MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Mini Project Report

of

Computer Networks LAB

CAPTURING RETRANSMITTED PACKETS

SUBMITTED

BY

NAME	REG NO	ROLL NO	SECTION
Prathik Jayaraja Shetty	200905045	11	B
Utkarsh Tawakley	200905071	16	B

ABSTRACT

To capture packets that get retransmitted in a TCP connection.

Using a C to read a pcap file and read packet by packet using the <pcap.h> header file.

Checks if the packet is like a previously transmitted packet by comparing TCP/IP headers.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 General Introduction

TCP:

Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite and it lies in the Transport Layer. TCP is used for reliable data transfer as it is a connection-oriented protocol. A connection between client and server is established before data transfer occurs.

TCP DATAGRAM:

Source Port Number & Destination Port Number: The source and destination port numbers that identify the TCP connection.

Sequence Number: The sequence number of the first data byte in the packet. Unless the syn flag is set in which case the sequence number is the isn (initial sequence number) and the first data byte has a sequence number of isn + 1.

Acknowledgement Number: If the ack bit is set, this number is the next sequence number the sender of the packet expects to receive. After a connection is established, the ack bit will always be set and this number will always be used.

Ack (Acknowledgement) Flag: The ack flag says that the ack number is valid. The packet is at least an acknowledgement of data that has been received.

Receive Window Size: An advertisement of the amount of open space in the receive buffer for the given tcp connection.

Data: Data from whatever you're doing (telnet, ftp, http, and so on) goes here. Since TCP provides a reliable end-to-end stream of data to whatever your application is, it's up to you to implement your own protocol inside it, just like telnet and ftp have done.

For a successful 3-way-handshake connection to be established the following steps occur: The sender begins the process by sending sequence number and SYN flag. The receiver side sends a randomly generated random sequence number and a calculated acknowledgement number along with the SYN and ACK flags. Sender finally replies with a sequence number, acknowledgement number and ACK flag

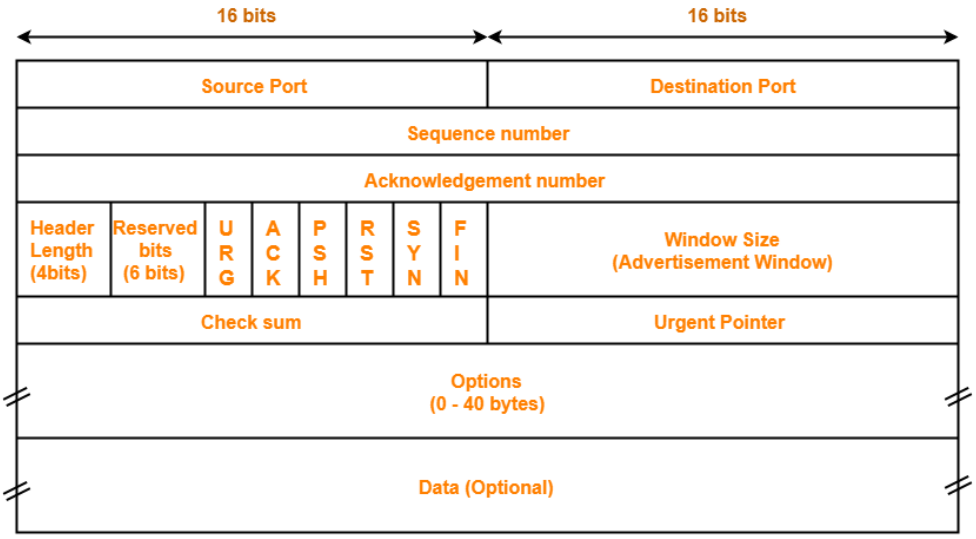

Figure 1.1: TCP Datagram Header

If any packets fail to reach their destination during transmission, it is called Packet Loss. Packet loss mainly occurs due to congestion or corruption of data. This is where TCP connection are advantageous as it detects packet loss and perform retransmissions to ensure reliable data transfer.

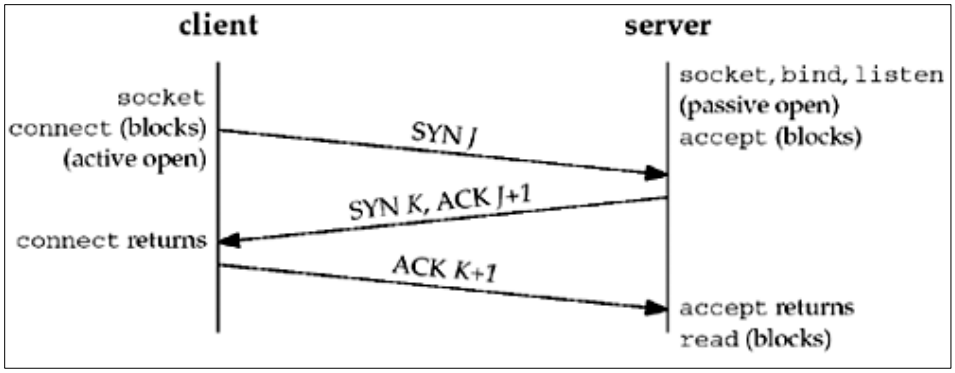

Figure 1.2: TCP Connection Establishment

IP:

Internet Protocol (IP) is a set of rules the governs the format of data sent on the internet. IP address is used for identification and location of a network device. The most used format is IPv4 which is a 32-bit number.

IPv4 Datagram Header:

Size of the header is 20 to 60 bytes.

Version: Version of the IP protocol (4 bits), which is 4 for IPv4

HLEN: IP header length (4 bits), which is the number of 32-bit words in the header. The minimum value for this field is 5 and the maximum is 15.

Type Of Service: Low Delay, High Throughput, Reliability (8 bits)

Total Length: Length of header + Data (16 bits), which has a minimum value 20 bytes, and the maximum is 65,535 bytes.

Identification: Unique Packet Id for identifying the group of fragments of a single IP datagram (16 bits)

Flags: 3 flags of 1 bit each: reserved bit (must be zero), do not fragment flag, more fragments flag (same order).

Protocol: Name of the protocol to which the data is to be passed (8 bits)

Source IP Address: 32 bits IP address of the sender

Destination IP Address: 32 bits IP address of the receiver

Option: Optional information such as source route, record route. Used by the Network administrator to check whether a path is working or not.

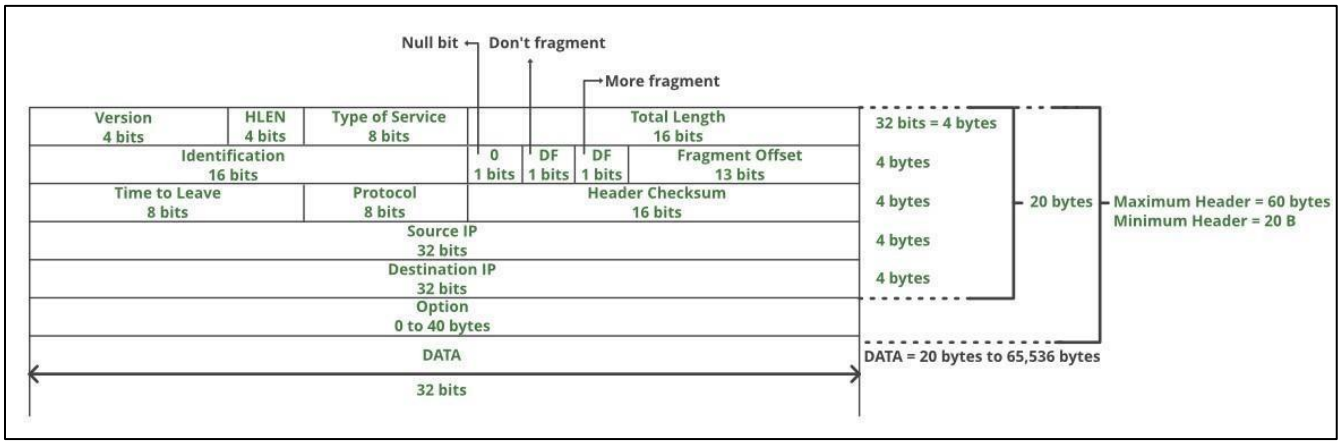

Figure 1.3: IP Datagram Header

1.2 Hardware and Software Requirements

A PC with the latest Linux OS, internet access, Wireshark, C text editor, terminal.

2. PROBLEM DEFINITION

Design a tool to capture the retransmitted packets and figure out how many packets sent and received were retransmitted.

3. OBJECTIVES

1. Read packet information from a .pcap file.
2. Check whether the packet currently being sniffed is being retransmitted.
3. Print details of the packet.

4. METHODOLOGY

SOURCE CODE:

```
#include <pcap.h>

#include <stdio.h>

#include <netinet/ip.h>

#include <netinet/tcp.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <net/ethernet.h>

#include <string.h>
```

```
void process_packet(u_char *,const struct pcap_pkthdr * , const u_char *);
void find_retransmissions(const u_char * , int );
int main()
{
    pcap_t *handle;
    char errbuff[PCAP_ERRBUF_SIZE];
    handle = pcap_open_offline("smallFlows.pcap", errbuff);
    pcap_loop(handle, -1, process_packet, NULL);
}
void process_packet(u_char *args,const struct pcap_pkthdr * header,const u_char *buffer)
{
    int size = header->len;
    struct ethhdr *eth = (struct ethhdr *)buffer;
    if(eth->h_proto == 8) //Check if IPv4
    {
        struct iphdr *iph = (struct iphdr*)(buffer +sizeof(struct ethhdr));
        if(iph->protocol == 6) //Check if TCP
        {
```

```
            find_retransmissions(buffer,size);
        }
    }
}
void find_retransmissions(const u_char * Buffer, int Size)
{
    static struct iphdr  previous_packets[20000];
    static struct tcphdr  previous_tcp[20000];
    static int index = 0;
    static int retransmissions = 0;
    int retransmission = 0;
    struct sockaddr_in source,dest;
    unsigned short iphdrlen;
    // IP header
    struct iphdr *iph = (struct iphdr *)(Buffer  + sizeof(struct ethhdr));
    previous_packets[index] = *iph;
    iphdrlen =iph->ihl*4;
    memset(&source, 0, sizeof(source));
```

```
source.sin_addr.s_addr = iph->saddr;

memset(&dest, 0, sizeof(dest));

dest.sin_addr.s_addr = iph->daddr;

// TCP header

struct tcphdr *tcph=(struct tcphdr*)(Buffer + iphdrlen + sizeof(struct ethhdr));

previous_tcp[index]=*tcph;

index++;

int header_size =  sizeof(struct ethhdr) + iphdrlen + tcph->doff*4;

unsigned int segmentlength;

segmentlength = Size - header_size;

/* First check if a same TCP packet has been received */

for(int i=1;i<index-1;i++)

{

   // Check if packet has been resent

   unsigned short temphdrlen;

   temphdrlen = previous_packets[i].ihl*4;

   // First check IP header

   if ((previous_packets[i].saddr == iph->saddr) // Same source IP address
```

```
    && (previous_packets[i].daddr == iph->daddr) // Same destination Ip address
    && (previous_packets[i].protocol == iph->protocol) //Same protocol
    && (temphdrlen == iphdrlen)) // Same header length
{
    // Then check TCP header
    if((previous_tcp[i].source == tcph->source) // Same source port
        && (previous_tcp[i].dest == tcph->dest) // Same destination port
        && (previous_tcp[i].th_seq == tcph->th_seq) // Same sequence number
        && (previous_tcp[i].th_ack==tcph->th_ack) // Same acknowledge number
        && (previous_tcp[i].th_win == tcph->th_win) // Same window
        && (previous_tcp[i].th_flags == tcph->th_flags) // Same flags
        && (tcph->syn==1 || tcph->fin==1 ||segmentlength>0)) // Check if SYN or FIN are
    {                                                       // set or if tcp.segment 0
        // At this point the packets are almost identical
        //  Now Check previous communication to check for retransmission
        for(int z=index-1;z>=0;z--)
        {
            // Find packets going to the reverse direction
```

```
if ((previous_packets[z].daddr == iph->saddr) // Swapped IP source addresses
   && (previous_packets[z].saddr ==iph->daddr) // Same for IP dest addreses
   && (previous_packets[z].protocol == iph->protocol)) // Same protocol
{
   if((previous_tcp[z].dest==tcph->source) // Swapped ports
      && (previous_tcp[z].source==tcph->dest)
      && (previous_tcp[z].th_seq-1 != tcph->th_ack) // Not Keepalive
      && (tcph->syn==1          // Either SYN is set
         || tcph->fin==1        // Either FIN is set
         || (segmentlength>0)) // Either segmentlength >0
      && (previous_tcp[z].th_seq>tcph->th_seq) // Next sequence number is
                                    // bigger than the expected
      && (previous_tcp[z].ack  != 1))  // Last seen ACK is set
   {
      retransmission = 1;
      retransmissions++;
      break;
   }
```

```
                }

            }

        }

    }

    if (retransmission == 1)

    {

        printf("\n\n******************IPv4 TCP Packet******************\n");

        printf("    | IP Version       : %d\n",(unsigned int)iph->version);

        printf("    | Source IP        : %s\n" , inet_ntoa(source.sin_addr) );

        printf("    | Destination IP   : %s\n" , inet_ntoa(dest.sin_addr) );

        printf("    | Source Port      : %u\n",  ntohs(tcph->source));

        printf("    | Destination Port : %u\n",  ntohs(tcph->dest));

        printf("    | Protocol         : %d\n",(unsigned int)iph->protocol);

        printf("    | Payload Length   : %d Bytes\n",Size - header_size);

        printf("\nTotal Retransmissions: %d\n",retransmissions);

    }

}
```

5. IMPLEMENTATION DETAILS

A .pcap file is generated from Wireshark from a client-server program that has multiple retransmissions occurring.

OUTPUT:

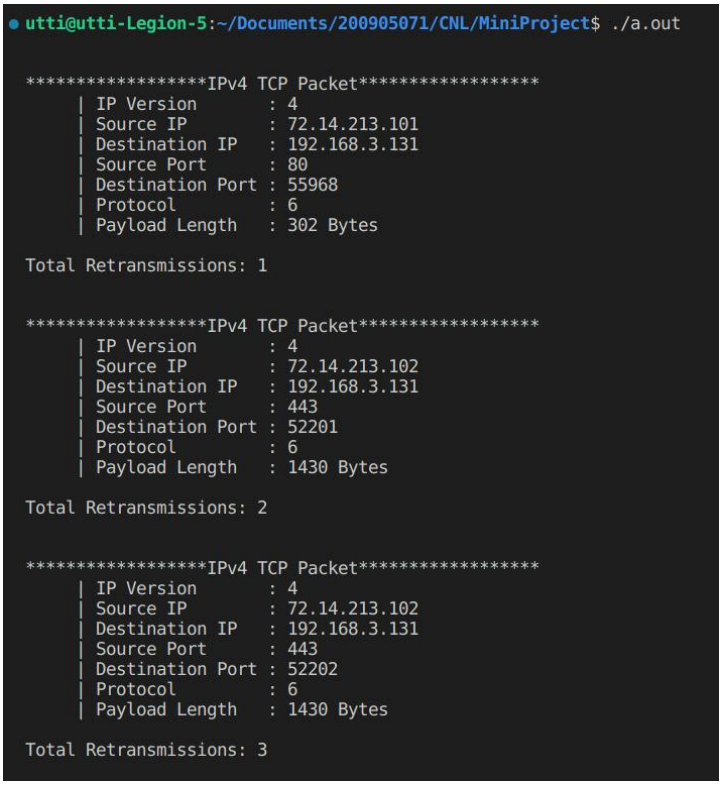

Figure: 5.1

```
*****************IPv4 TCP Packet*****************
     | IP Version       : 4
     | Source IP        : 72.14.213.102
     | Destination IP   : 192.168.3.131
     | Source Port      : 443
     | Destination Port : 52203
     | Protocol         : 6
     | Payload Length   : 1430 Bytes

Total Retransmissions: 4


*****************IPv4 TCP Packet*****************
     | IP Version       : 4
     | Source IP        : 72.14.213.102
     | Destination IP   : 192.168.3.131
     | Source Port      : 443
     | Destination Port : 52205
     | Protocol         : 6
     | Payload Length   : 1430 Bytes

Total Retransmissions: 5


*****************IPv4 TCP Packet*****************
     | IP Version       : 4
     | Source IP        : 72.14.213.102
     | Destination IP   : 192.168.3.131
     | Source Port      : 443
     | Destination Port : 52207
     | Protocol         : 6
     | Payload Length   : 1430 Bytes

Total Retransmissions: 6
```

Figure: 5.2

```
     | Source IP        : 204.14.234.85
     | Destination IP   : 192.168.3.131
     | Source Port      : 443
     | Destination Port : 57246
     | Protocol         : 6
     | Payload Length   : 6 Bytes

Total Retransmissions: 42


*****************IPv4 TCP Packet*****************
     | IP Version       : 4
     | Source IP        : 204.14.234.85
     | Destination IP   : 192.168.3.131
     | Source Port      : 8443
     | Destination Port : 57246
     | Protocol         : 6
     | Payload Length   : 6 Bytes

Total Retransmissions: 43


*****************IPv4 TCP Packet*****************
     | IP Version       : 4
     | Source IP        : 204.14.234.85
     | Destination IP   : 192.168.3.131
     | Source Port      : 443
     | Destination Port : 57248
     | Protocol         : 6
     | Payload Length   : 6 Bytes

Total Retransmissions: 44


*****************IPv4 TCP Packet*****************
     | IP Version       : 4
     | Source IP        : 204.14.234.85
     | Destination IP   : 192.168.3.131
     | Source Port      : 8443
     | Destination Port : 57248
     | Protocol         : 6
     | Payload Length   : 6 Bytes

Total Retransmissions: 45
utti@utti-Legion-5:~/Documents/200905071/CNL/MiniProject$
```

Figure: 5.3

6. CONTRIBUTION SUMMARY

Prathik Shetty:

Implemented code to check if a packet was retransmitted.

Utkarsh Tawakley:

Implemented of code for reading packets from a .pcap file.

7. REFERENCES

- https://stackoverflow.com/questions/65238453/how-to-find-tcp-retransmissions-while-sniffing-packets-in-c
- https://en.wikipedia.org/wiki/Transmission_Control_Protocol#TCP_segment_structure
- https://www.geeksforgeeks.org/what-is-transmission-control-protocol-tcp/
- James F. Kurose & Keith W. Ross, "*Computer Networking A Top-Down Approach*", Pearson Education