

Computer Networks (CS39006)

Spring Semester (2021-2022)

Lab 1

Prof. Sudip Misra

Department of Computer Science and Engineering

Indian Institute of Technology Kharagpur

Email: smisra@sit.iitkgp.ernet.in

Website: <http://cse.iitkgp.ac.in/~smisra/>

Research Lab: cse.iitkgp.ac.in/~smisra/swan/





Wireshark

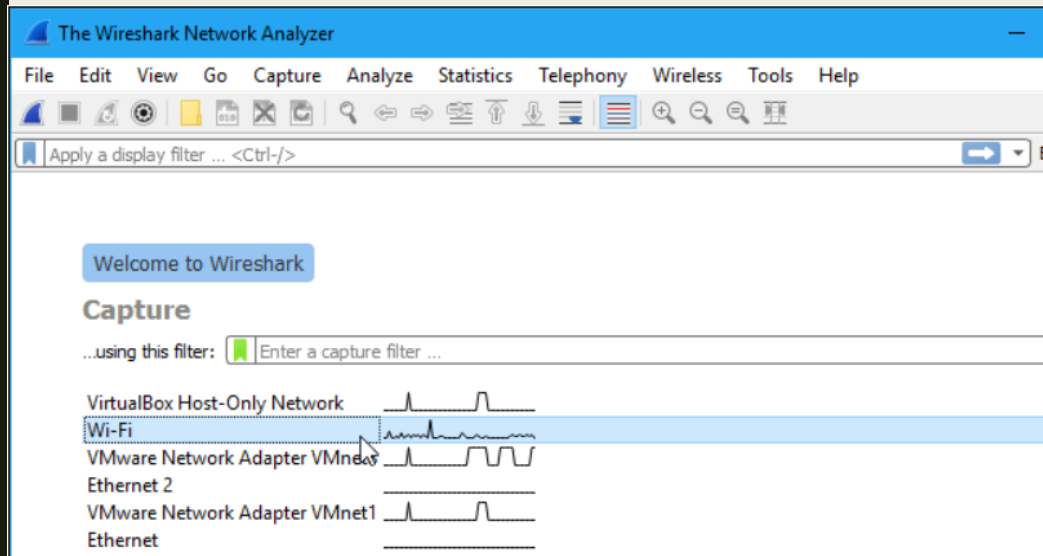
- ❑ Network sniffing tool.
- ❑ Provides GUI to decode many protocols and filters.
- ❑ Lets the user put network interface controllers into promiscuous mode.
- ❑ Understands the structure of different networking protocols.
- ❑ It can parse and display the fields, along with their meanings as specified by different networking protocols.
- ❑ Uses pcap to capture packets, so it can only capture packets on the types of networks that pcap supports.
- ❑ Similar to tcpdump.

Difference between Wireshark and tcpdump

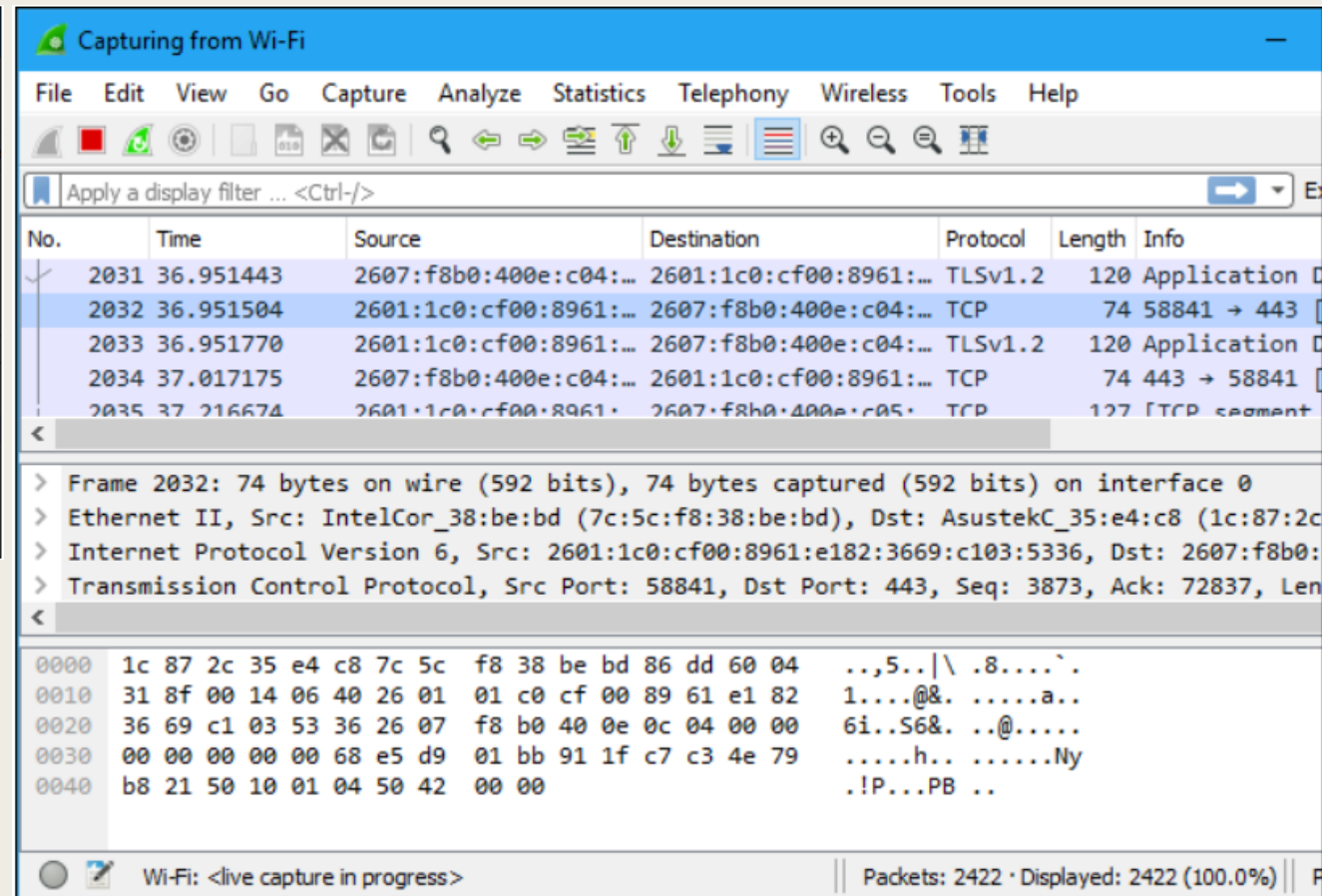


Sr No	Wireshark	Tcpdump
1	Wireshark is a graphical user interface tool that helps you to catch data packets.	Tcpdump is a CLI-based packet capturing tool.
2	It does packet analysis, and it can decode data payloads if the encryption keys are identified, and it can recognize data payloads from file transfers such as smtp, http, etc.	Tcpdump only provides a simple analysis of such types of traffic, such as DNS queries.
3	It has advanced network interfaces	It has system based conventional interfaces
4	Wireshark is good for complex filters	Tcpdump is used for simple filters.
5	It provides decoding of protocol-based packet capturing.	It is less efficient in decoding compared to Wireshark.

Capturing packets using wireshark



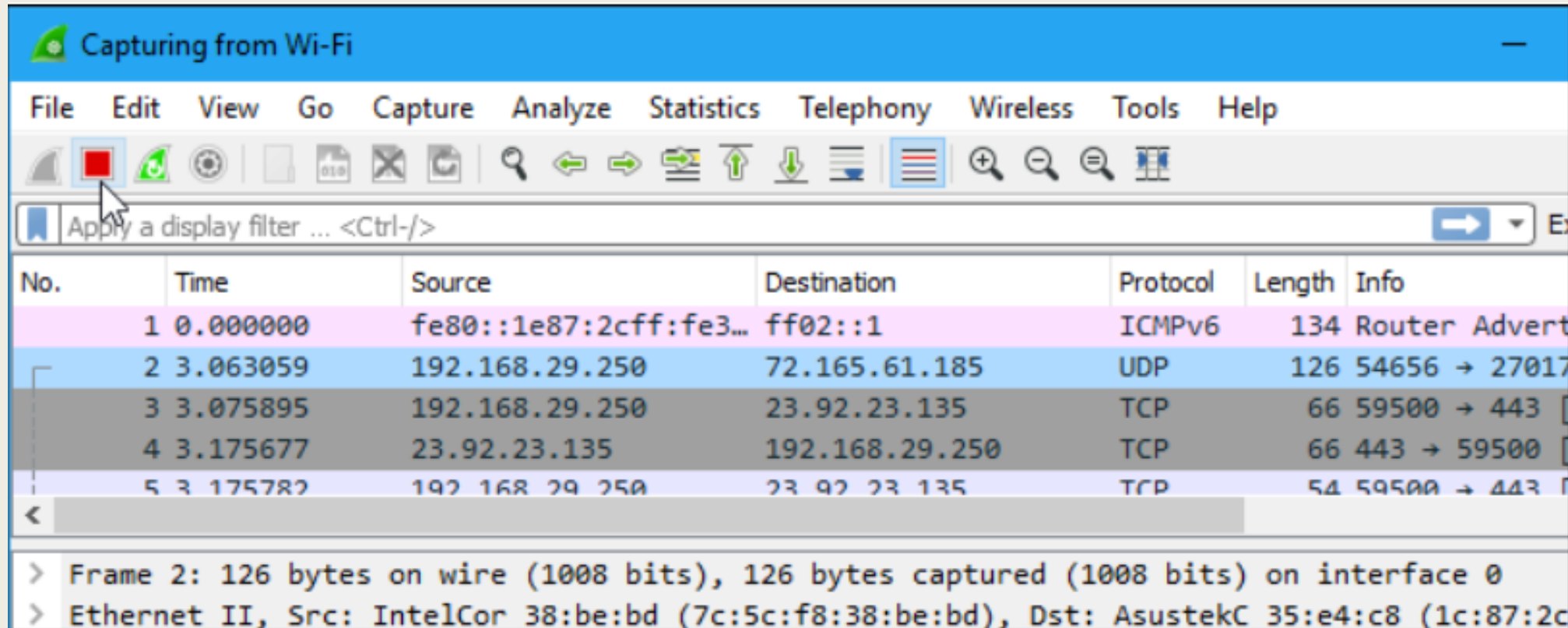
- Double-click the name of a network interface under Capture to start capturing packets on that interface.
- For example, if you want to capture traffic on your wireless network, click your wireless interface.



<https://www.howtogeek.com/104278/how-to-use-wireshark-to-capture-filter-and-inspect-packets/>

Stop packet capturing

Click the red “Stop” button near the top left corner of the window when you want to stop capturing traffic.



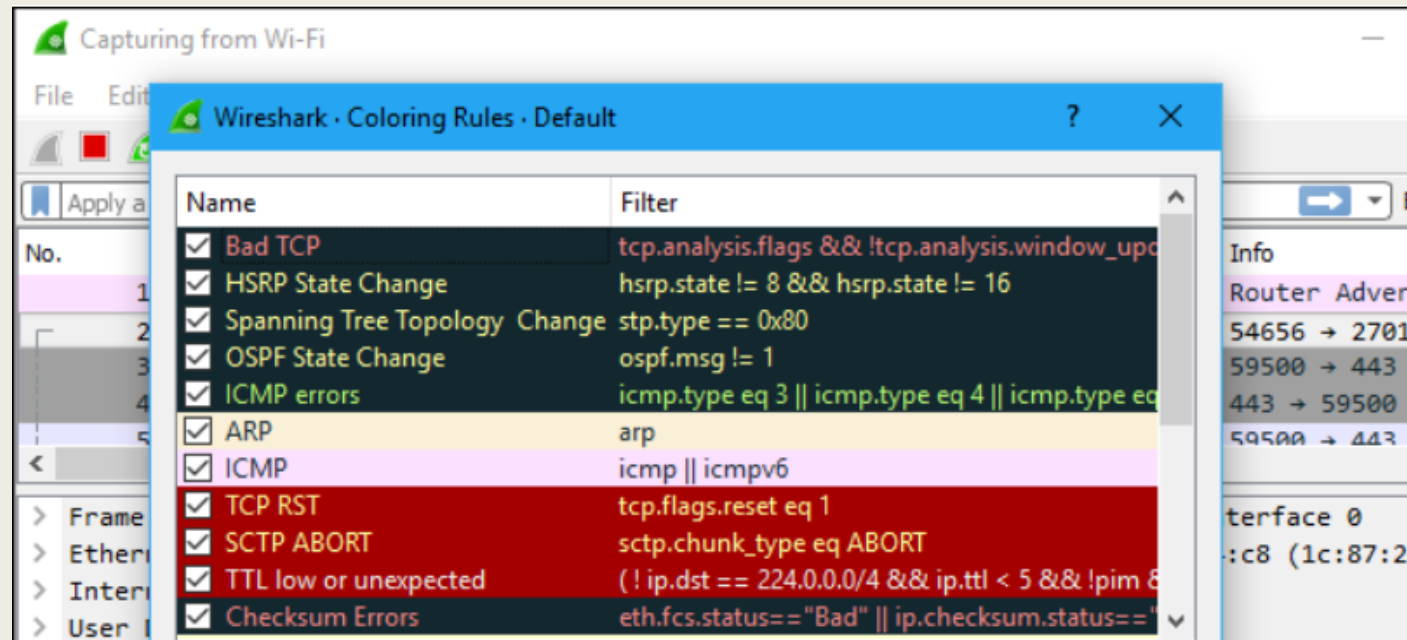
<https://www.howtogeek.com/104278/how-to-use-wireshark-to-capture-filter-and-inspect-packets/>

Color Coding



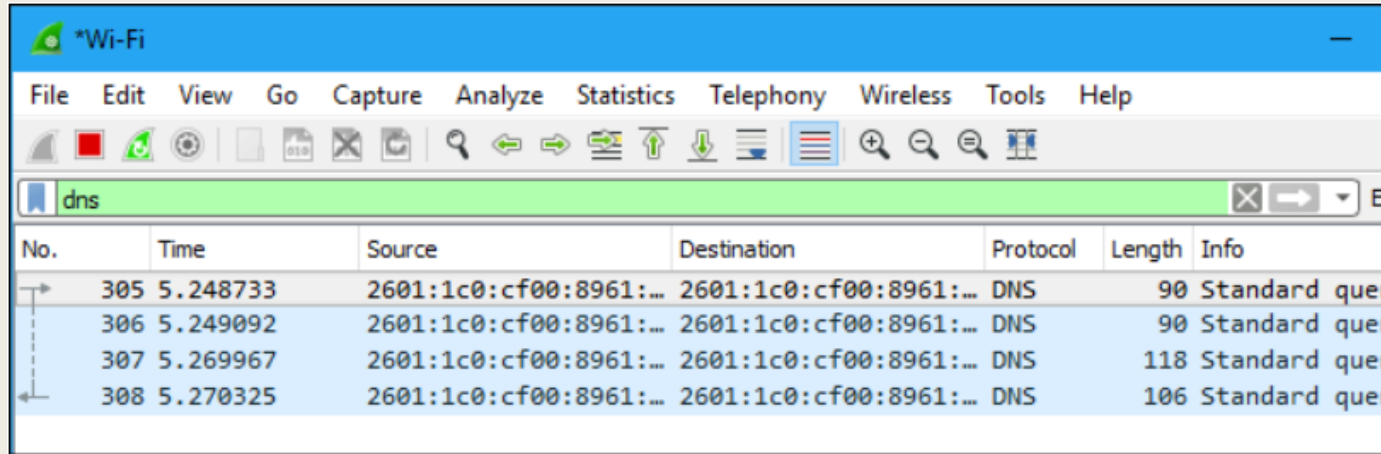
Wireshark uses colors to help you identify the types of traffic at a glance. By default,

- light purple is TCP traffic
- light blue is UDP traffic
- black identifies packets with errors



<https://www.howtogeek.com/104278/how-to-use-wireshark-to-capture-filter-and-inspect-packets/>

Packet Filtering

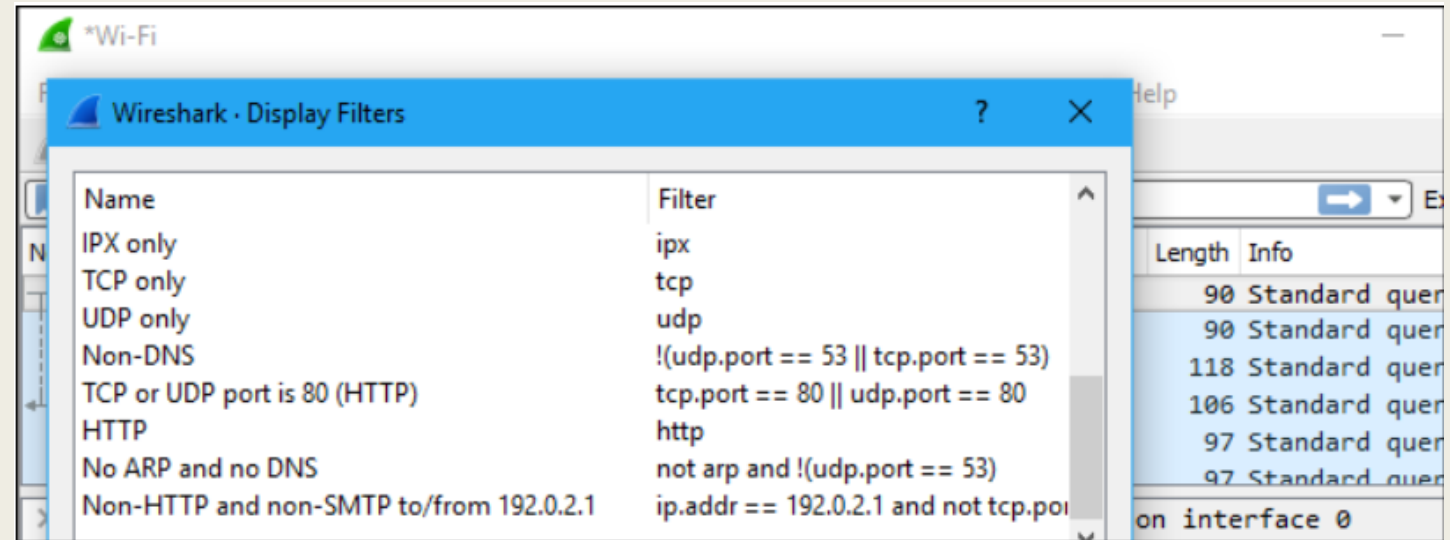


The screenshot shows the Wireshark interface with the 'dns' filter applied. The packet list table is as follows:

No.	Time	Source	Destination	Protocol	Length	Info
305	5.248733	2601:1c0:cf00:8961::...	2601:1c0:cf00:8961::...	DNS	90	Standard query
306	5.249092	2601:1c0:cf00:8961::...	2601:1c0:cf00:8961::...	DNS	90	Standard query
307	5.269967	2601:1c0:cf00:8961::...	2601:1c0:cf00:8961::...	DNS	118	Standard query
308	5.270325	2601:1c0:cf00:8961::...	2601:1c0:cf00:8961::...	DNS	106	Standard query

Type the filter in the filter box.

Click Analyze > Display Filters to choose a filter from among the default filters included in Wireshark.

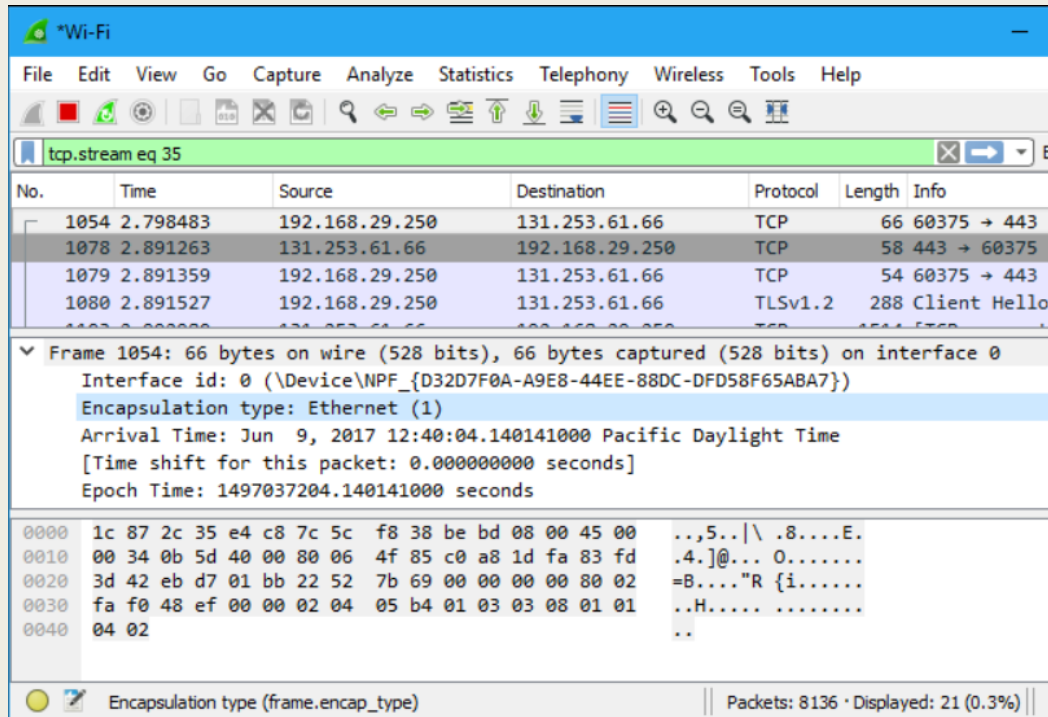


<https://www.howtogeek.com/104278/how-to-use-wireshark-to-capture-filter-and-inspect-packets/>

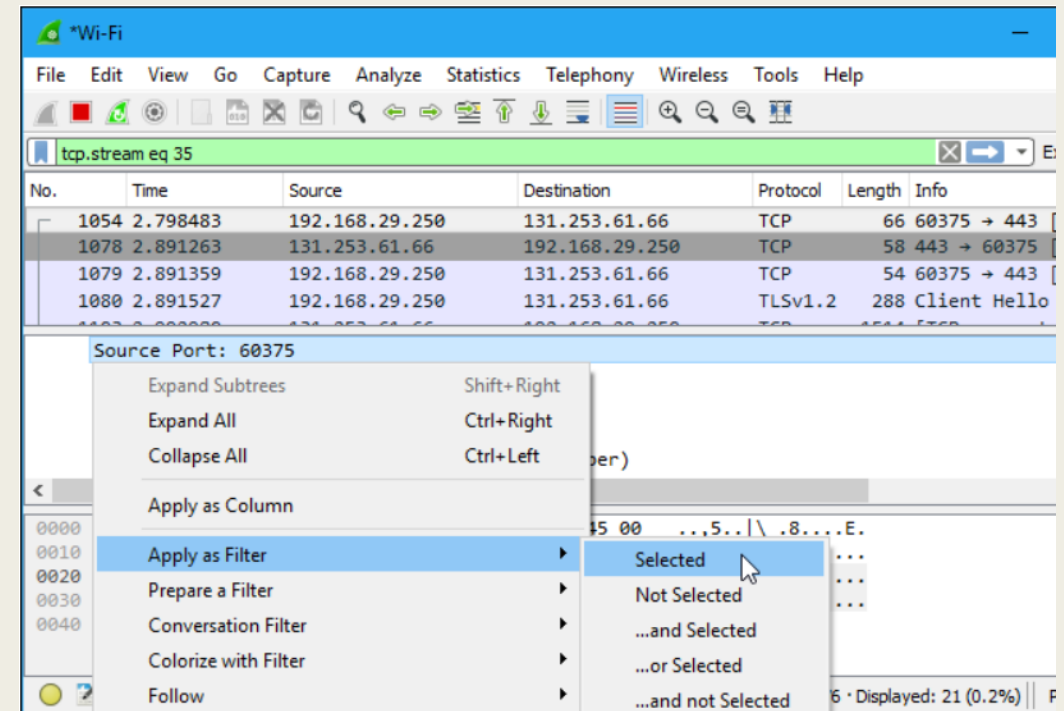
Inspecting Packets



Click a packet to select it and you can dig down to view its details.



Right-click one of the details and use the Apply as Filter submenu to create a filter based on it.



<https://www.howtogeek.com/104278/how-to-use-wireshark-to-capture-filter-and-inspect-packets/>

Application Layer



- The application layer provides services to the user.
- Communication is provided using a logical connection, which means that the two application layers assume that there is an imaginary direct connection through which they can send and receive messages.

Providing Services



All communication networks that started before the Internet were designed to provide services to network users.

Most of these networks, however, were originally designed to provide one specific service.

For example, the telephone network was originally designed to provide voice service: to allow people all over the world to talk to each other. This network, however, was later used for some other services, such as facsimile (fax), enabled by users adding some extra hardware at both ends.

Providing Services



Need two application programs to interact with each other: one running on a computer somewhere in the world, the other running on another computer somewhere else in the world.

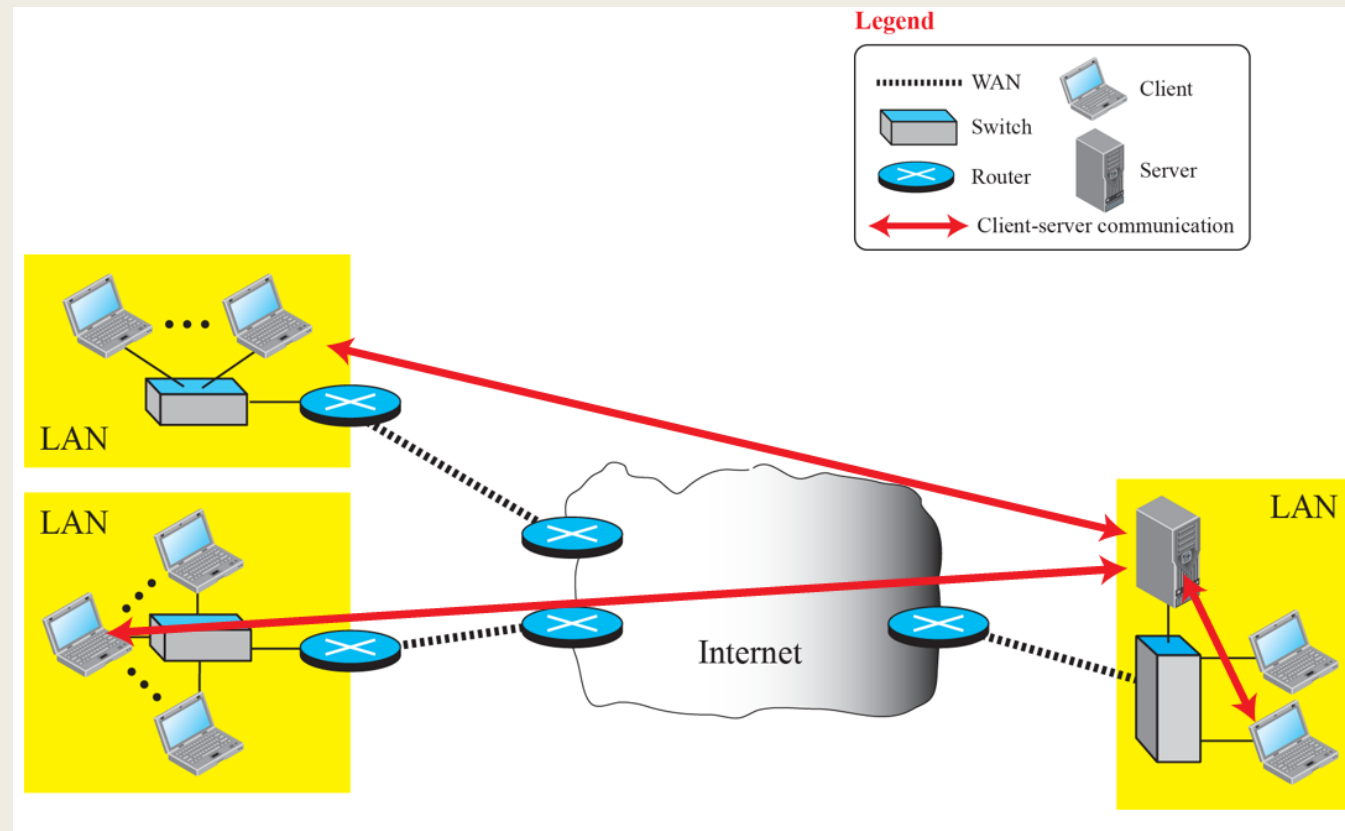
The two programs need to send messages to each other through the Internet infrastructure.

Two paradigms have been developed : the client-server paradigm and the peer-to-peer paradigm.

Client Server Paradigm



- The client/server paradigm divides software into two categories, clients and servers.
- A client is software that initiates a connection and sends requests, whereas a server is software that listens for connections and processes requests.

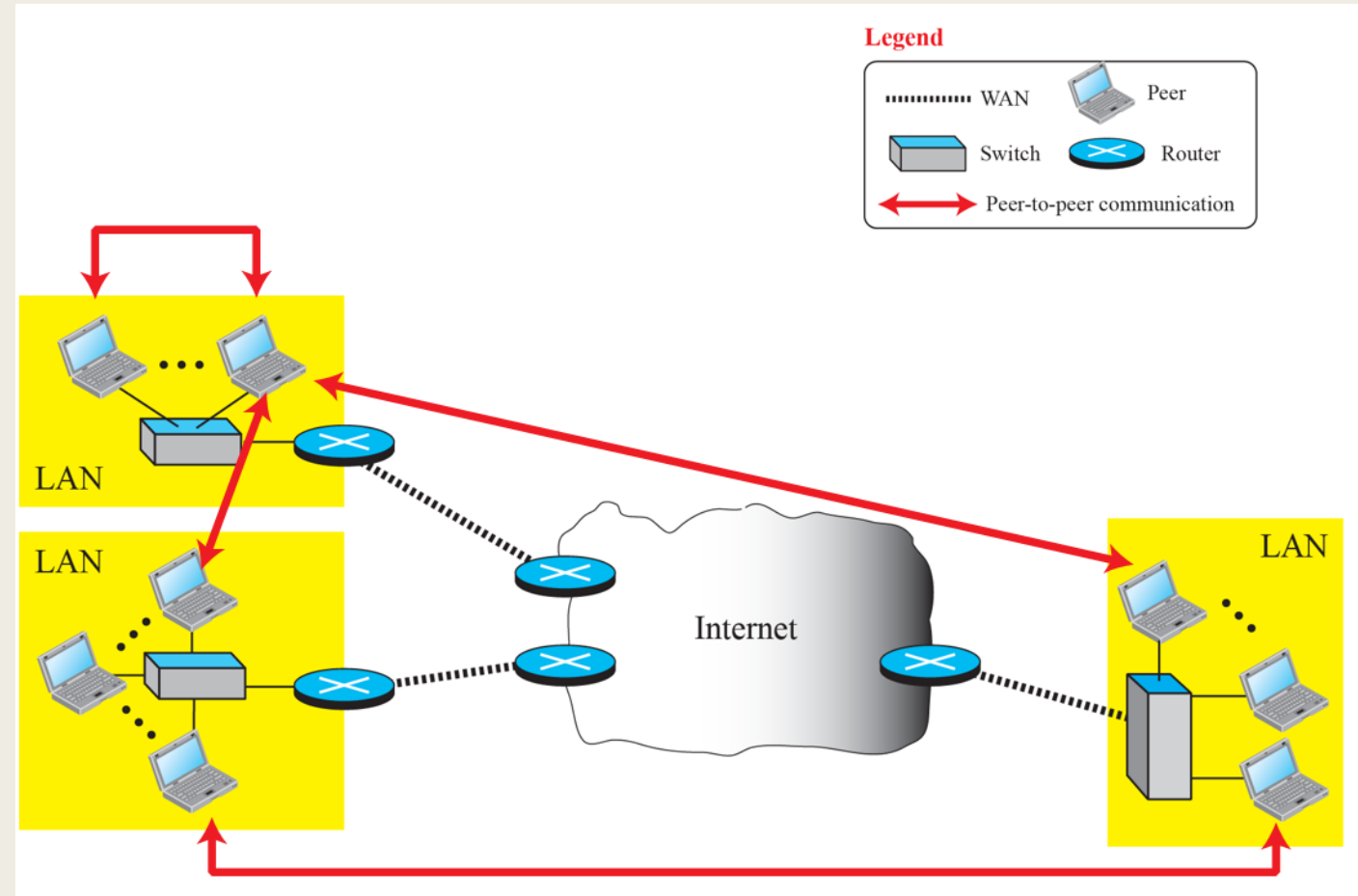


Source: B. A. Forouzan, "Data Communications and Networking,"
McGraw-Hill Forouzan Networking Series, 5E.

Peer to Peer Paradigm



- A peer-to-peer network allows computer hardware and software to communicate without the need for a server.
- Unlike client-server architecture, there is no central server for processing requests in a P2P architecture.
- The peers directly interact with one another without the requirement of a central server.



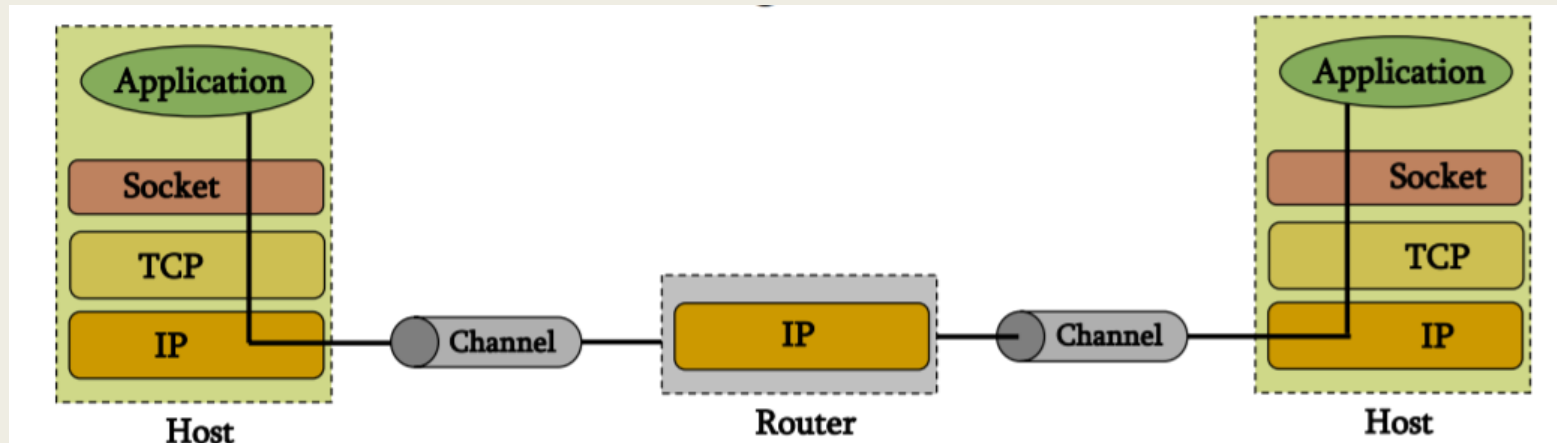
Sockets

A **socket** is one endpoint of a **two way** communication link between two programs running on the network.

The socket mechanism provides a means of inter-process communication (IPC) by establishing named contact points between which the communication take place.

Types:

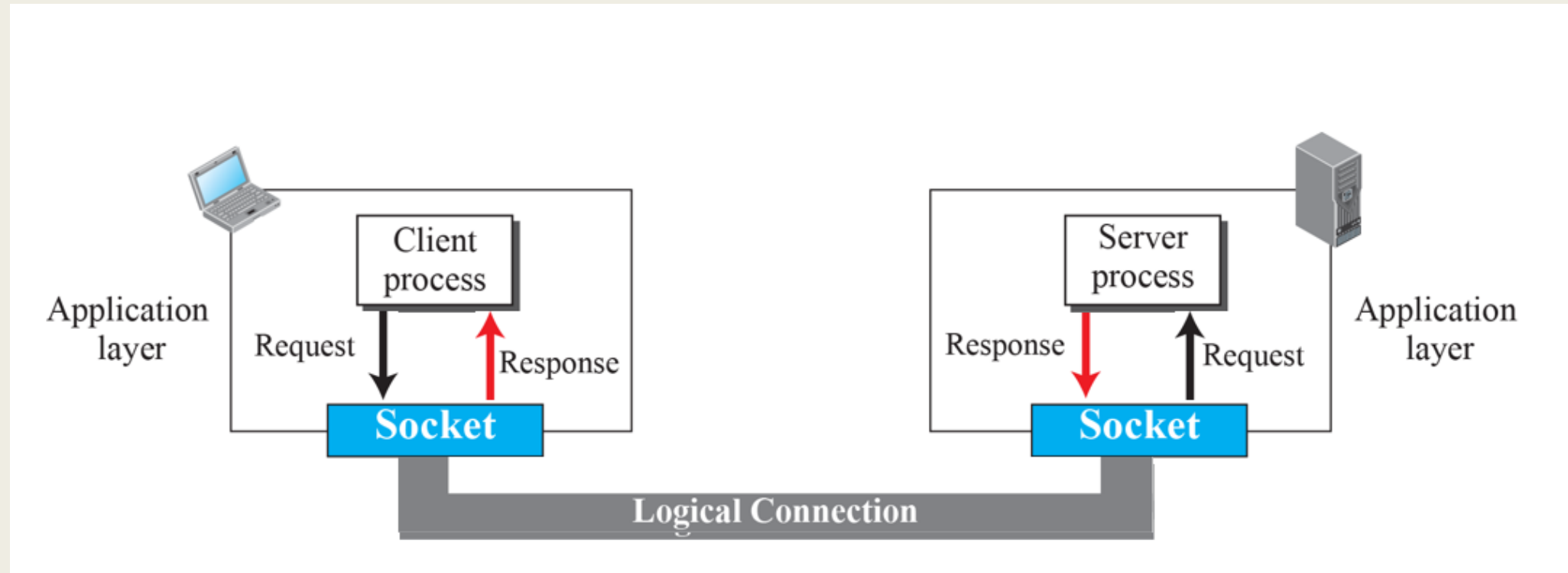
- Stream Sockets referred as "SOCK_STREAM"
- Datagram Sockets referred as "SOCK_DGRAM"
- Raw Sockets referred as "SOCK_RAW"



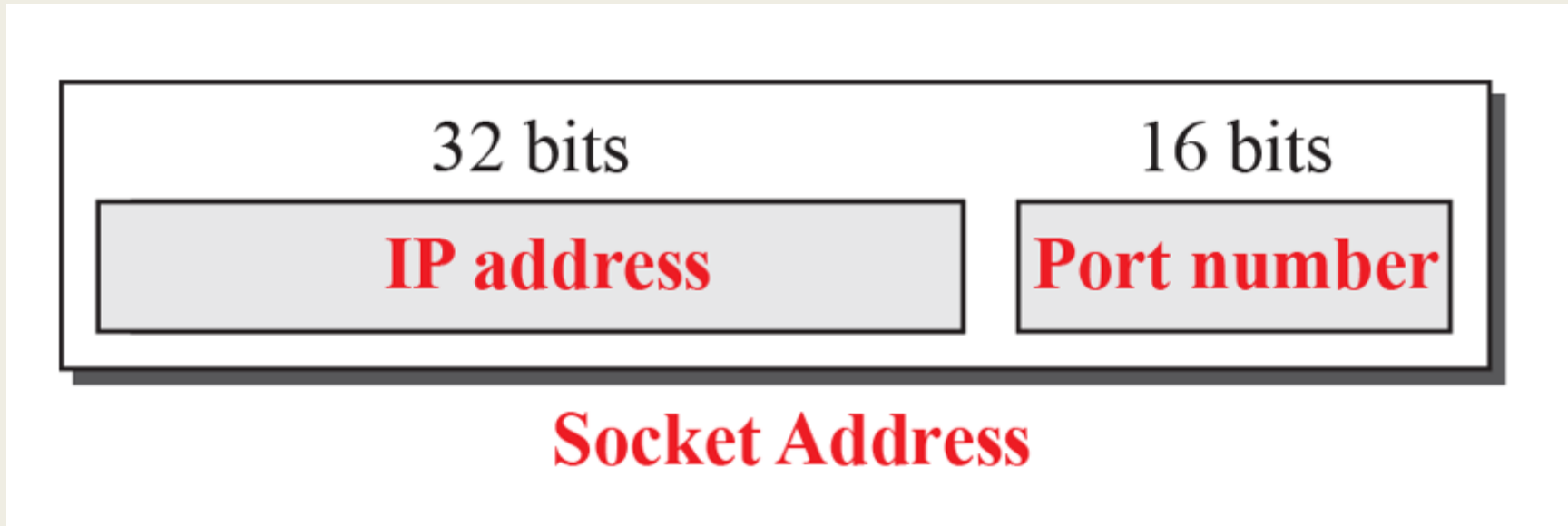
<https://www.csd.uoc.gr/~hy556/material/tutorials/cs556-3rd-tutorial.pdf>

Use of sockets in process-to-process communication

- Interprocess communication is the mechanism provided by the operating system that allows processes to communicate with each other.
- A pair of processes communicating over a network employs a pair of sockets, one for each process.



Socket Address

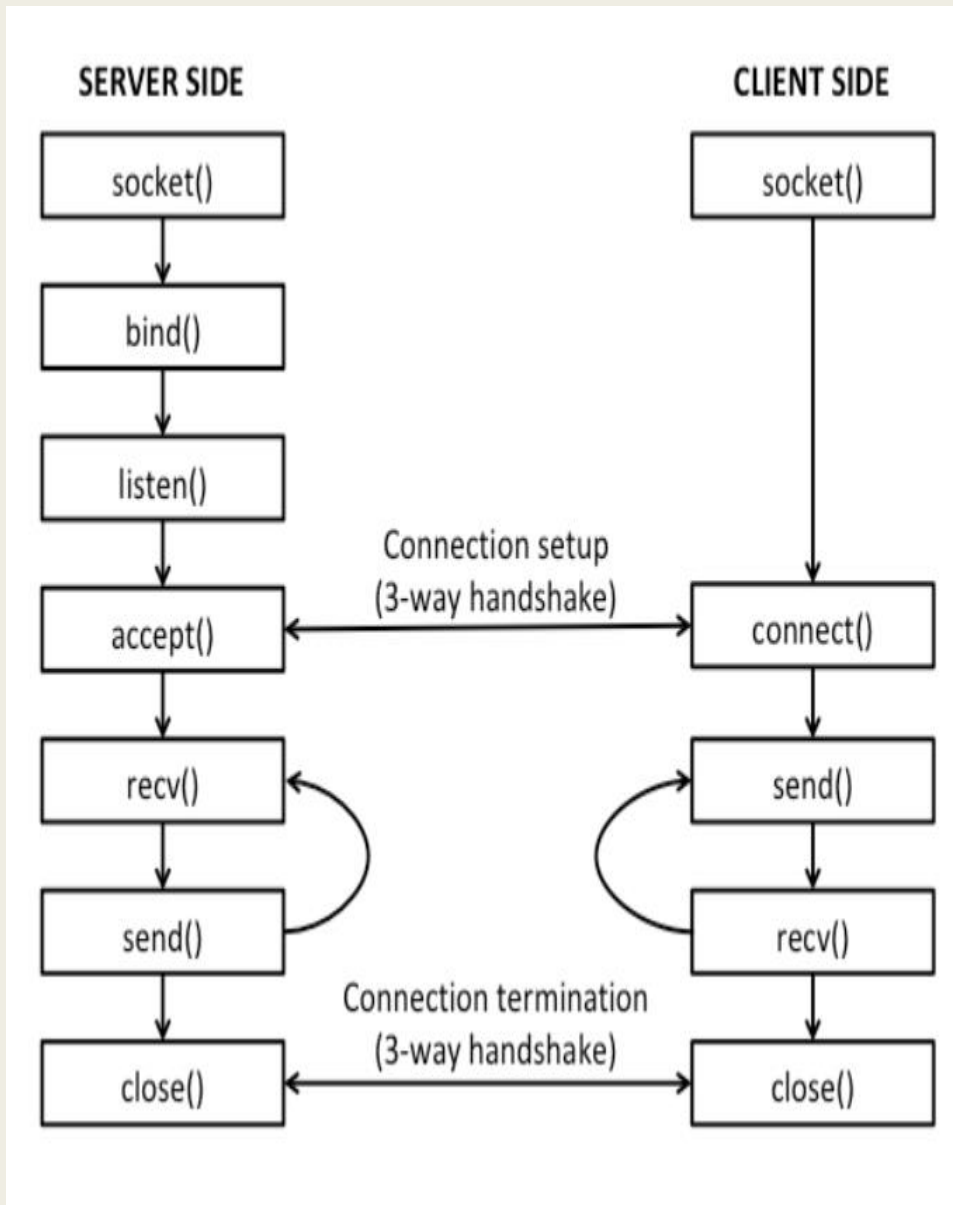




Socket Header Files

Item	Description
#include<netinet/in.h>	Contains definitions for the internet protocol family such as IPV4 or IPV6.
#include<arpa/nameser.h>	Contains the definitions used to support the construction of queries and the inspection of answers received from a Domain Name Server available in a network.
#include<netdb.h>	Contains the definitions for network database operations.
#include<resolv.h>	Contains resolver global definitions and variables.
#include<sys/socket.h>	Contains the definitions of socket addresses
#include<sys/socketvar.h>	Defines the kernel structure per socket and contains buffer queues.
#include<sys/types.h>	Contains the definitions of various data types.
#include<sys/un.h>	Contains the definition of the structures for the UNIX inter-process communication domain.
#include<sys/ndd_var.h>	Contains the definition of the structures for the operating system Network Device Driver (NDD) domain.
#include<sys/atmsock.h>	Contains the definitions of the structures for the Asynchronous Transfer Mode (ATM) protocol in the operating system NDD domain.

System Calls



- The `socket()` system call is used to create a socket descriptor on both the client and the server.
- `bind()` system call is used to bind the function with the particular IP and port.
- `listen()` system call hears to the various connections present in the network and selects the particular client to serve.
- The `connect()` function is then called on the client with three arguments, namely the socket descriptor, the remote server address and the length of the address data structure.
- `Accept()` extracts a connection on the buffer of pending connections in the system.
- A `send()` is used to send the message either from the client to server or vice versa.
- The `recv()` function is used to receive the messages at both the ends.
- The `close()` call is used to close the connection.



System Calls: socket()

The socket function is **used to create a new socket descriptor**.

It takes three arguments:

- the address family of the socket to be created
- the type of socket to be created
- the protocol to be used with the socket.

Syntax:

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
int socket(int domain, int type, int protocol);
```

Cont...



The address family of the socket to be created are:

- [AF_INET address family](#)
Members of AF_INET address family are IPv4 addresses.
- [AF_INET6 address family](#)
Members of AF_INET address family are IPv6 addresses.
- [AF_UNIX address family](#)
Members of AF_UNIX address family are names of Unix domain sockets
- [AF_UNIX_CCSD address family](#)
Members of AF_IPX address family are IPX addresses. IPX addresses are used to identify clients and servers on an IPX network. IPX is a networking protocol that conducts the activities and affairs of the end-to-end process of timely, managed and secured data.



System Calls: bind()

- bind() system call binds the service function with the particular port and IP.
- bind() returns integer type.
- Return -1 if binding failed
- Return 0 if binding is successful

Syntax:

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

sockfd is the socket file descriptor returned by socket().

my_addr is a pointer to a struct sockaddr that contains information about your address, namely, port and IP address.

addrlen can be set to sizeof(struct sockaddr)



System Calls: connect()

- connect() system call is the request send by client to the server for connection.
- Connect system call return integer:
- Return -1 if connection failed
- Return 0 if connection is established

Syntax:

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

sockfd is socket file descriptor, as returned by the socket() call
serv_addr is a struct sockaddr containing the destination port and IP address
addrlen can be set to sizeof(struct sockaddr).



System Calls: listen()

- listen() function listen to the various connections present in the network.
- On success, it return a non-negative integer that is a file descriptor of the client
- On error, -1 is returned

Syntax:

```
int listen(int sockfd, int backlog);
```

sockfd is the socket file descriptor from the socket() system call.

backlog is the number of connections allowed on the incoming queue.



System Calls: accept()

- accept() system call selects the particular connection from the waiting queue.
- On success, these system calls return a non-negative integer that is a file descriptor for the accepted socket of the client
- On error, -1 is returned

Syntax:

```
#include <sys/socket.h>
```

```
int accept(int sockfd, void *addr, int *addrlen);
```

sockfd is the socket descriptor return by socket().

addr is is a pointer to sockaddr.

addrlen is a local integer variable that should be set to sizeof(struct sockaddr_in).

System Calls: send()



- Used to transmit message to the other socket.
- Upon successful completion, *send()* returns the number of bytes sent.
- -1 shall be returned to indicate the error.

Syntax:

```
int send(int sockfd, const void *msg, int len, int flags);
```

sockfd is the socket descriptor you want to send data to (whether it's the one returned by `socket()` or the one got with `accept()`.)

msg is a pointer to the data you want to send.

len is the length of that data in bytes.

flags specifies the type of message transmission. Mostly the value of flag is 0 which allows to use a regular `send()`, with a standard behavior.



System Calls: `recv()`

- Used to receive message to the other socket.
- Upon successful completion, `recv()` returns the number of bytes received.
- -1 shall is returned to indicate the error.

Syntax:

```
int recv(int sockfd, void *buf, int len, unsigned int flags);
```

`sockfd` is the socket descriptor to read from

`buf` is the buffer to read the information into

`len` is the maximum length of the buffer

`flags` can be set to 0 to represent the standard behaviour of `recv`

`recv()` returns the number of bytes actually read into the buffer, or -1 on error.

System Calls: close()



Syntax:

```
close(sockfd);
```

This will prevent any more reads and writes to the socket.

Anyone attempting to read or write the socket on the remote end will receive an error.

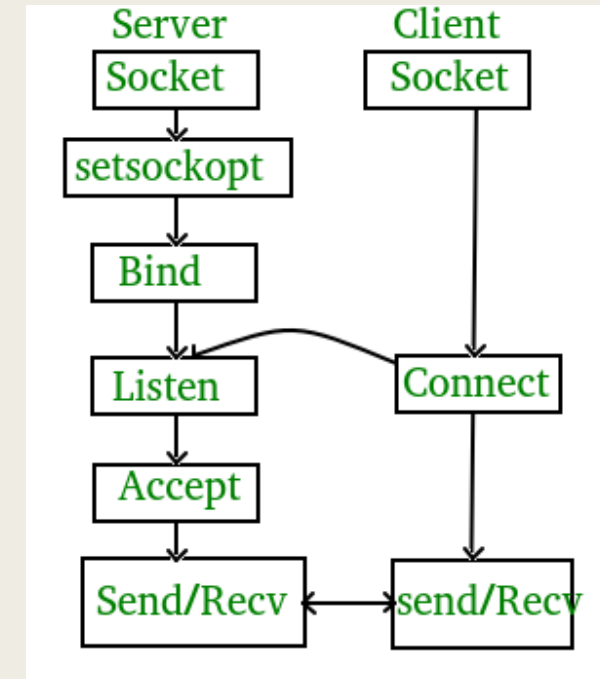
TCP Server and Client

TCP Server

- Using create(), Create TCP socket.
- Using bind(), Bind the socket to server address.
- Using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection
- Using accept(), At this point, connection is established between client and server, and they are ready to transfer data.
- Go back to Step 3.

TCP Client

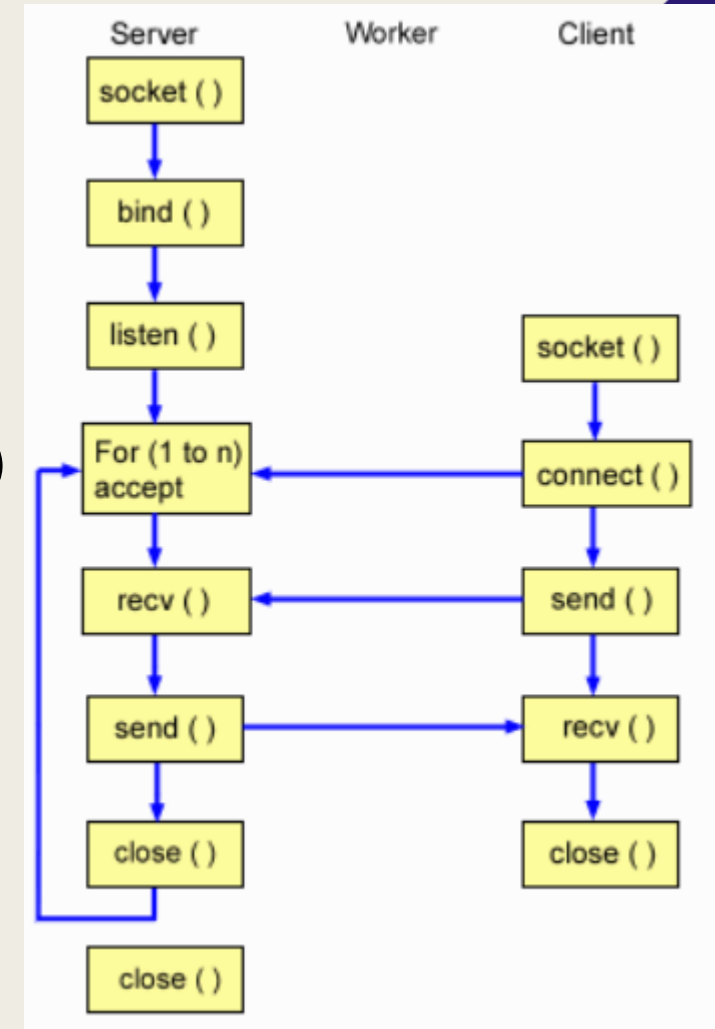
- Create TCP socket.
- connect newly created client socket to server.



<https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/>

Iterative TCP Server

- Server opens multiple passive TCP sockets each bound to a different port.
- Server keeps an array of function pointers to associate each socket with a service function.
- Server uses select to determine which socket (port) to service next.
- When a connection is ready, server calls accept to start handling a connection.
- Server calls the proper service function.



UDP Server and Client



UDP Server

- Create a UDP socket.
- Bind the socket to the server address.
- Wait until the datagram packet arrives from the client.
- Process the datagram packet and send a reply to the client.
- Go back to Step 3.

UDP Client

- Create a UDP socket.
- Send a message to the server.
- Wait until response from the server is received.
- Process reply and go back to step 2, if necessary.
- Close socket descriptor and exit.

Iterative UDP Server



- Server opens multiple UDP sockets each bound to a different port.
- Server keeps an array of function pointers to associate each socket with a service function.
- Server uses select to determine which socket (port) to service next and calls the proper service function.



Thank You!!!