Multiplexing and Demultiplexing

Introduction:

- These services are provided in every network architecture.
- Multiplexing and Demultiplexing are fundamental techniques in Computer Networks.
- They enable the efficient sharing of network resources over multiple data sources.
- Multiplexing and Demultiplexing are performed by using two internet protocols TCP and UDP.
- Two special fields are added in header file Source port number and destination port number

What is Multiplexing and Multiplexer ?

Multiplexing:

- Multiplexing is a set of technique that allows the simultaneous transmission of multiple signals across a single data link.

Multiplexer:

- Multiplexer is a device that combines several signals into a single signal or one one signal.

What is Demultiplexing and Demultiplexer ?

Demultiplexing:

- Delivering received segments at the receiver side to the correct app layer processes is called demultiplexing.

Demultiplexer:

- Demultiplexer is a device that performs the inverse operation.

Need of MUX and DEMUX

1-Real Time Communication

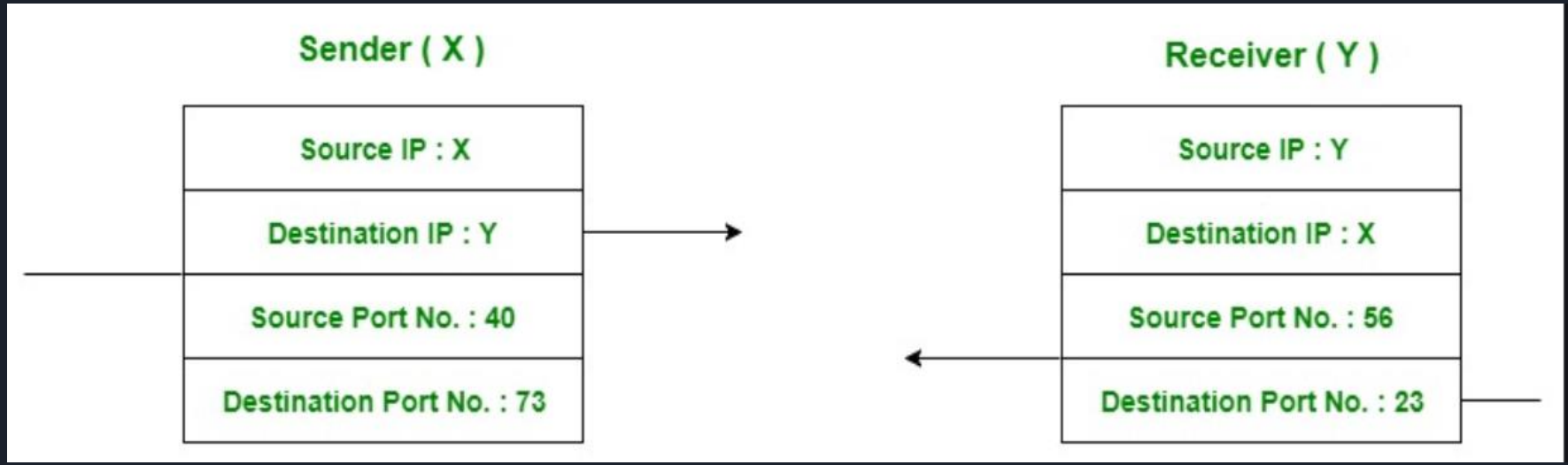
- Applications like voice and video calls require real time data sharing , necessitating efficient data transmission.

2-Improved Network Utilization

- Multiplexing allows multiple users to share the network without causing congestion.

How Multiplexing and Demultiplexing is done?

Sender and Receiver:



Example of Multiplexing and Demultiplexing

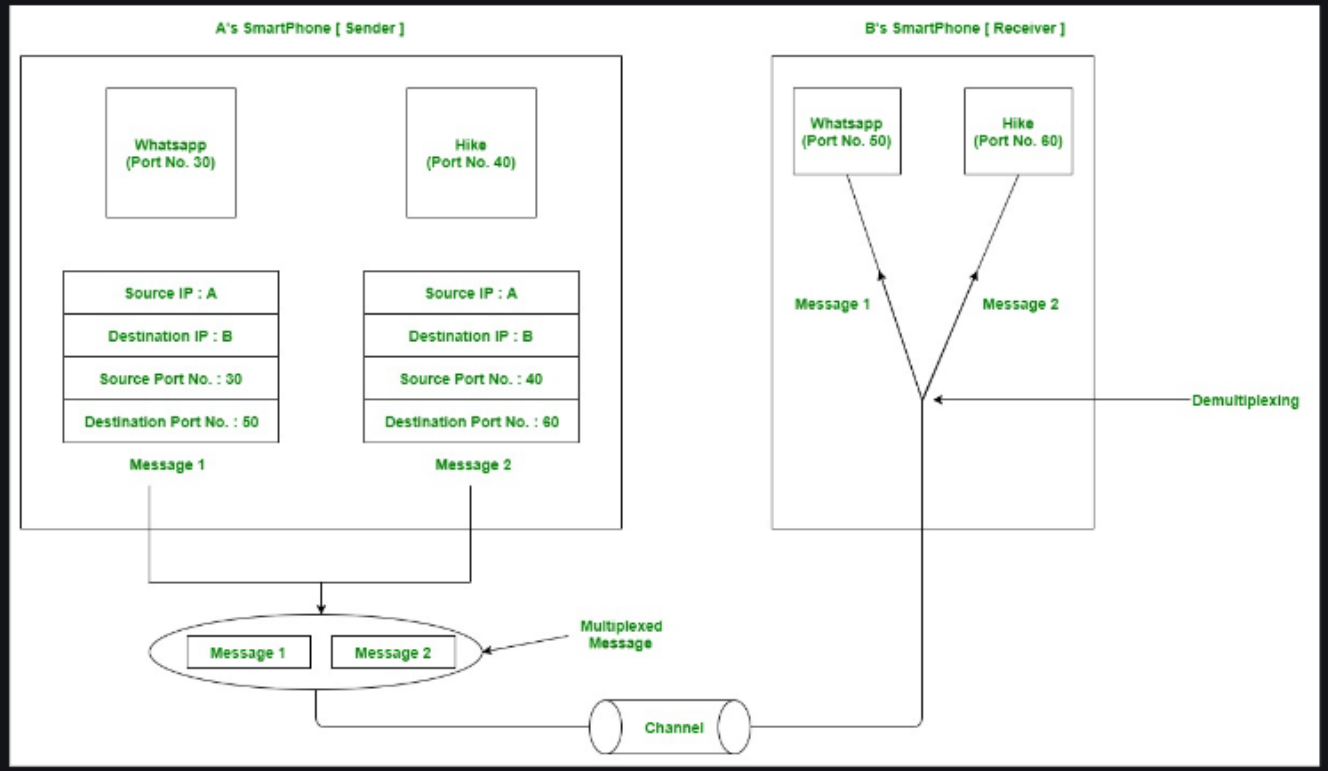


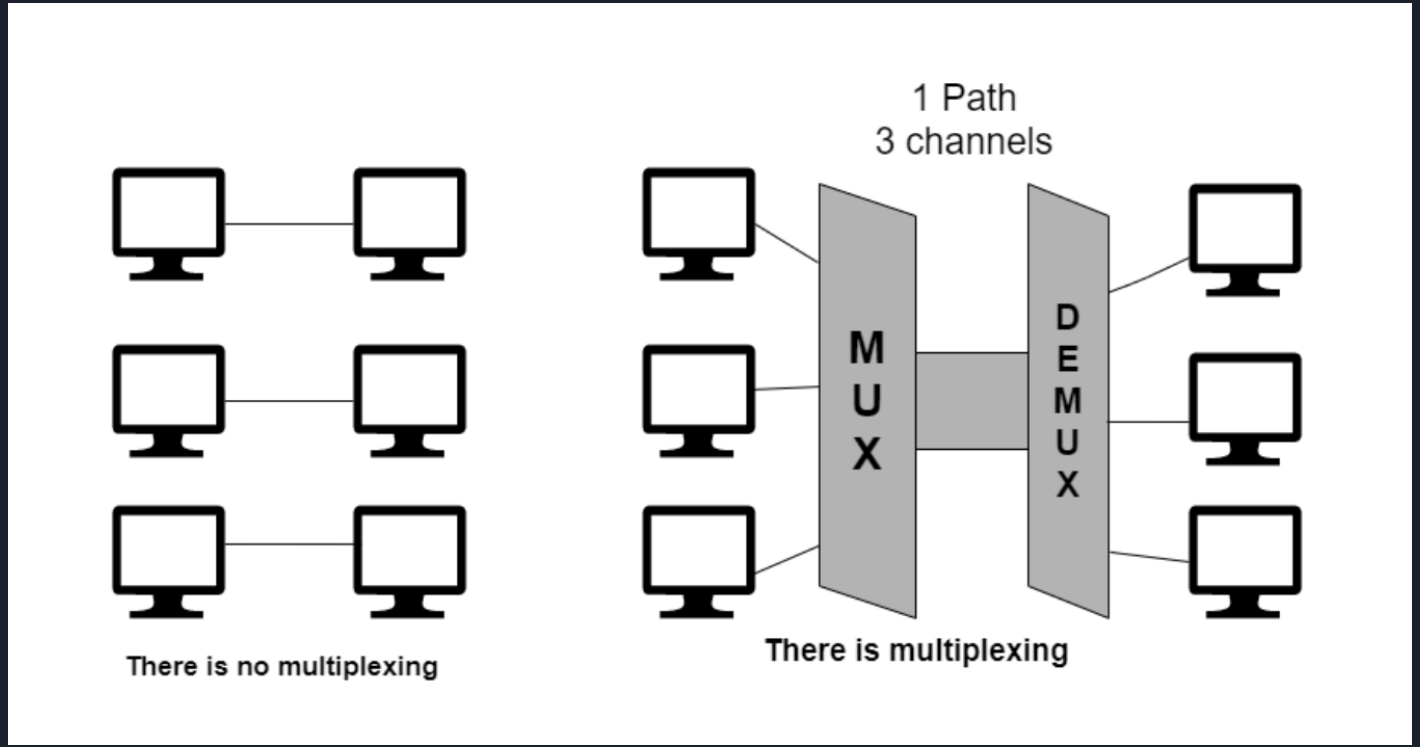
Figure – Message transfer using WhatsApp and hike messaging application

Explanation:

- A is the sender and B is the receiver.
- A wants to send message to B in Whatsapp and Hike both.
- For sending message on whatsapp A must mention the IP address of B and destination port number of whatsapp.
- For sending message on Hike A must mention the IP address of B and destination port number of Hike.
- Now messages will be wrapped as a single message and sent to receiver(Called Multiplexing).
- Now messages will be sent to whatsapp and hike A/C to port number(Called Demultiplexing).

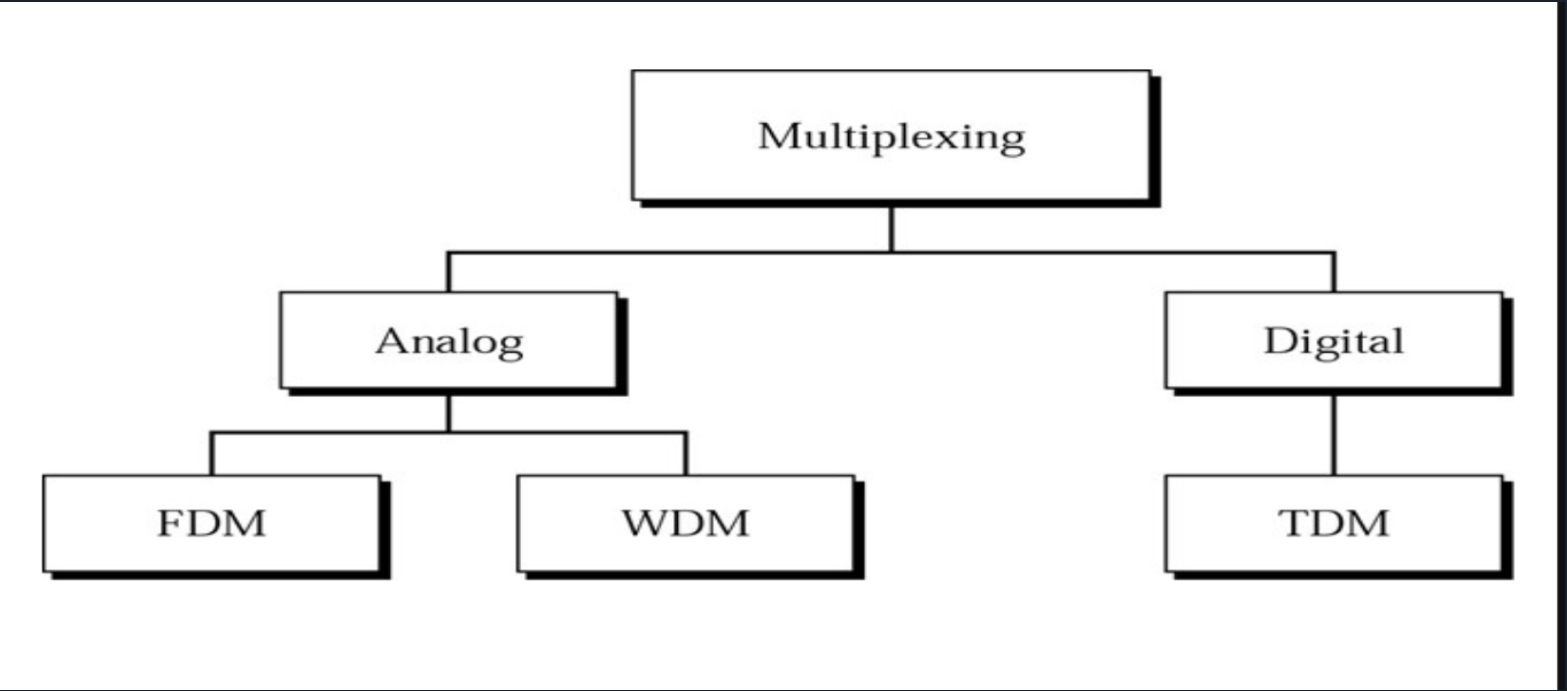
Diagramatic View

Multiplexing and No-Multiplexing



Categories of Multiplexing

Categories of Multiplexing:



Frequency Division Multiplexing

Frequency Division Multiplexing:

- FDM is applied when the bandwidth of a link is greater than the combined bandwidth of the signal to be transmitted.
- In FDM , Signals generated by each device modulate different carriers frequencies. These modulated signals are combined into a single composite signal that can be transported by the link.

Frequency Division Multiplexing(continue):

Frequency Division Multiplexing

Frequency Division Multiplexing(continue):

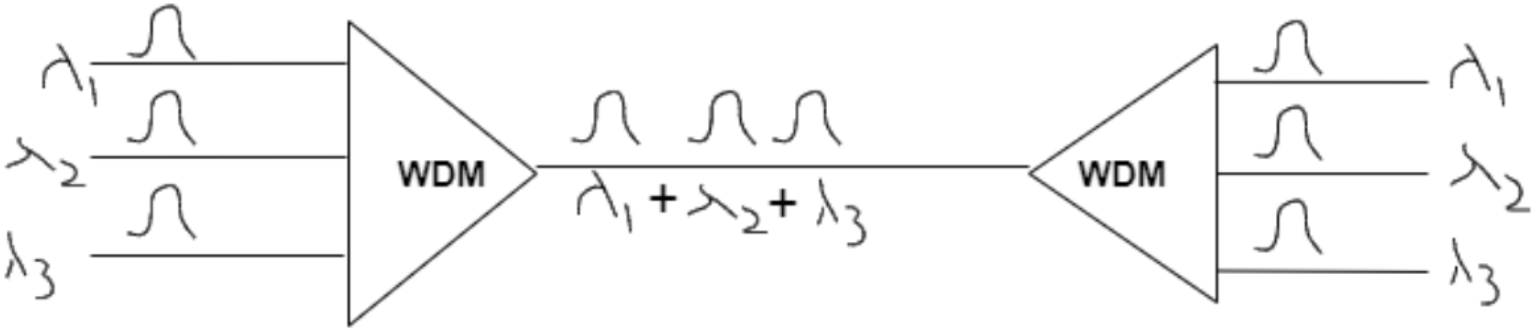
- From the previous Diagram;
- Transmission path is divided into three parts
- Each part represent a channel
- Each channel carry one transmission

Wave-Division Multiplexing

Wave Division Multiplexing

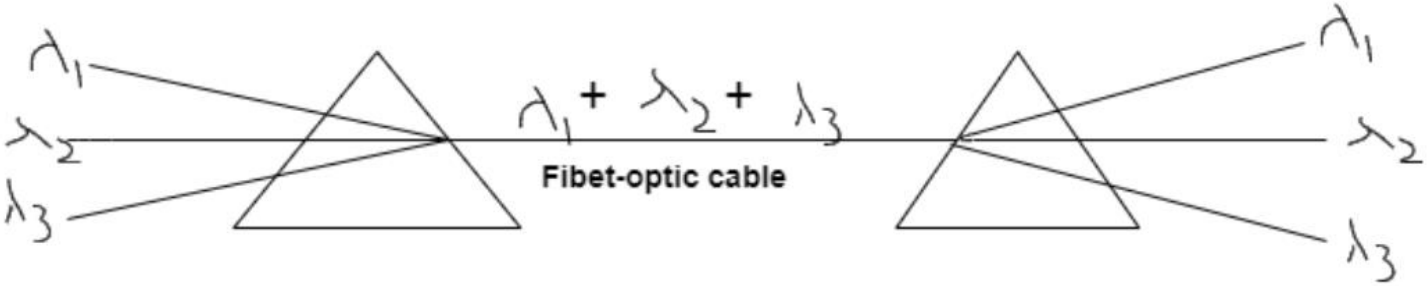
- This is similar to FDM
- Light signals are transmitted through optical fibre.
- Various optical signals are grouped together in the form of composite signals.
- This composite signal is transmitted through an optical fibre cable.

Wave Division Multiplexing(continue):



The above Figure indicates Wavelength Divison Multiplexing

Wave Division Multiplexing(continue):



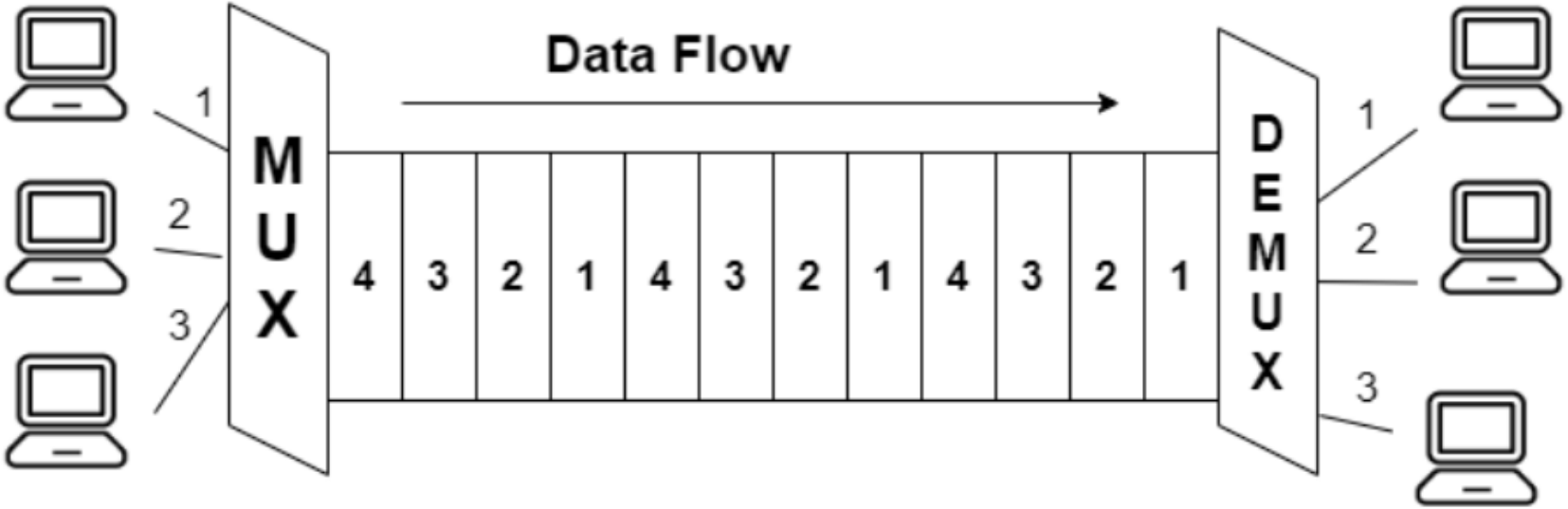
Use of Prism

Time-Division Multiplexing

Time Division Multiplexing(continue):

- Link is divided on the basis of time instead of Frequency
- In TDM , the data rate capacity should be much more greater than the data rate that is required by the sending and receiving device

Time Division Multiplexing(continue):



Types of Multiplexing and Demultiplexing

Types of MUX and DEMUX:

1-Connection Oriented Multiplexing and Demultiplexing

2-Connectionless Multiplexing and Demultiplexing

What is connectionless multiplexing and demultiplexing?

- Used in UDP(User Datagram Protocol)
- connectionless transport layer protocol
- no dedicated connection is establshed between sender and receiver
- packet(datagram)
- use port to transfer data to process

Connectionless Multiplexing

- Happen at the sending host
- application is identified by unique port number
- multiplexer add source port number in datagram
- network layer add source IP address
- two-tuple

Connectionless demultiplexing

- happen at the receiving host
- demultiplexer identifies destination port number

What is connection oriented multiplexing and demultiplexing?

- used in TCP(transmission control protocol)
- connection oriented transport layer protocol
- connection is established between sender and reciever
- use socket to transfer data to process

Connection oriented multiplexing

- identified by unique IP address and port number(socket)
- 4 tupples:
- source IP address
- source port
- destination IP address
- destination port

Connection oriented demultiplexing

- demultiplexer extract destination IP address and port number
- forward the segment to appropriate socket

Connectionless multiplexing and demultiplexing in detail....

Create UDP socket

- can be created using python
- code line:

clientSocket = socket(AF_INET, SOCK_DGRAM)

- socket is created...

- socket:

provide low level networking capabilities

create network socket

- AF_INET(address family-internet)

a constant terminology

create socket for IPv4 communication

- SOCK_DGRAM

Socket Kind Datagram

indicate type of socket

datagram->connectionless communication

send chunks

IPv4?

- network layer protocol for TCP/IP
- 32 bit addressing
- 4 decimal numbers->192.168.0.1
- connectionless
- best effort deliverey
- now->IPv6

Sides of application

Two sides

- client side
- server side

Assigning port no. at client side

- two ways
- automatically assign by transport layer
- range from 1024 to 65535(why not from 0 to 1023)
- manually assigning port number
- use method bind()

clientSocket.bind((’’, 19157))

- client socket

-> socket object

-> created using socket module

- bind

-> associate socket with local address and port number

- empty strings

-> bind to all available networks

- 19157

-> any port number

Assigning port no. at server side

- assigned by application developer
- well known port no. (0 to 1023)

How host A will send chunk to Host B?

- Host A perform multiplexing
- Host B perform demultiplexing

Transport layer segment includes:

- source port no.
- destination port no.
- length
- checksum

two-tuple

- destination IP address
- Destination port number

Scenario: What will happen?

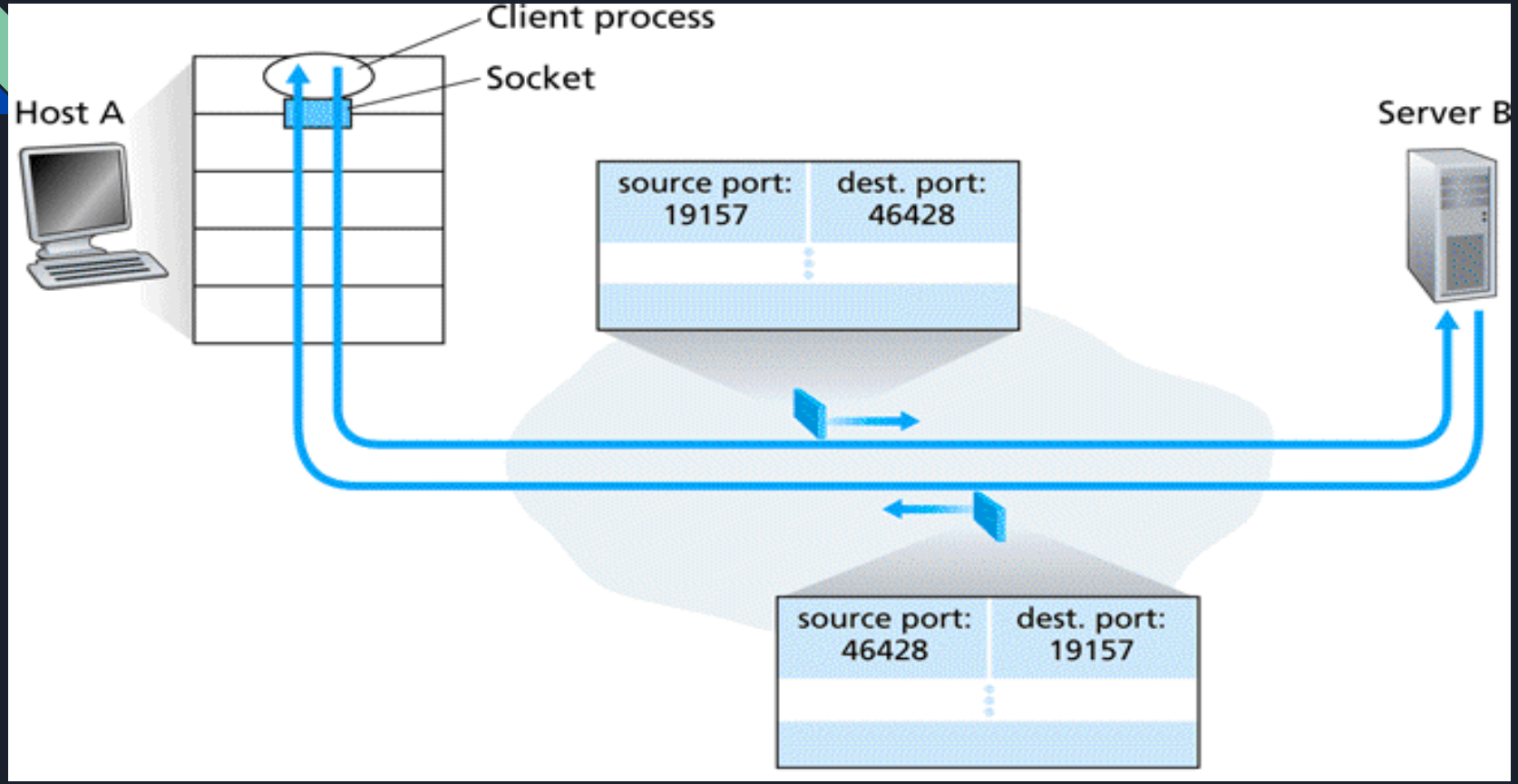
- differnt source IP address
- different source port number
- same destination address
- same port number

purpose of source port number:

- serve as return address
- recvfrom() method

->extract source port number

->use it as destination address


Client process
Socket
Host A
Server B
source port: 19157
dest. port: 46428
source port: 46428
dest. port: 19157

Connection-Oriented Multiplexing and Demultiplexing

TCP DEMULTIPLEXING

TCP sockets:

- used to identify application

TCP connection establishment:

- three-way handshake to establish a reliable connection

Three-way handshake to establish a reliable connection

1. Client sends SYN (Synchronize) Packet
2. Server responds with SYN_ACK (Acknowledge) Packet
3. Clients sends ACK Packet

When TCP segment arrives from the network to a host

4 tupples:

- source IP address
- source port number
- destination IP address
- destination port number

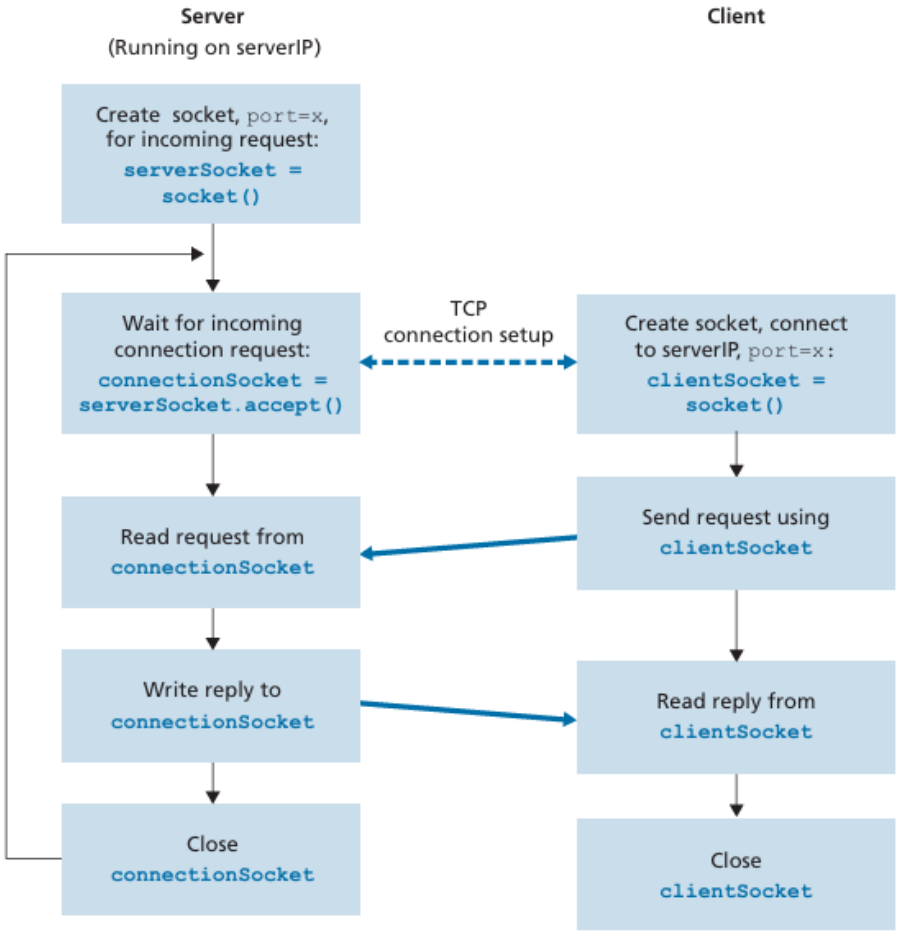


Figure 2.29 ♦ The client-server application using TCP

TCP client-server programming example

- Client Process

```
clientSocket = socket(AF_INET, SOCK_STREAM)

clientSocket.connect((serverName,12000))
```

Server Process

connectionSocket, addr = serverSocket.accept()

connectionsocket identify by :

1. the source port number in the segment
2. the IP address of the source host
3. the destination port number in the segment
4. its own IP address

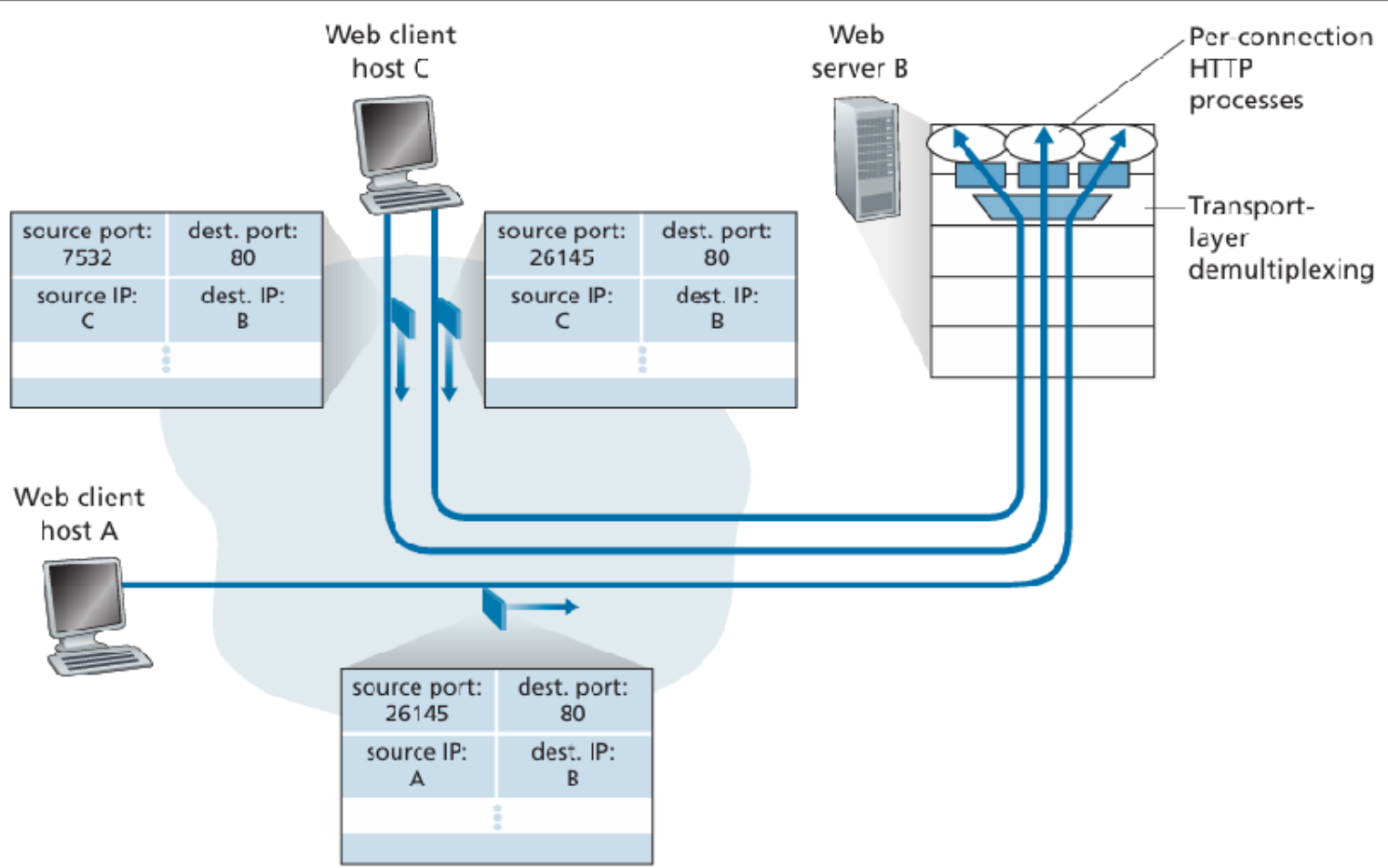


Figure 3.5 ♦ Two clients, using the same destination port number (80) to communicate with the same Web server application

Web Servers and TCP

- Each process has its own connectionsocket through which requests arrive and sent
- Not always a one-to-one correspondence between connection sockets and processes
- Process create a new thread with a new connection socket for each new client connection.

Persistent using:

- the client and server exchange HTTP messages via the same server socket

Non-Persistent using:

- a new TCP connection is created and closed for every request/response