**Packet Sniffing**

**INTERNSHIP PROJECT**

**Mohd Sameer**

Roll No. 200106045

HBTU, Kanpur

*Under the Guidance of*

**Prof. Manoj Misra**

Department of Computer Science

Indian Institute of Technology,Roorkee

July, 2023

**CANDIDATE’S DECLARATION**

I,hereby, declare that the work, which is being presented in the dissertion titles “Packet Sniffing” for the partial fulfillment of the requirements of the awards of the internship submitted to the department of Computer science and Engineering,Indian Institure of Technology, Roorkee, Uttarakhand (India) is an authentic record of my own work carried out during the period from 29 mat 2023 to 8 july,2023 , under the guidance of Dr. Manoj Misra,Professor,Department of Computer Science and Engineering,IIT Roorkee.

The matter submitted in this dissertation has not bee submitted by me for the award of any other degree of this or any other institute.

Date-

Place-Roorkee Mohd Sameer

**CERTIFICATE**

\*This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.\*

Date- Dr.Manoj Misra

Place –Roorkee Professor

Dept. Computer Science &Engineering

IIT Roorkee

**ACKNOWLEDGEMENT**

With a deep sense of gratitude, I sincerely thank to Prof. Manoj Misra for his outstanding guidance & providing me this golden opportunity of carrying out my Summer Internship project work. Words can never qualify nor quantify how helpful your guidance & advice has been. I would be forever grateful for your support and cooperation.

I would like to thank Mr. Avnish Kumarfor giving suggestion to complete my project.

I’m also grateful to all members of Lab for providing such a smooth and familiar environment to work.

I would like to express my sincere gratitude for giving me the privilege to explore the world of Packet Sniffing and enhance my knowledge.

|  |  |
| --- | --- |
|  | **Mohd Sameer** |
| Date: 03.07.2023 | Department of Electronics Engineering |
| Place: Roorkee | Harcourt Butler Technical University, Kanpur |

**ABSTRACT**

Packet sniffers is used for capturing and monitoring the network traffic through the installation of software on our device.it is also known as packet analyzer or packet capture.

Socket programming is a fundamental concept in network communication, allowing software applications to establish connections and exchange data over a network. It involves the use of sockets, which act as endpoints for communication between different devices or processes. In socket programming, a client-server model is commonly used, where a server listens for incoming connections, and clients initiate connections to the server.

Wireshark is a free and open-source software which is available for the packet analyzer.It is used for network troubleshooting, analysis, communications protocol development, and education.

Generally, By using the sniffing software, we can capture the traffic in a device's network. It capture everything that is being transmitted through the device over the network. Although it is a non-malicious technique to capture the capture. Packet sniffing is used for inspect the data packets and extract valuable information such as passwords, IP addresses, etc.

**Contents**

[1.0 **Introduction** 7](#_Toc139385568)

[2.0 **Objective** 8](#_Toc139385569)

[3.0 **Literature Review** 9](#_Toc139385570)

[3.1 Client-Server Communication in Socket Programing 9](#_Toc139385571)

[3.1.1 TCP(Transport Control Protocol) 10](#_Toc139385572)

[3.1.2 UDP (User Datagram Protocol) 19](#_Toc139385573)

[3.2 Wireshark 23](#_Toc139385574)

[3.2.1 Analyzing data packets on Wireshark 23](#_Toc139385575)

[3.3 Packet Sniffing 25](#_Toc139385577)

[4.0 **Program for Packet Sniffer** 26](#_Toc139385579)

[5.0 **Program Output** 30](#_Toc139385580)

[6.0 **Results** 31](#_Toc139385583)

[7.0 **Conclusion** 31](#_Toc139385584)

[8.0 **Future Scope** 32](#_Toc139385585)

[**References** 33](#_Toc139385586)

# Introduction

A packet sniffer is a piece of hardware or software used to track network traffic. It is sometimes referred to as a network analyzer, packet analyzer.Sniffers which is used to analyze data packet streams that go between connected computer as well as connection over wider internet.. Using a sniffer, it's possible to capture almost any information for example, which websites that a user visits, what is viewed on the site, the contents and destination of any email along with details about any downloaded files. Protocol analyzers are often used by companies to keep track of network use by employees and are also a part of many reputable antivirus software package.[1]

A computer network is a connection among a set of computers through some networking hardware like router, switch, etc. and communication channel. Primary goal of developing a computer network is to share resources of different computers. To share the resources, therefore, we need support from some programs. These programs are essential for developing any software for a networked system (and for Internet). Such programs create socket to send and receive data to/from the network. So, these programs are commonly popular as Socket Programs. One way to conceptualize sockets in computer networking is as interfaces that can "plug into" one another over a network. Once so “plugged in”, the programs can communicate. This note provides a short introduction on the socket programming. Languages, like C, C++, JAVA etc, can be used in socket programming. But in our opinion, exploring networks through socket programming in C is the most exciting. In this document, we write socket programs in C..

# 

# Objective

The objective of socket programming is to establish communication between two or more devices over a network. Its facilitates the implementation of client-server architectures, where a server listens for incoming connections from clients and responds to their requests.it also allows developers to implement various network protocols, such as TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).

Wireshark , the network protocol analyzer, is to capture and analyze network traffic for the purpose of troubleshooting, network analysis, and security monitoring. Its provides a comprehensive set of tools and features to capture, inspect, and interpret network packets, allowing users to gain insights into network behavior and diagnose network-related issues.

Packet Sniffing which is used to capture and analyze network packets flowing through a network interface. Packet sniffing involves intercepting and examining the packets to gain insights into network communication, troubleshoot issues, and analyze network behavior.

**Yhn tk se 15per**

# 3.0 Literature Review

trators to fish out intruders or anomalous connections. The need to capture this information has

lead to the development of packet sniffers.

A number of research works exist in the development of packet sniffers. However, the search for

the ideal packet sniffer continues. Psniffer will come with additional functionalities such as 3D

pie charts, a GUI and with little memory requirements.

Psniffer when installed in a network will help monitor netwo

The information running through networks is a valuable source of evidence for network adminis-

trators to fish out intruders or anomalous connections. The need to capture this information has

lead to the development of packet sniffers.

A number of research works exist in the development of packet sniffers. However, the search for

the ideal packet sniffer continues. Psniffer will come with additional functionalities such as 3D

pie charts, a GUI and with little memory requirements.

Psniffer when installed in a network will help monitor ne

The information running through networks is a valuable source of evidence for network adminis-

trators to fish out intruders or anomalous connections. The need to capture this information has

lead to the development of packet sniffers.

A number of research works exist in the development of packet sniffers. However, the search for

the ideal packet sniffer continues. Psniffer will come with additional functionalities such as 3D

pie charts, a GUI and with little memory requirements.

Psniffer when installed in a network will help monitor network traffic and keeps log of all con-

nections to the network, which is then analyzed for the detection of suspicious activities.

The information running through networks is a valuable source of evidence for network adminis-

trators to fish out intruders or anomalous connections. The need to capture this information has

lead to the development of packet sniffers.

A number of research works exist in the development of packet sniffers. However, the search for

the ideal packet sniffer continues. Psniffer will come with additional functionalities such as 3D

pie charts, a GUI and with little memory requirements.

Psniffer when installed in a network will help monitor network traffic and keeps log of all con-

nections to the network, which is then analyzed for the detection of suspicious activities.

The information running through networks is a valuable source of evidence for network adminis-

trators to fish out intruders or anomalous connections. The need to capture this information has

lead to the development of packet sniffers.

A number of research works exist in the development of packet sniffers. However, the search for

the ideal packet sniffer continues. Psniffer will come with additional functionalities such as 3D

pie charts, a GUI and with little memory requirements.

Psniffer when installed in a network will help monitor network traffic and keeps log of all con-

nections to the network, which is then analyzed for the detection of suspicious activities.

The information running through networks is a valuable source of evidence for network adminis-

trators to fish out intruders or anomalous connections. The need to capture this information has

lead to the development of packet sniffers.

A number of research works exist in the development of packet sniffers. However, the search for

the ideal packet sniffer continues. Psniffer will come with additional functionalities such as 3D

pie charts, a GUI and with little memory requirements.

Psniffer when installed in a network will help monitor network traffic and keeps log of all con-

nections to the network, which is then analyzed for the detection of suspicious activities.

The information running through networks is a valuable source of evidence for network adminis-

trators to fish out intruders or anomalous connections. The need to capture this information has

lead to the development of packet sniffers.

A number of research works exist in the development of packet sniffers. However, the search for

the ideal packet sniffer continues. Psniffer will come with additional functionalities such as 3D

pie charts, a GUI and with little memory requirements.

Psniffer when installed in a network will help monitor network traffic and keeps log of all con-

nections to the network, which is then analyzed for the detection of suspicious activities.

The information running through networks is a valuable source of evidence for network adminis-

trators to fish out intruders or anomalous connections. The need to capture this information has

lead to the development of packet sniffers.

A number of research works exist in the development of packet sniffers. However, the search for

the ideal packet sniffer continues. Psniffer will come with additional functionalities such as 3D

pie charts, a GUI and with little memory requirements.

Psniffer when installed in a network will help monitor network traffic and keeps log of all con-

nections to the network, which is then analyzed for the detection of suspicious activities.

## 3.1 Client-Server Communication in Socket Programing

The client-server model is one of the most used communication paradigms in networked systems. Clients normally communicates with one server at a time. From a server’s perspective, at any point in time, it is not unusual for a server to be communicating with multiple clients. Client need to know of the existence of and the address of the server, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established. Client and servers communicate by means of multiple layers of network protocols

The transport layer comprises two types of protocol-

A, TCP (Transport Control Protocol)

B. UDP (User Datagram Protocol)

### 3.1.1 TCP(Transport Control Protocol)

TCP provides a connection oriented service, since it is based on connections between clients and servers. TCP provides reliability. When a TCP client send data to the server, it requires an acknowledgement in return. If an acknowledgement is not received, TCP automatically retransmit the data and waits for a longer period of time. TCP is instead a byte-stream protocol, without any boundaries at all. [2]

**TCP Socket API**

The sequence of function calls for the client and a server participating in a TCP connection is presented in Figure-1 (98 percent)

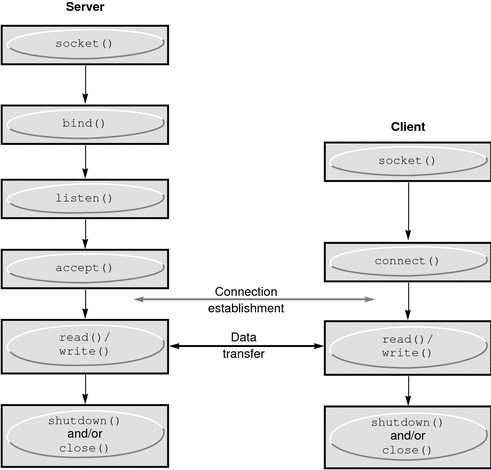


Fig 1: TCP client-server

As shown in the figure-1, the steps for establishing a TCP socket on the client side are the following:

* Create a socket using the socket() function;
* Connect the socket to the address of the server using the connect() function;
* Send and receive data by means of the read() and write() functions.

The steps involved in establishing a TCP socket on the server side are as follows:

* Create a socket with the socket() function;
* Bind the socket to an address using the bind() function;
* Listen for connections with the listen() function;
* Accept a connection with the accept() function system call. This call typically blocks until a client connects with the server.
* Send and receive data by means of the write() and read() functions.

**The Socket() Function**

The first step is to call the socket function, specifying the type of communication protocol (TCP based on IPv4, TCP based on IPv6, UDP).The function is defined as follows:

#include<sys/socket.h>  
int socket (int family,int tpe, int protocol);

where family specifies the protocol family (AF\_INET for the IPv4 protocols), type is a constant described the type of socket (SOCK\_STREAM for stream sockets and SOCK\_DGRAM for datagram sockets.

**The connect() Function**

The connect() function is used by a TCP client to establish a connection with a TCP server.The function is defined as follows

#include<sys/socket.h>  
int connect(int sockfd,const struct sockaddr \*servaddr, socklen\_t addrlen);

where sockfd is the socket descriptor returned by the socket function.

**The bind() Function**

The bind() assigns a local protocol address to a socket. With the Internet protocols, the address is the combination of an IPv4 or IPv6 address (32-bit or 128-bit) address along with a 16 bit TCP port number.The function is defined as follows:

#include<sys/socket.h>  
int bind(int sockfd,const struct sockaddr \*servaddr, socklen\_t addrlen);

where sockfd is the socket descriptor, myaddr is a pointer to a protocol-specific address and addrlen is the size of the address structure.

**The listen() Function**

The listen() function converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket. It is defined as follows:

#include<sys/socket.h>  
int listen(int sockfd,int backlog);

where sockfd is the socket descriptor and backlog is the maximum number of connections the kernel should queue for this socket.

**The accept() Function**

The accept() is used to retrieve a connect request and convert that into a request. It is defined as follows:

#include <sys/socket.h>  
int accept(int sockfd,const struct sockaddr \*cliaddr, socklen\_t addrlen);

where sockfd is a new file descriptor that is connected to the client that called the connect(). The cliaddr and addrlen arguments are used to return the protocol address of the client.

**The send() Function**

Since a socket endpoint is represented as a file descriptor, we can use read and write to communicate with a socket as long as it is connected. However, if we want to specify options we need another set of functions.

#include<sys/socket.h>  
ssize\_t send(int sockfd,const void \*buf, size\_t nbytes,int flags);

where buf and nbytes have the same meaning as they have with write. The additional argument flags is used to specify how we want the data to be transmitted. We will not consider the possible options in this course. We will assume it equal to 0.

**The receive() Function**

The recv() function is similar to read(), but allows to specify some options to control how the data are received. We will not consider the possible options in this course. We will assume it is equal to 0.receive is defined as follows:

#include<sys/socket.h>  
ssize\_t send(int sockfd,const void \*buf, size\_t nbytes,int flags);

**The close() Function**

The normal close() function is used to close a socket and terminate a TCP socket. It returns 0 if it succeeds, -1 on error. It is defined as follows:

#include<unistd.h>  
int close(int sockfd);

**Example**

Develop a socket program in UNIX/Linux that uses.The server sends a string (input by the user) to the client and the client prints the string on the screen after receiving it.

Program on the Server Side

#include<stdio.h>

#include<stdlib.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<netinet/in.h>

int main(){

    char server\_message[256]="socket programming";

    int server\_socket=socket(AF\_INET,SOCK\_STREAM,0);    // create socket

    struct sockaddr\_in server\_address;

    server\_address.sin\_family=AF\_INET;

    server\_address.sin\_port=htons(8000);

    server\_address.sin\_addr.s\_addr= INADDR\_ANY;

    bind(server\_socket,(struct sockadd \*) & server\_address, sizeof(server\_address));    //bind

    listen(server\_socket,5);       //listen

    int cliemt\_sock;

    cliemt\_sock= accept(server\_socket,NULL,NULL);   //accept

    send(cliemt\_sock,server\_message, sizeof(server\_message),0);    //send

    close(server\_socket);    //close

    return 0;

}

**Program on the client side**

#include<stdio.h>

#include<stdlib.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<netinet/in.h>

int main(){

    int network\_socket;

    network\_socket = socket(AF\_INET,SOCK\_STREAM,0);      //socket

    struct sockaddr\_in server\_address;~

    server\_address.sin\_family=AF\_INET;

    server\_address.sin\_port=htons(8000);

    server\_address.sin\_addr.s\_addr= INADDR\_ANY;

    int connection\_status;      //connect

    connection\_status=connect(network\_socket,(struct sockadd \*) & server\_address, sizeof(server\_address));

    if(connection\_status==-1){

        printf("error");

    }

    char msg[256];

    recv(network\_socket,&msg , sizeof(msg),0);    //recv

    printf("the server send the ddata --> %s \n",msg);

    close(network\_socket);    //close

    return 0;

}

**Echo Client Server**

The client reads a line of text from its standard input and writes the line to the server. The server reads the line from its network input and echoes the line back to the client. The client reads the echoed line and prints it on its standard output.[3]

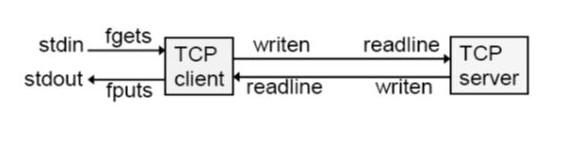


Fig 2: Echo TCP client-server

**Program on Echo Client Side**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

void error\_handling(char \*message);

#define Buf\_SIZE 1024

int main(int argc, char \*argv[]){

    int sock;

    char message[Buf\_SIZE];

    int str\_len;

    struct sockaddr\_in ser\_adr;

    if (argc !=3){

        printf("Usage : %s <IP> <port>\n",argv[0]);

        exit(1);

    }

    // socket creation

    sock=socket(AF\_INET,SOCK\_STREAM, 0);

    if(sock==-1)

        error\_handling("socket() error");

    memset(&ser\_adr,0,sizeof(ser\_adr));

    ser\_adr.sin\_family=AF\_INET;

        fputs("Input message (Q to quit): ", stdout);

        fgets(message,Buf\_SIZE,stdin);

        if(!strcmp(message,"q\n")||!strcmp(message,"Q\n"))

            break;

        write(sock,message,strlen(message));

        str\_len=read(sock,message, Buf\_SIZE-1);

        message[str\_len]=0;

        printf("Message from the server : %s ",message);

    close(sock);

    return 0;

}

void error\_handling(char\*message){

    fputs(message,stderr);

    fputc('\n',stderr);

    exit(1);

}

**Program on the Echo-Server side**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

void error\_handling(char \*message);

#define Buf\_SIZE 1024

int main(int argc, char \*argv[]){

    int ser\_sock, client\_sock;

    char message[Buf\_SIZE];

    int str\_len, i;

    struct sockaddr\_in ser\_adr;

    struct sockaddr\_in clnt\_adr;

    socklen\_t clnt\_adr\_sz;

    if (argc !=2){

        printf("Usage : %s <port>\n",argv[0]);

        exit(1);

    }

    // socket creation

    ser\_sock=socket(AF\_INET,SOCK\_STREAM, 0);

    if(ser\_sock==-1)

        error\_handling("socket() error");

    memset(&ser\_adr,0,sizeof(ser\_adr));

    ser\_adr.sin\_family=AF\_INET;

    ser\_adr.sin\_addr.s\_addr=htonl(INADDR\_ANY);

    ser\_adr.sin\_port=htons(atoi(argv[1]));

    // binding

    if(bind(ser\_sock,(struct sockaddr\*)&ser\_adr,sizeof(ser\_adr))==-1)

        error\_handling("bind() error");

    //listening

    if(listen(ser\_sock,5==-1))

        error\_handling("listen() error");

    clnt\_adr\_sz=sizeof(clnt\_adr);

    for(i=0;i<5;i++){

        client\_sock=accept(ser\_sock,(struct sockaddr\*)&clnt\_adr,&clnt\_adr\_sz);

        if(client\_sock==-1)

            error\_handling("accept() error");

        else

            printf("connected client %d\n",i+1);

        while((str\_len=read(client\_sock,message,Buf\_SIZE))!=0)

            write(client\_sock,message,str\_len);

        close(client\_sock);

    }

close(ser\_sock);

    return 0;

}

void error\_handling(char\*message){

    fputs(message,stderr);

    fputc('\n',stderr);

    exit(1);

}

**(89 percent)**

**The fork() function**

The fork() function is the only way in Unix to create a new process. It is defined as follows:

#include <unistd.h>

pid\_t fork(void);

#include<stdio.h>

#include<unistd.h>

int gval=10;

int main(int argc, char \*argv[]){

    pid\_t pid;

    int lval=20;

    gval++, lval+=5;

    pid=fork();

    if(pid==0){   // child process

        gval+=2, lval +=2;

    }else{         // parent process

        gval-=2, lval-=2;

    }

    if(pid==0){

        printf("child process %d %d",gval,lval);

    }else{

        printf("parent process %d %d",gval,lval);

    }

}

### 3.1.2 UDP (User Datagram Protocol)

UDP speeds up transmissions by enabling the transfer of data before an agreement is provided by the receiving party. As a result, UDP is beneficial in time-sensitive communications, including voice over IP (VoIP), domain name system ([DNS](https://www.techtarget.com/searchnetworking/definition/domain-name-system)) lookup, and video or audio playback.[4]

UDP is an alternative to Transmission Control Protocol ([TCP](https://www.techtarget.com/searchnetworking/definition/TCP)). Both UDP and TCP run on top of IP and are sometimes referred to as UDP/IP or [TCP/IP](https://www.techtarget.com/searchnetworking/definition/TCP-IP). However, there are important differences between the two. For example, UDP enables process-to-process communication, while TCP supports host-to-host communication.

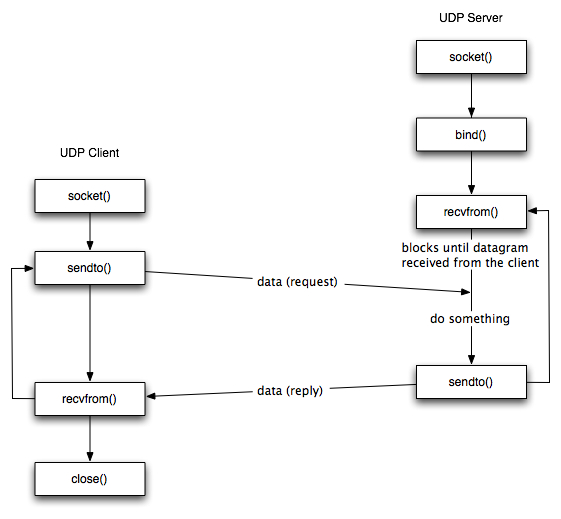


Fig-2: TCP client-server

Example

Develop a socket program in UNIX/Linux that uses. The server sends a string (input by the user) to the client and the client prints the string on the screen after receiving it.

**Program on the UDP-Server side**

#include <stdio.h>

#include <strings.h>

#include <sys/types.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include<netinet/in.h>

#define PORT 5000

#define MAXLINE 1000

int main()

{

    char buffer[100];

    char \*message = "Hello UDP Client";

    int listenfd, len;

    struct sockaddr\_in servaddr, cliaddr;

    bzero(&servaddr, sizeof(servaddr));

    listenfd = socket(AF\_INET, SOCK\_DGRAM, 0);

    servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    servaddr.sin\_port = htons(PORT);

    servaddr.sin\_family = AF\_INET;

    bind(listenfd, (struct sockaddr\*)&servaddr, sizeof(servaddr));

    len = sizeof(cliaddr);

    int n = recvfrom(listenfd, buffer, sizeof(buffer),

            0, (struct sockaddr\*)&cliaddr,&len);

    buffer[n] = '\0';

    puts(buffer);

    sendto(listenfd, message, MAXLINE, 0,

        (struct sockaddr\*)&cliaddr, sizeof(cliaddr));

}

**Program on the UDP-Client side**

#include <stdio.h>

#include <strings.h>

#include <sys/types.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include<netinet/in.h>

#include<unistd.h>

#include<stdlib.h>

#define PORT 5000

#define MAXLINE 1000

int main()

{

    char buffer[100];

    char \*message = "Hello Server";

    int sockfd, n;

    struct sockaddr\_in servaddr;

    bzero(&servaddr, sizeof(servaddr));

    servaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    servaddr.sin\_port = htons(PORT);

    servaddr.sin\_family = AF\_INET;

    sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

    if(connect(sockfd, (struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0)

    {

        printf("\n Error : Connect Failed \n");

        exit(0);

    }

    sendto(sockfd, message, MAXLINE, 0, (struct sockaddr\*)NULL, sizeof(servaddr));

    recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr\*)NULL, NULL);

    puts(buffer);

    close(sockfd);

}

Types of Socket Programming

1. **Stream Sockets**
2. **Datagram Sockets**
3. **Raw Socket**

a. **Stream Sockets:**-Stream socket allows processes to use the [Transfer Control Protocol (TCP)](https://www.geeksforgeeks.org/what-is-transmission-control-protocol-tcp/) for communication. A stream socket provides a sequenced, constant or reliable, and two-way (bidirectional) flow of data. After the establishment of connection, data can be read and written to these sockets in a byte stream. The socket type of stream socket is SOCK\_STREAM

b. **Datagram Sockets:**Datagram sockets allow processes to use the User Datagram Protocol (UDP). It is a two-way flow of communication or messages. It can receive messages in a different order from the sending way and also can receive duplicate messages. These sockets are preserved with their boundaries. The socket type of datagram socket is SOCK\_DGRAM

**c. Raw Sockets:**Raw Socket provide user access to the Internet Control Message Protocol (ICMP). Raw sockets are not used for most applications. These sockets are the same as the datagram oriented, their characteristics are dependent on the interfaces. They provided support in developing new communication protocols or for access to more facilities of an existing protocol. Only the superusers can access the Raw Sockets. The socket type of Raw Socket is SOCK\_RAW.

## 3.2 Wireshark

Wireshark is a network traffic analyzer, or "sniffer", for Linux, macOS, BSD and other Unix and Unix-like operating systems and for Windows. It uses libpcap and npcap as packet capture and filtering libraries.

Wireshark is an open-source network protocol analysis software program, widely considered the industry standard. A global organization of network specialists and software developers supports Wireshark and continues to make updates for new network technologies and encryption methods.[5]

Government agencies, corporations, non-profits, and educational institutions use Wireshark for troubleshooting and teaching purposes. There truly isn’t a better way to learn low-level networking than to look at traffic under the Wireshark microscope.

### 3.2.1 Analyzing data packets on Wireshark

Wireshark shows you three different panes for inspecting packet data. The Packet List, the top pane, lists all the packets in the capture. When you click on a packet, the other two panes change to show you the details about the selected packet. You can also tell if the packet is part of a conversation. Here are details about each column in the top pane:

* **No.**: This is the number order of the packet captured. The bracket indicates that this packet is part of a conversation.
* **Time**: This column shows how long after you started the capture this particular packet was captured. You can change this value in the Settings menu to display a different option.
* **Source**: This is the address of the system that sent the packet.
* **Destination**: This is the address of the packet destination.
* **Protocol**: This is the type of packet. For example: TCP, DNS, DHCPv6, or ARP.
* **Length**: This column shows you the packet’s length, measured in bytes.
* **Info**: This column shows you more information about the packet contents, which will vary depending on the type of packet.[7]

### 

## 3.3 Packet Sniffing

When any data has to be transmitted over the computer network, it is broken down into smaller units at the sender’s node called data packets and reassembled at receiver’s node in original format. It is the smallest unit of communication over a computer network. It is also called a block, a segment, a datagram or a cell. The act of capturing data packet across the computer network is called **packet sniffing**. It is similar to as wire tapping to a telephone network. It is mostly used by crackers and hackers to collect information illegally about network. It is also used by ISPs, advertisers and governments. **ISPs** use packet sniffing to track all your activities such as:

* who is receiver of your email
* what is content of that email
* what you download
* sites you visit
* what you looked on that website
* downloads from a site
* streaming events like video, audio, etc. (90percent)

**Packet Sniffer**

### Packet sniffing is done by using tools called packet sniffer. It can be either filtered or unfiltered. Filtered is used when only specific data packets have to be captured and Unfiltered is used when all the packets have to be captured. WireShark, Smart Sniff are examples of packet-sniffing tools.[7]

# 4.0 Program for Packet Sniffer

#include <stdlib.h>

#include <stdio.h>

#include <pcap.h>

#include<netinet/ip.h>

#include<netinet/ip\_icmp.h>

#include<netinet/ether.h>

#include <netinet/tcp.h>

#include <netinet/udp.h>

#include <time.h>

#include<string.h>

void got\_packet(u\_char \*args, const struct pcap\_pkthdr \*header,const u\_char \*packetData)

{

    struct ether\_header\* ethernetHeader;

    struct ip\* ipHeader;

    struct icmphdr\* icmpHeader;

    struct tcphdr\* tcpHeader;

    struct udphdr\* udpHeader;

    char sourceIP[INET\_ADDRSTRLEN];

    char destIP[INET\_ADDRSTRLEN];

    struct ether\_addr sourceMAC, destMAC;

    time\_t ticks;

    ethernetHeader = (struct ether\_header\*)packetData;

    memcpy(&sourceMAC, ethernetHeader->ether\_shost, sizeof(struct ether\_addr));

    memcpy(&destMAC, ethernetHeader->ether\_dhost, sizeof(struct ether\_addr));

    if (ntohs(ethernetHeader->ether\_type) == ETHERTYPE\_IP) {

        ipHeader = (struct ip\*)(packetData + sizeof(struct ether\_header));

        inet\_ntop(AF\_INET, &(ipHeader->ip\_src), sourceIP, INET\_ADDRSTRLEN);

        inet\_ntop(AF\_INET, &(ipHeader->ip\_dst), destIP, INET\_ADDRSTRLEN);

    if(ipHeader->ip\_p == IPPROTO\_ICMP ||ipHeader->ip\_p == IPPROTO\_TCP ||ipHeader->ip\_p == IPPROTO\_UDP){

        ticks = time(NULL);

        printf("time:");    // display the time

        printf(ctime(&ticks));

            printf("Source MAC: %02X:%02X:%02X:%02X:%02X:%02X\n", sourceMAC.ether\_addr\_octet[0],            //display source and destination mac address

        sourceMAC.ether\_addr\_octet[1], sourceMAC.ether\_addr\_octet[2], sourceMAC.ether\_addr\_octet[3],

        sourceMAC.ether\_addr\_octet[4], sourceMAC.ether\_addr\_octet[5]);

            printf("Destination MAC: %02X:%02X:%02X:%02X:%02X:%02X\n", destMAC.ether\_addr\_octet[0],

        destMAC.ether\_addr\_octet[1], destMAC.ether\_addr\_octet[2], destMAC.ether\_addr\_octet[3],

        destMAC.ether\_addr\_octet[4], destMAC.ether\_addr\_octet[5]);

        if(ipHeader->ip\_p == IPPROTO\_ICMP) {                                                  // for icmp packet

            icmpHeader = (struct icmphdr\*)(packetData + sizeof(struct ether\_header) + sizeof(struct ip));

            printf("Protocol: ICMP\n");

            printf("Source IP: %s\n", sourceIP);

            printf("Destination IP: %s\n", destIP);

            unsigned int dataLen = ntohs(ipHeader->ip\_len) - sizeof(struct ip) - sizeof(struct icmphdr);

            printf("Data Length: %u\n", dataLen);

            printf("\n\n");

        }else if(ipHeader->ip\_p == IPPROTO\_TCP) {                                          // for tcp packet

            tcpHeader = (struct tcphdr\*)(packetData + sizeof(struct ethhdr) + sizeof(struct ip));

            printf("Protocol: TCP\n");

            printf("Source IP: %s\n", sourceIP);

            printf("Destination IP: %s\n", destIP);

            printf("Source Port: %u\n", ntohs(tcpHeader->th\_sport));

            printf("Destination Port: %u\n", ntohs(tcpHeader->th\_dport));

            unsigned int dataOffset = (tcpHeader->doff \* 4);

            unsigned int dataLen = ntohs(ipHeader->ip\_len) - sizeof(struct ip) - dataOffset;

            printf("Data Length: %u\n", dataLen);

            printf("\n\n");

        }else if (ipHeader->ip\_p == IPPROTO\_UDP) {                                          // for udp packet

            udpHeader = (struct udphdr\*)(packetData + sizeof(struct ethhdr) + sizeof(struct ip));

            printf("Protocol: UDP\n");

            printf("Source IP: %s\n", sourceIP);

            printf("Destination IP: %s\n", destIP);

            printf("Source Port: %u\n", ntohs(udpHeader->uh\_sport));

            printf("Destination Port: %u\n", ntohs(udpHeader->uh\_dport));

            unsigned int dataOffset = sizeof(struct udphdr);

            unsigned int dataLen = ntohs(udpHeader->uh\_ulen) - sizeof(struct udphdr);

            printf("Data Length: %u\n", dataLen);

            printf("\n\n");

        }

    }

    }

}

int main()

{

    pcap\_t \*handle;

    char errbuf[PCAP\_ERRBUF\_SIZE];

    struct bpf\_program fp;

    char filter\_exp[] = "";  // for all tcp, udp, icmp packet

    bpf\_u\_int32 net;

    handle = pcap\_open\_live("enp1s0", BUFSIZ, 1, 1000, errbuf);   // live session

    pcap\_compile(handle, &fp, filter\_exp, 0, net);

    if (pcap\_setfilter(handle, &fp) !=0) {

        pcap\_perror(handle, "Error:");

        exit(EXIT\_FAILURE);

    }

    pcap\_loop(handle, -1, got\_packet, NULL);                    // Capture packet

    pcap\_close(handle);

    return 0;

}

# 5.0 Program Output

### 

### In order to learn more about the network, we made this code to intercept and capture real-time data as it is transmitted via Ethernet or Wi-Fi. we use libpcap library for capturing the packet. After seeing the result we can observer that we are able to know the time of capturing and Source & Destination address. It shows which protocol used like icmp, tcp or udp. Its also describe source and destination ip and data length of the packet.

# 6.0 Results

Through looking into the packets and we are able to detects the problems like network congestion, packet loss, or faulty configuration, packet sniffing can be used to find network faults It can be used for detecting and evaluating security risks such malware infections, network intrusions, and unwanted access attempts. It can be used to evaluate network protocols and spot potential areas for development or optimization. Any data delivered can be recorded by a packet sniffer and sent to a command and control (C&C) server for additional examination.

# 7.0 Conclusion

Network sniffers have a variety of applications, including monitoring bandwidth, boosting productivity, ensuring the supply of business services, boosting security, etc. Some of the top use packet sniffer programs are Wireshark, TCPdump, and WinDump. Some of them are free available in the market. All of them vary in features and functionalities.

In a society where people are more dependent on networked technology to complete personal and professional duties, packet sniffing attacks are a severe hazard.It is very important to protects from the attacks of packet sniffing as so much sensitive data being sent over the internet.

# 8**.0 Future Scope**

Here, we are outlining some potential trends and developments that may shape the future of packet sniffing:

**8.1 Increased Encryption**

As more and more communication moves towards encrypted protocols such as HTTPS, packet sniffing becomes less effective in capturing meaningful data.

**8.2 Advanced Traffic Analysis**

As network traffic becomes more complex and sophisticated, the need for advanced traffic analysis tools will increase. Machine learning and artificial intelligence techniques can be employed to analyze large volumes of network traffic and identify patterns, anomalies, and potential security threats more effectively.

**8.3 Cloud and Virtualization**

Packet sniffing tools and techniques will need to adapt to these dynamic and distributed environments to capture and analyze traffic within virtual networks and cloud infrastructure.

**8.4 Privacy and Legal Implications**

With the growing concerns around privacy and data protection, there will likely be stricter regulations governing packet sniffing activities. Future developments may include legal restrictions on the use of packet sniffing techniques, requirements for explicit user consent, and increased focus on data anonymization and encryption.

# References

[1]. *What is a Packet Sniffer?* (2023). Retrieved from AO Kaspersky: https://www.kaspersky.com/resource-center/definitions/what-is-a-packet-sniffer

# [2]. Campbell, A. T. (2022). Socket Programming. Retrieved from cs.dartmouth: https://www.cs.dartmouth.edu/~campbell/cs50/socketprogramming.html

[3]. Stevens W.R.(2003) UNIX Network Programming (Volume-1) Addison Wesley

[4] *User Datagram Protocol*. (2023). Retrieved from SDxCentral: https://sdxcentral.com/resources/glossary/user-datagram-protocol-udp/

[5] Kinzie, K. (2022, Aug 19). *How to Use Wireshark: Comprehensive Tutorial + Tips*. Retrieved from Varonis: <https://www.varonis.com/blog/how-to-use-wireshark>

[6] *Cyber Forensic*. (n.d.). Retrieved from secupent: http://www.secupent.com/Cyber-Forensic.html#:~:text=Packet%20Sniffing%20When%20any%20data,communication%20over%20a%20computer%20network.

[7] Mittal, D. (2019) What is Sniffing? Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/what-is-packet-sniffing/#article-meta-div