto: %d***** seqno: %d *****ackno: %d*****”, my-
info.proto,myinfo.tcpseq, myinfo.tcpack);
strptr=inet_ntoa(ip_hdr->ip_src);
strncpy(s1, strptr,17);
strptr=inet_ntoa(ip_hdr->ip_dst);
strncpy(s2, strptr,17);
myinfo.window=(uint16_t)(window);
if((int)readPkts%1000000==0) {
printf(“Sent %d packets.”,(int)readPkts);
}
memcpy(&sendPDUS[sendCount],&myinfo, sizeof(struct pduinfo));
sendCount++;
if(sendCount==sendSize) {
sentbytes=send(srvsocket, &sendPDUS, sendSize*sizeof(struct pduinfo),0);
tSentbytes+=sentbytes;
sentPDUs+=sendCount;
sendSize=1;
    if(sentbytes<=0) {
perror(“Cannot send message on data channel.”);
close(srvsocket);
close(srvctrl);
return(NULL);
}
sendCount=0;
}
} // if(sample==1 && sampleThisPDU==1)

```

```
    if(pkts>0 && (readPkts+1)>pkts) {
/* Read enough pkts lets break. */
printf("ENOUGH!");
break;
}
}
rpStatus=read_post(&inStream, dataPtr,myfilter);
if(rpStatus!=0) {
readPkts++;
inputPDUs++;
DPMIactive=1;
}
else { /* TEMPORARY ENTRY! */
gettimeofday(&tv2,NULL);
dif=(tv2.tv_sec-tv1.tv_sec)*1000000 + (tv2.tv_usec-tv1.tv_usec);
dif=dif/1000000.0;
printf("Sent %g pdus at %g pdus/s. ", readPkts, (double)readPkts/dif);
}
}
if(sendCount>0){
sentbytes=send(srvsocket, &sendPDUS, sendCount*sizeof(struct pduinfo),0);
if(sentbytes<=0){
printf("Error sending the last batch of %d pdus.", sendCount);
}
}
tSentbytes+=sentbytes;
gettimeofday(&tv2,NULL);
dif=(tv2.tv_sec-tv1.tv_sec)*1000000 + (tv2.tv_usec-tv1.tv_usec);
dif=dif/1000000.0;
printf("Sent %g pdus (%d bytes) at %g pdus/s. ", readPkts, tSentbytes, (double)readPkts/dif);
//End Packet processing
if(!closestream(&inStream)){
printf("Problems closing stream?");
}
if (verbose) {
printf("There was a total of %g pkts that matched the filter.",pktCount);
}
printf("<-dataThread()");
terminate=1;
threadCount--;
pthread_exit(NULL);
}
```

Appendix D

ControlThread Thread

```
#ifndef __cplusplus
extern "C" {
#endif
#include <cap_utils.h>
// get the interface
#ifdef __cplusplus
}
#endif

    #include <pthread.h>
#include <unistd.h>
#include <getopt.h>
#include <sys/time.h>
#include <time.h>
#include <iostream>
#include <iomanip>
#include <string.h>
#include <math.h>
#include <signal.h>
#include <string>

    #include <stdio.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/ioctl.h>

    #include "global_variables.h"
#include "globals_functions.h"
" #include "../packetprocessor/globals_structures.h"

    struct timeval tid1,tid2;
struct itimerval mydifftime;

    void* controlConnection(void* ptr){
printf("->controlThread()");
```

```
    while(SOCKID===-1 && terminate==0) {
printf("Waiting for dataThread to provide ID.");
sleep(1);
}
printf("Got the ID: %d", SOCKID);
if(terminate==1){
printf("<-controlThread()");
close(srvctrl);
pthread_exit(NULL);
}
/* Send the SOCKID to the control socket. */
CIM.sockid=htonl(SOCKID);
CIM.dataPort=htonl(dataPort);
printf("Sending CIM.sockid=%d CIM.dataPort=%d", ntohl(CIM.sockid), ntohl(CIM.dataPort));
readbytes=send(srvctrl,&CIM, sizeof(CIM),0);
if(readbytes<=0){
perror("Cannot send message to control channel.");
close(srvsocket);
close(srvctrl);
pthread_exit(NULL);
}

    /* Setup sampler */
tid1.tv_sec=tid2.tv_sec=SAMPLETIME;
tid1.tv_usec=tid2.tv_usec=0;
mydifftime.it_interval=tid2;
mydifftime.it_value=tid1;

    if(signal(SIGALRM, statisticsSampler)==SIG_ERR){
perror("Problems with statisticsSampler signal.");
return 0;
}

    if(setitimer(ITIMER_REAL, &mydifftime, NULL)==-1){
perror("Problems with setitimer.");
return 0;
}
printf("Setup statistics sampler, every %d seconds.", (int)mydifftime.it_interval.tv_sec);

    /* Main loop.. */
struct ctrlmsg *themessage;
struct sampletype *stmmsg;
struct PPfilterMod *PPfM;
struct identification *idmsg;
struct synchmessage *syncmmsg;
struct synchmessage syncmmsgResp;
struct PPstatus myStatus;
in_addr src,srcmask,dst,dstmask;
char *strptr;
```

```

char s1[17], s2[17], s3[17],s4[17];
struct timeval tv;
FILE* datasent;
datasent=fopen("datasent.txt","w");

    while(true && terminate==0) {
ioctlreturn=ioctl(srvctrl, FIONREAD, &buffercontent);
printf("Checking CTRL channel, got %d bytes of data.", buffercontent);
gettimeofday(&tv,NULL);
if((unsigned)buffercontent>=sizeof(struct ctrlmsg)) {
readbytes=recv(srvctrl, &servermessage, sizeof(struct ctrlmsg),0);
if(readbytes== -1) {
if(errno == EINTR) {
perror("***EINTR***");
} else {
perror("Someother error.");
}
} else if (readbytes==0) {
perror("Server has closed control channel.");
close(srvctrl);
close(srvsocket);
} else {
/* Handle the Control Message */
themessage=(struct ctrlmsg*)servermessage;
fprintf(datasent,"ServerCTRL sent (%d bytes) a : %d ",readbytes,ntohl(themessage->msgtype));
switch(ntohl(themessage->msgtype)){
case 1: // Stop sampling
if(sample==1) {
printf("Stopping sampling.");
}
sample=0;
break;
case 2: // Start sampling
if(sample==0) {
printf("Starting sampling.");
}
sample=1;
break;
case 3: // SampleType
stmsg=(struct sampletype*)servermessage;
sampletype=ntohl(stmsg->type);
noPdus=ntohl(stmsg->nopdus);
batchSize=ntohl(stmsg->batchsize);
starttime=ntohl(stmsg->starttime);
currPduCount=0;
currBatchCount=0;
printf("Change sample settings: Type=%d npPdus=%d batchSize=%d start-time=%d", sampletype, noPdus,batchSize, starttime);
break;

```

```

case 4: // Change filter
PPfM=(struct PPfilterMod*)servermessage;
myFilters[ntohl(PPfM->position)]=PPfM->theFilter;
printf("Got a filter for position %d", ntohl(PPfM->position));
printf("Active filters.");
for(int i=0;i<10;i++){
if(myFilters[i].index!=0){
src.s_addr=myFilters[i].IPsrc;
srcmask.s_addr=myFilters[i].IPsrcMask;
dst.s_addr=myFilters[i].IPdst;
dstmask.s_addr=myFilters[i].IPdstMask;
strptr=inet_ntoa(src);
strncpy(s1, strptr,17);
strptr=inet_ntoa(srcmask);
strncpy(s2, strptr,17);
strptr=inet_ntoa(dst);
strncpy(s3, strptr,17);
strptr=inet_ntoa(dstmask);
strncpy(s4, strptr,17);
printf("[%d]:%d %s/%s %s/%s %0d %d/%0x %d/%0x",
i,
ntohl(myFilters[i].index),
s1,s2,s3,s4,
ntohs(myFilters[i].IP_PROTO),
ntohs(myFilters[i].SrcPort),
ntohs(myFilters[i].SrcPortMask),
ntohs(myFilters[i].DstPort),
ntohs(myFilters[i].DstPortMask));
}
}
fflush(stdout);
printf("Done.");

break;
case 5: // Change Identification Method
idmsg=(struct identification*)servermessage;
identificationMethod=ntohl(idmsg->type);
printf("Change identification settings: %d", identificationMethod);
break;
case 6: // Get the current status of PP
printf("Got Status request:");
printf("avgInput %d avgFilter %d avgSent %d",avgInput,avgFiltered, avgSent);
myStatus.input=htonl(CinputPDUs);
myStatus.filtered=htonl(CfilteredPDUs);
myStatus.sent=htonl(CsentPDUs);
myStatus.avgInput=htonl(avgInput);
myStatus.avgFiltered=htonl(avgFiltered);
myStatus.avgSent=htonl(avgSent);
myStatus.maxInputRate=htonl(maxInputRate);
myStatus.maxFilterRate=htonl(maxFilterRate);

```

```

myStatus.maxSentRate=htonl(maxSentRate);
myStatus.dpmiSilenceTime=htonl(silenceTime);
myStatus.dpmiTimeoutCount=htonl(DPMItimeoutCount);
myStatus.samplingMethod=htonl(sampletype);
myStatus.collecting=htonl(sample);
myStatus.NoPdus=htonl(noPdus);
myStatus.BatchSize=htonl(batchSize);
myStatus.StartTime=htonl((int)starttime);
myStatus.identificationMethod=htonl(identificationMethod);
memcpy(&myStatus.theFilter, &myFilters, sizeof(myFilters));

    printf("Statistics");
printf("LocationCurrentMaxAverage");
printf("Input %d%d%d", inputPDUs, maxInputRate, ntohl(myStatus.avgInput));
printf("Filter %d%d%d", filteredPDUs, maxFilterRate, ntohl(myStatus.avgFiltered));
printf("Sent %d%d%d", sentPDUs, maxSentRate, ntohl(myStatus.avgSent));
printf("DPMIactivity = %d Timeouts %d ", DPMIactive, DPMItimeoutCount);
printf("DPMI last heard %d seconds ago.", silenceTime);

    printf("Status: ");
if(sample==1){
printf("Collecting.");
}else {
printf("Idle.");
}
printf("SamplingType: %d nopdus %d batchsize %d starttime %d.", sampletype,
noPdus, batchSize,(int)starttime);
printf("Identification: type %d ", identificationMethod);
readbytes=send(srvctrl,&myStatus, sizeof(myStatus),0);
if(readbytes<=0){
perror("Cannot send message to control channel.");
terminate=1;
close(srvsocket);
close(srvctrl);
pthread_exit(NULL);
} else {
printf("*****Sent %d bytes.", readbytes);
}
break;
case 7:
printf("Got Shutdown message.");
terminate=1;
break;
case 8: // Synchronize PP and DE
fprintf(datasent,"we got a message from data evaluator");
syncmsg=(struct synchmessage*)servermessage;
syncmsgResp.DEtime.tv_sec=syncmsg->DEtime.tv_sec;

    DESENDtime.tv_sec=ntohl(syncmsg->DEtime.tv_sec);
DESENDtime.tv_usec=ntohl(syncmsg->DEtime.tv_usec);

```

```

    syncmsgResp.PParrTime.tv_sec=htonl(tv.tv_sec);
    syncmsgResp.PParrTime.tv_usec=htonl(tv.tv_usec);

    SendSyncMsg++;
    fprintf(datasent,"Got a sync. message for the %d time. ",SendSyncMsg );
    printf("Got a sync. message for the %d time. ",SendSyncMsg);
    readbytes=send(srvctrl,&syncmsgResp, sizeof(syncmsgResp),0);
    if(readbytes<=0){
    printf("Cannot send message to control channel.");
    } else {
    fprintf(datasent,"*****Sent %d bytes.", readbytes);
    }
    break;

    default:
    printf("Not supported.");
    break;
    }
    } else {
    sleep(1);
    }
    }
    printf("<-controlThread()");
    pthread_exit(NULL);
    }

void statisticsSampler(int signo){
// printf("statisticsSampler->");
int i=0;
struct timeval NOWtime;
int averages[4];
CinputPDUs=inputPDUs/((int)tid2.tv_sec);
CfilteredPDUs=filteredPDUs/((int)tid2.tv_sec);
CsentPDUs=sentPDUs/((int)tid2.tv_sec);

    for(i=8;i>=0;i-){
    historyInput[i+1]=historyInput[i];
    historyFilter[i+1]=historyFilter[i];
    historySent[i+1]=historySent[i];
    }
    historyInput[0]=CinputPDUs;
    historyFilter[0]=CfilteredPDUs;
    historySent[0]=CsentPDUs;

    if(DPMIactive==0){
/* Timeout of dpmi */
DPMItimeoutCount++;
} else {

```

```

gettimeofday(&lastDPMIread, NULL);
DPMItimeoutCount=0;
}

    if(CinputPDUs>maxInputRate) {
maxInputRate=CinputPDUs;
    }
    if(CfilteredPDUs>maxFilterRate){
maxFilterRate=CfilteredPDUs;
    }
    if(CsentPDUs>maxSentRate){
maxSentRate=CsentPDUs;
    }
gettimeofday(&NOWtime, NULL);
silenceTime=NOWtime.tv_sec-lastDPMIread.tv_sec;

    averages[0]=0;
averages[1]=0;
averages[2]=0;
averages[3]=0;

    for(i=0;i<10;i++){
// printf("%d : %d %d %d ",i, historyInput[i], historyFilter[i], historySent[i]);
averages[0]+=historyInput[i];
averages[1]+=historyFilter[i];
averages[2]+=historySent[i];
if(myFilters[i].index!=-1){
(averages[3])++;
}
}

    avgInput=averages[0]/(10);
avgFiltered=averages[1]/(10);
avgSent=averages[2]/(10);
printf("%c[2J %c[H",27,27);
printf("Statistics %d", (int)NOWtime.tv_sec);
printf("LocationCurrentMaxAverage (10 intervals) [p/s]");
printf("Input %d%d%d", CinputPDUs, maxInputRate, avgInput);
printf("Filter %d%d%d", CfilteredPDUs, maxFilterRate, avgFiltered);
printf("Sent %d%d%d", CsentPDUs, maxSentRate, avgSent);
printf("DPMIactivity = %d Timeouts %d ", DPMIactive, DPMItimeoutCount);
printf("DPMI last heard %d seconds ago.", silenceTime);
if(averages[3]==0){
printf("No filters present.");
} else {
printf("%d filters present.", averages[3]);
for(i=0;i<10;i++){
if(myFilters[i].index>-1){
printf("[%d]: %d %0lx %0lx %hx %hx %hx ",i,
myFilters[i].index,

```

```
(long unsigned int)myFilters[i].IPsrc,
(long unsigned int)myFilters[i].IPdst,
myFilters[i].IP_PROTO,
myFilters[i].SrcPort,
myFilters[i].DstPort);
}
}
}
printf("Status: ");
if(sample==1){
printf("Collecting.");
}else {
printf("Idle.");
}
printf("SamplingType: %d nopdus %d batchsize %d starttime %d.", samplotype,
noPdus, batchSize,(int)starttime);
printf(" : curr %d current %d %d.", currPduCount, currBatchCount,(int)NOWtime.tv_sec);
printf("Identification: type %d ", identificationMethod);
DPMIactive=0;
inputPDUs=0;
filteredPDUs=0;
sentPDUs=0;
return;
}
```

Appendix E

Globals Of PacketProcessor Package

E.1 Globals

```
#ifndef _THEGLOBALS
#define _THEGLOBALS
#ifdef __cplusplus
extern "C" {
#endif
#include <cap_utils.h>
// get the interface
#ifdef __cplusplus
}
#endif

    #include <pthread.h>
#include <unistd.h>
#include <getopt.h>
#include <time.h>
#include <iostream>
#include <iomanip>
#include <string.h>
#include <math.h>
#include <signal.h>
#include <netdb.h>
#include <string>
#include <stdio.h>
#include <errno.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
using namespace std;
#include <openssl/sha.h>
#include <stdio.h>
```

```

#include <sys/types.h>
#define APP_VERSION_MAJOR 0
#define APP_VERSION_MINOR 1
#define ETHERTYPE_VLAN 0x8100
#define SERVER_PORT 3490

    /* The handler for the alarm, i.e. DPMI timeout */
void alarmHandler(int signo);

    /* timeout counter */
extern int timeout;// Counter for the number of timeouts that has occurred.
extern int timeout_value; // Timeout time in seconds.
extern int sampleAlgorithm, identificationMethod, noPdus, batchSize, currP-
duCount, currBatchCount;
extern pthread_t dataThread, controlThread;
extern int terminate; /* Set to 1 to terminate all threads !*/
extern int threadCount;

    void cleanup_main(int);
void* controlConnection(void* ptr);
void* dataHandlerThread(void* ptr);

    extern double pkts;
extern struct pduinfo myinfo;
extern int srvsocket, srvctrl, ioctl_return;
extern struct hostent *he;
extern struct sockaddr_in server_addr,server_addrCtrl,myDataAddr, myDataAd-
drctrl;
extern char servermessage[1500];
extern int SOCKID,i, buffercontent, readbytes, sumofbytes, Serverport, verbose;
extern char* servername;
extern struct ctrlInitMsg CIM;
extern int dataPort, ctrlPort, sentbytes, sendCount;
extern struct pduinfo sendPDUS[100];

    /* Libcap 0.5 variables.. */
extern char *filename;
extern struct filter* myfilter;
extern struct stream inStream; // Stream to read from
extern int streamType; // Stream type
extern int l; // Temporary variable
extern char *nic; // Holder of NIC string
extern int portnumber; // Port to listen to
extern int rpStatus; // read post status
extern char d; // For dealing with packets
extern char* data; // -"-
extern char** dataPtr;// -"-
extern double pktCount;

    /* END of Libcap 0.5 variables ..*/

```

```
#endif
```

E.2 Global Variables

```
#ifndef _THEGLOBALS
#define _THEGLOBALS
#define _DEBUG
#ifdef _DEBUG
#define _DEBUG_MSG(x) (x);
fflush(stdout);
#else
#define _DEBUG_MSG(x)
#endif

    /* timeout counter */
extern int timeout; // Counter for the number of timeouts that has occurred.
extern int timeout_value; // Timeout time in seconds.
extern pthread_t dataThread, controlThread;
extern int terminate; /* Set to 1 to terminate all threads !*/
extern int threadCount;
extern double pkts;
extern struct pduinfo myinfo;
extern struct flag tcp_f;
extern int srvsocket, srvctrl, ioctl_return;
extern struct hostent *he;
extern struct sockaddr_in server_addr, server_addrCtrl, myDataAddr, myDataAddrCtrl;
extern char servermessage[1500];
extern int SOCKID, i, buffercontent, readbytes, sumofbytes, Serverport, verbose;
extern char* servername;
extern struct ctrlInitMsg CIM;
extern int dataPort, ctrlPort, sentbytes, sendCount;
extern struct pduinfo sendPDUS[512];
extern int sendSize;
extern struct timeval DESENDtime;
extern struct pduinfo myinfoSync;
extern int SendSyncMsg;
/* Status variables */
extern int streamType; //for filtering purpose
extern struct timeval lastDPMIread;
extern int DPMIactive;
extern int DPMItimeoutCount;
extern int sample;
extern int sampletype;
extern int identificationMethod;
extern int noPdu;
extern int batchSize;
extern int currPduCount;
```

```
extern int currBatchCount;
extern uint32_t starttime;
extern struct PPfilter myFilters[10];
extern int inputPDUs;
extern int filteredPDUs;
extern int sentPDUs;
extern int CinputPDUs;
extern int CfilteredPDUs;
extern int CsentPDUs;
extern int maxInputRate, maxFilterRate,maxSentRate;
extern int avgInput;
extern int avgFiltered;
extern int avgSent;
extern int silenceTime;
extern int historyInput[10];
extern int historyFilter[10];
extern int historySent[10];
extern int SAMPLETIME;

    /* Libcap 0.5 variables.. */
extern char *filename;
extern struct filter* myfilter;
extern struct stream inStream; // Stream to read from
extern int streamType; // Stream type
extern int l; // Temporary variable
extern char *nic; // Holder of NIC string
extern int portnumber; // Port to listen to
extern int rpStatus; // read post status
extern char d; // For dealing with packets
extern char* data;
extern char** dataPtr;
extern double pktCount;

    /* END of Libcap 0.5 variables ..*/
#endif
```

E.3 Global Functions

```
/* The handler for the alarm, i.e. DPMI timeout */
void alarmHandler(int signo);
void cleanup_main(int);
void statisticsSampler(int signo);

    /* Threads */
void* controlConnection(void* ptr);
void* dataHandlerThread(void* ptr);
```

E.4 Global Structures

```
struct pt{
time_t tv_sec;
uint64_t tv_nsec;
} __attribute__((packed));
typedef struct pt tpico;

    struct tcp_flags{
unsigned char FIN;
unsigned char SYN;
unsigned char RST;
unsigned char PUSH;
unsigned char ACK;
unsigned char URG;
}__attribute__((packed));
typedef struct tcp_flags t_flag;

    struct pduinfo{
char mpid[8];
char nic[8];
uint16_t caplen;
uint16_t len;
uint32_t ipdst;
uint32_t ipsrc;
uint16_t proto;
uint16_t portsrc;
uint16_t portdst;
uint32_t tcpseq;
uint32_t tpack;
uint16_t window;
tpico ts;
t_flag flags;
// unsigned char identification[20];
} __attribute__((packed));

    struct ctrlmsg{
int msgtype;
char data[36];
}__attribute__((packed));

    struct sampletype{
int msgtype;
uint32_t type;
uint32_t nopdus;
uint32_t batchsize;
uint32_t starttime;
char padding[20];
} __attribute__((packed));
```

```
    struct identification{
int msgtype;
int type;
char padding[32];
} __attribute__((packed));

    struct synchmessage{
int msgtype;
struct timeval DEtime;
struct timeval PParrTime;
struct timeval PPsendTime;
char padding[32];
} __attribute__((packed));

    struct bufferContent{
int free;
struct pduinfo myPdu;
};

    struct dataHandler{
int listenSocket;
int port;
};

    struct controlHandler{
int listenSocket;
int port;
};

    struct ctrlInitMsg{
int sockid;
int dataPort;
}__attribute__((packed));

    struct PPfilter{
int index;
uint32_t IPsrc;
uint32_t IPsrcMask;
uint32_t IPdst;
uint32_t IPdstMask;
uint16_t IP_PROTO;
uint16_t SrcPort;
uint16_t SrcPortMask;
uint16_t DstPort;
uint16_t DstPortMask;
} __attribute__((packed));

    struct PPfilterMod{
int msgtype;
int position;
```

```
struct PPfilter theFilter;
char padding[2];
}__attribute__((packed));

    struct PPstatus{
int input;
int filtered;
int sent;
int avgInput;
int avgFiltered;
int avgSent;
int dpmiTimeoutCount;
int dpmiSilenceTime;
int maxInputRate;
int maxFilterRate;
int maxSentRate;
int collecting;
int samplingMethod;
int NoPdus;
int BatchSize;
int StartTime;
int identificationMethod;
struct PPfilter theFilter[10];
};

    struct PProcessor{
struct PProcessor* next;
/* Used for identification; IPAddress of connector, port and ParentID of the call-
ing process*/
uint32_t IP;
unsigned short port;
int ppid;
pthread_t thread_id;
struct bufferContent* buffer;
int bufferSize;
int PPwriting;
int writePos;
int readPos;
int writeOK;
int terminate;
int PPdataSocket;
int PPctrlSocket;
int ctrlPort;
int statusCount;
struct PPstatus lastStatus;
};
```

```
    struct pt{
time_t tv_sec;
uint64_t tv_nsec;
} __attribute__((packed));
typedef struct pt tpico;

    struct tcp_flags{
unsigned char FIN;
unsigned char SYN;
unsigned char RST;
unsigned char PUSH;
unsigned char ACK;
unsigned char URG;
} __attribute__((packed));
typedef struct tcp_flags t_flag;

    struct pduinfo{
char mpid[8];
char nic[8];
uint16_t caplen;
uint16_t len;
uint32_t ipdst;
uint32_t ipsrc;
uint16_t proto;
uint16_t portsrc;
uint16_t portdst;
uint32_t tcpseq;
uint32_t tcpack;
uint16_t window;
tpico ts;
t_flag flags;
// unsigned char identification[20];
} __attribute__((packed));

    struct ctrlmsg{
int msgtype;
char data[36];
} __attribute__((packed));

    struct sampletype{
int msgtype;
uint32_t type;
uint32_t nopdus;
uint32_t batchsize;
uint32_t starttime;
char padding[20];
} __attribute__((packed));

    struct identification{
int msgtype;
```

```
int type;
char padding[32];
} __attribute__((packed));

    struct synchmessage{
int msgtype;
struct timeval DEtime;
struct timeval PParrTime;
struct timeval PPsendTime;
char padding[32];
} __attribute__((packed));

    struct bufferContent{
int free;
struct pduinfo myPdu;
};

    struct dataHandler{
int listenSocket;
int port;
};

    struct controlHandler{
int listenSocket;
int port;
};

    struct ctrlInitMsg{
int sockid;
int dataPort;
}__attribute__((packed));

    struct PPfilter{
int index;
uint32_t IPsrc;
uint32_t IPsrcMask;
uint32_t IPdst;
uint32_t IPdstMask;
uint16_t IP_PROTO;
uint16_t SrcPort;
uint16_t SrcPortMask;
uint16_t DstPort;
uint16_t DstPortMask;
} __attribute__((packed));

    struct PPfilterMod{
int msgtype;
int position;
struct PPfilter theFilter;
char padding[2];
```

```
}_attribute((packed));

    struct PPstatus{
int input;
int filtered;
int sent;
int avgInput;
int avgFiltered;
int avgSent;
int dpmiTimeoutCount;
int dpmiSilenceTime;
int maxInputRate;
int maxFilterRate;
int maxSentRate;
int collecting;
int samplingMethod;
int NoPdus;
int BatchSize;
int StartTime;
int identificationMethod;
struct PPfilter theFilter[10];
};

    struct PProcessor{
struct PProcessor* next;
/* Used for identification; IPAddress of connector, port and ParentID of the calling process*/
uint32_t IP;
unsigned short port;
int ppid;

    pthread_t thread_id;
struct bufferContent* buffer;
int bufferSize;
int PPwriting;
int writePos;
int readPos;
int writeOK;
int terminate;
int PPdataSocket;
int PPctrlSocket;
int ctrlPort;

    int statusCount;
struct PPstatus lastStatus;
};
```

Appendix F

ServerThreaded Thread

```
#include "sys.h"
#include "debug.h"
#include <pthread.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
#include <stdint.h>
#include <sys/socket.h>

using std::string;

/* Contains definitions of structures */
#include "global_structures.h"
#include "global_functions.h"
#include "global_variables.h"

#define MYPOR 3499 // the port users will be connecting to
#define BACKLOG 10 // how many pending connections queue will hold
#define MATPORT 8601 //the port matlab will be connecting to

/* Global variables */
/* All others have to extern them. */
pthread_t dataThread, controlThread, ctrlThread, calcThread, sampleThread;
struct PProcessor* PPlist;
struct PProcessor* availPPlist;
struct RTT *firstRTT, *lastRTT;
int NoPPs;
int terminate; /* Set to 1 to terminate all threads !*/
int myShutdown; /* Set to 1 to shutdown all threads, nicer than terminate */
int threadCount;
int collecting;
int sampletype;
```

```

int nopdus;
int batchsize;
uint32_t starttime;
int identificationmethod;
int DPMISILENTTIME;
uint32_t basetime; // Arrivaltime of the first PDU to the system
uint32_t youngest; // Arrivaltime of the youngest (newest) PDU to the system.
uint32_t housekeepingCounter; // Counter that keeps track on the n# of house-
keeping operations.double samplefreq; // Sample frequency.
int MAIN_IRQ_EXEC;
double samplefreq; // Sample frequency.
int noSN, noIN, noLN, noBDN;
string FILENAME;
int fileNumber_del, fileNumber_bps;
int dataCount;
int rPos_del, rPos_bps, wPos_del, wPos_bps;
int DONOTSAMPLE;
struct PPfilterMod theFilters[10];
streamDistinguisher *firstNode;
streamDistinguisher *Nodes;
streamDistinguisher *currNode;
streamDistinguisher *freeSDNodes;
RoundTripNode *startNode;
RoundTripNode *thisNode;
int IDLETIME;
int slcount;
int pTimeCount;
int readpos, writepos;
string host, database, table_del, table_bps, user, password;
int OUT_DESTINATIONS;
/* Semaphores for the PPlist */
int CDactive, DEactive, CCactive;
int CCsample, CCfree, SSsample, SSfree;
struct PProcessor* DEobject;
pthread_mutex_t FreeSDLock=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t SampleSNLock=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t FreeINLock=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t SampleINLock=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t FreeRTTLock=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t RTTLock=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t SampleLNLock=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t FreeRTN=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t SampleBDNs=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t cout_mutex=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t AvailPPs=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t ActiveSD=PTHREAD_MUTEX_INITIALIZER;

void cleanup_main(int);
int main(void)
{

```

```

int mainPID;
readpos=0;writepos=0;
signal(SIGINT,cleanup_main);
mainPID=pthread_self();
DPMISILENTTIME=defaultDPMISILENTTIME;
IDLETIME=5;
terminate=0;
myShutdown=0;
threadCount=0;
NoPPs=0;
PPlist=0;
noSN=noIN=noLN=noBDN;
DNOTSAMPLE=0;
FILENAME= "OUTPUT";
startNode=thisNode=0;
Nodes=freeSDNodes=firstNode=currNode=0;
fileNumber_del=fileNumber_bps=0;
dataCount=0;
rPos_del=rPos_bps=wPos_del=wPos_bps=0;
collecting=1;
sampletype=0;
nopdus=0;
batchsize=0;
starttime=0;
identificationmethod=0;
basetime=0;
samplefreq=1;
MAIN_IRQ_EXEC=0;
youngest=0;
housekeepingCounter=0;
CCsample=0;
CCfree=0;
SSsample=0;
SSfree=0;
for(int i=0;i<10;i++){
theFilters[i].position=i;
theFilters[i].theFilter.index=0;
}
host="localhost";
password="";
user="";
database="myDB";
table_del="delay";
table_bps="throughput";
OUT_DESTINATIONS=1;
struct dataHandler myData;
struct controlHandler myControl;
struct matlabHandler matHandle;
matHandle.port=MATPORT;
myData.port=MYPORT;

```

```
myControl.port=MYPORT+1;
PPlist=0;
NoPPs=0;
terminate=0;
PPlist=0;
availPPlist=0;

/* Allocating a bunch of PProcessors */
PProcessor *myPP;
printf("PreAllocating PProcessors; ");
for(int i=0;i<5;i++){
myPP = new(struct PProcessor);
myPP->buffer=(struct bufferContent*)malloc(BUFFERsize*sizeof(bufferContent));
bzero(myPP->buffer, BUFFERsize*sizeof(bufferContent));
myPP->bufferSize=BUFFERsize;
myPP->writePos=0;
myPP->readPos=0;
myPP->writeOK=1;
myPP->terminate=0;
myPP->statusCount=0;
myPP->next=availPPlist;
availPPlist=myPP;
printf("%d : %p -> %p ",i, myPP, myPP->next);
}

    if(pthread_create(&outputThread, NULL, output_thread,NULL)){
perror("Cannot create Output Thread. ");
terminate=1;
abort();
} else {
if(pthread_create(&calcThread, NULL, calculator, NULL)){
perror("Cannot create Calculator Thread.");
terminate=1;
abort();
} else {
if(pthread_create(&sampleThread, NULL, sample_thread, &matHandle)) {
perror("Cannot create SAMPLE thread.");
terminate=1;
abort();
} else {
if(pthread_create(&dataThread, NULL, dataConnection, &myData)) {
perror("Cannot create data connection thread.");
terminate=1;
abort();
} else {
if(pthread_create(&controlThread, NULL, controlConnection, &myControl)) {
perror("Cannot create control connection thread.");
terminate=1;
abort();
}
}
```

```

}
}
}
}
}
pthread_join(controlThread,NULL);
pthread_join(dataThread,NULL);
pthread_join(calcThread,NULL);
pthread_join(sampleThread,NULL);
printf("MAIN: Got the BASE THREADS. Now are all DataHandlers dead?");

/* Initializing a kill of remaining PProcessors. This should not be done before the calculator thread has completed. Perhaps the calculator thread should free the resources as well. */
struct PProcessor *tmpKill;
tmpKill=PPlist;
if(PPlist!=0) {
printf("Got some Data Handlers to kill.");
while(tmpKill!=0) {
printf("Waiting for %d ", (int)tmpKill->thread_id);
printf(" WriteOK %d ", tmpKill->writeOK);
printf("WritePos %d ", tmpKill->writePos);
printf(" readPos %d ", tmpKill->readPos);
pthread_join(tmpKill->thread_id, NULL);
tmpKill=tmpKill->next;
}
} else {
printf("No Data Handlers to kill. ");
}
printf("These threads are still alive %d.", threadCount);
return(0);
}

void cleanup_main(int signo){
printf("Cleanup_main, called %d times. There are %d threads.", terminate+1, threadCount);
terminate++;
return;
}

```


Appendix G

DataConnection Thread

```
#include "sys.h"
#include "debug.h"
#include <pthread.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
using std::string;
#include "global_structures.h"
#include "global_variables.h"
#include "global_functions.h"

void* dataConnection(void* ptr){
struct dataHandler* input=0;
input=(struct dataHandler*)ptr;
struct sockaddr_in listenSockaddr, PPAAddr;
socklen_t PPAAddr_len=sizeof(PPAAddr);
int listenSocket, ppSocket, readPPID;
int readbytes;
int terminateThread=0;
threadCount++;
listenSocket=socket(AF_INET, SOCK_STREAM, 0);
if(listenSocket<0) {
threadCount--;
pthread_exit(NULL);
return(0);
}
listenSockaddr.sin_family=AF_INET;
listenSockaddr.sin_addr.s_addr=htonl(INADDR_ANY);
listenSockaddr.sin_port=htons(input->port);
```

```

if(bind(listenSocket,(struct sockaddr*)&listenSockaddr, sizeof(listenSockaddr))<0){
printf("Cannot bind to the %d port.", input->port);
printf("dataConnection (DCthread) aborted due to error.");
pthread_exit(NULL);
}
listen(listenSocket, 30);
printf("DataConnection waiting on port %d.", input->port);
printf("Grabbing a PP ");
struct PProcessor* myPP;
getAvailPP();
if(availPPlist!=0) {
myPP=availPPlist;
availPPlist=availPPlist->next;
printf("from availPPlist, %p. Availnow points at %p.",myPP, availPPlist);
} else {
myPP = new(struct PProcessor);
myPP->buffer=(struct bufferContent*)malloc(BUFFERsize*sizeof(bufferContent));
myPP->bufferSize=BUFFERsize;
myPP->writePos=0;
myPP->readPos=0;
myPP->writeOK=1;
myPP->terminate=0;
myPP->statusCount=0;
printf(" by creation, %p. availPPlist %p ", myPP, availPPlist);
}
freeAvailPP();
struct timeval accept_timeout;
fd_set rset;
FD_ZERO(&rset);
FD_SET(listenSocket, &rset);
accept_timeout.tv_sec=5;
accept_timeout.tv_usec=0;
int selectReturn;
while(terminateThread==0 && terminate==0) {
while( (selectReturn=select(listenSocket+1, &rset, NULL, NULL, &accept_timeout))<=0
&& terminateThread==0 && terminate==0) {
if(terminateThread!=0 —— terminate!=0) {
printf("DCThread2: got a break signal.");
break;
}
if(selectReturn===-1) {
delete(myPP);
close(listenSocket);
pthread_exit(NULL);
} else if (selectReturn==0) {
FD_SET(listenSocket, &rset);
accept_timeout.tv_sec=5;
accept_timeout.tv_usec=0;
//printf("DCThread: Timeout happend, no connection attempt during the last
%d seconds ", (int)accept_timeout.tv_sec);

```

```

}
}
printf("DCThread: Left second while with %d %d %d ", selectReturn, terminateThread, terminate);
if(terminateThread!=0 —— terminate!=0) {
printf("DCThread1: got a break signal.");
break;
}
ppSocket = accept(listenSocket, (struct sockaddr*)&PAddr, &PAddr_len);
if(ppSocket===-1) {
} else {
/* Proper connection */
myPP->next=0;
myPP->PPdataSocket=ppSocket;
myPP->PPctrlSocket=0;
myPP->IP=(uint32_t)ntohl((PAddr.sin_addr.s_addr));
myPP->port=ntohs(PAddr.sin_port);
myPP->statusCount=0;
myPP->terminate=0;
myPP->writePos=0;
myPP->readPos=0;
myPP->writeOK=1;
if(myPP->buffer==0){
myPP->buffer=(struct bufferContent*)malloc(BUFFERsize*sizeof(bufferContent));
}
bzero(myPP->buffer, BUFFERsize*sizeof(bufferContent));
myPP->bufferSize=BUFFERsize;
readPPID=htonl(ppSocket);

/* Send the socket Identifier */
printf("The assigned PP has terminate = %d ", myPP->terminate);
myPP->terminate=0;
printf("DCThread: Accept from %s:%d ", inet_ntoa(PAddr.sin_addr), ntohs(PAddr.sin_port));
readbytes=send(ppSocket, &readPPID,sizeof(readPPID),0);
printf("DCThread: Identifier sent to PP %d, sent %d bytes .", ntohl(readPPID),
readbytes);
if(readbytes<0) {
terminateThread=1;
} else {
printf("DCThread: Setting up dataHandlerThread.");
myPP->ppid=ppSocket;
/* Insert into list */
while(CCactive==1 && DEactive){
/* Waiting for CC and DE to finish */
}
CDactive=1; /* Signal that Pplist is used by CD.*/
/* Put the entry first */
myPP->next=Pplist;
Pplist=myPP;
NoPPs++; // Increase the count of PPS.

```

```
CDactive=0; /* Signal that PPlist is free */
/* Thread of the DataHandler, and give it the PProcessor
structure as input. */
printf("DCthread: pthread_create(dataHandler).");
if(pthread_create(&myPP->thread_id, NULL, dataHandlerThread, myPP)) {
terminateThread=1;
}
printf("DCthread: Allocating PProcessor stuff.");
/* DataHandler started, PProcessor added, allocate new PProcessor. */
if(availPPlist!=0 && CCactive==0){
CDactive=1;
myPP=availPPlist;
availPPlist=availPPlist->next;
CDactive=0;
} else {
myPP = new(struct PProcessor);
myPP->buffer=(struct bufferContent*)malloc(BUFFERsize*sizeof(bufferContent));
bzero(myPP->buffer, BUFFERsize*sizeof(bufferContent));
myPP->bufferSize=BUFFERsize;
myPP->writePos=0;
myPP->readPos=0;
myPP->writeOK=1;
myPP->terminate=0;
myPP->statusCount=0;
}
printf("DCthread: PProcessor allocation done.");
} // else readbytes>=0
} // else ppSocket != -1
} // while terminateThread==0
delete(myPP);
close(listenSocket);
threadCount--;
printf("dataConnection (DCthread) terminated.");
pthread_exit(NULL);
}
```

Appendix H

ControlConnection Thread

```
#include "sys.h"
#include "debug.h"

#include <pthread.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdint.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
using std::string;
#include "global_structures.h"
#include "global_variables.h"
#include "global_functions.h"

void* controlConnection(void* ptr){
printf("controlConnection (CCthread) started.");
struct controlHandler* input=0;
input=(struct controlHandler*)ptr;
struct sockaddr_in listenSockaddr, PPAAddr;
socklen_t PPAAddr_len;
struct timeval timeout_tv,dtime;
time_t currttime,timelimit;
int listenSocket, ppSocket;
int readbytes;
int terminateThread=0;
FILE *ccfile;
ccfile=fopen("ctrlmsgdemo.txt","w");
struct ctrlInitMsg CIM;
```

```

struct PProcessor* tmp;
threadCount++;
listenSocket=socket(AF_INET, SOCK_STREAM, 0);
if(listenSocket<0) {
return(0);
}
listenSockaddr.sin_family=AF_INET;
listenSockaddr.sin_addr.s_addr=htonl(INADDR_ANY);
listenSockaddr.sin_port=htons(input->port);

    if(bind(listenSocket,(struct sockaddr*)&listenSockaddr, sizeof(listenSockaddr))<0){
printf("CCthread: Cannot bind to the %d port.", input->port);
printf("controlConnection (CCthread) aborted due to error.");
terminate=1;
pthread_exit(0);
}
timeout_tv.tv_sec=5;
timeout_tv.tv_usec=0;
gettimeofday(&dtime,NULL);
timelimit=dtime.tv_sec;
timelimit+=10;
listen(listenSocket, 30);
printf("ControlConnection waiting on port %d.", input->port);
struct timeval accept_timeout;
fd_set rset;
FD_ZERO(&rset);
FD_SET(listenSocket, &rset);
accept_timeout.tv_sec=5;
accept_timeout.tv_usec=0;
int selectReturn;
int ppCount;
struct ctrlmsg PPsample;
struct sampletype STinit;
struct identification IDinit;
struct PPfilterMod PPFilterMod;
struct synchmessage syncMsg;
int ioctl_return,buffercontent;
sizeof(IDinit),sizeof(PPFilterMod));
int sbytes;

    while(terminateThread==0 && terminate==0) {
currtime=dtime.tv_sec;
fprintf(ccfile, "currtime:%ld timelimit:%ld", currtime, timelimit);
while( (selectReturn=select(listenSocket+1, &rset, NULL, NULL, &accept_timeout))<=0
&& terminateThread==0 && terminate==0) {
if(terminateThread!=0 — terminate!=0) {
printf("CCThread2: got a break signal.");
break;
}
}
if(selectReturn==-1) {

```

```

close(listenSocket);
pthread_exit(NULL);
} else if (selectReturn==0) {
FD_SET(listenSocket, &rset);
accept_timeout.tv_sec=5;
accept_timeout.tv_usec=0;
}
}
printf("CCThread: Left second while with %d %d %d ", selectReturn, terminateThread, terminate);
if(terminateThread!=0 —— terminate!=0) {
printf("CCThread1: got a break signal.");
break;
}
ppSocket = accept(listenSocket, (struct sockaddr*)&PPAddr, &PPAddr_len);
if(ppSocket===-1) {
} else {
/* Proper connection */
readbytes=0;
buffercontent=0;
ioctl_return=ioctl(ppSocket, FIONREAD, &buffercontent);
printf("CCThread: Got a connection, ppSocket %d has %d bytes.", ppSocket, buffercontent);

while((unsigned)buffercontent<sizeof(CIM) && terminate ==0){
ioctl_return=ioctl(ppSocket, FIONREAD, &buffercontent);
printf("CCThread: Waiting for ppSocket %d to deliver data, it has %d / %d.", ppSocket, buffercontent, sizeof(CIM));
sleep(1);
}
printf("CCThread: ppSocket %d has %d bytes CIM = %d & terminate = %d.", ppSocket, buffercontent, sizeof(CIM),terminate);
if(terminate==1){
break;
}
readbytes=recv(ppSocket, &CIM,sizeof(CIM),0);
if(readbytes<0) {
terminateThread=1;
} else {

/* Got the connection */
printf("Got %d bytes vs %d bytes.", readbytes, sizeof(CIM));
printf("CCThread: CTRL conn. for PP sockid = %d dataport = %d .", ntohl(CIM.sockid), ntohl(CIM.dataPort));
tmp=PPlist;
ppCount=0;
syncMsg.msgtype=htonl(8);
syncMsg.DEtime.tv_sec=htonl(dtime.tv_sec);
syncMsg.DEtime.tv_usec=htonl(dtime.tv_usec);
sbytes=send(ppSocket, &syncMsg,sizeof(syncMsg),0);

```

```

if(sbytes<=0){
fprintf(ccfile,“Cannot send sync message to control channel.”);
} else {
fprintf(ccfile,“*****Sent %d bytes.”, sbytes);
}

while(tmp!=0){
fprintf(ccfile,“PProcessor[%d]: sockid = %d dataPort = %d”, ppCount, tmp-
>PPdataSocket, tmp->port);
if((int)tmp->port == (int)ntohl(CIM.dataPort) && (int)tmp->PPdataSocket
== (int)ntohl(CIM.sockid)) {
fprintf(ccfile,“Found the accociated PProcessor entry, %d .”, ppCount);
tmp->PPctrlSocket=ppSocket;
fprintf(ccfile,“Sending current settings to it.”);
flush(ccfile);
if(collecting==1) {
PPsample.msgtype=htonl(2);
} else {
PPsample.msgtype=htonl(1);
}
sbytes=send(ppSocket, &PPsample, sizeof(PPsample),0);
if(sbytes<0){
} else {
STinit.msgtype=htonl(3);
STinit.type=htonl(sampletype);
STinit.nopdus=htonl(nopdus);
STinit.batchsize=htonl(batchsize);
STinit.starttime=htonl(starttime);
fprintf(ccfile,“ST type=%d nopdus=%d batchsize=%d starttime=%d”, ntohl(STinit.type),
ntohl(STinit.nopdus), ntohl(STinit.batchsize), ntohl(STinit.starttime));
sbytes=send(ppSocket, &STinit, sizeof(PPsample),0);
if(sbytes<0){
} else {
IDinit.msgtype=htonl(5);
IDinit.type=htonl(identificationmethod);
fprintf(ccfile,“ID type = %d.”, ntohl(IDinit.type));
sbytes=send(ppSocket, &IDinit, sizeof(IDinit), 0);
if(sbytes<0){
} else {
for(int i=0;i<10;i++){
if(theFilters[i].theFilter.index!=0){
printf(“Sending filter %d.”, i);
PPFilterMod.msgtype=htonl(4);
PPFilterMod.position=htonl(i);
memcpy(&PPFilterMod.theFilter,&theFilters[i].theFilter, sizeof(struct PPfilter-
Mod));
sbytes=send(ppSocket, &PPFilterMod, sizeof(PPFilterMod), 0);
printf(“CCthread: Sent %d bytes.”, sbytes);
if(sbytes<0){
} else {

```

```

printf("Sent Filter %d.", i);
}
}
}
}
}
}
}
}
break;
} else {
ppCount++;
tmp=tmp->next;
}
}
if(tmp==0) {
printf("No Match..Didnt find any PProcessor with dataport %d and sockid %d.",
ntohl(CIM.dataPort),ntohl(CIM.sockid));
PPsample.msgtype=htonl(7);
sbytes=send(ppSocket, &PPsample, sizeof(PPsample),0);
if(sbytes<0){
} else {
printf("Told the PP to shutdown.");
}
close(ppSocket);
}
} // else readbytes>=0
} // else ppSocket != -1
} // while terminateThread==0
close(listenSocket);
threadCount--;
Dout(dc::notice, "Leaving CCthread " << pthread_self());
printf("controlConnection (CCthread) terminated.");
pthread_exit(NULL);
}

```


Appendix I

DataHandler Thread

```
#include "sys.h"
#include "debug.h"
#include <pthread.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/ioctl.h>
using std::string;
#include "global_structures.h"
#include "global_variables.h"
#include "global_functions.h"

void* dataHandlerThread(void* ptr){
struct PProcessor* input=0;
int readBytes=0;
input=(struct PProcessor*)ptr;
struct timeval tv1, tv2;
long int pduCount=0;
int ioctl_return, buffercontent;
int tReadbytes=0;
int mySocket=input->PPdataSocket;
char *pp;
in_addr PacketIP;
char pIP[17];
FILE *syncfile;
syncfile=fopen("SyncTime.txt", "w");
int ppPort=0;
```

```

threadCount++;
struct timeval readtimeout;
fd_set rset;
FD_ZERO(&rset);
FD_SET(input->PPdataSocket, &rset);
readtimeout.tv_sec=2;
readtimeout.tv_usec=0;
int overwriteCount=0;
int selectreturn;
int minRead=sizeof(struct pduinfo);
setsockopt(input->PPdataSocket, SOL_SOCKET, SO_RCVLOWAT, &minRead,
sizeof(minRead));
while(input->terminate==0 && terminate == 0) {
while( (selectreturn=select(input->PPdataSocket+1, &rset, NULL, NULL, &readtimeout))<=0
&& input->terminate==0 && terminate==0){
if(input->terminate!=0 —— terminate!=0) {
printf(“DThread %d: got a break signal.”, input->PPdataSocket);
input->terminate=1;
break;
}
if(selectreturn==-1) {
printf(“DHThread %d: Select error.”, input->PPdataSocket);
input->terminate=1;
break;
} else if (selectreturn==0) {
FD_SET(input->PPdataSocket, &rset);
readtimeout.tv_sec=2;
readtimeout.tv_usec=0;
}
}
if(input->terminate!=0 —— terminate != 0){
break;
}
input->PPwriting=1;
if( (readBytes=recv(input->PPdataSocket, &input->buffer[input->writePos].myPdu,
sizeof(struct pduinfo),0))<0) {
printf(“DHthread %d: Problems reading data (%d vs %d).”, input->PPdataSocket,
readBytes, sizeof(struct pduinfo));
input->terminate=1;
input->PPwriting=0;
} else {

/*check if special */
if((strcmp(input->buffer[input->writePos].myPdu.mpid,“11111111”,8)==0)&&(strcmp(input-
>buffer[input->writePos].myPdu.nic,“11111111”,8)==0)){
ppPort=(int)input->port;
PacketIP.s_addr=input->IP;
pp=inet_ntoa(PacketIP);
strncpy(pIP, pp,17);
fprintf(syncfile,“we have got a sync message from the packetprocessor”);

```

```

fprintf(syncfile, "dataevaluator time %d:%d packetprocessor time %d:%d packet-
ProcessorID %s:%d %d",
input->buffer[input->writePos].myPdu.tcpsseq, input->buffer[input->writePos].myPdu.tcpack,
input->buffer[input->writePos].myPdu.ipsrc, input->buffer[input->writePos].myPdu.ipdst, pIP, ppPort, input-
>writePos);
printf("dataevaluator time %d:%d packetprocessor time %d:%d packetProces-
sorID %s:%d",
input->buffer[input->writePos].myPdu.tcpsseq, input->buffer[input->writePos].myPdu.tcpack,
input->buffer[input->writePos].myPdu.ipsrc, input->buffer[input->writePos].myPdu.ipdst, pIP, ppPort);
fflush(syncfile);
} else {

    input->PPwriting=0;
if(readBytes==0) {
input->terminate=1;
printf("DHtread %d: Remote side closed the connection.", input->PPdataSocket);
ioctl_return=ioctl(input->PPdataSocket, FIONREAD, &buffercontent);
printf("Checking DATA channel, got %d bytes of data.", buffercontent);
printf("Occured after %d bytes, writepos = %d.", tReadbytes, input->writePos);
} else {

    if(pduCount==0) {
gettimeofday(&tv1, NULL);
}
if(readBytes<0) {
printf("WHOOA!!! %d", readBytes);
}
pduCount++;
if((int)readBytes<(int)sizeof(struct pduinfo)) {
printf("Shiute! %d vs. %d", readBytes, sizeof(struct pduinfo));
printf("Occured after %d bytes, writepos = %d.", tReadbytes, input->writePos);
}
tReadbytes+=readBytes;

    input->buffer[input->writePos].free++;
if(input->buffer[input->writePos].free>1) {
if(input->buffer[input->writePos].free>(overwriteCount+1)){
printf("Overwriting data at %d for the %d time (%d)", input->writePos, (input-
>buffer[input->writePos].free)-1,
overwriteCount);
overwriteCount++;
}
} else {
if(overwriteCount>0) {
printf("Reader read, clear overwriteCount.");
overwriteCount=0;
}
}
fflush(stdout);
}
/* Increment the write position */

```

```
if( (input->writePos)+1>=input->bufferSize ) {
input->writePos=0;
} else {
input->writePos++;
}
}
}
}
} // while terminateThread==0 && terminate == 0
if(input->PPdataSocket!=0) {
close(input->PPdataSocket);
input->PPdataSocket=0;
}
input->writeOK=-1;
printf("DataHandler Thread (DHthread) %d terminating.",mySocket);
printf("input->terminate=%d && terminate = %d", input->terminate, terminate);
gettimeofday(&tv2,NULL);
double dif;
dif=(tv2.tv_sec - tv1.tv_sec)*1000000 + (tv2.tv_usec-tv1.tv_usec);
dif=dif/1000000.0;
printf("Processed %ld pdus in %.21f seconds. %g pdu/sec", pduCount,dif, (double)pduCount/dif);
printf("Got %d bytes. ", tReadbytes);
threadCount--;
pthread_exit(NULL);
}
```

Appendix J

Calculator Thread

```
#include "sys.h"
#include "debug.h"
#include <pthread.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
#include <sys/time.h>
#include <time.h>
using std::string;
#include "global_structures.h"
#include "global_variables.h"
#include "global_functions.h"
#include "streamDistinguisher.h"
#include "streamIdentifier.h"
#include "RoundTripNode.h"
#include "ptime.h"

    struct PProcessor* findOldest(void);
int checkCleanPPlist(void);
void getoldest(struct PProcessor* thePP, struct pduinfo* myPDU);
int cleanPPlist(void);
#define CLEANTHRESHOLD 1
int oRp1;
int oRp2;
int oRp3;
int cc_state;
int noSD;
int pCount;
void CC_IRQ(int signo);
```

```
void* calculator(void* ptr){
struct PProcessor *oldest;
struct pduinfo thepdu;
ptime *arrivalTime;
streamDistinguisher *firstNode, *currNode, *freeSDNodes;
RoundTripNode *startNode, *thisNode;
sIcount=0;
pTimeCount=0;
pCount=0;
noSD=BufferSize_StreamDistinguisher;
while(SSfree==1){
usleep(1);
}
CCfree=1;
cc_state=0;
/* pre-allocate data structure objects */
int removeCandidates=0;
int removed=0;
int housekeeping=0;
int idleCount=0;
int idle=0;
printf("Allocating %d StreamNodes.", noSN);
while(SSfree==1){
usleep(1);
}
oRp1=oRp2=oRp3=0;
struct tm ptm;
char timeStr[25];
time_t myTime;
FILE *infile;
infile=fopen("CALCULATOR.input.txt", "w");
in_addr src, dst;
char *strptr;
char s1[17], s2[17];
while(terminate==0){
cc_state=1;
oldest=findOldest();
cc_state=2;
if(oRp3==oRp2) {
}

if(oldest==0){
cc_state=10;
// printf("No oldest.");
/* No oldest found. */
housekeeping=0;
removeCandidates=checkCleanPPlist();
cc_state=12;
if(removeCandidates>0 && MAIN_IRQ_EXEC==1){
```

```

printf("No old pdu found, and %d are remove candidates.", removeCandidates);
/* There are data to remove. */
cc_state=13;
removed=cleanPPlist();
MAIN_IRQ_EXEC=0;
printf("Removed %d/%d entries from PPlist. sleep(6)", removed, removeCandidates);
cc_state=15;
} else {
fflush(stdout);
cc_state=17;
/* Forced Sample */

    if(myShutdown==1 && PPlist==0){
printf("CALCthread: shutdown noted and PPlist empty, pushing out data.");
youngest=youngest+IDLETIME+1;
}
if(idle==0){
idleCount++;
if(idleCount>2*IDLETIME){
idleCount=0;
youngest=youngest+IDLETIME+1;
idle=1;
printf("CALC: Idle Threshold reached, triggering streamDistinguisher deletion.");
//deleting the streamDistinguisher and RoundtripNode when idle time is 24*5=120
sleeps reached
getFreeSD();
if(firstNode!=0){
while(firstNode->next!=0){
currNode=firstNode;
getFreeRTN();
while(currNode->startptr->previous!=0){
thisNode=currNode->startptr;
currNode->startptr=currNode->startptr->previous;
delete(thisNode);
}

    if(currNode->startptr->previous==0){thisNode=currNode->startptr;currNode->startptr=0;delete(thisNode);}
freeRTNDone();
firstNode=firstNode->next;
printf("After 10 sleeps deleting the streamDistinguisher Node and respective
RoundTripNodes %p",currNode);fflush(stdout);
delete(currNode);
}
if(firstNode->next==0){
getFreeRTN();
while(firstNode->startptr->previous!=0){
thisNode=firstNode->startptr;
firstNode->startptr=firstNode->startptr->previous;

```

```
delete(thisNode);
}
if(firstNode->startptr->previous==0){thisNode=firstNode->startptr;firstNode-
>startptr=0;delete(thisNode);}
freeRTNDone();
delete(firstNode);
}
}
freeSDDone();
}
}
//conditioned_sample();
sleep(1);
}
} else {
// printf("got one old.");
idle=0;
idleCount=0;
cc_state=20;
getoldest(oldest, &thepdu);
oRp3=oRp2;
oRp2=oRp1;
oRp1=oldest->readPos;
cc_state=24;
arrivalTime =new ptime(thepdu);
sIcount++;
pTimeCount++;
if(basetime==0){
myTime=(time_t)thepdu.ts.tv_sec;
printf("myTime = %d ts.tv_sec= %d ",(int)myTime, (int)thepdu.ts.tv_sec);
ptm = *gmtime(&myTime);
printf("%s", timeStr);
ptm.tm_mday--;
ptm.tm_hour=0;
ptm.tm_min=0;
ptm.tm_sec=0;
strftime(timeStr, sizeof(timeStr), "%Y-%m-%d %H:%M:%S", &ptm);
myTime=mktime(&ptm);
basetime=(uint32_t)myTime;
youngest=basetime;
}

if(youngest<arrivalTime->sec()){
youngest=arrivalTime->sec();
}
src.s_addr=thepdu.ipsrc;
dst.s_addr=thepdu.ipdst;
strptr=inet_ntoa(src);strncpy(s1, strptr,17);
strptr=inet_ntoa(dst);strncpy(s2, strptr,17);
getFreeSD();
```

```

    if(firstNode!=0){
currNode=firstNode;
currNode->cmp4same(firstNode,the pdu,infile);
currNode=currNode->cmp4opp(firstNode,the pdu,infile);
fflush(infile);
}
freeSDDone();

    if(freeSDNodes==0){
getFreeSD();
if(firstNode==0){
firstNode=new streamDistinguisher(the pdu);
fprintf(infile,"new object is created ");
}
freeSDDone();
}
cc_state=26;
housekeeping++;

    if(housekeeping>100000) {
printf("Housekeeping: writePos = %d readPos =%d", oldest->writePos, oldest-
>readPos);fflush(stdout);
housekeeping=0;
}
}
}
pthread_exit(NULL);
}

    struct PProcessor* findOldest(void){
time_t min_sec =(unsigned long long int)0xFFFFFFFFFULL;
uint64_t min_psec=(unsigned long long int)0xFFFFFFFFFULL;
struct PProcessor* tmp=0;
struct PProcessor* minPPNode=0;
tmp=PPlist;
int PPcounter, freeCounter;
PPcounter=freeCounter=0;
if(oRp3==oRp2) {
printf("checking for the oldest pdu.");
fflush(stdout);
}
cc_state=100;
while(tmp!=NULL){
PPcounter++;
printf("CCthread: Checking %p - %lu.%012llu vs %lu.%012llu ", tmp, min_sec,
min_psec,tmp->buffer[tmp->readPos].myPdu.ts.tv_sec, tmp->buffer[tmp->readPos].myPdu.ts.tv_psec);
fflush(stdout);

    if(tmp->writePos==tmp->readPos){

```

```

printf(“writePos==readPos.”);
fflush(stdout);
cc_state=101;

    if(tmp->writeOK==-1){
printf(“WriteOK==-1, to be terminated.”);
fflush(stdout);
cc_state=102;
    }

    if(tmp->PPwriting==1){
printf(“PP writing at location, waiting for it to complete.”);
fflush(stdout);
cc_state=103;
while(tmp->PPwriting==1 && tmp->writePos==tmp->readPos){
// wait for them
printf(“tmp->PPwriting=1 and write==readPos: %d!” ,cc_state);
fflush(stdout);
cc_state++;
usleep(1);
}
cc_state=104;

    if(tmp->buffer[tmp->readPos].free>0){
cc_state=105;
if(minPPNode==0){
cc_state=106;
min_sec=tmp->buffer[tmp->readPos].myPdu.ts.tv_sec;
min_psec=tmp->buffer[tmp->readPos].myPdu.ts.tv_nsec;
minPPNode=tmp;
} else {
cc_state=107;

    if( tmp->buffer[tmp->readPos].myPdu.ts.tv_sec < min_sec ){
cc_state=108;
min_sec=tmp->buffer[tmp->readPos].myPdu.ts.tv_sec;
min_psec=tmp->buffer[tmp->readPos].myPdu.ts.tv_nsec;
minPPNode=tmp;
printf(“p3=p2:updating new oldest, %p ” ,minPPNode);fflush(stdout);
} else if ( ( tmp->buffer[tmp->readPos].myPdu.ts.tv_sec == min_sec ) &&
(tmp->buffer[tmp->readPos].myPdu.ts.tv_nsec<=min_psec) ){
cc_state=109;
min_sec=tmp->buffer[tmp->readPos].myPdu.ts.tv_sec;
min_psec=tmp->buffer[tmp->readPos].myPdu.ts.tv_nsec;
minPPNode=tmp;
printf(“p3=p2:updating new oldest, %p ” ,minPPNode);fflush(stdout);
}
}
} else {
freeCounter++;

```

```

}
} //if(tmp->PPwriting==1)
} else {
cc_state=110;
if(tmp->buffer[tmp->readPos].free>0){
cc_state=111;

    if(minPPNode==0){
min_sec=tmp->buffer[tmp->readPos].myPdu.ts.tv_sec;
min_psec=tmp->buffer[tmp->readPos].myPdu.ts.tv_psec;
minPPNode=tmp;
printf("(initial cmp): updating new oldest, %p: %lu.%012llu",minPPNode, min_sec,
min_psec);fflush(stdout);
} else {
cc_state=112;

    if( tmp->buffer[tmp->readPos].myPdu.ts.tv_sec < min_sec ){
cc_state=113;
min_sec=tmp->buffer[tmp->readPos].myPdu.ts.tv_sec;
min_psec=tmp->buffer[tmp->readPos].myPdu.ts.tv_psec;
minPPNode=tmp;
printf("(ts.tv_sec<min_sec):updating new oldest, %p: %lu.%012llu",minPPNode,
min_sec, min_psec);fflush(stdout);
} else if ( (tmp->buffer[tmp->readPos].myPdu.ts.tv_sec == min_sec) &&
(tmp->buffer[tmp->readPos].myPdu.ts.tv_psec<=min_psec) ){
cc_state=115;
min_sec=tmp->buffer[tmp->readPos].myPdu.ts.tv_sec;
min_psec=tmp->buffer[tmp->readPos].myPdu.ts.tv_psec;
minPPNode=tmp;
printf("(ts.tv_sec==min_sec,tv_psec<min_psec):updating new oldest, %p: %lu.%012llu",minPPNode,
min_sec,
min_psec);fflush(stdout);
} else {
cc_state=120;
}
}
} else {
freeCounter++;
}
}
tmp=tmp->next;
} // while(tmp->next!=NULL)
if(PPcounter==freeCounter) {
/* No data is available. */
printf("No data avail.");fflush(stdout);
return(NULL);
}
;fflush(stdout);
return minPPNode;
}

```

```
int checkCleanPPlist(void){
struct PProcessor* tmp, *moveMe;
int PPcounter;
tmp=PPlist;
PPcounter=0;
while(tmp!=NULL){
moveMe=0;

    if(tmp->writePos==tmp->readPos){
if(tmp->writeOK==-1) {
// This is probably a good time to kill this.. W
// WriteOk-1 means that the PP has stopped sending, write=read means that
we've read everything.
moveMe=tmp; // OK remove this.
PPcounter++;
}
} // if(tmp->writePos
tmp=tmp->next;
}
return(PPcounter); // Returns the number of PProcessors that are removal can-
didates.
}

void getoldest(struct PProcessor* thePP, struct pduinfo* myPDU){
memcpy(myPDU, &thePP->buffer[thePP->readPos].myPdu, sizeof(struct pduinfo));
thePP->buffer[thePP->readPos].free=0;
thePP->readPos++;
if(thePP->readPos>=thePP->bufferSize){
thePP->readPos=0;
}
return;
}

int cleanPPlist(void){
struct PProcessor* tmp, *tmp2, *moveMe;
int PPcounter;
tmp=PPlist;
PPcounter=0;
while(tmp!=NULL){
moveMe=0;
if(tmp->writePos==tmp->readPos){
printf("write==read, ");
if(tmp->writeOK==-1) {
moveMe=tmp; // OK remove this.
printf(" writeOk==-1 ");
} else {
printf(" writeOk==%d ", tmp->writeOK);
}
} printf("moveMe= %p ", moveMe);
tmp=tmp->next;
```

```

if(moveMe!=0){
printf("moveMe->terminate = %d", moveMe->terminate);
if(moveMe->terminate==0){
moveMe->terminate=1;
printf("setting moveMe->terminate=1");
} else {
printf("Checking ctrlSocket = %d (should be 0).",
moveMe->PPctrlSocket);
if (moveMe->PPctrlSocket==0){
printf("Waiting for DE & CD = %d && %d ", DEactive, CDactive);
while(DEactive==1 && CDactive ==1){
/* wait for ConnectionData and DE control to finish */
}
CCactive=1;
PPcounter++;
printf("Locating the pointer that points to moveMe.");
tmp2=PPlist;
while(tmp2->next!=moveMe && tmp2->next!=0){
tmp2=tmp2->next;
}
/* moveMe was first */
if(PPlist==moveMe){
PPlist=moveMe->next;
} else {
if(tmp2!=0){ /* moveMe was intermediate or last */
tmp2->next=moveMe->next;
} else {
// This should have been caught by the PPlist==moveMe
}
}
printf("Updated PPlist, moving to availPPlist.");
moveMe->next=0;
moveMe->terminate=0;
bzero(moveMe->buffer, BUFFERsize*sizeof(bufferContent));
if(availPPlist==0){
availPPlist=moveMe;
} else {
moveMe->next=availPPlist;
availPPlist=moveMe;
}
CCactive=0;
}
}
} // while(tmp->next!=NULL)
}
return(PPcounter); // Returns the number of PProcessors that were removed
from the PPlist.
}

```

```

void getAvailPP(void){

```

```
pthread_mutex_lock(&AvailPPs);
return;
}

void freeAvailPP(void){
pthread_mutex_unlock(&AvailPPs);
return;
}

void getActiveSD(void){
pthread_mutex_lock(&ActiveSD);
return;
}

void freeActiveSD(void){
pthread_mutex_unlock(&ActiveSD);
}

void getFreeSD(void){
pthread_mutex_lock(&FreeSDLock);
return;
}

void freeSDDone(void){
pthread_mutex_unlock(&FreeSDLock);
return;
}

void getFreeRTN(void){
pthread_mutex_lock(&FreeRTN);
return;
}

void freeRTNDone(){
pthread_mutex_unlock(&FreeRTN);
return;
}

void getFreeRTT(void){
pthread_mutex_lock(&FreeRTTLock);
return;
}

void freeRTTDone(void){
pthread_mutex_unlock(&FreeRTTLock);
return; }
```

Appendix K

StreamDistinguisher Class

```
#ifndef _SDclass
#define _SDclass
#include <arpa/inet.h>
#include <stdio.h>
#include "global_structures.h"
#include "RoundTripNode.h"

class streamDistinguisher{
public:
FILE* thisStream;
int Nodenum;
uint32_t ipsrc;
uint32_t ipdst;
uint8_t proto;
uint16_t portsrc;
uint16_t portdst;
uint32_t iniSeqNo;
uint32_t iniAckNo;
RoundTripNode *startptr;
streamDistinguisher *next;
qd_real lowerInterval,upperInterval;
qd_real rttSum,sumsquare,rttMean,rttVariance,min,max;
qd_real sampleCount;

    streamDistinguisher();
streamDistinguisher();
streamDistinguisher(struct pduinfo &thispdu);
streamDistinguisher* addNode(streamDistinguisher* thisNode, struct pduinfo
&thispdu, FILE* infile);
int cmpIP(uint32_t ips, uint32_t ipd, uint32_t s, uint32_t d);
int cmpTP(uint16_t tps, uint16_t tpd, uint16_t s, uint16_t d);
void cmp4same(streamDistinguisher* thisNode,struct pduinfo &thispdu,FILE*
infile);
streamDistinguisher* cmp4opp(streamDistinguisher* thisNode,struct pduinfo &this-
pdu, FILE* infile);
```

```
};  
#endif
```

Appendix L

StreamDistinguisher Implementation

```
#include <signal.h>
#include <sys/time.h>
#include <inttypes.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <stdio.h>
#include <iostream>
#include "global_structures.h"
#include "global_variables.h"
#include "streamDistinguisher.h"
#include "global_functions.h"
#include "RoundTripNode.h"

    streamDistinguisher::streamDistinguisher(){
return;
}

streamDistinguisher::streamDistinguisher(struct pduinfo &thispdu){
ipsrc=(thispdu.ipsrc);
ipdst=(thispdu.ipdst);
proto=thispdu.proto;
portsrc=thispdu.portsrc;
portdst=thispdu.portdst;
char* ptr;
in_addr src,dst;
char s1[17],s2[17],s3[50];
src.s_addr=thispdu.ipsrc;
dst.s_addr=thispdu.ipdst;
ptr=inet_ntoa(src);strncpy(s1,ptr,17);
ptr=inet_ntoa(dst);strncpy(s2,ptr,17);
sprintf(s3, "./output/%s_%s_%d_%d.txt",s1,s2,portsrc,portdst);
```

```

thisStream=fopen(s3, "w");
getFreeRTN();
startptr=new RoundTripNode(thispdu);
freeRTNDone();
next=0;
lowerInterval=(qd_real)(double)thispdu.ts.tv_sec;
upperInterval=(qd_real)(double)SamplePeriod;
upperInterval+=(qd_real)(double)thispdu.ts.tv_sec;
iniSeqNo=thispdu.tcpseq;
iniAckNo=thispdu.tcpack;
sampleCount=(qd_real)(double)0.0;
sumsquare=(qd_real)(double)0.0;
min=(qd_real)(double)0.0;
max=(qd_real)(double)0.0;
rttSum=(qd_real)(double)0.0;
rttMean=(qd_real)(double)0.0;
rttVariance=(qd_real)(double)0.0;
return;
}

```

```

streamDistinguisher:: streamDistinguisher(){
}

```

```

streamDistinguisher* streamDistinguisher::addNode(streamDistinguisher* thisNode,struct
pduinfo &thispdu, FILE* infile){
streamDistinguisher* sd;
sd=new streamDistinguisher(thispdu);
thisNode->next=sd;
thisNode=sd;
return thisNode;
}

```

```

void streamDistinguisher::cmp4same(streamDistinguisher* thisNode,struct pduinfo
&thispdu,FILE* infile){
streamDistinguisher* sd;
RoundTripNode* rtn,*rtnThisNode,*preNode;
char* ptr;
in_addr thissrc, thisdst,src,dst;
char s1[17],s2[17],s3[17],s4[17];
sd=thisNode;
infile=sd->thisStream;
uint8_t newproto;
uint16_t newportsrc;
uint16_t newportdst;
thissrc.s_addr=thispdu.ipsrc;
thisdst.s_addr=thispdu.ipdst;

```

```

src.s_addr=sd->ipsrc;
dst.s_addr=sd->ipdst;
bzero(s1,17);
bzero(s2,17);
bzero(s3,17);
bzero(s4,17);
ptr=inet_ntoa(thissrc);strncpy(s1, ptr,17);
ptr=inet_ntoa(thisdst);strncpy(s2, ptr,17);
ptr=inet_ntoa(src);strncpy(s3,ptr,17);
ptr=inet_ntoa(dst);strncpy(s4,ptr,17);
newproto=thispdu.proto;
newportsrc=thispdu.portsrc;
newportdst=thispdu.portdst;
if(sd->proto==newproto){

    if(((strcmp(s3,s1)==0)&&(strcmp(s4,s2)==0))&&((sd->portsrc==newportsrc)&&(sd-
>portdst==newportdst))){
getFreeRTN();
rtn=sd->startptr;
if(rtn->sent_tcpseq!=thispdu.tcpseq){
rtnThisNode=new RoundTripNode(sd->startptr,thispdu);
rtn->next=rtnThisNode;
sd->startptr=rtnThisNode;
}else{
while(rtn->previous!=0){
if(rtn->sent_tcpseq==thispdu.tcpseq){
rtn->sent_tcpseq=(thispdu.tcpseq);
rtn->sent_tcpack=(thispdu.tcpack);
rtn->sent_window=thispdu.window;
rtn->sentimesec=thispdu.ts.tv_sec;
rtn->sentimepsec=thispdu.ts.tv_nsec;
rtn->sent_len=thispdu.len;
if(rtn->sent_len==0){
rtn->tmp_tcpseq=rtn->sent_tcpseq+1;
}else{
rtn->tmp_tcpseq=rtn->sent_tcpseq+rtn->sent_len;
}
}
rtn=rtn->previous;
}if((rtn->previous==0)&&(rtn->sent_tcpseq==thispdu.tcpseq)){
rtn->sent_tcpseq=(thispdu.tcpseq);
rtn->sent_tcpack=(thispdu.tcpack);
rtn->sent_window=thispdu.window;
rtn->sentimesec=thispdu.ts.tv_sec;
rtn->sentimepsec=thispdu.ts.tv_nsec;
rtn->sent_len=thispdu.len;
if(rtn->sent_len==0){
rtn->tmp_tcpseq=rtn->sent_tcpseq+1;
}else{
rtn->tmp_tcpseq=rtn->sent_tcpseq+rtn->sent_len;
}
}
}
}

```

```

}
}
}
freeRTNDone();
fflush(infile);
return;
}else{
if(sd->next==0){
sd=addNode(sd, thispdu, infile);
fprintf(infile, "%p newnode ", sd);
fflush(infile);
return;
}else{
sd=sd->next;
cmp4same(sd, thispdu, infile);
fflush(infile);
}
fflush(infile);
return;
}
}
return;
}

```

```

streamDistinguisher* streamDistinguisher::cmp4opp(streamDistinguisher* thisNode, struct
pduinfo &thispdu, FILE* infile){
streamDistinguisher* sd;
char* ptr;
in_addr thissrc, thisdst, src, dst;
char s1[17], s2[17], s3[17], s4[17];
sd=thisNode;
infile=sd->thisStream;
uint8_t newproto;
uint16_t newportsrc;
uint16_t newportdst;
thissrc.s_addr=thispdu.ipsrc;
thisdst.s_addr=thispdu.ipdst;
src.s_addr=sd->ipsrc;
dst.s_addr=sd->ipdst;
bzero(s1,17);
bzero(s2,17);
bzero(s3,17);
bzero(s4,17);
ptr=inet_ntoa(thissrc);strncpy(s1, ptr,17);
ptr=inet_ntoa(thisdst);strncpy(s2, ptr,17);
ptr=inet_ntoa(src);strncpy(s3,ptr,17);
ptr=inet_ntoa(dst);strncpy(s4,ptr,17);
newproto=thispdu.proto;
newportsrc=thispdu.portsrc;

```

```
newportdst=thispdu.portdst;

    if(sd->proto==newproto){
if(((strcmp(s3,s2)==0)&&(strcmp(s4,s1)==0))&&((sd->portsrc==newportdst)&&(sd-
>portdst==newportsrc))){
getFreeRTN();
thisNode->startptr->cmpRTT(sd,sd->startptr,thispdu,infile);
freeRTNDone();
fflush(infile);
return sd;
}else{
if(sd->next==0){
return sd;
}else{
sd=sd->next;
sd=cmp4opp(sd,thispdu,infile);
fflush(infile);
}
fflush(infile);
return sd;
}
}
return sd;
}
```


Appendix M

RoundTripNode Class

```
#ifndef _RTNclass
#define _RTNclass
#include <arpa/inet.h>
#include "global_structures.h"
#include "streamDistinguisher.h"
#include <string.h>
#include <qd/qd_real.h>

    class streamDistinguisher;

    class RoundTripNode{
public:
RoundTripNode *next, *previous
; // int Nodenum;
uint32_t sent_ipsrc,sent_ipdst
; uint16_t sent_tpsrc,sent_tpdst;
uint32_t sent_tcpseq;
uint32_t sent_tcpack;
uint32_t tmp_tcpseq;
uint16_t sent_window;
uint16_t sent_len;
uint32_t sentimesec, rectimesec,rttsec;
uint64_t sentimepsec, rectimepsec,rttpsec;
qd_real sentime, rectime, rtt;
char sent_mpid[8];
char rec_mpid[8];
char sent_dag[8];
char rec_dag[8];

    RoundTripNode();
sinRoundTripNode();
RoundTripNode(RoundTripNode* rtn, struct pduinfo &newpdu);
RoundTripNode(struct pduinfo &sentpdu);

    void cmpRTT(streamDistinguisher* sd, RoundTripNode* rtn, struct pduinfo
```

```
&recpdu, FILE* infile);  
void computePDU(streamDistinguisher* sd, RoundTripNode* rttNode, struct pduinfo  
&thispud, FILE* infile);  
void deleteNode(streamDistinguisher* sd, RoundTripNode* rtn, FILE* infile);  
};  
#endif
```

Appendix N

RoundTripNode Implementation

```
labelRoundTripNode Implementation
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
#include <sys/time.h>
#include <inttypes.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <qd/qd_real.h>
#include <qd/qd_inline.h>
#include "global_structures.h"
#include "RoundTripNode.h"
#include "global_variables.h"
#include "global_functions.h"
#include "locationconfig.h"
#include <iostream>
#include <iomanip>
// #define PICODIVIDER (int)10
#define PICODIVIDER (double)1.e12
#define DIVIDE (double)1.e12

class RoundTripNode;
using namespace std;
struct RTT *rtt;

RoundTripNode::RoundTripNode(){
return;
```

```
}

RoundTripNode::sinRoundTripNode(){
}

RoundTripNode::RoundTripNode(struct pduinfo &sentpdu){
  bzero(&sent_mpid, 8);
  bzero(&sent_dag, 8);
  bzero(&rec_mpid, 8);
  bzero(&rec_dag, 8);
  strncpy(sent_mpid, sentpdu.mpid, 8);
  strncpy(sent_dag, sentpdu.nic, 8);
  sent_ipsrc=sentpdu.ipsrc;
  sent_ipdst=sentpdu.ipdst;
  sent_tpsrc=sentpdu.portsrc;
  sent_tpdst=sentpdu.portdst;
  sent_tcpseq=(sentpdu.tcpseq);
  sent_tcpack=(sentpdu.tcpack);
  sent_window=sentpdu.window;
  sentimesec=sentpdu.ts.tv_sec;
  sentimepsec=sentpdu.ts.tv_nsec;
  sent_len=sentpdu.len;
  if(sent_len==0){
    tmp_tcpseq=sent_tcpseq+1;
  }else{
    tmp_tcpseq=sent_tcpseq+sent_len;
  }
  rectimesec=0;
  rectimepsec=0;
  rttsec=0;
  rttpsec=0;
  next=0;
  previous=0;
  return;
}

RoundTripNode::RoundTripNode(RoundTripNode* rtn, struct pduinfo &new-
pdu){
  bzero(&sent_mpid, 8);
  bzero(&sent_dag, 8);
  bzero(&rec_mpid, 8);
  bzero(&rec_dag, 8);
  strncpy(sent_mpid, newpdu.mpid,8);
  strncpy(sent_dag, newpdu.nic, 8);
  sent_ipsrc=newpdu.ipsrc;
  sent_ipdst=newpdu.ipdst;
  sent_tpsrc=newpdu.portsrc;
  sent_tpdst=newpdu.portdst;
```

```

sent_tcpack=(newpdu.tcpack);
sent_tcpseq=(newpdu.tcpseq);
sent_window=(newpdu.window);
sent_len=newpdu.len;
tmp_tcpseq=sent_tcpseq+sent_len;
sentimesec=newpdu.ts.tv_sec;
sentimepsec=newpdu.ts.tv_nsec;
rectimesec=0;
rectimepsec=0;
rttsec=0;
rttpecc=0;
next=0;
previous=rtn;
return;
}

```

```

void RoundTripNode::cmpRTT(streamDistinguisher* sd, RoundTripNode* rtn, struct
pduinfo &recpdu, FILE* infile){
char* ptr;
in_addr src, dst,r1,r2;
char source[17],destination[17],s1[17],s2[17];
src.s_addr=rtn->sent_ipsrc;
dst.s_addr=rtn->sent_ipdst;
r1.s_addr=recpdu.ipsrc;
r2.s_addr=recpdu.ipdst;
ptr=inet_ntoa(src);strncpy(source, ptr,17);
ptr=inet_ntoa(dst);strncpy(destination, ptr,17);
ptr=inet_ntoa(r1);strncpy(s1,ptr,17);
ptr=inet_ntoa(r2);strncpy(s2,ptr,17);
fflush(infile);
if((rtn->rectimesec==0)&&(rtn->tmp_tcpseq==recpdu.tcpack)&&(strncmp(recpdu.nic,rtn-
>sent_dag,4)==0)){//
computePDU(sd, rtn, recpdu, infile);
fflush(infile);
return;
}else{
if(rtn->previous==0){
if((rtn->rectimesec==0)&&(rtn->tmp_tcpseq==recpdu.tcpack)&&(strncmp(recpdu.nic,rtn-
>sent_dag,4)==0)){//
computePDU(sd, rtn, recpdu, infile);
return;
}else{return;}
}else{
rtn=rtn->previous;
cmpRTT(sd,rtn,recpdu,infile);
}
}
}
}

```

```

void RoundTripNode::computePDU(streamDistinguisher* sd, RoundTripNode*
rtn, struct pduinfo &thispdu,FILE* infile){
RoundTripNode* rttNode;
rttNode=rtn;
double a=1.0;
in_addr src,dst;
uint32_t seqno=rtn->sent_tcpseq;
char s1[17],s2[17];
char* ptr;
char mean[11],variance[11],min[11],max[11],rtt[11],recv_time[20];
struct RTT* tmp_rtt;
// struct rttllok* tmp_rl;
bzero(rtt,10);
bzero(s1,17);
bzero(s2,17);
qd_real squareMean;
qd_real squareSampleCount;
src.s_addr=rttNode->sent_ipsrc;ptr=inet_ntoa(src);strncpy(s1,ptr,17);
dst.s_addr=rttNode->sent_ipdst;ptr=inet_ntoa(dst);strncpy(s2,ptr,17);
rttNode->sentime=(qd_real)(double)rttNode->sentimesec+(qd_real)(double)(rttNode-
>sentimepsec/PICODIVIDER);
rttNode->rectime=(qd_real)(double)thispdu.ts.tv_sec+(qd_real)(double)(thispdu.ts.tv_psec/(PICODIVI
rttNode->rtt=(rttNode->rectime>rttNode->sentime)?(rttNode->rectime):rttNode-
>sentime;
rttNode->rtt=(rttNode->rectime>rttNode->sentime)?rttNode->sentime:rttNode-
>rectime;
cout<<setiosflags(ios::fixed);
cout.precision(18);
strncpy(rttNode->rec_mpid, thispdu.mpid,8);
strncpy(rttNode->rec_dag, thispdu.nic, 8);
rttNode->rectimesec=thispdu.ts.tv_sec;
rttNode->rectimepsec=thispdu.ts.tv_psec;
(rttNode->rttsec)=(rttNode->rectimesec)-(rttNode->sentimesec);
(rttNode->rttpsec)=(rttNode->rectimepsec)-(rttNode->sentimepsec);

if((sd->lowerInterval==rttNode->rectimesec)&&(rttNode->rectimesec+1==sd-
>upperInterval)){
if(sd->max<rttNode->rtt){sd->max=rttNode->rtt;}
if(sd->min>rttNode->rtt){sd->min=rttNode->rtt;}
sd->sampleCount+=1;
sd->rttSum+=rttNode->rtt;
sd->sumsquare+=(qd_real)(to_double(rttNode->rtt)*(to_double(rttNode->rtt)));
sd->rttMean=(qd_real)(to_double(sd->rttSum)/to_double(sd->sampleCount));
squareMean=(qd_real)(to_double(sd->rttMean)*to_double(sd->rttMean));
squareSampleCount=(qd_real)(to_double(sd->sampleCount)*to_double(sd->sampleCount));
sd->rttVariance=((to_double(sd->sumsquare)/to_double(sd->sampleCount))-(to_double(squareMean)));

printf(mean, "%1.7lf", to_double(sd->rttMean));
printf(variance, "%1.7lf", to_double(sd->rttVariance));

```

```

sprintf(min, "%1.7lf", to_double(sd->min));
sprintf(max, "%1.7lf", to_double(sd->max));
sprintf(rtt, "%1.7lf", to_double(rttNode->rtt));
sprintf(recv_time, "%12.6lf", to_double(rttNode->rectime));
pthread_mutex_lock(&FreeRTTLock);
if(firstRTT==0){
firstRTT=new RTT;
writepos=1;
bzero(firstRTT->ipsrc,17);bzero(firstRTT->ipdst,17);
strncpy(firstRTT->ipsrc,s1,17);strncpy(firstRTT->ipdst,s2,17);
firstRTT->portsrc=rttNode->sent_tpsrc;
firstRTT->portdst=rttNode->sent_tpdst;
firstRTT->upperInterval=to_double(sd->upperInterval);
firstRTT->lowerInterval=to_double(sd->lowerInterval);
firstRTT->tcpseq=seqno;firstRTT->tcpack=thispdu.tcpack;
bzero(firstRTT->sent_mpid,8);bzero(firstRTT->rec_mpid,8);
bzero(firstRTT->sent_dag,8);bzero(firstRTT->rec_dag,8);
bzero(firstRTT->mean,10);bzero(firstRTT->variance,10);
bzero(firstRTT->min,10);bzero(firstRTT->max,10);
bzero(firstRTT->recv_time,20);bzero(firstRTT->rtt,10);
memcpy(firstRTT->sent_mpid,rttNode->sent_mpid,8);
memcpy(firstRTT->sent_dag,rttNode->sent_dag,8);
memcpy(firstRTT->rec_mpid,rttNode->rec_mpid,8);
memcpy(firstRTT->rec_dag,rttNode->rec_dag,8);
strncpy(firstRTT->mean,mean,10);
strncpy(firstRTT->variance,variance,10);
strncpy(firstRTT->min,min,10);
strncpy(firstRTT->max,max,10);
strncpy(firstRTT->rtt,rtt,10);
strncpy(firstRTT->recv_time,recv_time,20);
firstRTT->count=1;
firstRTT->previous=0;
firstRTT->next=0;
lastRTT=firstRTT;
tmp_rtt=firstRTT;
fprintf(infile, "[%d]", tmp_rtt->count);
fprintf(infile, "%s:%d%s:%d%u%s", s1, rttNode->sent_tpsrc, s2, rttNode->sent_tpdst, seqno, (rtt));
fflush(infile);
}
else if((firstRTT!=0)&&(lastRTT!=0)){
tmp_rtt=new RTT;
writepos++;
tmp_rtt->previous=lastRTT;
bzero(tmp_rtt->ipsrc,17);bzero(tmp_rtt->ipdst,17);
strncpy(tmp_rtt->ipsrc,s1,17);strncpy(tmp_rtt->ipdst,s2,17);
tmp_rtt->portsrc=rttNode->sent_tpsrc;
tmp_rtt->portdst=rttNode->sent_tpdst;
tmp_rtt->upperInterval=to_double(sd->upperInterval);
tmp_rtt->lowerInterval=to_double(sd->lowerInterval);
tmp_rtt->tcpseq=seqno;tmp_rtt->tcpack=thispdu.tcpack;

```

```

bzero(tmp_rtt->sent_mpid,8);bzero(tmp_rtt->sent_dag,8);
bzero(tmp_rtt->rec_mpid,8);bzero(tmp_rtt->rec_dag,8);
bzero(tmp_rtt->mean,10);bzero(tmp_rtt->variance,10);
bzero(tmp_rtt->min,10);bzero(tmp_rtt->max,10);
bzero(tmp_rtt->recv_time,20);bzero(tmp_rtt->rtt,10);
memcpy(tmp_rtt->sent_mpid,rttNode->sent_mpid,8);
memcpy(tmp_rtt->sent_dag,rttNode->sent_dag,8);
memcpy(tmp_rtt->rec_mpid,rttNode->rec_mpid,8);
memcpy(tmp_rtt->rec_dag,rttNode->rec_dag,8);
strncpy(tmp_rtt->mean,mean,10);
strncpy(tmp_rtt->variance,variance,10);
strncpy(tmp_rtt->min,min,10);
strncpy(tmp_rtt->max,max,10);
strncpy(tmp_rtt->rtt,rtt,10);
strncpy(tmp_rtt->recv_time,recv_time,20);
tmp_rtt->next=0;
tmp_rtt->count=lastRTT->count+1;
lastRTT->next=tmp_rtt;
lastRTT=tmp_rtt;
fprintf(infile, "[%d]", tmp_rtt->count);
fprintf(infile, "%s:%d%s:%d%u%s", s1, rttNode->sent_tpsrc, s2, rttNode->sent_tpdst, seqno, (rtt));
fflush(infile);
}
pthread_mutex_unlock(&FreeRTTLock);
} else if ((rttNode->rectimesec >= sd->upperInterval)) {
sd->lowerInterval = rttNode->rectimesec;
sd->upperInterval = sd->lowerInterval + SamplePeriod;
sd->sampleCount = a;
sd->rttSum = rttNode->rtt;
sd->sumsquare = (qd_real)((to_double(rttNode->rtt))*(to_double(rttNode->rtt)));
sd->rttMean = (qd_real)(to_double(sd->rttSum)/to_double(sd->sampleCount));
squareMean = (qd_real)(to_double(sd->rttMean)*to_double(sd->rttMean));
sd->rttVariance = (to_double(sd->sumsquare)/to_double(sd->sampleCount)-(to_double(squareMean)));
sd->max = rttNode->rtt;
sd->min = rttNode->rtt;
sprintf(mean, "%1.7lf", to_double(sd->rttMean));
sprintf(variance, "%1.7lf", to_double(sd->rttVariance));
sprintf(min, "%1.7lf", to_double(sd->min));
sprintf(max, "%1.7lf", to_double(sd->max));
sprintf(rtt, "%1.7lf", to_double(rttNode->rtt));
sprintf(recv_time, "%12.6lf", to_double(rttNode->rectime));
pthread_mutex_lock(&FreeRTTLock);
tmp_rtt = new RTT;
writepos++;
tmp_rtt->previous = lastRTT;
bzero(tmp_rtt->ipsrc, 17); bzero(tmp_rtt->ipdst, 17);
strncpy(tmp_rtt->ipsrc, s1, 17); strncpy(tmp_rtt->ipdst, s2, 17);
tmp_rtt->portsrc = rttNode->sent_tpsrc; tmp_rtt->portdst = rttNode->sent_tpdst;
tmp_rtt->upperInterval = to_double(sd->upperInterval); tmp_rtt->lowerInterval = to_double(sd->lowerInterval);

```

```

tmp_rtt->tcpseq=seqno;tmp_rtt->tcpack=thispdu.tcpack;
bzero(tmp_rtt->sent_mpid,8);bzero(tmp_rtt->sent_dag,8);
bzero(tmp_rtt->rec_mpid,8);bzero(tmp_rtt->rec_dag,8);
bzero(tmp_rtt->mean,10);bzero(tmp_rtt->variance,10);
bzero(tmp_rtt->min,10);bzero(tmp_rtt->max,10);
bzero(tmp_rtt->recv_time,20);bzero(tmp_rtt->rtt,10);
memcpy(tmp_rtt->sent_mpid,rttNode->sent_mpid,8);
memcpy(tmp_rtt->sent_dag,rttNode->sent_dag,8);
memcpy(tmp_rtt->rec_mpid,rttNode->rec_mpid,8);
memcpy(tmp_rtt->rec_dag,rttNode->rec_dag,8);
strncpy(tmp_rtt->mean,mean,10);
strncpy(tmp_rtt->variance,variance,10);
strncpy(tmp_rtt->min,min,10);
strncpy(tmp_rtt->max,max,10);
strncpy(tmp_rtt->rtt,rtt,10);
strncpy(tmp_rtt->recv_time,recv_time,20);
tmp_rtt->next=0;
tmp_rtt->count=lastRTT->count+1;
lastRTT->next=tmp_rtt;
lastRTT=tmp_rtt;
fprintf(infile, "[%d]", tmp_rtt->count);
fprintf(infile, "%s:%d%s:%d%u%s", s1, rttNode->sent_tpsrc, s2, rttNode->sent_tpdst, seqno, (rtt));
fflush(infile);
pthread_mutex_unlock(&FreeRTTLock);
}
deleteNode(sd, rttNode, infile);
fflush(infile);
return;
}

```

```

void RoundTripNode::deleteNode(streamDistinguisher* sd, RoundTripNode* rtn,
FILE* infile){
RoundTripNode* tmpNode,*preNode,*thisNode;
tmpNode=rtn;
if((tmpNode->previous==0)&&(tmpNode->next==0)){
return;
}else if((tmpNode->previous!=0)&&(tmpNode->next!=0)){
preNode=tmpNode;
while(preNode->previous!=0){preNode=preNode->previous;}
if(preNode->previous==0){
while(preNode==tmpNode){
thisNode=preNode->next;
preNode->previous=0;
delete(thisNode);
}
}
}else if((tmpNode->previous!=0)&&(tmpNode->next==0)){
preNode=tmpNode;
while(preNode->previous!=0){preNode=preNode->previous;}
}

```

```
if(preNode->previous==0){
while(preNode==tmpNode){
thisNode=preNode->next;
preNode->previous=0;
delete(thisNode);
}
}
sd->startptr=tmpNode; return;
}else if((tmpNode->previous==0)&&(tmpNode->next!=0)){
return;
}
return;
}
```

Appendix O

Globals Of Dataevaluator

O.1 Globals

```
/* Globals */ #ifndef GLOBALSDE
#define GLOBALSDE
#include "streamDistinguisher.h"
#include "RoundTripNode.h"
#include <qd/qd_real.h>
/* First we start with defines, then structures, then variables */

/* The number of PDU that the DHandler->Calculator buffer should be able
to hold. */
#define BUFFERsize 1500000
#define defaultDPMISILENTTIME 300

    struct pt{
time_t tv_sec;
uint64_t tv_psec;
} __attribute__((packed));
typedef struct pt tpico;

    struct tcp_flags{
unsigned char FIN;
unsigned char SYN;
unsigned char RST;
unsigned char PUSH;
unsigned char ACK;
unsigned char URG;
}__attribute__((packed));
typedef struct tcp_flags flag;

    struct pduinfo{
char mpid[8];
char nic[8];
uint16_t caplen;
```

```

uint16_t len;
uint32_t ipdst;
uint32_t ipsrc;
uint16_t proto
; uint16_t portsrc;
uint16_t portdst;
uint32_t tcpseq;
uint32_t tpack;
uint16_t window;
tpico ts;
flag flags;
} __attribute__((packed));

    struct RTT{
char ipsrc[17],ipdst[17];
uint16_t portsrc,portdst;
uint32_t lowerInterval;
uint32_t upperInterval;
uint32_t tcpseq,tpack;
char sent_mpid[8],rec_mpid[8];
char sent_dag[8],rec_dag[8];
int count;
char mean[10], variance[10], min[10], max[10],rtt[10],recv_time[18];
RTT *previous,*next;
}__attribute__((packed));

    typedef struct rttllock{
struct RTT* first;
pthread_mutex_t mutex;
}rttllok;

    struct ctrlmsg{
int msgtype;
char data[36];
}__attribute__((packed));

    struct sampletype{
int msgtype;
uint32_t type;
uint32_t nopdus;
uint32_t batchsize;
uint32_t starttime;
char padding[20];
} __attribute__((packed));

    struct identification{
int msgtype;
int type;
char padding[32];
} __attribute__((packed));

```

```
    struct synchmessage{
int msgtype;
struct timeval DEtime;
struct timeval PParrTime;
struct timeval PPsendTime;
char padding[32];
} __attribute__((packed));
```

```
    struct bufferContent{
int free;
struct pduinfo myPdu;
};
```

```
    struct dataHandler{
int listenSocket;
int port;
};
```

```
    struct controlHandler{
int listenSocket;
int port;
};
```

```
    struct matlabHandler{
int listenSocket;
int port;
};
```

```
    struct ctrlInitMsg{
int sockid;
int dataPort;
}__attribute__((packed));
```

```
    struct PPfilter{
int index;
uint32_t IPsrc;
uint32_t IPsrcMask;
uint32_t IPdst;
uint32_t IPdstMask;
uint16_t IP_PROTO;
uint16_t SrcPort;
uint16_t SrcPortMask;
uint16_t DstPort;
uint16_t DstPortMask;
} __attribute__((packed));
```

```
    struct PPfilterMod{
int msgtype;
int position;
```

```
struct PPfilter theFilter;
char padding[2];
}_attribute((packed));

    struct PPstatus{
int input;
int filtered;
int sent;
int avgInput;
int avgFiltered;
int avgSent;
int dpmiTimeoutCount;
int dpmiSilenceTime;
int maxInputRate;
int maxFilterRate;
int maxSentRate;
int collecting;
int samplingMethod;
int NoPdus;
int BatchSize;
int StartTime;
int identificationMethod;
struct PPfilter theFilter[10];
};

    struct PProcessor{
struct PProcessor* next;
/* Used for identification; IPAddress of connector, port and ParentID of the calling process*/
uint32_t IP;
unsigned short port;
int ppid;
pthread_t thread_id;
struct bufferContent* buffer;
int bufferSize;
int PPwriting;
int writePos;
int readPos;
int writeOK;
int terminate;
int PPdataSocket;
int PPctrlSocket;
int ctrlPort;
int statusCount;
struct PPstatus lastStatus;
};

#endif
```

O.2 Global Variables

```
/* The global variables */
#include "RoundTripNode.h"
#include "RoundTripTime.h"
#include <string>
#ifdef CWDDEBUG
pthread_t cout_mutex=PTHREAD_MUTEX_INITIALIZER;
#endif

    // #define _DEBUG
#ifdef _DEBUG
#define _DEBUG_MSG(x) (x);fflush(stdout);
#else
#define _DEBUG_MSG(x)
#endif

    // #define _DEBUG_SS
#ifdef _DEBUG_SS
#define _DEBUGSS_MSG(x) (x);fflush(stdout);
#else
#define _DEBUGSS_MSG(x)
#endif
// #define _DEBUG_CC
#ifdef _DEBUG_CC
#define _DEBUG_MSGCC(x) (x);fflush(stdout);
#else
#define _DEBUG_MSGCC(x)
#endif

#define BufferSize_StreamDistinguisher 5
#define BufferSize_RoundTripNode 10
#define SamplePeriod 1
extern pthread_t dataThread, controlThread, calcThread, sampleThread, out-
putThread;
#ifdef CWDDEBUG
extern pthread_mutex_t cout_mutex=PTHREAD_MUTEX_INITIALIZER;
#endif
extern struct PProcessor* PPlist;
extern struct PProcessor* availPPlist;
extern streamDistinguisher* freeSDNodes;
extern streamDistinguisher* firstNode;
extern streamDistinguisher* currNode;
extern RoundTripNode* startNode;
extern int readpos, writepos;

extern struct RTT *firstRTT,*lastRTT;
extern int noSN, noIN, noLN, noBDN;
extern int NoPPs;
extern int terminate; /* Set to 1 to terminate all threads !*/
```

```
extern int myShutdown; /* Shutdown signal, nicer form of termination, gives
tools time to clean up*/
extern int threadCount;
extern int IDLETIME;
extern int sIcount;
extern int pTimeCount;
extern int collecting;
extern int sampletype;
extern int nopdus;
extern int batchsize;
extern uint32_t starttime;
extern int identificationmethod;
extern struct PFilterMod theFilters[10];
extern int DPMISILENTTIME;
extern uint32_t basetime; // Arrivaltime of the first PDU to the system
extern uint32_t youngest; // Arrivaltime of the youngest (newest) PDU to the
system.
extern uint32_t housekeepingCounter; // Counter that keeps track on the n# of
housekeeping operations.
extern double samplefreq; // Sample frequency.
extern int DONOTSAMPLE;

/* MAIN_IRQ_EXEC
Set to 1 by Main IRQ when executed.
calculator.cpp sets it to 0 when removing PPs, as
the MAIN_IRQ needs to execute once inbetween checks.
*/

extern int MAIN_IRQ_EXEC;

/* Semaphores for the PPlist */
extern int CDactive, DEactive, CCactive;
// CCsample=1 if Calculator(CC) is accessing a sample list
// CCfree=1 if CC is accessing a free list
extern int CCsample, CCfree, SSsample, SSfree;

// Locks using pthreads.. The preferred way.
extern pthread_mutex_t ActiveSD;
extern pthread_mutex_t FreeSDLock;
extern pthread_mutex_t FreeINLock, SampleINLock;
extern pthread_mutex_t FreeLNLock, SampleLNLock;
extern pthread_mutex_t FreeRTTLock;
extern pthread_mutex_t AvailPPs;
extern pthread_mutex_t FreeSDLock;
extern pthread_mutex_t FreeRTN;
extern pthread_mutex_t RTTLock;
extern struct PProcessor* DEobject;
extern std::string host, database, table_del, table_bps, user, password;
extern std::string FILENAME;
// Where do we send the output.
```

```
// 0=No output, 1-files, 2-Database, 3-file&database
extern int OUT_DESTINATIONS;
extern int fileNumber_del, fileNumber_bps, dataCount, rPos_del, rPos_bps, wPos_del,
wPos_bps;
```

O.3 Global Functions

```
/* Global functions */

/* Threads definitions */
void* dataConnection(void*);
void* controlConnection(void*);
void* dataHandlerThread(void *);
void* control(void*);
void* calculator(void*);
void* sample_thread(void*);
void* output_thread(void*);
void* mathread(void*);

void CC_IRQ(int);
void MAIN_IRQ(int);

void getFreeSD(void);
void freeSDDone(void);
void getSampleSN(void);
void sampleSNDone(void);
void getActiveSD(void);
void freeActiveSD(void);
void getFreeIN(void);
void freeINDone(void);
void getSampleIN(void);
void sampleINDone(void);
void getFreeRTT(void);
void freeRTTDone(void);
void getFreeRTN(void);
void freeRTNDone(void);
void getAvailPP(void);
void freeAvailPP(void);
```

O.4 Global Structures

```
/* Globals */
#ifdef GLOBALSDE
#define GLOBALSDE

#include "streamDistinguisher.h"
```

```
#include "RoundTripNode.h"
#include <qd/qd_real.h>
/* First we start with defines, then structures, then variables */

/* The number of PDU that the DHandler->Calculator buffer should be able
to hold. */
#define BUFFERsize 1500000
#define defaultDPMISILENTTIME 300

struct pt{
time_t tv_sec;
uint64_t tv_nsec;
} __attribute__((packed));
typedef struct pt tpico;

struct tcp_flags{
unsigned char FIN;
unsigned char SYN;
unsigned char RST;
unsigned char PUSH;
unsigned char ACK;
unsigned char URG;
}__attribute__((packed));
typedef struct tcp_flags flag;

struct pduinfo{
char mpid[8];
char nic[8];
uint16_t caplen;
uint16_t len;
uint32_t ipdst;
uint32_t ipsrc;
uint16_t proto;
uint16_t portsrc;
uint16_t portdst;
uint32_t tcpseq;
uint32_t tcpack;
uint16_t window;
tpico ts;
flag flags;
// unsigned char identification[20];
} __attribute__((packed));

struct RTT{
char ipsrc[17],ipdst[17];
uint16_t portsrc,portdst;
uint32_t lowerInterval;
uint32_t upperInterval;
uint32_t tcpseq,tcpack;
char sent_mpid[8],rec_mpid[8];
```

```
char sent_dag[8],rec_dag[8];
int count;
char mean[10], variance[10], min[10], max[10],rtt[10],recv_time[18];
RTT *previous,*next;
}__attribute__((packed));

    typedef struct rttllock{
struct RTT* first;
pthread_mutex_t mutex;
}rttllok;

    struct ctrlmsg{
int msgtype;
char data[36];
}__attribute__((packed));

    struct sampletype{
int msgtype;
uint32_t type;
uint32_t nopdus;
uint32_t batchsize;
uint32_t starttime;
char padding[20];
} __attribute__((packed));

    struct identification{
int msgtype;
int type;
char padding[32];
} __attribute__((packed));

    struct synchmessage{
int msgtype;
struct timeval DEtime;
struct timeval PParrTime;
struct timeval PPsendTime;
char padding[32];
} __attribute__((packed));

    struct bufferContent{
int free;
struct pduinfo myPdu;
};

    struct dataHandler{
int listenSocket;
int port;
};

    struct controlHandler{
```

```
int listenSocket;
int port;
};

    struct matlabHandler{
int listenSocket;
int port;
};

    struct ctrlInitMsg{
int sockid;
int dataPort;
}_attribute__((packed));

    struct PPfilter{
int index;
uint32_t IPsrc;
uint32_t IPsrcMask;
uint32_t IPdst;
uint32_t IPdstMask;
uint16_t IP_PROTO;
uint16_t SrcPort;
uint16_t SrcPortMask;
uint16_t DstPort;
uint16_t DstPortMask;
}_attribute__((packed));

    struct PPfilterMod{
int msgtype;
int position;
struct PPfilter theFilter;
char padding[2];
}_attribute__((packed));

    struct PPstatus{
int input;
int filtered;
int sent;
int avgInput;
int avgFiltered;
int avgSent;
int dpmiTimeoutCount;
int dpmiSilenceTime;
int maxInputRate;
int maxFilterRate;
int maxSentRate;
int collecting;
int samplingMethod;
int NoPdus;
int BatchSize;
```

```
int StartTime;
int identificationMethod;
struct PPfilter theFilter[10];
};

    struct PProcessor{
struct PProcessor* next;
/* Used for identification; IPAddress of connector, port and ParentID of the call-
ing process*/
uint32_t IP;
unsigned short port;
int ppid;
pthread_t thread_id;
struct bufferContent* buffer;
int bufferSize;
int PPwriting;
int writePos;
int readPos;
int writeOK;
int terminate;
int PPdataSocket;
int PPctrlSocket;
int ctrlPort;
int statusCount;
struct PPstatus lastStatus;
};

    #endif
```


Bibliography

- [1] Telecommunication Networks Group, <http://tstat.tlc.polito.it/measure.shtml#histo> (September 2009).
- [2] Richard Stevens M., TCP/IP Illustrated 3rd Edition, ISBN:0201633469.
- [3] Naylor W., Opderbeck H., Mean Round-Trip Times in the ARPANET *IETF RFC 619*, 1974.
- [4] Almes G., Kaldindi S., Zekauskas M., A Round-trip Delay Metric for IPPM *IETF RFC 2681*, September 1999.
- [5] Internetworking Research Group, Ohio University, <http://jarok.cs.ohiou.edu/software/tcptrace/#histo> (September 2009).
- [6] Mathew G. Naugle., Illustrated TCP/IP, Wiley Computer Publishing, ISBN:0471196568, Pub Date: 11/01/98.
- [7] Transmission control Protocol, DARPA INTERNET PROGRAM, *IETF RFC 793*, September 1981.
- [8] Yolanda Tsang, Mehmet Yildiz, Paul Barford, Rober Nowak, Network Radar: Tomography From Round Trip Time Measurements.
- [9] Arutur Ziviani, Encyclopedia of Internet Technologies And Applications, Internet Measurements. ISBN: 978-1-59140-993-9.
- [10] Arlos P., Fiedler M. and Nilsson A., A Distributed Passive Measurement Infrastructure.
- [11] Internet Protocol, *IETF RFC 791*, September 1981.
- [12] Mathis M., Mahdavi J., Floyd S. and Romanow A., TCP Selective Acknowledgement Options, *IETF RFC 2018* October 1996.

