Undergraduate Project Report
2016/17

Identifying TCP Loss and Reordering in Networks

Name:	Weiqiu Liu
Programme:	Telecoms
Class:	2013215108
QM Student No.	130801337
BUPT Student No.	2013213152

Date 15/05/2017

Table of Contents

Abstract

With the development of modern network communications, the quality of network service become more and more important. Loss rate is a key criterion for measuring the status of networks. Many software can be used to classify the retransmission and reordering packet, such as Wireshark, but few of them could identify the loss. This project develops a python program that can read and split a passive network trace file to analyze the TCP loss rate in each flow. Two algorithms are specified and implemented to identify the TCP loss and reordering packets.

Based on the requirement of analyzing large-scale files, this program would dump each split flow out to the disk and load them while processing. In this project, one large-scale file from CRAWDAD websites is used to test the efficiency. Traffic Control module in Linux is used to test the accuracy of loss rate.

Users can run this program in Linux terminal with the Libpcap library by inputting the passive network trace file. Then they can discover which packet is lost and what is the loss rate in each flow by searching the output log file.

摘要 (Chinese translation of the Abstract)

随着当今网络通信的发展，网络服务的质量也变的越来越重要。丢包率是一个衡量网络状态的重要指标。虽然许多软件可以区别重传包和乱序包，比如 Wireshark，但这其中几乎没有可以识别丢包的软件。本项目将开发一个 Python 程序来读取、分解一个静态网络抓包文件并在每个网络流中分析 TCP 丢包率。两个算法将被定义和实现来识别 TCP 丢包和乱序。

基于分析大型文件的需求，本程序会将每个分解后的流写入硬盘并在处理它们时加载至内存。本项目使用来自 CRAWDAD 网站的大型文件来测试运行效率，使用 Linux 流量控制模块来测试丢包率结果准确性。

用户可以在 Linux 终端和 Libpcap 库环境中，输入一个静态网络抓包文件并运行本程序，然后通过程序输出的日志文件查看丢包情况和每个流的丢包率。

Chapter 1: Introduction

Many applications use TCP to achieve their requirements to send message with other terminals. TCP traffic composes the majority of network traffic in the unit of flows. Although technologies and services using new application layer protocols is booming in the Internet, TCP remains the dominant protocol in transport layer. Therefore, the efficiency and dependability of TCP is vital important.

TCP loss recovery is a complex algorithm that ensure the efficiency and reliability. Network congestion leads to the packet dropped (loss) [1]. When packet loss occurred in congestion, reliable delivery protocol TCP would attempt to make up for the packet gap by using loss recovery algorithms to retransmit the loss packet. The retransmission of a packet can be a tool to infer the loss packet but it is far from enough.

Many existing TCP analysers of software can identify the retransmission packets and reordering packets from an individual flow, such as “tcprs” and “Wireshark”. Few tools focus on the loss packet and loss rate in flows. This research firstly did a literature survey focused on difficulties in identifying TCP loss, then developed and implemented the algorithms of TCP loss analyser named “tcpla” and tests efficiency and accuracy of this tool in large scale trace file. The next sections in this part will discuss TCP algorithms, network reordering, network loss and the research goals.

1.1 Transmission Control Protocol

Transmission Control Protocol (TCP) is a connection-oriented, reliable and stream-based transmission layer communication protocol, defined in IETF RFC 793[2]. TCP runs two different algorithms, Loss Recovery and Congestion Control, to ensure server or client response to packet loss correctly and efficiently.

Congestion control ensure the packets in a specified connection to avoid congestion. Congestion control uses several different algorithms such as slow-start and Additive Increase Multiplicative Decrease (AIMD). Slow-start begins initially with a congestion window size (cwnd) of 1, 2 or 10.[3] The value of the Congestion Window will be increased by one with each acknowledgement (ACK) received, effectively doubling the window size each round-trip time. AIMD approach is to increase the transmission rate (window size), probing for usable bandwidth, until loss occurs. The policy of additive increase may, for instance, increase the congestion window by a fixed amount every round trip time. When congestion is detected, the transmitter decreases the transmission rate by a multiplicative factor. A growing cwnd size allows server or client to transport an increasing number of packets unless congestion happened, finally arriving at the stable status. Congestion Control decreases the possibility of congestion happening but it still may occur. Congestion happens when network resources are filled over the queues limitation so that packets lost[4].

Loss recovery control the reaction of TCP packet loss. Loss Recovery influences the efficiency of Transmission Control Protocol directly. If TCP consumes much time on recovering from the loss, network delay would be longer but throughput would be dropped. If the recovery strategy are too stressful, it would lead to spurious retransmissions of packets seem to be lost so that produce much more crowded in connection and lead to the new loss of packet [4].

Fast retransmission is a useful application of Loss Recovery that makes TCP to trigger Loss Recovery after receiving at least 2 DUP ACK. Once the 2 DUP ACK are discovered, TCP would re-send the first packet which is suspected to be loss. Once the first suspect loss packet is retransmitted, TCP will then try to re-send other lost data according to the receiving of new cumulative acknowledgments until all data loss is retransmitted. In order to handle more cases of packet loss, TCP has many algorithms for loss recovery according to specified situations.

1.2 Network Reordering and Loss

Reordering packet is a visible situation which packet reached to the endpoint slower than a continuously transmitted packet in its flow. Reordering has influence on Loss Recovery as TCP might produce DUP ACK for each out-of-order packet. Algorithms, such as Fast Retransmit, depend on DUP ACK to identify loss, so that reordering is able to lead the Loss Recovery spuriously.[4] If the retransmission triggered by network reordering, it is likely the previous reordering packet is not lost even though it has been re-sent in the same flow.

One of the most crucial reasons of packet lost in a network is crowded. Other reasons about loss include data collision, hardware bugs and so on. Network loss can also trigger retransmission by either duplicate acknowledgments or retransmission timeout (RTO) timer expiring. TCP uses some regulations and timers such as RTO timeouts and the number of DUP ACK to calculate loss. If the loss detection was running incorrectly, TCP would respond to perceived loss packet with a wrongly response, so that produces more and more congestion. If loss recovery is run correctly and fast, the time cost in retransmission and performance influenced are minimized.

As both loss and reordering can cause the retransmission, some algorithms need to be implemented to identify whether it is network reordering or the real loss.

1.3 Research Goals

This research started as a literature survey about the related papers and tools in order to fully understand the project requirement and difficulties in identifying. Several papers provide many ideas about TCP traffic analysis and some reference books give whole details about TCP operation module.

Then the information learned from these sources was applied to the design of the on-paper algorithms. With the algorithms developing, implemented algorithms to tcpla has been done concurrently. In order to make the algorithms more accurate, the test results of tcpla would produce feedbacks to improve the algorithms. As the requirement of analysing large file, tcpla would occupy part of the disk storage while running and release it when finished.

When completed development and implementation, tcpla had the ability to analyse large scale passive network trace file and output the result to a log file. In this project, one large-scale file was used for test and profiler in python was used to analyse the running efficiency. Additionally, traffic control module in Linux was used to produce a loss rate so that whether tcpla can recovery loss rate was tested by feeding the trace file which is captured in a given loss rate. The results of each test would be used to deep the knowledge about TCP flow and improve the tcpla tool.

Chapter 2: Background

This chapter provides background information relevant to the research project and introduces many of the terms and tools used throughout the project and report. First, technology used is listed and described with each detailed functions or structures. Following the section on technologies is an integration of main difficulties in the whole project, including vantage point and identifying. Finally, previous work and existing tools in the field of network retransmission and reordering detection is discussed, including tcprs and p0f tools.

2.1 Technology Used

2.1.1 TCPDUMP and PCAP files

TcpDump is a packet analysis tool and can be simply defined as 'dump the traffic on a network', which capture the packet on the networks according to the user's definition [5]. TcpDump can capture data packets transmitted in the network. It supports filtering for network layer protocol, host, network or port, and provides 'and', 'or', 'not' and other logic statements to help user get rid of useless information. Linux as a network server, especially as a router or gateway, data acquisition and analysis is essential. TcpDump is one of the powerful network data acquisition and analysis tools in Linux. With its powerful functions and flexible interception strategy, TcpDump become a necessary network analysis tool to troubleshoot problems for every senior administrator.

In this project, TcpDump was used to capture the passive network trace file in a given loss rate. Use:

```
tcpdump 'tcp' –i lo –s 68 -w filename.pcap
```

in Linux root terminal to start packet capture. 'tcp' means only capture the packet which transport layer protocol is TCP. '-i lo' means capture the packet in the localhost interface. '-s 68' means the tool only captures 68 bytes of header so that does not contain the data bits. (The data bits are useless for the analysis.) These can reduce the size of the output file. '-w filename.pcap' make TcpDump write the captured traffic file out to the disk and save as 'filename.pcap'.

PCAP file is a type of passive network trace file. It uses hexadecimal bytes to store information. PCAP file is generated by packet-captured tools, such as TcpDump. The structure of PCAP file is in Figure 1. Each PCAP file has a PCAP header represents the captured information, such as LinkType. Each packet has a packet header in PCAP file introducing the timestamp and captured length of this packet. Following packet header is the packet data and then next is another packet header.

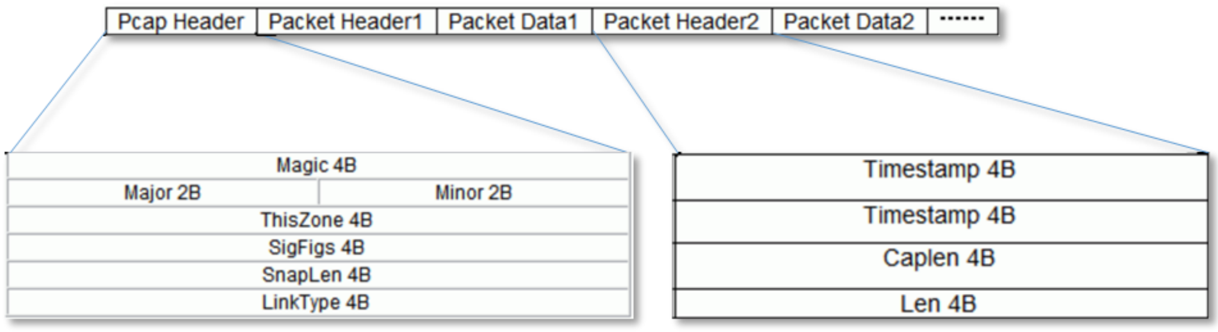


Figure 1 Structure of PCAP file [6]

2.1.2 Libpcap and Scapy in Python

Libpcap is the packet capture library available on most modern UNIX/LINUX platform [5]. Libpcap provides a system-independent user-level network packet capture interface and takes full account of application portability. Most network monitoring software or tools is based on Libpcap, such as the Wireshark and TcpDump. Libpcap can also be used as the passive network trace file reader. In this project, Libpcap mainly be used as the reader to open a PCAP file and read packet one by one. It would return a length of hexadecimal number contains the packet header and packet data. Then some decoding procedures need to be implemented.

Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. It can easily handle most classical tasks like scanning, trace routing, probing, unit tests, attacks or network discovery. Scapy normally runs on Linux with Libpcap and their respective python wrapper.[7] However, Scapy has many extra operations upon the packet that are not needed. In order to have a high efficiency, at the end of this project, tcpla removed the Scapy part and changed to decode hexadecimal bytes directly.

2.1.3 Traffic Control

TC is the traffic control module in Linux, which uses the queuing rule to set up the data packet queue and defines the sending method of each data packet in queue so as to control the flow. Processing of flows is controlled by three kinds of objects; QDISC (queue discipline), CLASS and FILTER. QDISC is the basis of traffic control, no matter when kernel needs to send packets through a network interface, it is necessary to add packets into the queue according to the configuration of the qdisc in this interface. The kernel will bring packets out as many as possible from qdisc, and then to the network adapter for processing. [8]

In this project, QDISC is mainly used even though CLASS and FILTER objects also has their useful functions. QDISC object add the loss rate in a specified network interface by using:

```
tc qdisc add dev lo root netem loss 10%
```

in Linux root terminal. This command means that add loss rate with 10% in the localhost interface. In order to delete this loss rate and come back to the normal state, use:

```
tc qdisc del dev lo root netem loss 10%
```

in Linux root terminal. It will not only delete the loss rate 10%, but also clear all user-defined limitation in the localhost interface. To be safe, in this project, once the PCAP file was generated in a specified loss rate, clear it to zero and re-set to the required number again.

2.2 Difficulties

2.2.1 Vantage Point

VP (Vantage Point) is the location that the packet capture tool worked at, and is crucial to the TCP network analysis [4]. Single VP provides a scoped viewpoint in networks, while using multiple VPs in a network system is either physically impossible or simply impractical, because it needs multiple tools in each endpoint where the packets is captured. Single VP detection is reasonable as TCP behaviour can be applied to infer loss in the lack of more extra information from any other VPs.

For the VP situated at the TCP receiver, the received packets can be assumed to the VP packet capture, eliminating the need to infer whether loss occurred between VP and TCP endpoint. For the VP located at the mid of two TCP endpoints, there are two cases may occur; packet is lost before it was captured or packet is lost after it was captured.

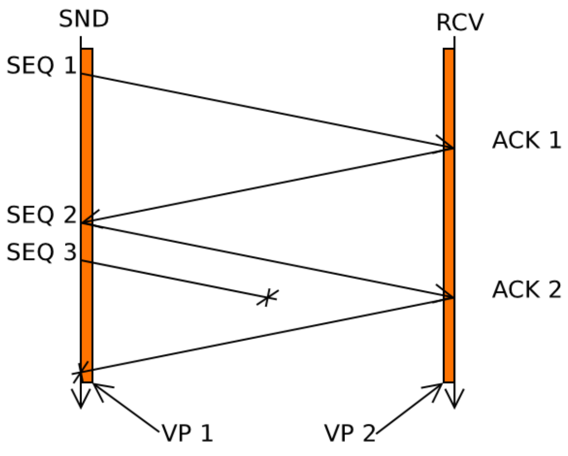


Figure 2 Two Vantage Points locate at two endpoint [4]

Figure 2 illustrate that two vantage points locate at the receiver and sender. In this case, it is trivial to observed either packet loss or reordering as the information from sender and receiver can be analysed. However, sender or receiver might be a commercial server or some other servers that cannot implement vantage point as the efficiency will be reduced. In practice, it is impossible to set the vantage points at both side. Thus, single vantage point has to be used.

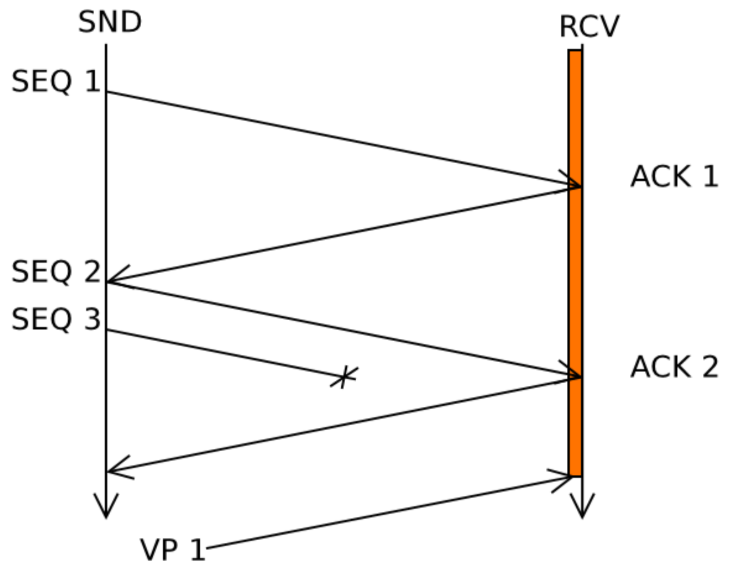


Figure 3 Single Vantage Point locates at the receiver [4]

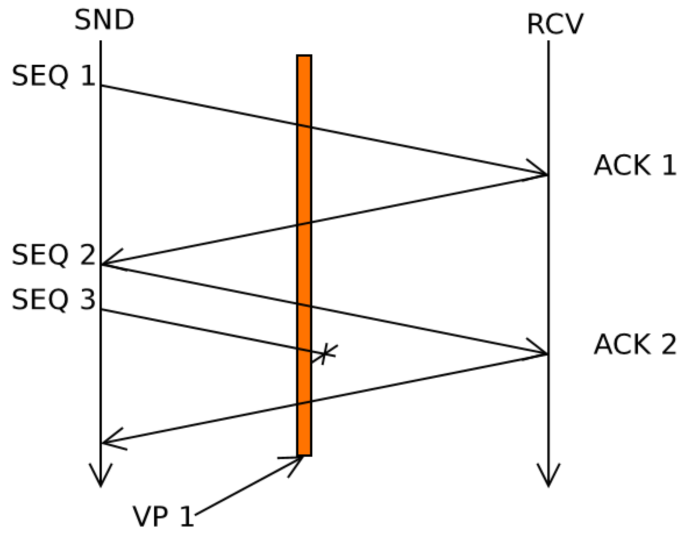


Figure 4 Single Vantage Point locates between endpoints [4]

Two modules of single vantage point are described in the following paragraphs, one for location at endpoint and another one for VP exists between endpoints. Figure 3 illustrates that single VP located at the receiver. In this case, it is impossible to observe whether packet loss occurred as it was not observed in the sender, but it will be trivial to check the ACK number sent back to the sender.

Figure 4 describes single VP locates between the sender and receiver. In this specified case, SEQ 3 was lost after captured and VP did not realize this event. In this situation, packet loss could be inferred by retransmission. For instance, if retransmission packet is same with SEQ 3, VP can find the SEQ is repeat so that infer the previous packet might loss. However, the situation in the real world might not be such nicely. The retransmission packet might not be completely same with the previous one.

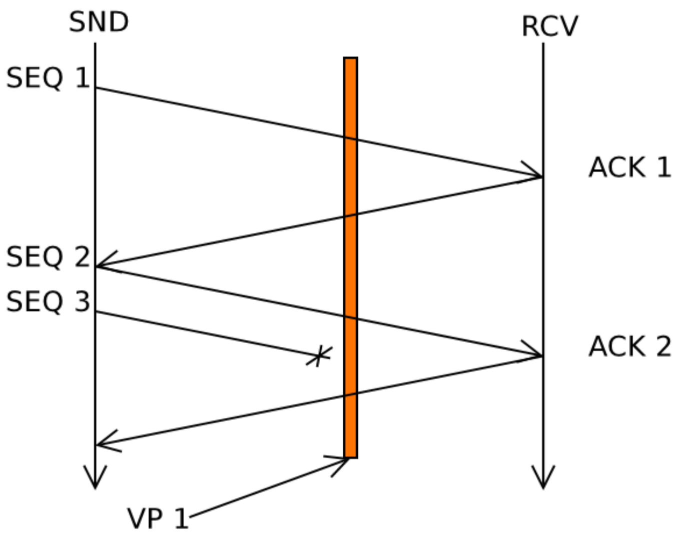


Figure 5 Another situation about loss for single VP between endpoints [4]

Another situation about packet loss is illustrated in Figure 5. In this specified case, the vantage point would not observe the first SEQ 3 packet as it lost before captured. When the retransmission of SEQ 3 captured at the vantage point, VP might not find there are loss occurred before, as it is just the first seen of SEQ 3. Therefore, the analyser should have the ability to find it is a retransmission even though whole sequence numbers in packet are new.

Reordering packet would also face to the same problems, like the loss. Reordering might occur in any node in the transport path. That is because each node in path might use different forwarding methods to treat different packets. It causes different length of transmission path so that different arriving time for packets. It is impossible to infer a reordering that happened after vantage point, unless the reordering packet triggered some unusual events, such as retransmission.

In this project, in order to consider most of the situations, the analyser will assume the input trace file is captured between two endpoints (like Figure 4 and Figure 5) so that whole details illustrated in this section will be included. It increases the accuracy but must have influence on the efficiency. Based on higher accuracy, a high efficiency is required.

2.2.2 Reordering and Loss

As the description in 1.2, both reordering and loss can cause retransmission. In other words, retransmission does not mean the loss. Even though there are many tools in any Operation System can identify the retransmission and reordering packet, few of them can identify the loss. Wireshark could set the tag '[Previous packet not captured]' to the advanced packets but these can be derived from network reordering or the real loss. Therefore, it is far from enough for current tools or software to get an accurate result about loss.

If a TCP packet loss occurred, it must trigger some events of the reliable protocol, such as duplicate ACK. For instance, if the number of duplicate ACK is more than or equal to 2 in some TCP implementations, Fast Retransmission in Loss Recovery algorithms will be triggered. Loss packet will be retransmitted according to this algorithm. Additionally, the sender has a timer called retransmission timeout (RTO). When server send a packet in the network, a RTO timer would be initialled. When server receive an ACK packet related to the individual flow, the RTO timer would be stopped as the data packet has been acknowledged by this ACK packet. If RTO timer expired, retransmission for relevant packet is triggered.

In Figure 6, the retransmission triggered by timer expired and it must be a packet loss. However, an expired RTO timer is not common in practice as there are so many data sequence packets transmitted at the same time. RTO timer usually expires when the network connection environment is terrible. In Figure 7, the retransmission triggered by duplicate ACK but this ACK is caused by network reordering. It is no loss occurred in this situation but duplicate ACK is also likely triggered by the real loss.

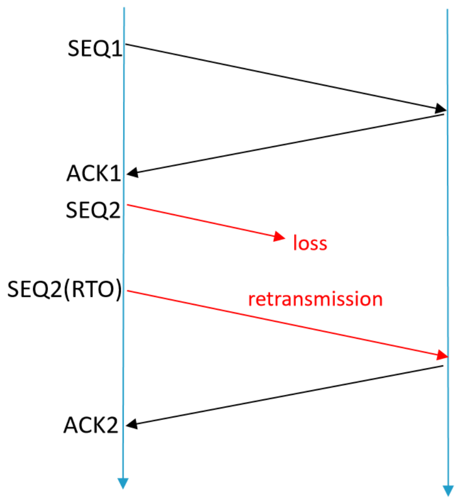


Figure 6 Retransmission caused by loss

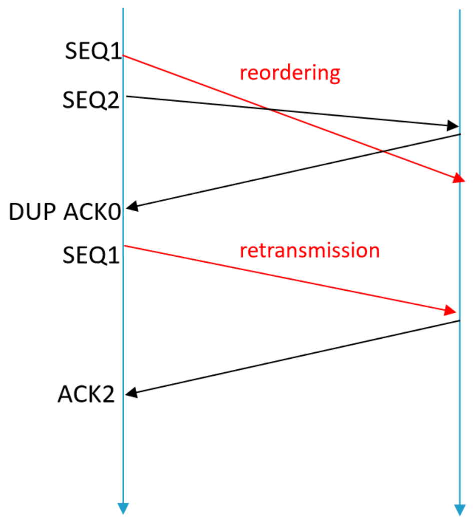


Figure 7 Retransmission caused by reordering

Additionally, there are many detailed problems while analysing loss. For instance, it is difficult to discover the loss packet before connection set up and after connection closed. If retransmission is occurred in SYN or FIN packet, it has a little different with the normal packet. For example, if a connection is re-built with the same port numbers and IP addresses after it was closed, some judgement should be implemented in. For acknowledgement packet, even though the current packet's ACK number is repeat with a previous one, it still can be a normal packet but not the DUP ACK, such as the window update packet.

To sum up, retransmission can be caused by both loss and reordering. It cannot simply treat the retransmission as packet loss so some algorithms must be implemented to identify whether the retransmission exactly caused by network loss or reordering. Meanwhile, there are many detailed problems related to analysing loss need to be considered.

2.3 Prior work

2.3.1 Tcprs

Tcprs is a traffic analyser that is used to identify the retransmission and reordering packet and developed by James E. Swaro in reference [4]. Comparing with other analysis tools, such as tcpcsm and tcptrace, tcprs is more accurate and precise in processing and resulting the passive trace file. Based on the estimation by its developer, tcprs is succeed in making TCP network analyser more accurate.

Tcprs used a mature classification hierarchy to the retransmission and reordering packet, detailed in

Figure 8. Originally classification for out-of-order packet list it into three different objects; reordering, retransmission, and not sure [9]. Now, based on this classification, tcprs can detect and identify the retransmission and reordering packet accurately.

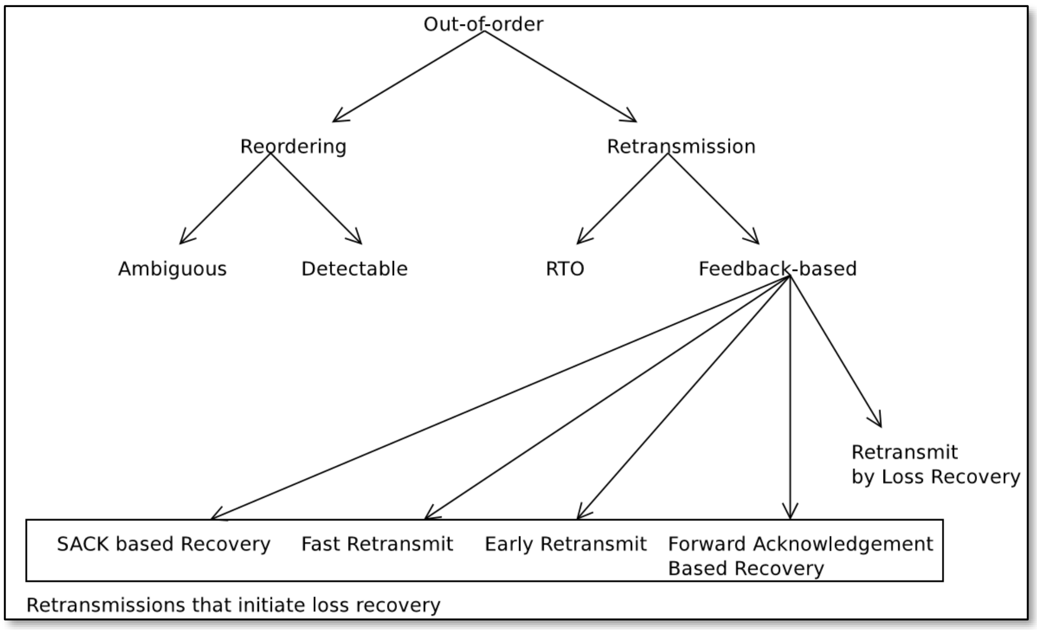


Figure 8 Classification developed in [4]

To identify whether the packet is reordering, tcprs compares the time gap between normal packet and out-of-order packet with the round trip time. The method of estimated RTT will be given in section 2.3.3. In Doctor James' paper, he introduced three methods about retransmission and reordering detection; timestamp detection, acknowledgement number detection and time gap detection. Each algorithm is able to identify the retransmission whose SEQ number is completely first seen.

2.3.2 Stack Fingerprinting

Stack Fingerprinting of Operation System and TCP strategy have been used to help developers in analysing and identifying retransmission. Many tools could be used to provide active or passive fingerprinting, such as p0f. Passive fingerprinting includes viewing on the packet stream and discovering unusual things in the header or responses which might include a specific operation type of TCP [10].

A reasonable potential method uses OS Fingerprinting to identify TCP Congestion Control module. Every OS apply a different algorithm to control their congestions. This configuration of control can be brought to construct the changes in slow start threshold and congestion window so that the prediction of slow start threshold is more precise. As the prediction for slow start threshold and congestion window becomes more precise, judging the connection state about congestion becomes easier. When the AIMD behaviour of the network is known before analysis, the analyser should have the ability to model the expected changes in window size with respect calculation. Therefore, congestion states would be more accurately defined as long as its information can be brought from the OS Fingerprint.

The p0f is a tool that utilizes an array of passive traffic fingerprinting mechanisms to identify the OS behind any incidental TCP/IP communications without interfering in any way [11]. Whenever p0f obtains a fingerprint from the observed traffic, it defers to the data read from database to identify the OS and obtain some ancillary data needed for other analysis tasks. However, Stack Fingerprinting cannot ensure that TCP operations will be same with the behaviour as the fingerprint tells. In other words, misclassification of behaviour may occur.

2.3.3 Estimate Round Trip Time (RTT)

Round Trip Time would be used in reordering packet detection. It is crucial to get an accurate RTT so that can classify reordering packet perfectly. The mathematical methods and equations are abundant enough to have a precise round trip time. In this project, the basic idea behind our RTT estimation technique is illustrated in Figure 9. The algorithm infers the RTT as the sum of i) the round trip delay from the measurement point to the receiver and then back to the vantage point (labelled d1 in Figure 9), and ii), the round trip delay between the vantage point, the sender and then back to the measurement point (labelled d2 in Figure 9).

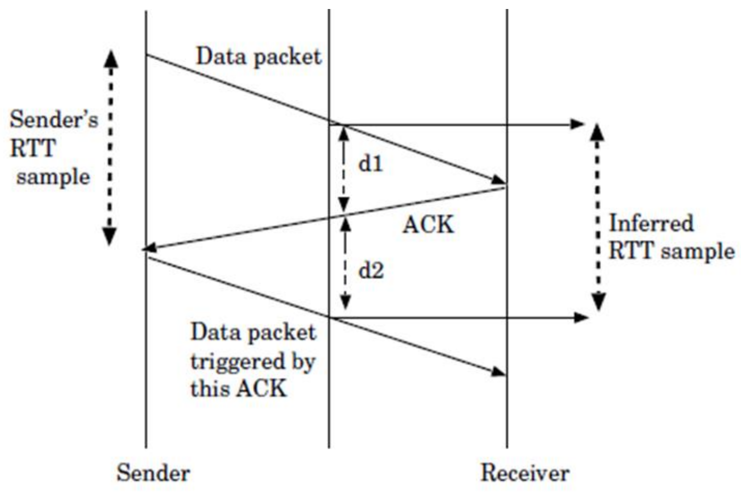


Figure 9 Estimation of Round Trip Time [12]

The sum of these two delays d1 + d2, as shown in Figure 9, is the estimation of RTT.

$$RTT_{sample} = T_{acknowledgment} - T_{packet}$$

The sample RTT is calculated as the timestamp of the packet's acknowledgement decrease by the timestamp of the packet [12]. The estimate RTT between each sender and receiver is:

$$RTT_{endpoints} = RTT_{send} + RTT_{recv}$$

$$RTTVAR_{endpoints} = RTTVAR_{send} + RTTVAR_{recv}$$

This average RTT of the network connection is an estimate of the actual RTT in the real flow. The software would use the adjacent two packets that satisfied the packet-acknowledgement relationship to estimate the forward or backward RTT.

Chapter 3: Design and Implementation

3.1 Read and Split Procedure

Using Libpcap to open and read PCAP files. Using f = pcap.pcapObject(filename) or f = pcap.pcap(filename) to return a file object f. Then f.next() would be the next packet, including timestamp, length of packet and hexadecimal information bits. To decode these hexadecimal bits, tcpla uses struct.unpack() function to translate these bits to the readable format, such as integer and string.

In order to save running memory, tcpla would create a defined object named ‘packet’ to represents each packet. This object only contains the useful variables, including IP addresses, port numbers, SEQ/ACK number, segment length, window size, direction and flags (SYN, FIN, RST).

Split procedure firstly creates a flowname String that is ‘source IP address_destination IP address_source port number_destination port number’. Then check whether this flowname has been seen, using a dict containing all flows’ name. If exists, append this packet to flowdict[flowname], which represents a list of all packets in individual flow, and set this packet’s direction to 1 (forward). If not exists, create another flowname String that is ‘destination IP address_source IP address_destination port number_source port number’ and check again. If new flowname exists in flowdict, append this packet and set its direction to 0 (reverse). If still not exists, create an empty list in flowdict[flowname] and append this packet with direction 1 (forward). After this procedure, the flowdict records all packets in order using their flowname as keywords.

3.2 Identify Loss and Reordering Algorithms

There are some terms used in this section:

- Out-of-order packet: the packet captured not in an increasing order of SEQ. It might be caused by reordering or retransmission. For example, the highest observed SEQ is y, if now there is a packet with SEQ number x captured and $x < y$, then this captured packet is an out-of-order packet.
- Reordering: a visible situation which packet reached to the endpoint slower than a continuously transmitted packet in its flow.
- Probably retransmission: in some situations, it is hard to judge a packet is reordering or retransmission so that set it to be probably retransmission. For example, if an out-of-order packet is first seen and the time gap between this packet and last normal packet is close to but more than

one RTT, it is likely a retransmission but might just be a reordering packet, as the Figure 10 shown.

- Current packet: the packet in the capture file that is currently being considered.
- Next packet: the packet following the current packet in the capture file.
- Observed packets: all of the packets that have been processed so far (from the same connection).
- Previous packet: the nearest packet captured previously and contains the bits from retransmission packets. For example, if there are three packets contains the same information in the order of A, B and C, then A will be the previous packet of B and B will be the previous packet of C.

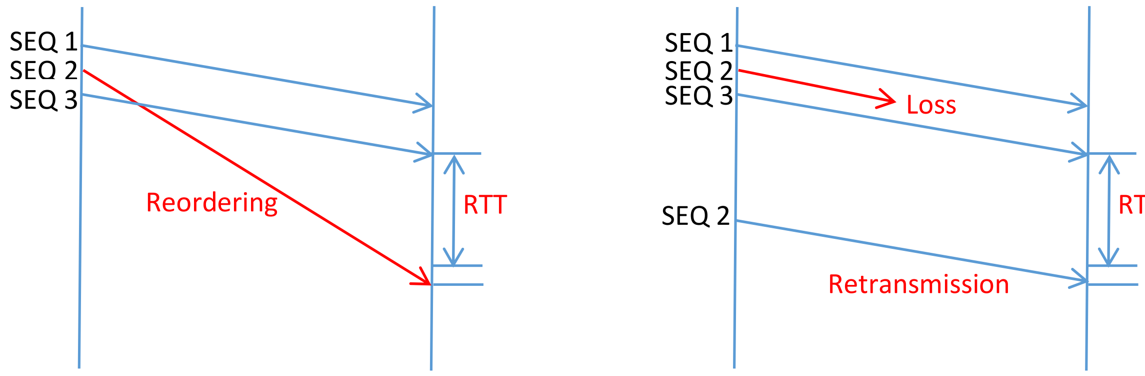


Figure 10 probably retransmission (it might be reordering)

3.2.1 Retransmission Detection

Algorithm:

```
--if current packet has been observed previously: it is a retransmission packet
--else if current packet has not been observed previously,
        --if the gap between last observed packet and this packet is less than RTT, it is a reordering
        --else, it is a probably retransmission packet
```

Flowchart:

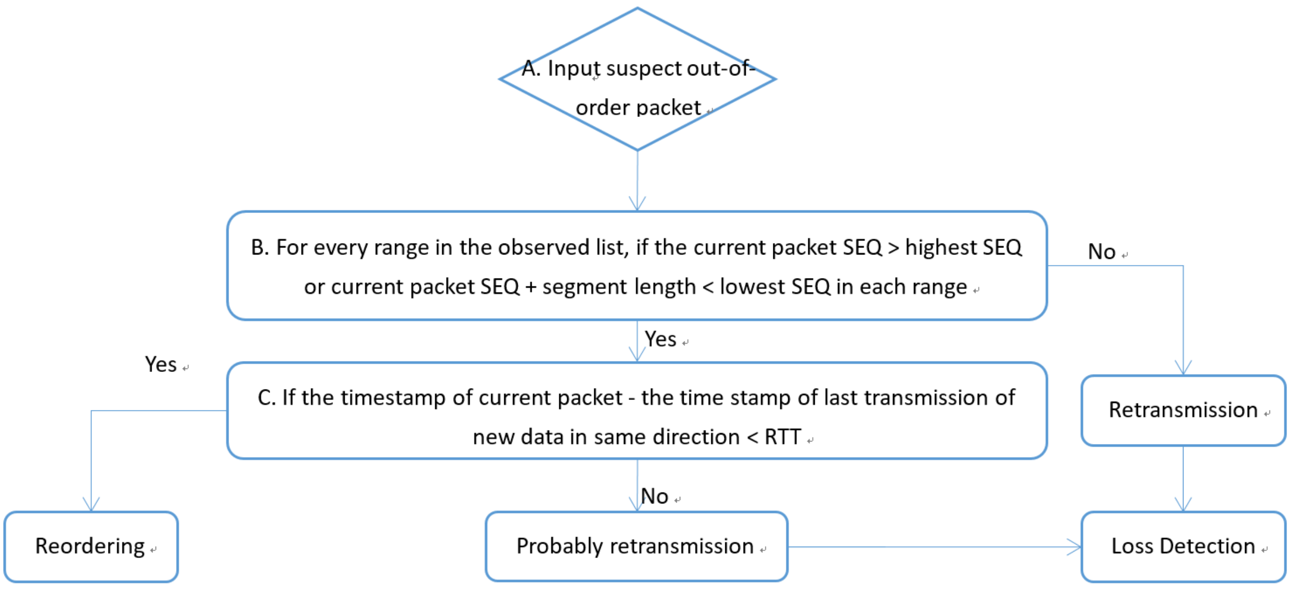

A. Input suspect out-of-order packet
B. For every range in the observed list, if the current packet SEQ > highest SEQ or current packet SEQ + segment length < lowest SEQ in each range
No
Yes
Yes
C. If the timestamp of current packet - the time stamp of last transmission of new data in same direction < RTT
Retransmission
No
Reordering
Probably retransmission
Loss Detection

Figure 11 Flowchart of retransmission detection

Annotations of Figure 11:

A. The flow chart is a loop so that input the next packet when finished judge one out-of-order packet is reordering or retransmission. The next packet will be checked whether it is out-of-order in advance. If it is not an out-of-order packet, treat it as the normal packet and record its SEQ or ACK. If it is out-of-order, input to the loop.

   The check procedure uses the sequence number. For example, if next packet SEQ is 10 but the highest previously observed packets SEQ is 20, the next packet is an out-of-order packet because 10 < 20 and it must be a reordering or retransmission packet. If next packet SEQ is 20 and the highest observed packets SEQ is 10, the next packet is a normal packet even if SEQ 11 to 19 have not been observed.

B. At first, there is a python list to store the observed packet SEQ. The list is in the form:

$$[\underbrace{[a, a + \text{segment length}]}_{\text{Range a}}, \underbrace{[b, b + \text{segment length}]}_{\text{Range b}}, \ldots, \underbrace{[n, n + \text{segment length}]}_{\text{Range n}}]$$

where $a + \text{segment length} < b - 1$, $b + \text{segment length} < c - 1$ and so on. That means the range a, b… n are not connected, so there must be some packet have not been observed (out-of-order). If all packets observed in normal, the list should have only one element and be [[start SEQ, end SEQ]]. For instance, if the first packet is SEQ x and segment length is y, the list is [[x, x + y]]. Then if the second packet is SEQ x + y + 1 and segment length is y, the observed list is [[x, x + 2y]]. In addition, the highest observed SEQ of range a is a + segment length, the lowest observed SEQ of range a is a and so on.

As the Figure 12 shows, in each range, if current packet SEQ > the highest SEQ or current packet SEQ + segment length < the lowest SEQ, it means current packet does not have bits which falls in any range in the observed list, the current packet must be an entire new data packet and it needs further detection. Otherwise, the current packet must have the repeated bits so it is a retransmission packet. Moreover, the retransmission packet might have new bits, so the program will detect this and append new range to or change the observed list.

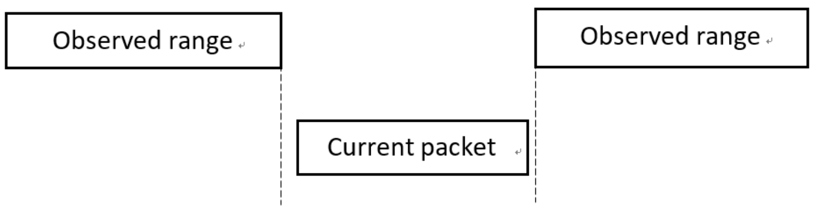


Figure 12 the current out-of-order packet is a new packet

C. In order for this part, the software needs a calculation of RTT. The time-gap is specified as the time-delta between the last transmission of new data (normal packet) and the out-of-order segment.

   As the Figure 13 illustrates, in the general case, network reordering occurs within less than a single round-trip time of the connection. Retransmits occurring from initiation of feedback-based loss recovery often occur on the receipt of a duplicate acknowledgment from the TCP receiver. It takes at minimum, one RTT for the duplicate acknowledgments for newly transmitted data to be received by the TCP sender [4].

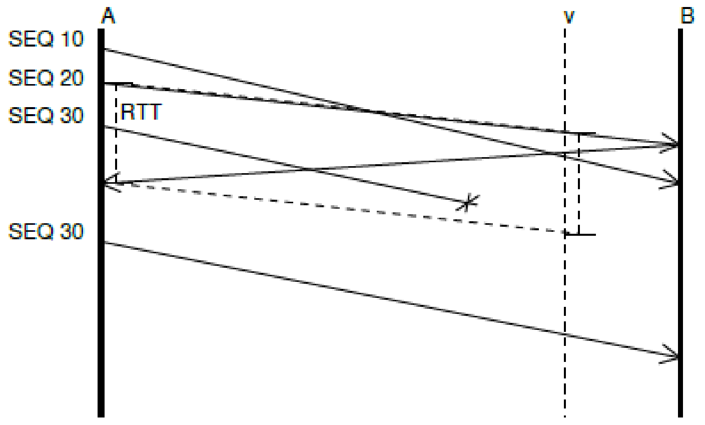


Figure 13 Retransmission Detection via Gap Analysis [4][3]

3.2.2 Loss Detection

Algorithm:

```
--if the retransmission packet is a 'probably retransmission', there is a probably data loss.
--else if the retransmission packet has been acknowledged before:
    --if the acknowledgement packet captured before the previous packet:
        --Previous packet probably lost.
    --else:
        --There is no data loss.
--else (if the retransmission packet has not been acknowledged before):
    --sort the retransmission packet in an increasing order of SEQ firstly,
    --if the bits of all retransmission packets cover the range of the previous packet SEQ to SEQ
    + segment length, before next ACK number that greater than previous packet SEQ + segment
    length:
        --if there are some ACK packet which number is in the range of previous packet SEQ
        to SEQ + segment length: the previous packet is lost.
        --else: the previous packet probably lost.
    --else: There is no data loss.
```

Flowchart:

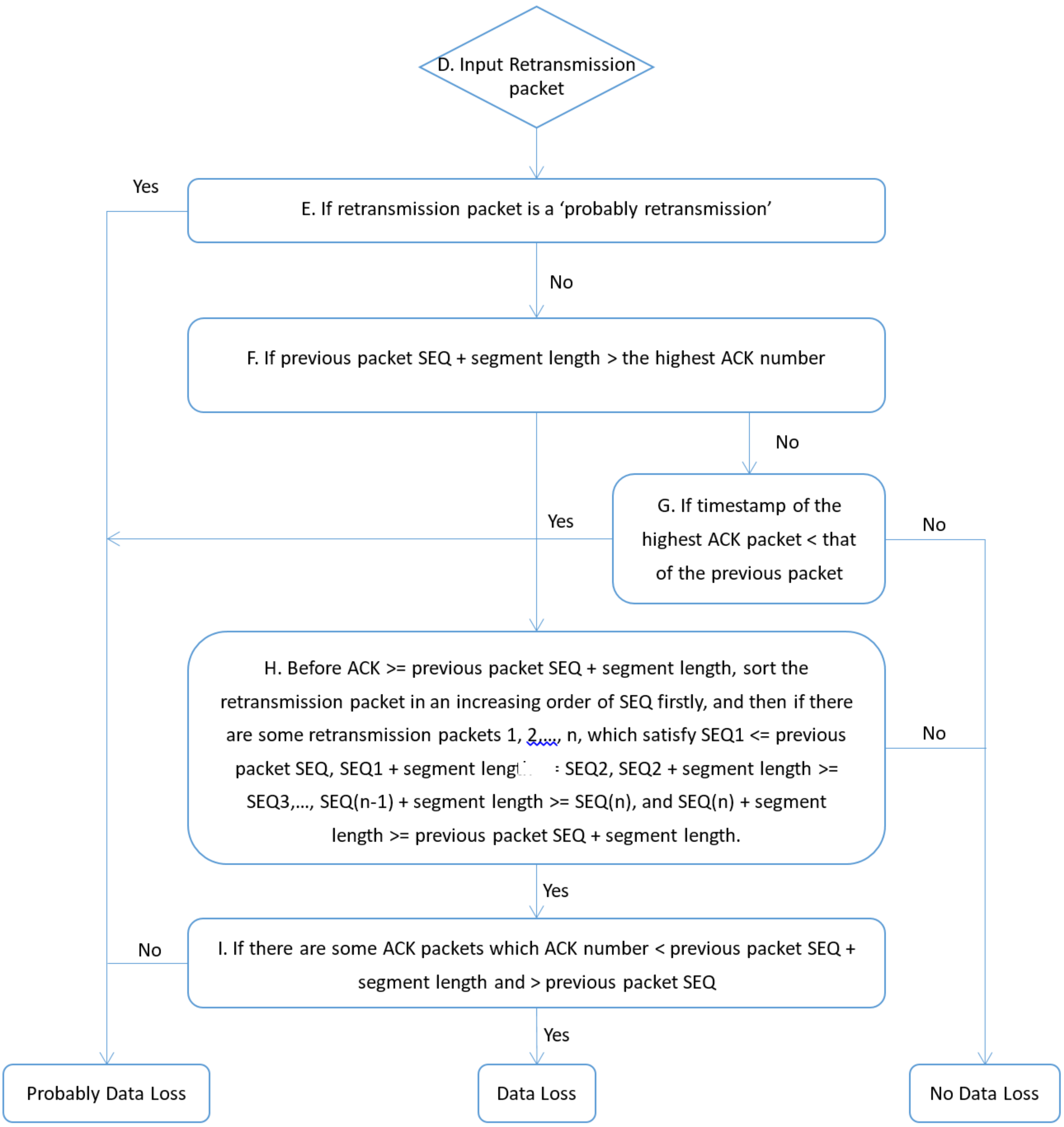

D. Input Retransmission packet
E. If retransmission packet is a 'probably retransmission'
Yes
No
F. If previous packet SEQ + segment length > the highest ACK number
No
Yes
G. If timestamp of the highest ACK packet < that of the previous packet
No
H. Before ACK >= previous packet SEQ + segment length, sort the retransmission packet in an increasing order of SEQ firstly, and then if there are some retransmission packets 1, 2,..., n, which satisfy SEQ1 <= previous packet SEQ, SEQ1 + segment length >= SEQ2, SEQ2 + segment length >= SEQ3,..., SEQ(n-1) + segment length >= SEQ(n), and SEQ(n) + segment length >= previous packet SEQ + segment length.
No
Yes
I. If there are some ACK packets which ACK number < previous packet SEQ + segment length and > previous packet SEQ
No
Yes
Probably Data Loss
Data Loss
No Data Loss

Figure 14 Flowchart of loss detection

Annotations of :

D. The software will conduct loss detection when retransmission detection finished. During loss detection, only retransmission packet will be inputted. That is because the data loss can only be detectable when retransmission occurred.

E. As the algorithm of retransmission detection, the probably retransmission packet can only be detected from the gap detection part. It also means the packet is a new data packet that be observed for the first time. Thus, if the probably retransmission packet is a retransmission packet, there must be a data packet lost before. If the probably retransmission packet is just a reordering packet (not retransmission), there are no loss. In other words, the probably retransmission packet can infer 'probably data loss'.

F. This step aims to check whether the bits in previous packet has been acknowledged. If it has not been acknowledged at this moment, in other words, the highest ACK number < previous packet SEQ + segment length, there might be a ACK packet acknowledge the previous packet after the retransmission packet, so further detection in step H is needed. If the bits in the previous packet has been acknowledged, next step G will check whether this ACK packet acknowledge the bits in this previous packet rather than other packets.

G. If the bits in previous packet have been acknowledged before the retransmission packet, there are two situations may occur; 1) this ACK packet appears before the previous packet, 2) this ACK packet appears after the previous packet(therefore it is between previous and retransmission packet). For case 1, it means this ACK packet acknowledged the bits in other packets rather than the previous packet. So the previous packet probably lost as the retransmission packet. For case 2, the ACK packet is more likely to be the acknowledgement of the previous packet. And it is hard to find other evidence to prove the previous packet lost. Therefore, in this situation, the software set the previous packet to be 'no loss'.

H. If there are bits in previous packet has not been acknowledged at that moment, at first, the software will find the first ACK packet which ACK number is greater than or equal to the previous packet SEQ + segment length. Then the software will generate a range from the previous packet to the ACK packet found above and detect the retransmission packets in this range.

   The software will sort the retransmission packets 1, 2, 3…n in the range in an increasing order of SEQ. SEQ(n) means the sequence number of packet n. Then, if SEQ1 <= previous packet

SEQ, SEQ1 + segment length >= SEQ2, SEQ2 + segment length >= SEQ3,…, SEQ(n-1) + segment length >= SEQ(n), and SEQ(n) + segment length >= previous packet SEQ + segment length, the retransmission packet is connected and cover the previous packet as the Figure 15.

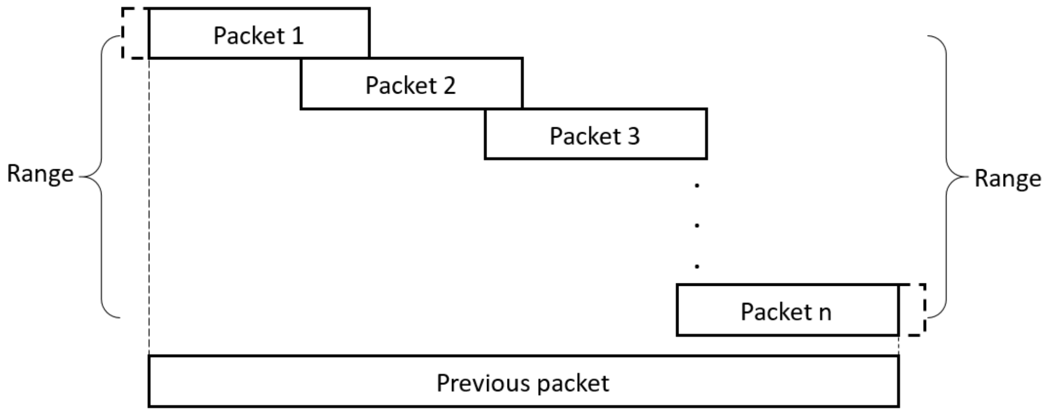


Figure 15 the retransmission packets cover previous packet

If all bits in packets in this range cannot cover the previous packet, in other words, if some bits of the previous packet are not retransmitted, the previous packet must not be lost. If all bits in packets in this range are connected and cover the previous packet, it has two possible situations and need a further detection.

I. If the answer is 'Yes' in the H box, it means all bits in packets in the range are connected and cover the previous packet. The further detection is to check whether there are some ACK packets which acknowledge the retransmission packets. If there are some ACK packet which ACK number is less than the previous packet SEQ + segment length and greater than the SEQ of previous packet, it means this ACK packet acknowledged a packet which is a retransmission of the previous packet. Therefore, the previous packet must be lost.

   If such ACK packet cannot be found, it means the ACK number of the next ACK packet is greater than the previous packet SEQ + segment length. Thus it cannot be determined that whether this ACK number acknowledges the previous packet or the retransmission packet. It is more likely to be the retransmission packet as every part of the previous packet have been retransmitted, so set it to be 'probably data loss'.

3.3 Analyse Large Scale Files

When handling large scale files, the software must run out of memory if no extra operations as there are too many packets to be recorded. Writing split flows out to the disk can ensure RAM not overflow, but it would also spend more running time. In order to have a trade-off between memory used and efficiency, six methods in Table 1 have been attempted and compared.

Table 1 Methods attempted

No.	Methods
1	read and split directly
2	cPickle.dump each packet
3	json.dump each packet
4	cPickle.dump flows for every 1 million packets
5	cPickle.dump silence flows for every 1m packets
6	Use profiler to analyze and improve tcpla

Details of each method:

1. Attempt to use Libpcap to read packet and decode it directly. Actually at first, tcpla used Scapy to read all packet to memory. But for large scale file, RAM cannot support for so much storage. Additionally, Scapy has many extra operations which are not needed, such as decodes the unneeded bits and stores it in a packet dict. Therefore, in order to read the large file efficiently, tcpla gave up to use Scapy and changed to use Libpcap directly. In this attempt, tcpla read and write packets one by one so it is not efficient.
2. Attempt to use cPickle library to dump packets. Based on attempt 1, tcpla used cPickle.dump to write each packet out to the flow file in disk. This method serializes the packet object and dump it out. Then tcpla would use cPickle.load to load the packet into memory when analysing flows. The problem of this attempt is same with attempt 1; reading and dumping packet one by one costs more time.
3. Attempt to use JSON library to dump packets. After utilizing cPickle library, tcpla tried to use json.dump and json.load to do the same processing in attempt 2. However, JSON does not work well for the user-defined object. In other words, JSON is not better than the method described in attempt 2.
4. Attempt to dump file by flows. One read one write mode is really slow as it needs 2 million writing for 2 million packet. Based on attempt 2, if tcpla read 1 million packet, split it into flows in memory and then dump flows to disk, much of the writing time would be saved. In this attempt, tcpla would read and split 1 million packets, then dump individual flows to the disk and read another 1 million packets again.
5. Attempt to identify active flows and only dump the silence flows. Active flow means the packets

in this flow are seen recently. Silence flow represents the flow whose packets have not been seen for a long time. Based on attempt 4, in this attempt, tcpla would record the 900,000th packet and compare it with the latest packet in each flow. If the latest packet in flow is later than the 900,000th packet, this flow is an active flow as the packets in this flow are seen recently. Otherwise, set the flow to be silence as tcpla has not seen the packet in this flow for at least 1 million packets. This method can reduce the number of writing times as the active flows would not be dumped so that save running time.

6. Attempt to use cProfile of Python to analyse program performance. The cProfile is a profiler for python program that can analyse running performance and give the number of call times and cumulate time for each function. Based on attempt 5, using the analysis results can improve tcpla in some detailed parts. According to the experimental results in 4.2.1, the final version of tcpla formed.

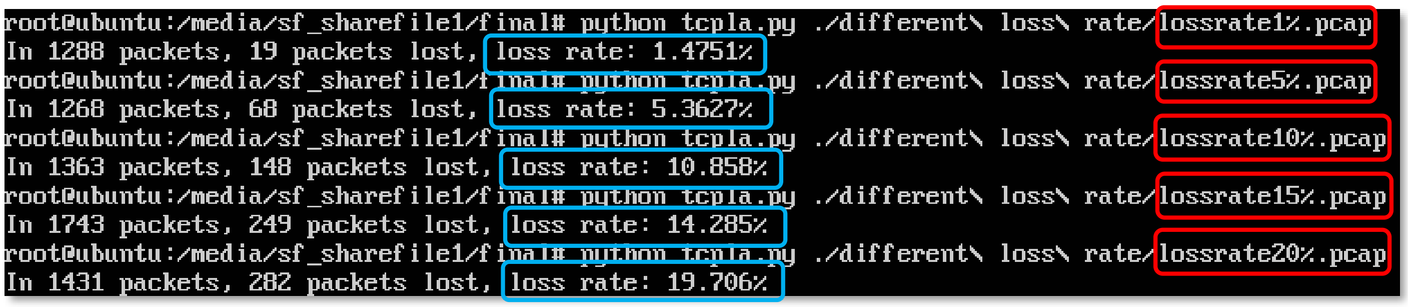
Chapter 4: Results and Discussion

4.1 Loss rate results

In this section, the experiment uses a TCP server program to listen a port and a TCP client program to request data packet so that produces the TCP packets in localhost interface. Then capture and use these packets to test the tcpla program. The TCP server and TCP client can simulate the real network traces but these are not always the same as there are so many factors have influence on the real network.

4.1.1 Results in different loss rate

Traffic control module in Linux kernel (2.1.3) is used to set different loss rate of local host interface with IP address 127.0.0.1. The traffic control module cannot control the loss rate exactly to the input value so there are some deviations on the results. Therefore, to ensure that tcpla get the correct results, Wireshark was used to help checking results.

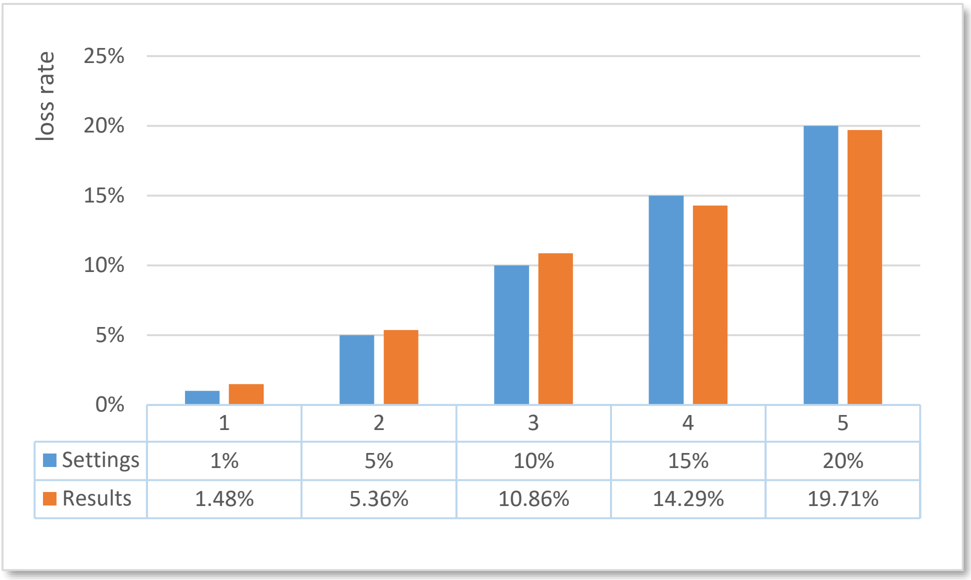


```
root@ubuntu:/media/sf_sharefile1/final# python tcpla.py ./different\ loss\ rate/lossrate1%.pcap
In 1288 packets, 19 packets lost, loss rate: 1.4751%
root@ubuntu:/media/sf_sharefile1/final# python tcpla.py ./different\ loss\ rate/lossrate5%.pcap
In 1268 packets, 68 packets lost, loss rate: 5.3627%
root@ubuntu:/media/sf_sharefile1/final# python tcpla.py ./different\ loss\ rate/lossrate10%.pcap
In 1363 packets, 148 packets lost, loss rate: 10.858%
root@ubuntu:/media/sf_sharefile1/final# python tcpla.py ./different\ loss\ rate/lossrate15%.pcap
In 1743 packets, 249 packets lost, loss rate: 14.285%
root@ubuntu:/media/sf_sharefile1/final# python tcpla.py ./different\ loss\ rate/lossrate20%.pcap
In 1431 packets, 282 packets lost, loss rate: 19.706%
```

Figure 16 screenshot of results in different loss rate

Figure 16 illustrates tcpla can recover the loss rate from different status. The red boxes represent the input files name and the blue boxes represent the results of tcpla. These files are named according to their loss rate configurations when these files are captured. These test files are submitted in the support documents.

It seems that some deviations might exist for around 1% in Figure 17, but in fact these deviations are caused by the traffic control module. Traffic control module cannot control the loss rate accurately to the exactly settings value. User can open the trace file in Wireshark to check the accuracy of loss rate results. (Moreover, sometimes the deviation between loss rate settings and practical value are greater than 1%).

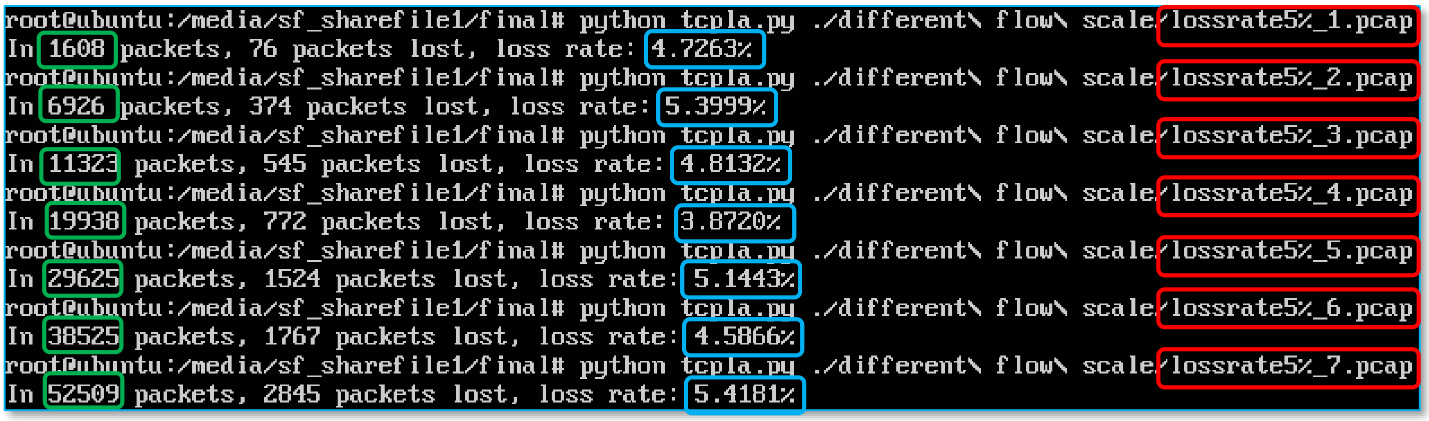


	1	2	3	4	5
🟦 Settings	1%	5%	10%	15%	20%
🟧 Results	1.48%	5.36%	10.86%	14.29%	19.71%

Figure 17 bar chart of the comparisons in Figure 16

4.1.2 Results of different flow scale

The tcpla has been tested in different loss rate, but it is only some small-scale file. In order to test the accuracy and deviations in flows containing more packets, this project set the loss rate to 5% and capture an increasing number of packets. Each of these trace files only contains one flow but has different number of packets in different flow.



```
root@ubuntu:/media/sf_sharefile1/final# python tcpla.py ./different\ flow\ scale/lossrate5%_1.pcap
In 1608 packets, 76 packets lost, loss rate: 4.7263%
root@ubuntu:/media/sf_sharefile1/final# python tcpla.py ./different\ flow\ scale/lossrate5%_2.pcap
In 6926 packets, 374 packets lost, loss rate: 5.3999%
root@ubuntu:/media/sf_sharefile1/final# python tcpla.py ./different\ flow\ scale/lossrate5%_3.pcap
In 11323 packets, 545 packets lost, loss rate: 4.8132%
root@ubuntu:/media/sf_sharefile1/final# python tcpla.py ./different\ flow\ scale/lossrate5%_4.pcap
In 19938 packets, 772 packets lost, loss rate: 3.8720%
root@ubuntu:/media/sf_sharefile1/final# python tcpla.py ./different\ flow\ scale/lossrate5%_5.pcap
In 29625 packets, 1524 packets lost, loss rate: 5.1443%
root@ubuntu:/media/sf_sharefile1/final# python tcpla.py ./different\ flow\ scale/lossrate5%_6.pcap
In 38525 packets, 1767 packets lost, loss rate: 4.5866%
root@ubuntu:/media/sf_sharefile1/final# python tcpla.py ./different\ flow\ scale/lossrate5%_7.pcap
In 52509 packets, 2845 packets lost, loss rate: 5.4181%
```

Figure 18 screenshot of results in different flow scale

Figure 18 illustrates the loss rate result in different flow scale. The red boxes represent the input files name and the blue boxes represent the results of tcpla. Additionally, the green boxes represent the number of data packets in the flow file. These flows were captured in the fixed loss rate 5% and were named by the increasing number of packets. These test files are submitted in the support documents.

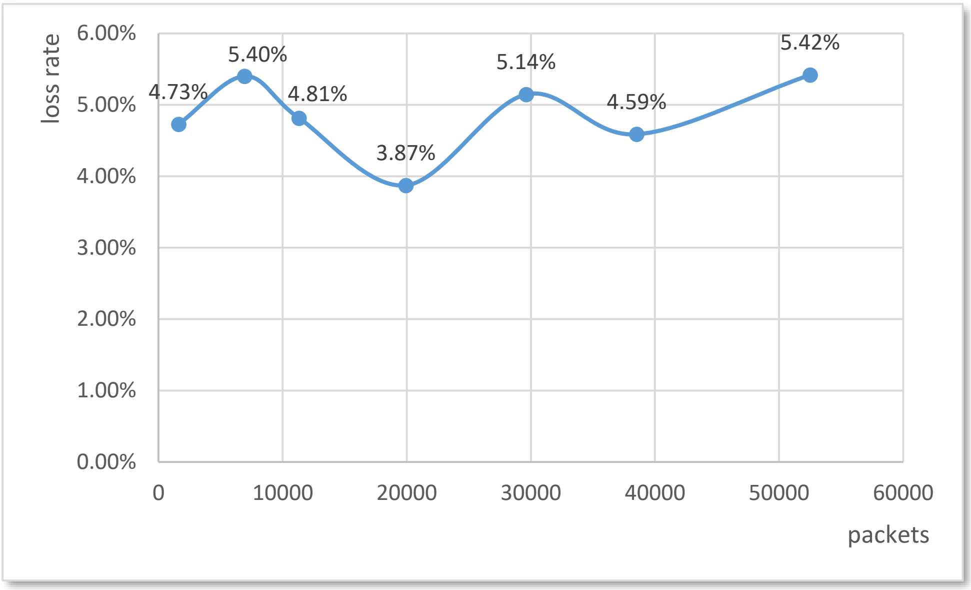


Figure 19 line chart of the results in Figure 18

With the increasing number of packet, the loss rate results fluctuate near the settings value 5%, as the Figure 19 shown. This fluctuation is reasonable because it is impossible for the traffic control module to control the loss rate to exactly 5%. In Figure 19, when 20,000 packets, the result 3.87% is a little far from the settings 5%. It is in an acceptable range as this situation might be occurred in a few packets flow and this flow result can be checked correctly by Wireshark.

In conclusion, the results about different loss rate and different flow scale prove that tcpla can recover the loss rate within a reasonable deviation. The experimental results are reliable and not the accidents as the procedures and outputs are repeatable.

4.2 Large scale file results

The large file used in this project is from CRAWDAD datasets website. CRAWDAD is the Community Resource for Archiving Wireless Data At Dartmouth, a wireless network data resource for the research community.[13] It is a 280 MB file including around 2.9 million packets. This large file is mainly used to test the performance of tcpla program using the method in Table 1, including the call times, cumulative time and total time of each function in tcpla.

4.2.1 Results of running time

Figure 20 illustrate the total time of running tcpla with the large file. The mode of 'one read one write' is slow in method 1, 2 and 3. Method 2 is slower than method 1 because the dump and load for 2.9 million packet in method 2 cost more time than just write and read the hexadecimal bits in method 1.

Method 3 is slower than method 2 because the JSON module does not work well for the user-defined object 'packet'.

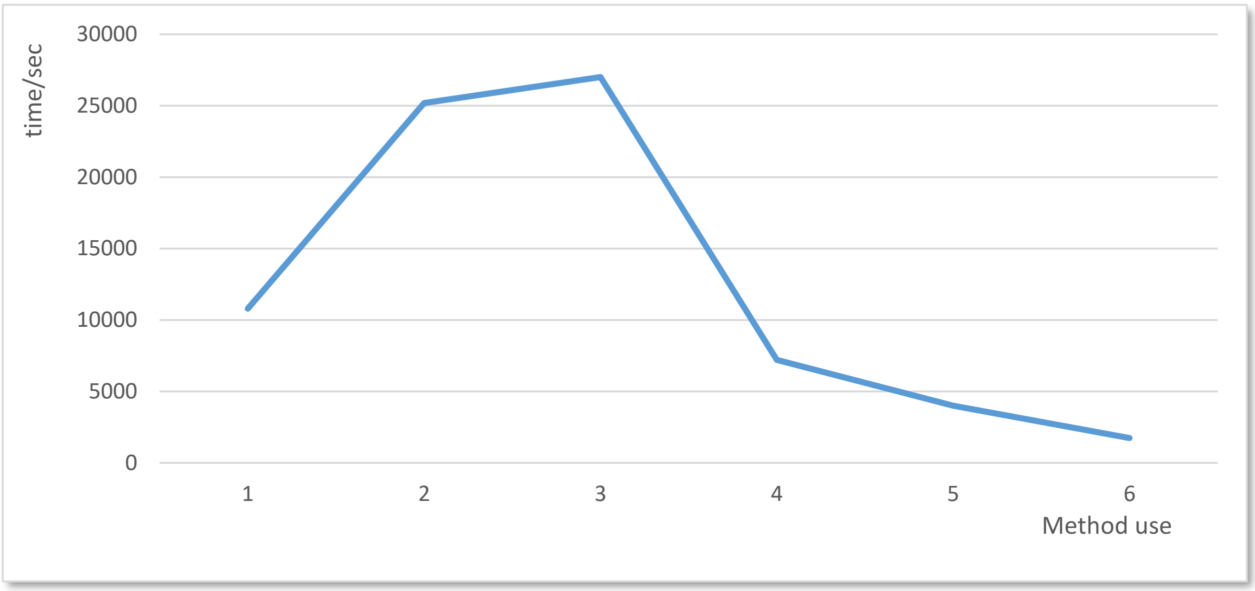
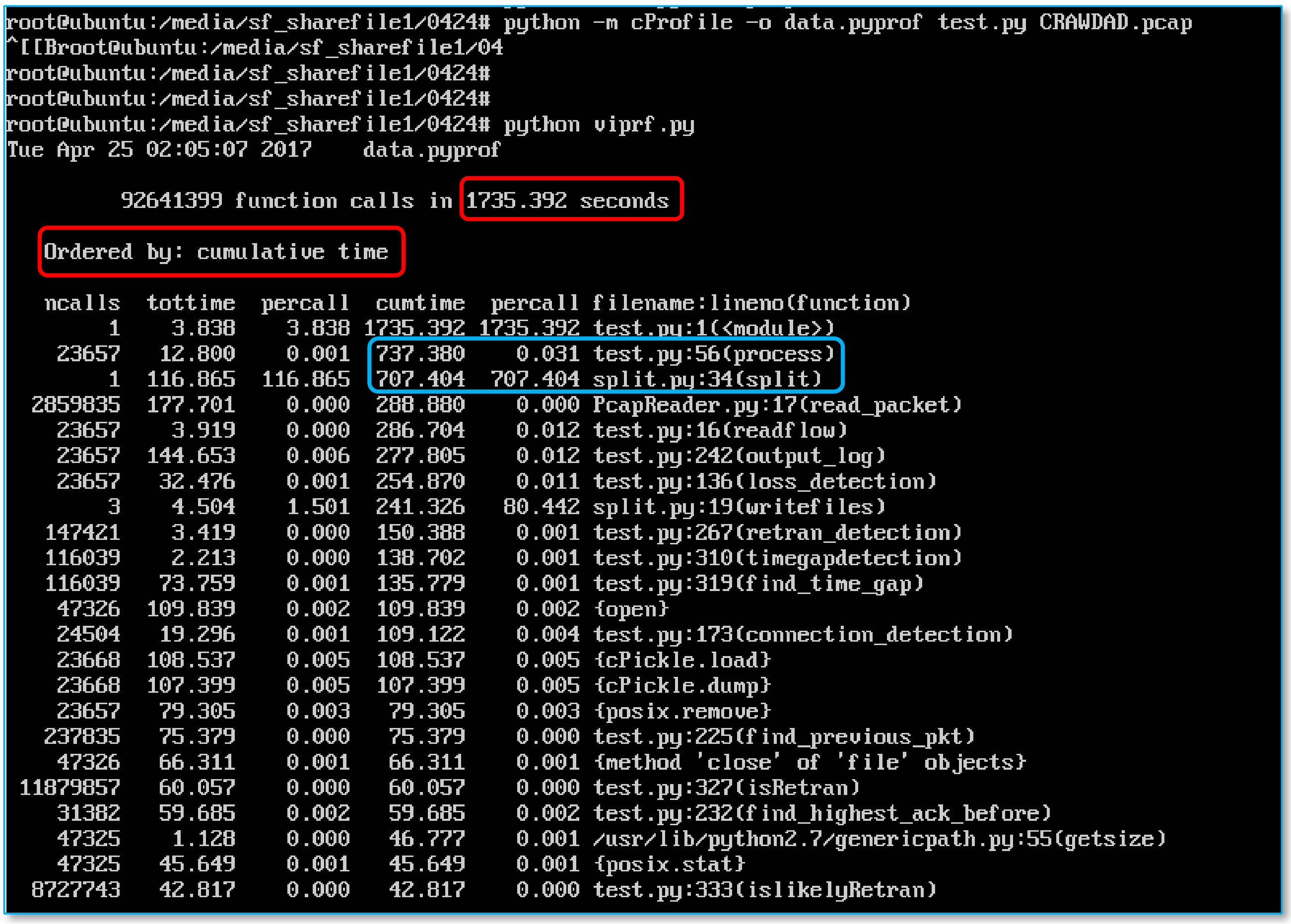


Figure 20 The total time of running tcpla using each method

After changing tcpla to dump flows rather than packets, the running time dropped to around 2 hours. That is because method 1, 2 and 3 need to call the dump or load function for 2.9 million times, but method 4, 5 and 6 only need to call dump or load for 23 thousand times as there are only 23 thousand flows in this file. Therefore, split trace files in memory and dump flows to disk can save much time. To ensure memory do not overflow, tcpla would only read 1 million packets for one time so that split and dump these 1 million packets then read another 1 million packets again. Method 5 is faster than method 4 because the identifying of the 'silence' flow (Silence flow represents the flow whose packets have been not seen for a long time) so that tcpla only dumps the silence flow to the disk and reduces the dump times.

4.2.2 Results of profiler

Finally, profiler in python helps to analyse the performance of tcpla and find the compositions of running time. The profiler result in Figure 21 can be used to improve the tcpla program.



```
root@ubuntu:/media/sf_sharefile1/0424# python -m cProfile -o data.pyprof test.py CRAWDAD.pcap
^[[Broot@ubuntu:/media/sf_sharefile1/04
root@ubuntu:/media/sf_sharefile1/0424#
root@ubuntu:/media/sf_sharefile1/0424#
root@ubuntu:/media/sf_sharefile1/0424# python viprf.py
Tue Apr 25 02:05:07 2017    data.pyprof

         92641399 function calls in 1735.392 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    3.838    3.838 1735.392 1735.392 test.py:1(<module>)
    23657   12.800    0.001  737.380    0.031 test.py:56(process)
        1  116.865  116.865  707.404  707.404 split.py:34(split)
  2859835  177.701    0.000  288.880    0.000 PcapReader.py:17(read_packet)
    23657    3.919    0.000  286.704    0.012 test.py:16(readflow)
    23657  144.653    0.006  277.805    0.012 test.py:242(output_log)
    23657   32.476    0.001  254.870    0.011 test.py:136(loss_detection)
        3    4.504    1.501  241.326   80.442 split.py:19(writefiles)
   147421    3.419    0.000  150.388    0.001 test.py:267(retran_detection)
   116039    2.213    0.000  138.702    0.001 test.py:310(timegapdetection)
   116039   73.759    0.001  135.779    0.001 test.py:319(find_time_gap)
    47326  109.839    0.002  109.839    0.002 {open}
    24504   19.296    0.001  109.122    0.004 test.py:173(connection_detection)
    23668  108.537    0.005  108.537    0.005 {cPickle.load}
    23668  107.399    0.005  107.399    0.005 {cPickle.dump}
    23657   79.305    0.003   79.305    0.003 {posix.remove}
   237835   75.379    0.000   75.379    0.000 test.py:225(find_previous_pkt)
    47326   66.311    0.001   66.311    0.001 {method 'close' of 'file' objects}
 11879857   60.057    0.000   60.057    0.000 test.py:327(isRetran)
    31382   59.685    0.002   59.685    0.002 test.py:232(find_highest_ack_before)
    47325    1.128    0.000   46.777    0.001 /usr/lib/python2.7/genericpath.py:55(getsize)
    47325   45.649    0.001   45.649    0.001 {posix.stat}
  8727743   42.817    0.000   42.817    0.000 test.py:333(islikelyRetran)
```

Figure 21 Profiler results

The result shows the total time of running tcpla in this CRAWDAD file is around half an hour. The form is ordered by cumulative time of each function. The cumulative time represents the total time of operating the functions. The two functions ‘process’ and ‘split’ in the blue box of Figure 21 are the main functions of tcpla. The ‘process’ function is to analyse the flow and the ‘split’ function is to split the trace files. With one result after another result from profiler, the running time decreased continuously. The running time of final version tcpla dropped over 50% comparing with the initial version.

Figure 22 is a part of the graph output of tcpla program by using ‘gprof2dot’ program to translate the profiler result. It illustrates that there are three part, split, readflow and process, constitute the mainline of tcpla. The ‘split’ procedure cost most of the running time, approximately 40.87%. The ‘process’ procedure is the loss identifying part, approximately 37.5%.

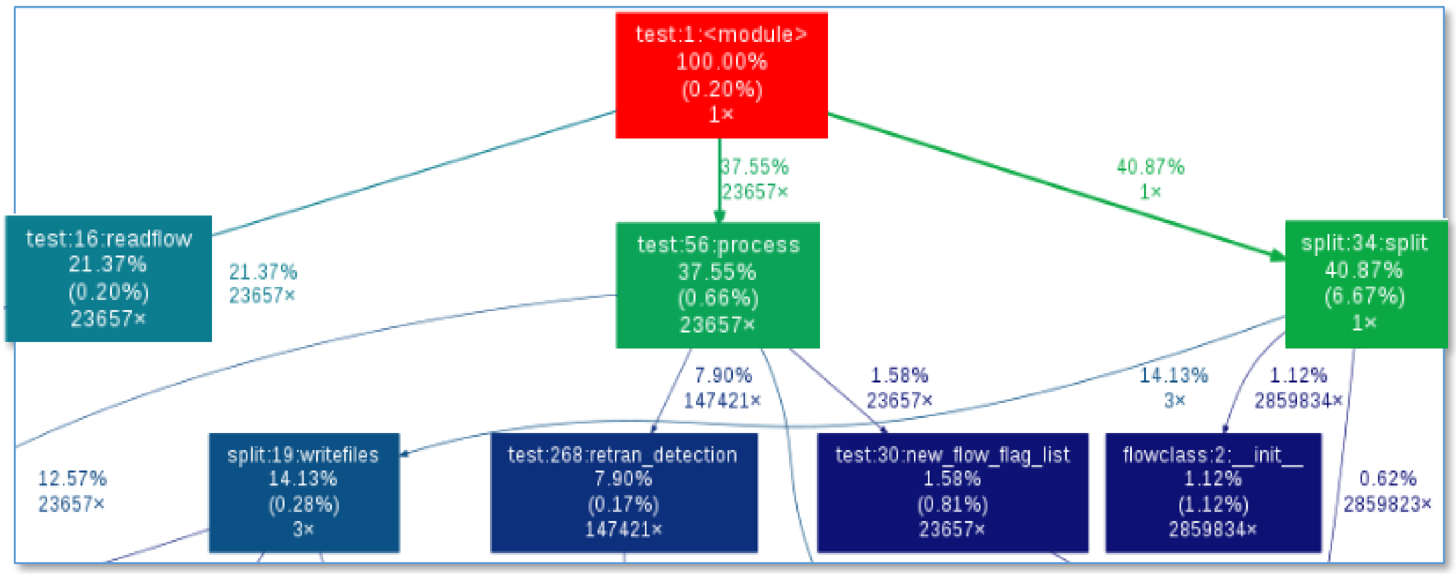


Figure 22 part of the translation graph of profiler result

Some other programs can identify retransmission and reordering packets in a high speed as it is enough for only recording the highest SEQ or ACK. It is more quickly to just identify retransmission and reordering. However, for the loss detection, the program needs all packets in this flow to determine whether this packet is acknowledged in somewhere or not.

To summarize, the analysis part is only occupied 1/3 of the total running time while 'split' procedure costs more, but the 'split' procedure is necessary and crucial because it needs the whole flow to identify loss.

Chapter 5: Conclusion and Further Work

5.1 Conclusion

This report has presented a complete research and design of a program for identifying TCP reordering and loss in networks. As the research goals described in section 1.3, the program tcpla has been developed using the new algorithms and has been tested in the large trace files. The results of the tcpla tool have been proved right and the efficiency of running tcpla has been improved to an acceptable level.

Literature survey has been made in advance and a literature review has been written during the learning of papers. The difficulties of this project are described in 2.2. Bearing all these requirements and complications of project in mind, two algorithms were developed and two flowcharts were drawn, including retransmission detection and loss detection. The retransmission detection is to discover the out-of-order packet and then identify whether it is retransmission or just reordering packet. After the retransmission detection of the whole flow, loss detection would analyse the retransmission packet and identify whether the retransmission is caused by reordering or real loss. Next, the algorithms were implemented to a python program. With testing and improving the program repeatedly, tcpla meets all the requirement of identifying reordering and loss. The program can split the input of network trace file into individual flows firstly and then analyse the loss status in each flow. The program would generate an output log file that contains the flow 'name', estimated round trip time, loss and reordering packets information and the loss rate in each flow. Finally, the tcpla was tested on a large-scale file derived from CRAWDAD website aiming to improve the efficiency and fix all bugs. According to the running time and the result, six methods have been attempted and compared. Profiler in python was used to analyse the performance of tcpla. With the improvement of the program, running time has been decreased by over 50%, to approximately half an hour. To get some experimental results, traffic control module in Linux has been used to configure the loss rate so that the developer can check whether tcpla can recover the loss rate from the settings value. The experimental results are sensible, repeatable and checkable.

To complete the project, a python program is achieved by using Libpcap, and in which both running the program and checking the output file is in a user friendly way. All the main tasks and targets have been satisfied according to the Gantt chart proposed in the project specification. However, before becoming a common-used tool in TCP networks, there are still some aspects described in section 5.2 need to be improved.

5.2 Further Work

As the description in 4.2, now the tcpla would spend around half an hour to analyse a 280 MB trace files. Although the running time has been shortened to over 50%, it should be better if it is reduced to a few minutes. Therefore, in future, tcpla should be improved in both ‘split’ and ‘process’ procedure, including reduce the complexity and optimize the algorithms.

The algorithms need the whole flow to identify the loss packet but the popular programs were not done by this way. However, it is impossible to 100% determine whether a packet lost if the program does not have the whole flow. Therefore, if tcpla could have a balance between accuracy and efficiency, in other words if the efficiency is improved with an almost correct result at the same time, the running time would be reduced to a few minutes and does not loss much accuracy.

Moreover, OS stack fingerprinting instructed in 2.3.2 can be used to help identifying the loss in future. There are many different TCP modes in the world. If tcpla knows the mode of the current TCP flows, it could induce the habit of the TCP sender and receiver by OS fingerprint. When the behaviour of the network is known before analysis, the analyser should have the ability to model the expected changes in window size with respect calculation. Therefore, loss packet would be detected more efficiently as long as its information can be brought from the OS Fingerprint.

References

[1] Paxson, V. E. (1997). Measurements and Analysis of End-to-End Internet Dynamics. University of California at Berkeley.

[2] J. Postel. Transmission Control Protocol. IETF RFC 793, Sep. 1981.

[3] Corbet, Jonathan. "Increasing the TCP initial congestion window". LWN. Retrieved 10 October 2012.

[4] Swaro, J. (2015). A Heuristic-Based Approach to Real-Time TCP State and Retransmission Analysis.

[5] Van Jacobson, Craig Leres and Steven McCanne, all of the Lawrence Berkeley National Laboratory, University of California, Berkeley, CA. Available from: http://www.tcpdump.org/.

[6] Vivian Lee. (2011). The Format of PCAP File Analysis and Decoding Code. Available from: http://blog.csdn.net/in7deforever/article/details/6460595.

[7] Philippe Biondi. (2016). Scapy: the python-based interactive packet manipulation program & library. Available from: http://www.secdev.org/projects/scapy/.

[8] Martin A. Brown. (2006). Linux Traffic Control. http://www.tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html.

[9] Rewaskar, S., Kaur, J., & Smith, F. D. (2006). A passive state-machine approach for accurate analysis of tcp out-of-sequence segments. Acm Sigcomm Computer Communication Review, 36(3), 51-64.

[10] Medeiros, J. P. S., Jr, A. M. B., & Pires, P. S. M. (2010). An Effective TCP/IP Fingerprinting Technique Based on Strange Attractors Classification. Data Privacy Management and Autonomous Spontaneous Security. Springer Berlin Heidelberg.

[11] M. Zalewski. (2012). p0f v3: Passive Fingerprinter. Available: http://lcamtuf.coredump.cx/p0f3.

[12] Jaiswal, S., Iannaccone, G., Diot, C., Kurose, J., & Towsley, D. (2002). Measurement and classification of out-of-sequence packets in a tier-1 IP backbone. ACM SIGCOMM Workshop on Internet Measurement 2002, Marseille, France, November (Vol.15, pp.113-114). DBLP.

[13] CRAWDAD website. Available: http://www.crawdad.org/.

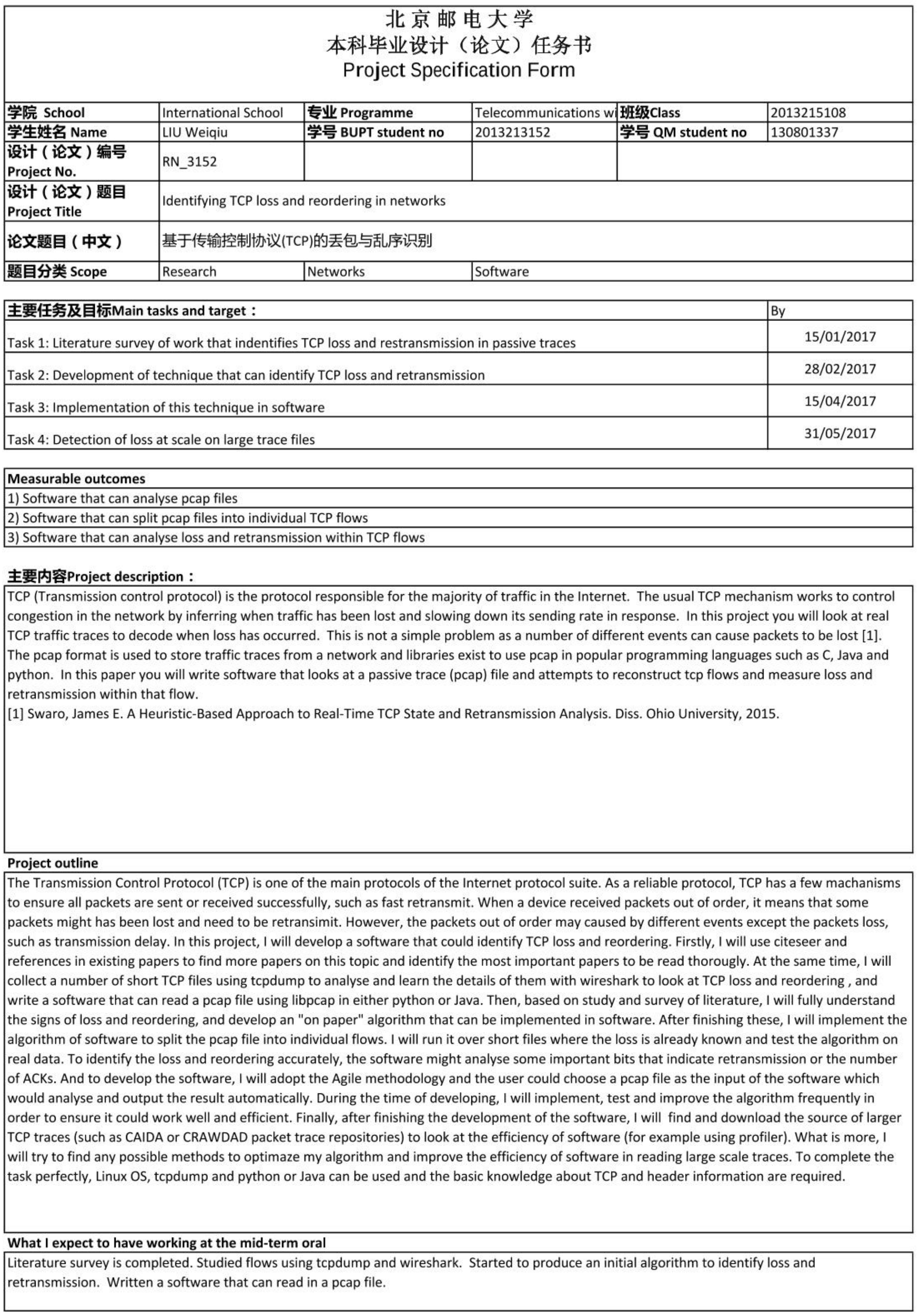
Acknowledgement

First, I would like to thank my supervisor for his invaluable guidance and support over the course of my project. He not only has the ability to predict the difficulties of the project, but also gives me many ideas and advices. He is very responsible for all students in his group as he hold the online meeting every week. With his weekly motivation and encouragement, I worked on this project efficiently and effectively.

Secondly, I would like to thank the administrator of CRAWDAD website who gave me a valid account to download the experimental data files. These data files helped me to improve the accuracy and efficiency of my program.

Finally, I would also like to thank my parents, my classmates and my girlfriend who have supported me in all my pursuits.

Appendix



北京邮电大学
本科毕业设计（论文）任务书
Project Specification Form

学院 School	International School	**专业 Programme**	Telecommunications wi	**班级Class**	2013215108
学生姓名 Name	LIU Weiqiu	**学号 BUPT student no**	2013213152	**学号 QM student no**	130801337
设计（论文）编号 Project No.	RN_3152				
设计（论文）题目 Project Title	Identifying TCP loss and reordering in networks				
论文题目（中文）	基于传输控制协议(TCP)的丢包与乱序识别				
题目分类 Scope	Research	Networks	Software		

主要任务及目标Main tasks and target：	By
Task 1: Literature survey of work that indentifies TCP loss and restransmission in passive traces	15/01/2017
Task 2: Development of technique that can identify TCP loss and retransmission	28/02/2017
Task 3: Implementation of this technique in software	15/04/2017
Task 4: Detection of loss at scale on large trace files	31/05/2017

Measurable outcomes
1) Software that can analyse pcap files
2) Software that can split pcap files into individual TCP flows
3) Software that can analyse loss and retransmission within TCP flows

主要内容Project description：

TCP (Transmission control protocol) is the protocol responsible for the majority of traffic in the Internet. The usual TCP mechanism works to control congestion in the network by inferring when traffic has been lost and slowing down its sending rate in response. In this project you will look at real TCP traffic traces to decode when loss has occurred. This is not a simple problem as a number of different events can cause packets to be lost [1]. The pcap format is used to store traffic traces from a network and libraries exist to use pcap in popular programming languages such as C, Java and python. In this paper you will write software that looks at a passive trace (pcap) file and attempts to reconstruct tcp flows and measure loss and retransmission within that flow.

[1] Swaro, James E. A Heuristic-Based Approach to Real-Time TCP State and Retransmission Analysis. Diss. Ohio University, 2015.

Project outline

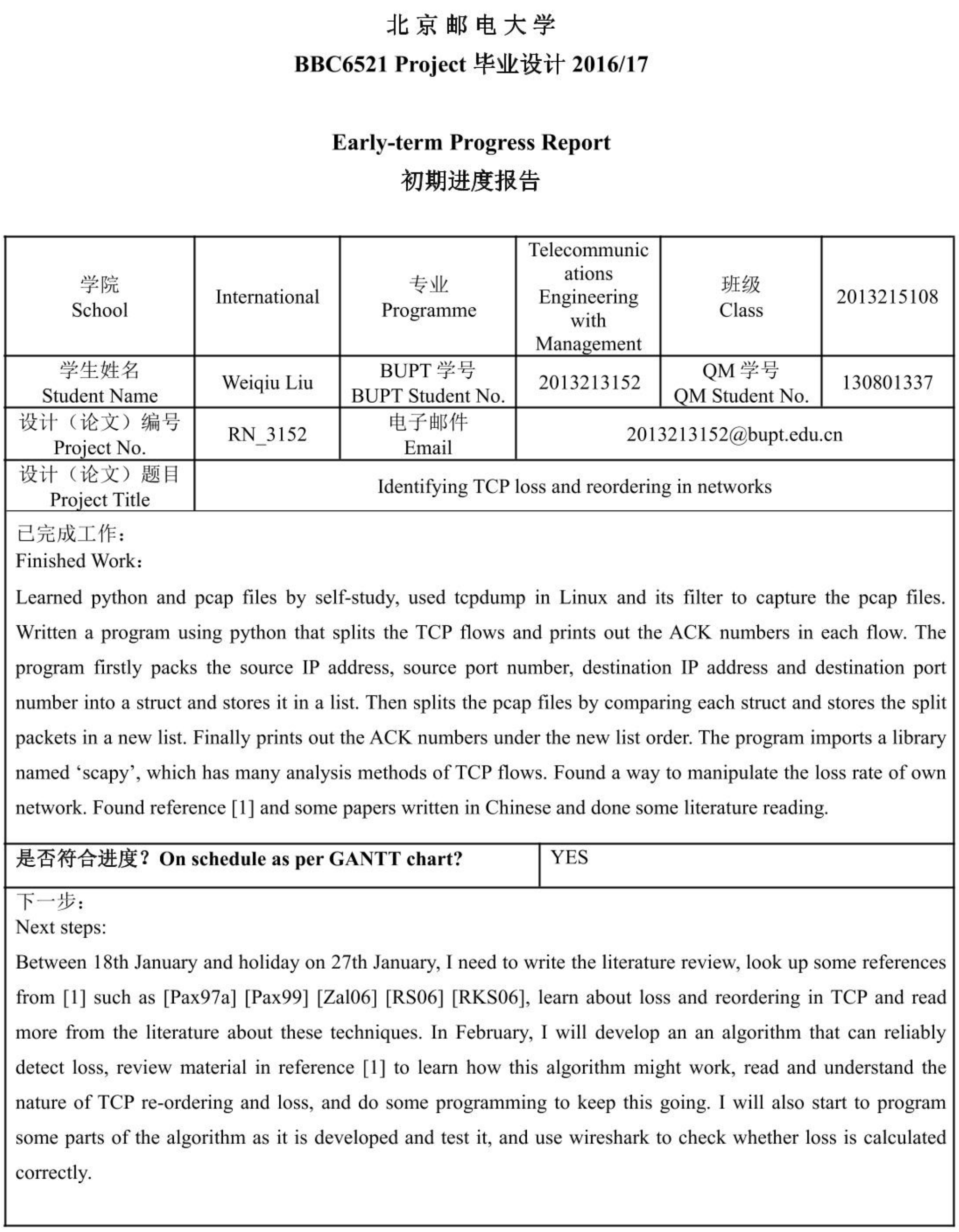
The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. As a reliable protocol, TCP has a few machanisms to ensure all packets are sent or received successfully, such as fast retransmit. When a device received packets out of order, it means that some packets might has been lost and need to be retransimit. However, the packets out of order may caused by different events except the packets loss, such as transmission delay. In this project, I will develop a software that could identify TCP loss and reordering. Firstly, I will use citeseer and references in existing papers to find more papers on this topic and identify the most important papers to be read thorougly. At the same time, I will collect a number of short TCP files using tcpdump to analyse and learn the details of them with wireshark to look at TCP loss and reordering , and write a software that can read a pcap file using libpcap in either python or Java. Then, based on study and survey of literature, I will fully understand the signs of loss and reordering, and develop an "on paper" algorithm that can be implemented in software. After finishing these, I will implement the algorithm of software to split the pcap file into individual flows. I will run it over short files where the loss is already known and test the algorithm on real data. To identify the loss and reordering accurately, the software might analyse some important bits that indicate retransmission or the number of ACKs. And to develop the software, I will adopt the Agile methodology and the user could choose a pcap file as the input of the software which would analyse and output the result automatically. During the time of developing, I will implement, test and improve the algorithm frequently in order to ensure it could work well and efficient. Finally, after finishing the development of the software, I will find and download the source of larger TCP traces (such as CAIDA or CRAWDAD packet trace repositories) to look at the efficiency of software (for example using profiler). What is more, I will try to find any possible methods to optimaze my algorithm and improve the efficiency of software in reading large scale traces. To complete the task perfectly, Linux OS, tcpdump and python or Java can be used and the basic knowledge about TCP and header information are required.

What I expect to have working at the mid-term oral

Literature survey is completed. Studied flows using tcpdump and wireshark. Started to produce an initial algorithm to identify loss and retransmission. Written a software that can read in a pcap file.

Fill in the sub-tasks and select the cells to show the extent of each task

	Nov		Dec		Jan		Feb		Mar		Apr		May	
Task 1: Literature survey of work that indentifies TCP loss and restransmission in passive traces														
Locate set of papers related to detection of TCP loss and retransmission in passive traces	■	■												
Use citeseer and references in existing papers to find more papers on this topic		■	■											
Identify the most important papers to be read thorougly			■	■										
Complete literature survey				■	■									
Task 2: Development of technique that can identify TCP loss and retransmission														
Collect a number of short TCP files to analyse using tcpdump		■												
Study these files with wireshark to look at TCP loss and reordering			■	■										
Understand signs of loss and reordering based on study and survey of literature						■	■							
Develop an "on paper" algorithm for TCP loss and reordering that can be implemented in task 3							■	■						
Task 3: Implementation of this technique in software														
Write software that can read a file from TCP dump using libpcap in either python or Java		■	■											
Write software that can split the pcap file into packets that come from individual TCP flows				■	■	■								
Implement the algorithm from task 2 part 4 and run it over short files where the loss is already known									■	■				
Test the algorithm on real data									■	■				
Task 4: Detection of loss at scale on large trace files														
Find the source of larger TCP traces (such as CAIDA or CRAWDAD packet trace repositories)										■	■			
Download and analyse larger packet traces										■	■			
Look at the efficiency of software (for example using a profiler)											■	■		
Improve the efficiency of software in reading large scale traces												■	■	



北 京 邮 电 大 学

BBC6521 Project 毕业设计 2016/17

Early-term Progress Report

初期进度报告

学院 School	International	专业 Programme	Telecommunications Engineering with Management	班级 Class	2013215108
学生姓名 Student Name	Weiqiu Liu	BUPT 学号 BUPT Student No.	2013213152	QM 学号 QM Student No.	130801337
设计（论文）编号 Project No.	RN_3152	电子邮件 Email	2013213152@bupt.edu.cn		
设计（论文）题目 Project Title	Identifying TCP loss and reordering in networks				

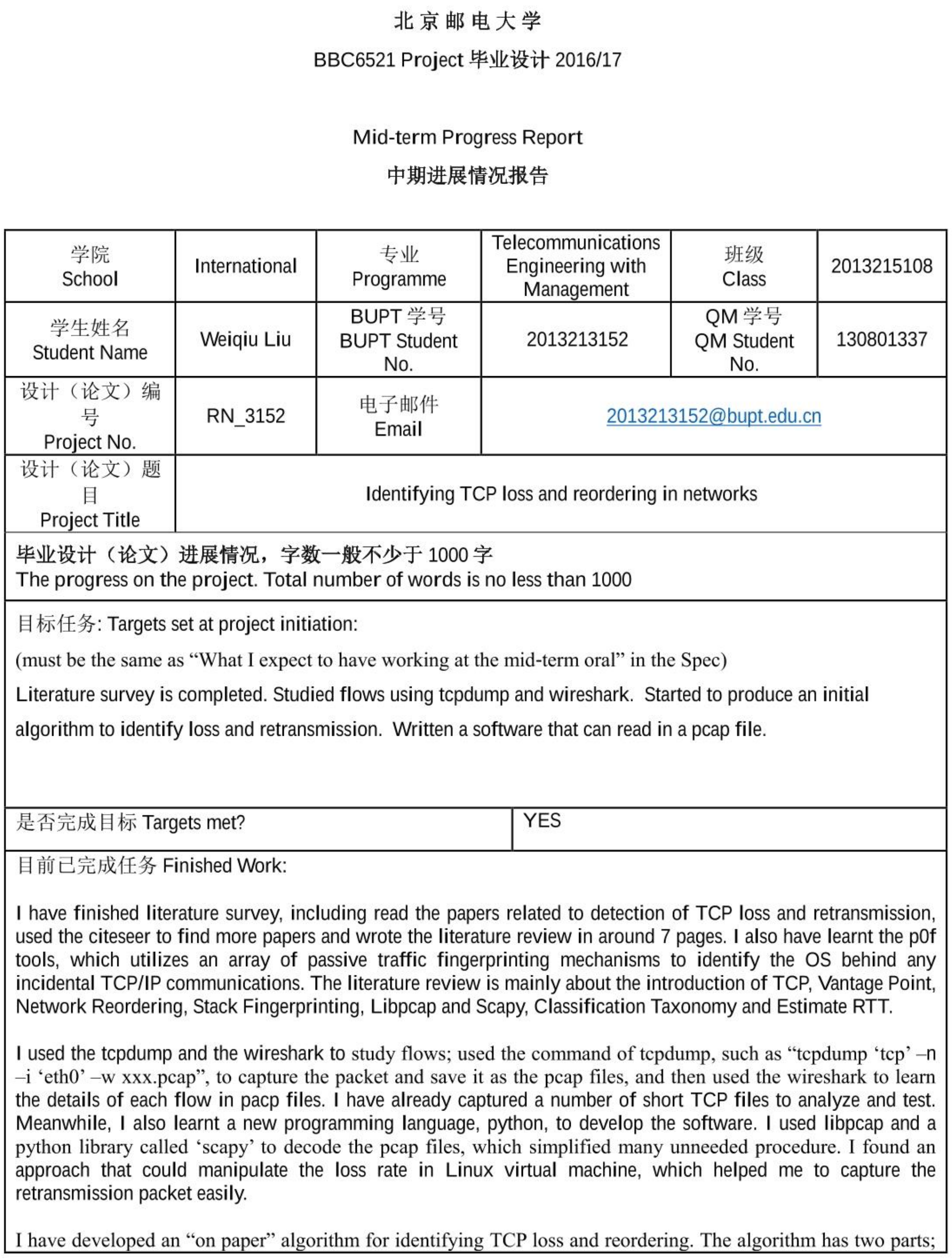
已完成工作：
Finished Work：

Learned python and pcap files by self-study, used tcpdump in Linux and its filter to capture the pcap files. Written a program using python that splits the TCP flows and prints out the ACK numbers in each flow. The program firstly packs the source IP address, source port number, destination IP address and destination port number into a struct and stores it in a list. Then splits the pcap files by comparing each struct and stores the split packets in a new list. Finally prints out the ACK numbers under the new list order. The program imports a library named 'scapy', which has many analysis methods of TCP flows. Found a way to manipulate the loss rate of own network. Found reference [1] and some papers written in Chinese and done some literature reading.

是否符合进度？On schedule as per GANTT chart?	YES

下一步：
Next steps:

Between 18th January and holiday on 27th January, I need to write the literature review, look up some references from [1] such as [Pax97a] [Pax99] [Zal06] [RS06] [RKS06], learn about loss and reordering in TCP and read more from the literature about these techniques. In February, I will develop an an algorithm that can reliably detect loss, review material in reference [1] to learn how this algorithm might work, read and understand the nature of TCP re-ordering and loss, and do some programming to keep this going. I will also start to program some parts of the algorithm as it is developed and test it, and use wireshark to check whether loss is calculated correctly.



北京邮电大学

BBC6521 Project 毕业设计 2016/17

Mid-term Progress Report

中期进展情况报告

学院 School	International	专业 Programme	Telecommunications Engineering with Management	班级 Class	2013215108
学生姓名 Student Name	Weiqiu Liu	BUPT 学号 BUPT Student No.	2013213152	QM 学号 QM Student No.	130801337
设计（论文）编号 Project No.	RN_3152	电子邮件 Email	2013213152@bupt.edu.cn		
设计（论文）题目 Project Title	Identifying TCP loss and reordering in networks				

毕业设计（论文）进展情况，字数一般不少于 1000 字
The progress on the project. Total number of words is no less than 1000

目标任务: Targets set at project initiation:

(must be the same as “What I expect to have working at the mid-term oral” in the Spec)

Literature survey is completed. Studied flows using tcpdump and wireshark. Started to produce an initial algorithm to identify loss and retransmission. Written a software that can read in a pcap file.

是否完成目标 Targets met?	YES

目前已完成任务 Finished Work:

I have finished literature survey, including read the papers related to detection of TCP loss and retransmission, used the citeseer to find more papers and wrote the literature review in around 7 pages. I also have learnt the p0f tools, which utilizes an array of passive traffic fingerprinting mechanisms to identify the OS behind any incidental TCP/IP communications. The literature review is mainly about the introduction of TCP, Vantage Point, Network Reordering, Stack Fingerprinting, Libpcap and Scapy, Classification Taxonomy and Estimate RTT.

I used the tcpdump and the wireshark to study flows; used the command of tcpdump, such as “tcpdump ‘tcp’ –n –i ‘eth0’ –w xxx.pcap”, to capture the packet and save it as the pcap files, and then used the wireshark to learn the details of each flow in pacp files. I have already captured a number of short TCP files to analyze and test. Meanwhile, I also learnt a new programming language, python, to develop the software. I used libpcap and a python library called ‘scapy’ to decode the pcap files, which simplified many unneeded procedure. I found an approach that could manipulate the loss rate in Linux virtual machine, which helped me to capture the retransmission packet easily.

I have developed an “on paper” algorithm for identifying TCP loss and reordering. The algorithm has two parts;

one for retransmission detection and another one for loss detection. Retransmission detection could identify retransmission and reordering packet, and the input of this part is the packet out-of-order. Loss detection could judge whether loss occur or not, and the input is the retransmission packet. I have also draw two flow charts for each part and wrote some detailed annotations in each decision point.

Until now, the software I developed could read the pcap files and split the pcap files into packets that comes from individual TCP flows. It uses the source IP, source port number, destination IP and destination port number to identify different flows. Moreover, I have also implemented part of the algorithm above so that the software could find out which packets are the retransmissions and estimate the RTT in each flow. When running the software, it would generate some log files that contains the information of the properties of packets in each flow. For instance, packet 1: '0x1000' will be written in log file if the packet 1 is a retransmission packet. When the software finished running, I read the log file to check whether the results is correct by comparing with the result from the wireshark. Most of the log files got the same answers with the wireshark, excluding some reordering packets.

尚需完成的任务 Work to do:

Firstly, I need to create some new categories, such as "probably retransmission" and "probably loss". Then, for flowchart, I need to spell everything out 100% clearly and describe the process fully. I will use some mathematical equations and precise sentences to spell out A, B, C and D in flow chart. Moreover, I will read another paper on this subject related to the loss detection. Initially I only look at data loss, so I will consider about the ACK loss and add the ACK loss detection into the algorithm.

For software, I will continue to implement the algorithms into codes. I will modify or change the codes as the algorithm developed. The software will be able to reach the title of the project-"identifying TCP loss and reordering in networks". I will also spend times in making the log files more readable. When finished developing software, I will use some data files to test it continually and find all bugs to improve. In addition, I will write the comments and a user manual for about the details of how to use this software.

After the function of the software has been realized, I will find the source of larger TCP traces, such as CAIDA or CRAWDAD packet trace repositories to experiment. I will try to look at the efficiency of software (using a profiler) and improve the efficiency of the software in reading large scale traces. I will consider that what happens when the program runs out of memory and how things can be made more efficient. At the end of the project, I will finish the project paper and prepare fully for the final viva.

能否按期完成设计（论文） Can finish the project on time or not:	YES

存在问题 Problems:

1. The annotation in the flow chart is not very clear. This makes it hard to understand my algorithms.
2. The blocks of the flow chart are mainly described in literature. It cannot make other people understand what I say directly.
3. The algorithm is not very accurate because there are some situations in retransmission and loss cannot be identified definitely.
4. The algorithm of loss detection only considered the data loss but not the ACK loss.
5. The software has not been tested on a large scale trace file and might have poor efficiency.

拟采取的办法 Solutions:

1. I should improve the annotations to make sure everything is spelled 100% clearly.
2. I should add some mathematical equations and precise sentences into the blocks to describe the process fully.
3. I should use more categories for classifying accurately, such as "probably retransmission" and "probably loss".
4. I should consider the flow thoroughly and add the ACK loss detection to the algorithm.
5. I should download datasets from CAIDA and CRAWDAD and use them to experiment. I should try to improve the efficiency according to the experimental results.

最终论文结构 Structure of the final report:

Identifying TCP loss and reordering in networks

日期 Date:	4th March, 2017

Risk Assessment

This project aims design a python program to identify TCP loss and reordering in networks. Since it is a software running in Linux, risk can come from factors shown in Table 2.

Table 2 Risk Assessment of the Project

Description of Risk	Description of Impact	Likelihood rating	Impact rating	Preventative actions
RAM overflow	The program will be stopped	Rare	Catastrophic	Ensure you have at least 100 MB free RAM if you input a extreme large file
Insufficient storage space in system	The program will be stopped	Rare	Catastrophic	Ensure you have at least twice the size of the input file storage space
Operation System crash	The program will be stopped	Unlikely	Catastrophic	Reboot your system and try again
Libpcap updates to not support part of the current code	The program unable to run	Rare	Catastrophic	Update the code of program when Libpcap updates

Environmental Impact Assessment

The project is a program-based so that has no impact on environment.