

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

**Network Statistics such as Transmission Speed, Average RTT.**

**Project Report**

***Submitted by***

*Aaryan Gupta – 200905276*

*Kopal Dixit - 200905244*

**Abstract**

For our mini project, we are sniffing data packets passed to us in a .pcap/.pcapng file and calculating different network statistics. For each packet we print the host IP address, destination IP address, source port address, destination port address, packet capture length, packet total length, sequence number and acknowledgement number. This project is done using the C language. The libpcap library was written as part of a more extensive program called TCPDump. The libpcap library allowed developers to write code to receive link-layer packets (Layer 2 in the OSI model) on different flavours of UNIX operating systems without having to worry about the idiosyncrasy of different operating systems' network cards and drivers. In our program, we allow the user to input any .pcap file for which we output its statistics.

1. **Introduction**
   1. **General Introduction to topic**

The Internet’s transport layer transports application-layer messages between application endpoints. A transport-layer protocol provides for logical communication between application processes running on different hosts. Logical communication implies that from the application's perspective, it is as if the hosts running the processes are directly connected. On the sending side, the transport layer converts the application-layer messages it receives from a sending application process into transport-layer packets, known as transport-layer segments.

Two of the transport layer protocols are:

1. Transmission Control Protocol (TCP)

TCP is said to be connection-oriented because before one application process can begin to send data to another, the two processes must first "handshake" with each other. The TCP protocol runs only in the end systems and not in the intermediate network element (routers and link-layer switches), the intermediate network elements do not maintain TCP connection state.

A TCP connection provides a full-duplex service: If there is a TCP connection between process A on one host and process B on another host, then application layer data can flow from process A to process B at the same time as application layer data flows from process B to process A.

1. User Datagram Protocol (UDP)

UDP takes messages from the application process, attaches source and destination port number fields, adds two other small fields, and passes the resulting segmentto the network layer. The network layer encapsulates the transport-layer segment into an IP datagram and then makes a best-effort attempt to deliver the segment to the receiving host. If the segment arrives at the receiving host, UDP uses the destination port number to deliver the segment's data to the correct application process.

* 1. **Hardware and Software Requirements**

Hardware Requirements:

For this project, the hardware requirements were as follows:

* Processor – 2 cores
* Memory – 4GB
* HDD – 0.5TB
* TCP Ports

Software Requirements:

For this project, the software requirements were as follows:

* A code editor – VS Code
* Wireshark – For capturing packets
* GCC Compiler

**Problem Definition**

In this project we have tried to calculate the network statistics like average speed and average RTT.

What is RTT?

Round-trip time (RTT) is the duration, measured in milliseconds, from when a browser sends a request to when it receives a response from a server. It’s a key performance metric for web applications.

Factors Influencing RTT

Actual round trip time can be influenced by:

* Distance
* Transmission medium
* Number of network hops
* Traffic levels
* Server response time

**Objectives**

The objectives of our project are as follows:

* Network Analysis by scanning various IP addresses and the data associated with them. An Ip packet can be identified as  
  Ip header Tcp header Data.  
  This IP datagram packet incorporates in it UDP as well as TCP packet data.
* To monitor and provide TCP/UDP source and destination ports, header length, packet size etc for TCP/UDP packet analysis.
* To monitor IP packet data with various associated parameters with ease.

**Methodology**

We have used pcap library to capture or send packets from a live network device or a file. We have used wireshark to capture packets and check them. We used pcap library to get header files and different data fields which exist in a TCP and UDP header.

**Implementation**

We have developed a command line interface in order to analyze packets present in a pcap file and present the network statistics such as throughput, average round trip time and transmission speed, for those packets.

The interface was made in the C language. The following header files were used for the implementation:

* stdio.h
* pcap.h
* stdlib.h
* netinet/in.h
* netinet/tcp.h
* netinet/udp.h
* netinet/ip.h
* unistd.h
* string.h

We obtained the IP header information and the TCP/UDP header information using the pcap library.

**CODE**

#include <stdio.h>

#include <pcap.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <netinet/tcp.h>

#include <netinet/udp.h>

#include <netinet/ip.h>

#include <unistd.h>

#include <net/ethernet.h>

#include <assert.h>

#include <string.h>

#define SIZE\_ETHERNET 14

/\* IP header \*/

struct sniff\_ip

{

u\_char ip\_vhl; /\* version << 4 | header length >> 2 \*/

u\_char ip\_tos; /\* type of service \*/

u\_short ip\_len; /\* total length \*/

u\_short ip\_id; /\* identification \*/

u\_short ip\_off; /\* fragment offset field \*/

#define IP\_RF 0x8000 /\* reserved fragment flag \*/

#define IP\_DF 0x4000 /\* don't fragment flag \*/

#define IP\_MF 0x2000 /\* more fragments flag \*/

#define IP\_OFFMASK 0x1fff /\* mask for fragmenting bits \*/

u\_char ip\_ttl; /\* time to live \*/

u\_char ip\_p; /\* protocol \*/

u\_short ip\_sum; /\* checksum \*/

struct in\_addr ip\_src, ip\_dst; /\* source and dest address \*/

};

#define IP\_HL(ip) (((ip)->ip\_vhl) & 0x0f)

#define IP\_V(ip) (((ip)->ip\_vhl) >> 4)

typedef u\_int tcp\_seq;

struct sniff\_tcp

{

u\_short th\_sport; /\* source port \*/

u\_short th\_dport; /\* destination port \*/

tcp\_seq th\_seq; /\* sequence number \*/

tcp\_seq th\_ack; /\* acknowledgement number \*/

u\_char th\_offx2; /\* data offset, rsvd \*/

#define TH\_OFF(th) (((th)->th\_offx2 & 0xf0) >> 4)

u\_char th\_flags;

#define TH\_FIN 0x01

#define TH\_SYN 0x02

#define TH\_RST 0x04

#define TH\_PUSH 0x08

#define TH\_ACK 0x10

#define TH\_URG 0x20

#define TH\_ECE 0x40

#define TH\_CWR 0x80

#define TH\_FLAGS (TH\_FIN | TH\_SYN | TH\_RST | TH\_ACK | TH\_URG | TH\_ECE | TH\_CWR)

u\_short th\_win; /\* window \*/

u\_short th\_sum; /\* checksum \*/

u\_short th\_urp; /\* urgent pointer \*/

};

struct sniff\_udp

{

u\_short sport; //source port

u\_short dport; //destination port

u\_short len; //datagram length

u\_short crc; //checksum

};

/\* This function can be used as a callback for pcap\_loop() \*/

void my\_packet\_handler(

u\_char \*args,

const struct pcap\_pkthdr \*header,

const u\_char \*packet)

{

struct ether\_header \*eth\_header;

struct sniff\_tcp \*t\_header;

struct sniff\_ip \*ip\_header;

struct sniff\_udp \*udp\_header;

eth\_header = (struct ether\_header \*)packet;

ip\_header = (struct ip \*)(packet + SIZE\_ETHERNET);

u\_int size\_ip = IP\_HL(ip\_header) \* 4;

printf("\n---------PACKET---------\n");

if (ip\_header->ip\_p == 6)

printf("PROTOCOL:TCP\n");

if (ip\_header->ip\_p == 17)

{

printf("PROTOCOL:UDP\n");

}

printf("Packet length:%d\n", header->len);

printf("IP Length:%d\n", ip\_header->ip\_len / 256);

printf("IP header length:%d\n", size\_ip);

//TCP PACKET

if (ip\_header->ip\_p == 6)

{

t\_header = (struct tcp\_hdr \*)(packet + SIZE\_ETHERNET + size\_ip);

u\_int size\_tcp = TH\_OFF(t\_header) \* 4;

const char \*payload = (u\_char \*)(packet + SIZE\_ETHERNET + size\_ip + size\_tcp);

printf("TCP header length:%d\n", size\_tcp);

printf("TCP segment length:%d\n", ip\_header->ip\_len / 256 - (size\_ip + size\_tcp));

printf("Source port:%u\nDestination port:%u\n", t\_header->th\_sport, t\_header->th\_dport);

printf("FLAGS:0x%x\n", t\_header->th\_flags);

printf("Sequence number:%u\n", t\_header->th\_seq);

printf("Acknowledgement number:%u\n\n\n", t\_header->th\_ack);

}

//UDP PACKET

else if (ip\_header->ip\_p == 17)

{

udp\_header = (struct udp\_hdr \*)(packet + SIZE\_ETHERNET + size\_ip);

printf("Source port:%u\nDestination port:%u\n", udp\_header->sport, udp\_header->dport);

printf("UDP datagram length:%u\n", udp\_header->len / 256);

}

}

void fileParser()

{

FILE \*f;

f = fopen("output.txt", "r");

char buf[256];

char temp[256];

int count = 0;

int i = 0, j = 0;

float size, speed, Mbps, prate;

for (int count = 0; fgets(buf, sizeof(buf), f) != NULL && count < 15; count++)

{

if (count < 11)

{

continue;

}

i = 0, j = 0;

while (!isdigit(buf[i]))

i++;

while (buf[i] != ' ')

{

if (buf[i] != ',')

{

temp[j++] = buf[i];

}

i++;

}

temp[j++] = 0;

if (count == 11)

{

speed = atof(temp);

while (buf[i++] != ' ')

;

char type[10];

int k = 0;

while (isalpha(buf[i]) || buf[i] == '/')

{

type[k++] = buf[i++];

}

type[k++] = 0;

if (strcmp(type, "kBps") == 0)

speed /= 1024.0f;

else if (strcmp(type, "bytes/s") == 0)

speed /= 1024.0f \* 1024.0f;

else if (strcmp(type, "MBps") != 0)

puts(type), assert(0);

printf("\*\*\t\tAVERAGE SPEED(MBps) : %4.2f MBps\n", speed);

}

else if (count == 12)

{

Mbps = atof(temp);

while (buf[i++] != ' ')

;

char type[10];

int k = 0;

while (isalpha(buf[i]))

{

type[k++] = buf[i++];

}

type[k++] = 0;

if (strcmp(type, "kbps") == 0)

Mbps /= 1024.0f;

else if (strcmp(type, "Mbps") != 0)

assert(0);

printf("\*\*\t\tAVERAGE SPEED(Mbps) : %4.2f Mbps\n", Mbps);

}

else if (count == 13)

{

size = atof(temp);

printf("\*\*\t\tAVERAGE PACKET SIZE : %4.2f bytes\n", size);

}

else if (count == 14)

{

prate = atof(temp);

printf("\*\*\t\tAVERAGE PACKET RATE/s : %4.2f kpackets/s\n", prate);

}

}

printf("\*\*\t\tAVERAGE RTT : %f seconds\n", size \* 2 / (speed \* (1 << 20)));

}

int main(int argc, char \*\*argv)

{

pcap\_t \*handle;

char error\_buffer[PCAP\_ERRBUF\_SIZE];

char fileName[100];

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Enter pcap file name to analyze:");

scanf("%s", fileName);

int f = fopen(fileName, "r");

if (fopen(fileName, "r") == 0)

{

printf("FILE NOT FOUND\nEXITING\n");

exit(0);

}

printf("Analyzing network packets in the given file...\n");

char command[100];

sprintf(command, "./shell.sh %s", fileName);

system(command);

handle = pcap\_open\_offline(fileName, error\_buffer);

pcap\_loop(handle, 0, my\_packet\_handler, NULL);

pcap\_close(handle);

fileParser();

return 0;

}

**Output:**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**References**

1. Computer Networking: A Top-Down Approach, 6th Edition

2. https://www.opensourceforu.com/2011/02/capturing-packets-c-program-libpcap/

3. <https://www.devdungeon.com/content/using-libpcap-c>

5. https://www.wireshark.org/docs/man-pages/capinfos.html