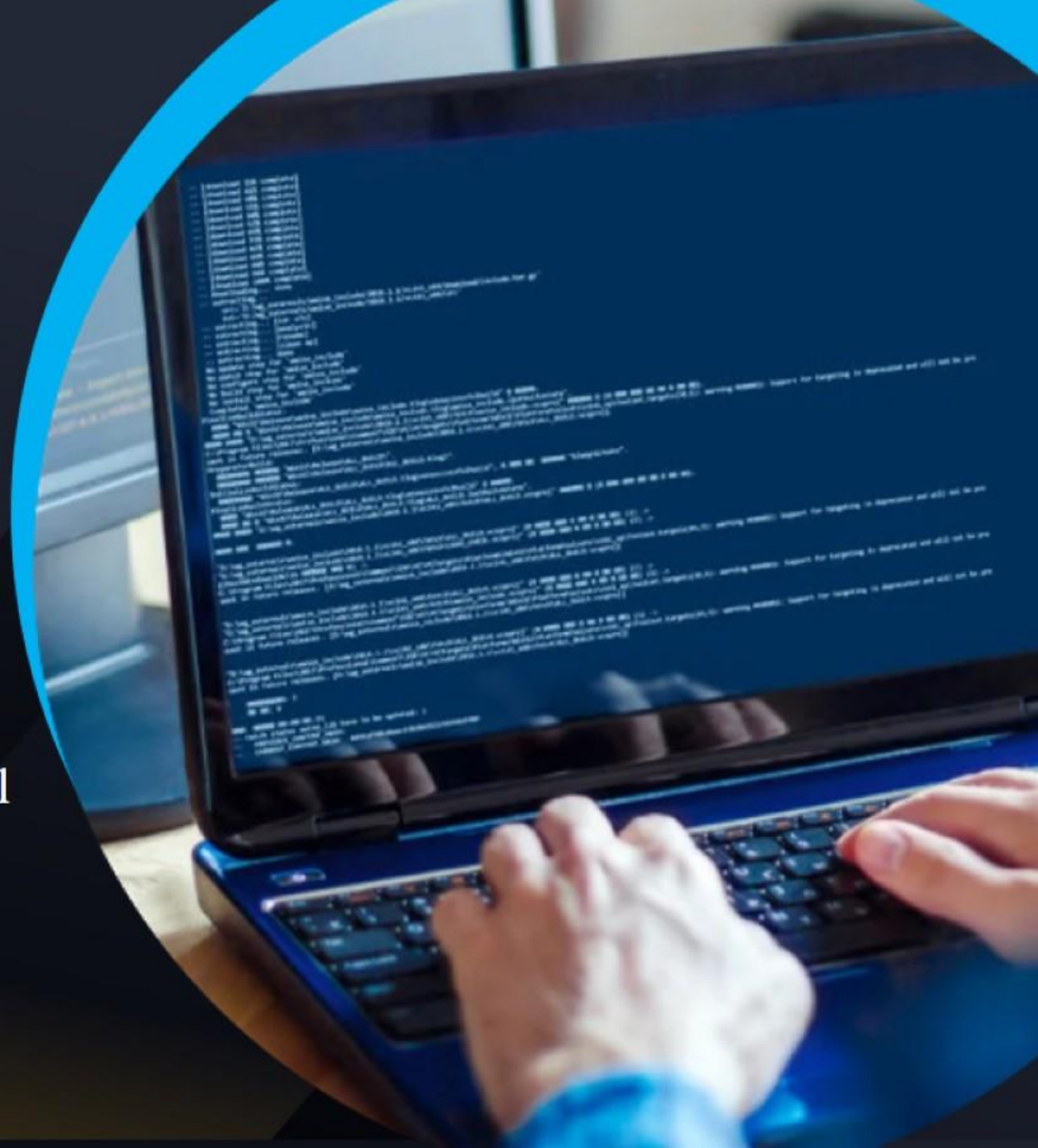


Network Programming in Python

Prepared by Dr. Hamada I. AbdulWakel

Lecture 4



The background features a series of vertical panels with rounded ends. The first panel shows hands typing on a laptop. The second panel has a graduation cap icon. The third panel has a play button icon. The fourth panel has a target icon. The fifth panel has an open book icon. The sixth panel has a magnifying glass icon. The seventh panel has a group of people icon. The eighth panel has a bar chart icon. The title text is overlaid on the right side of these panels.

Socket Programming Using Python – Part I

Lecture 4

Prepared by
Dr. Hamada I. AbdulWakel

www.midocean.edu.km

Lecture

04

RECAP: LECTURE 3

- BeautifulSoup and web scraping.
- Socket Programming in Python.

Lecture

04

LECTURE MAIN POINTS

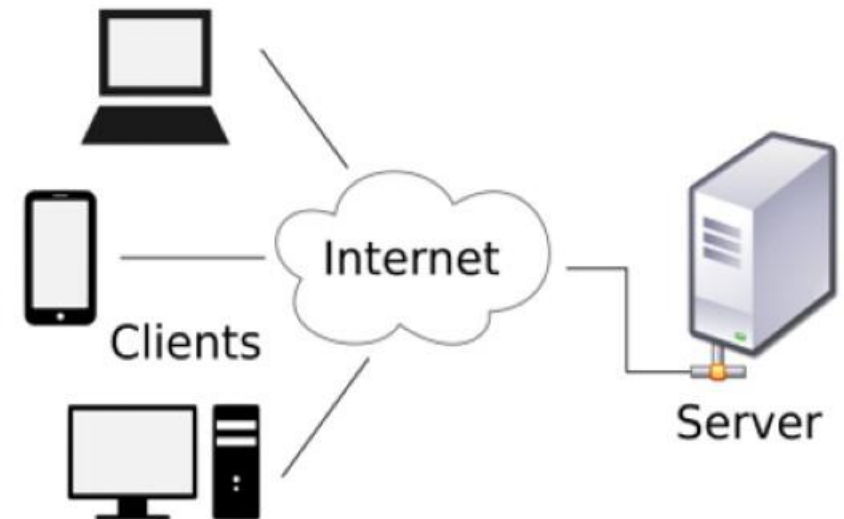
- The Client/Server Programming.
- The Communication Protocols.
- TCP and UDP Programming in Python.

Lecture

04

THE CLIENT/SERVER PROGRAMMING MODEL

- Every network application is based on the client-server model.
- An application consists of a server process and one or more client processes.
- A server manages some resource, and it provides some service for its clients by manipulating that resource.

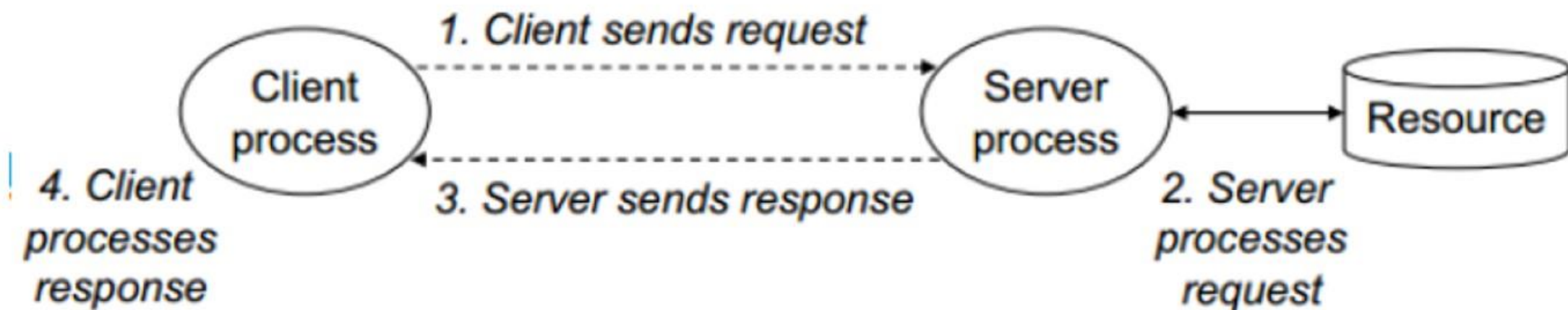


Lecture

04

THE CLIENT/SERVER PROGRAMMING MODEL

Cont'd



THE CLIENT/SERVER PROGRAMMING MODEL

Cont'd

- A client-server transaction consists of four steps.
 1. When a client needs service, it initiates a transaction by sending a request to the server.
 2. The server receives the request, interprets it, and manipulates its resources in the appropriate way.
 3. The server sends a response to the client, and then waits for the next request.
 4. The client receives the response and manipulates it.

Lecture

04

THE COOMUNICATION PROTOCOLS

- Any computer usually has:
 - A single physical connection to the network.
 - Data could be sent by different applications through the use of ports configured on the same physical network connection.
- A computer's address is a 32-bit IP address.
- The port number is a 16-bit number ranging from 0 to 65,535, with ports 0-1023 restricted by well-known applications like HTTP and FTP.

Lecture

04

THE COOMUNICATION PROTOCOLS

Cont'd

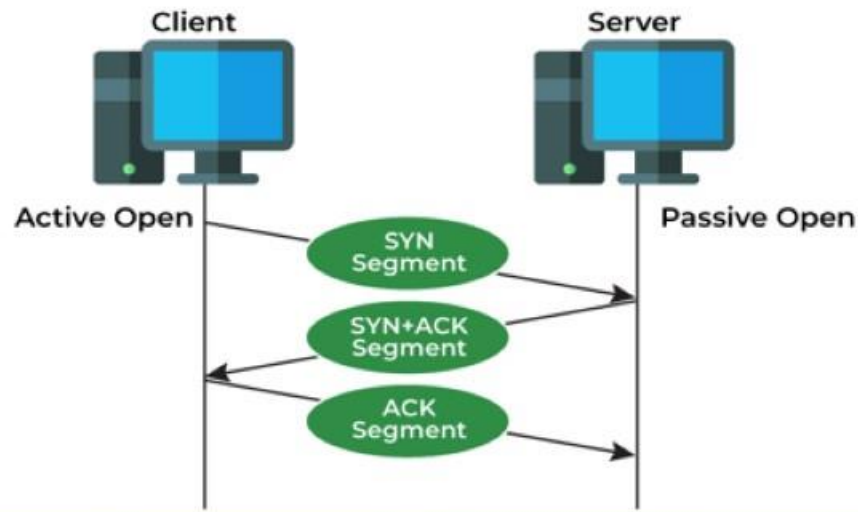
- Every packet of data has:
 - Source IP, Source Port, Destination IP, and Destination Port.
 - Allow computing devices and applications to send data via a network and verify its delivery.
- There are two types of client-server application communication protocol:
 - Transmission Control Protocol (TCP).
 - User Datagram Protocol (UDP).

Lecture

04

THE TRANSMISSION CONTROL PROTOCOL (TCP)

- A connection-oriented protocol that provides a reliable flow of data between end-to-end connection.
 - Three-way handshake. "Wanna connect?" "Sure!" "Alright then!".

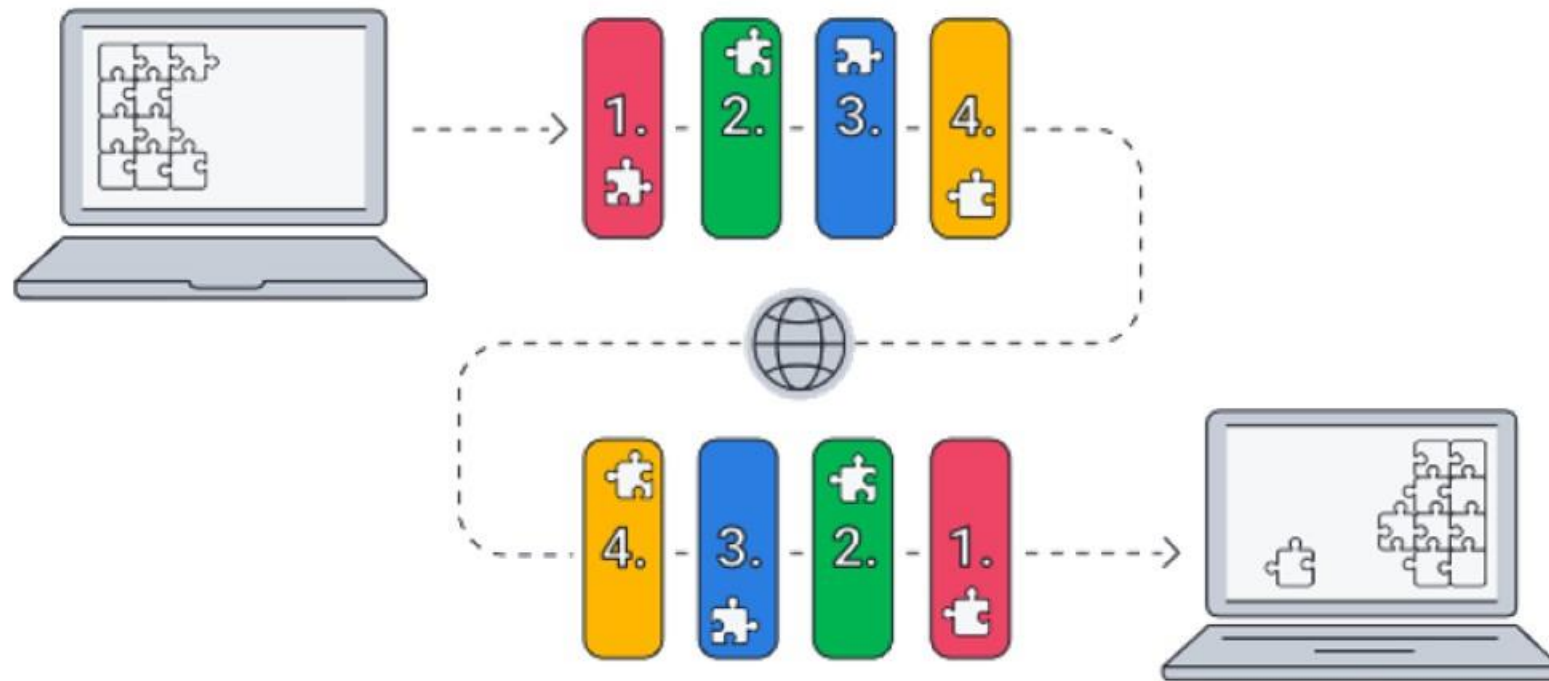


Lecture

04

THE TRANSMISSION CONTROL PROTOCOL (TCP)

Cont'd



THE TRANSMISSION CONTROL PROTOCOL (TCP)

Cont'd

- TCP features:
 - Slow overhead time of setting up.
 - Retransmission.
 - Bigger overhead.
 - In-order delivery.
 - Congestion control.
 - Delays transmission when network is congested.
 - Data doesn't always get sent out immediately (i.e., voice chat).

THE USER DATAGRAM PROTOCOL (UDP)

- A Connection-less protocol that sends independent packets of data (datagrams), from one computer to another.
 - No connection to create and maintain.

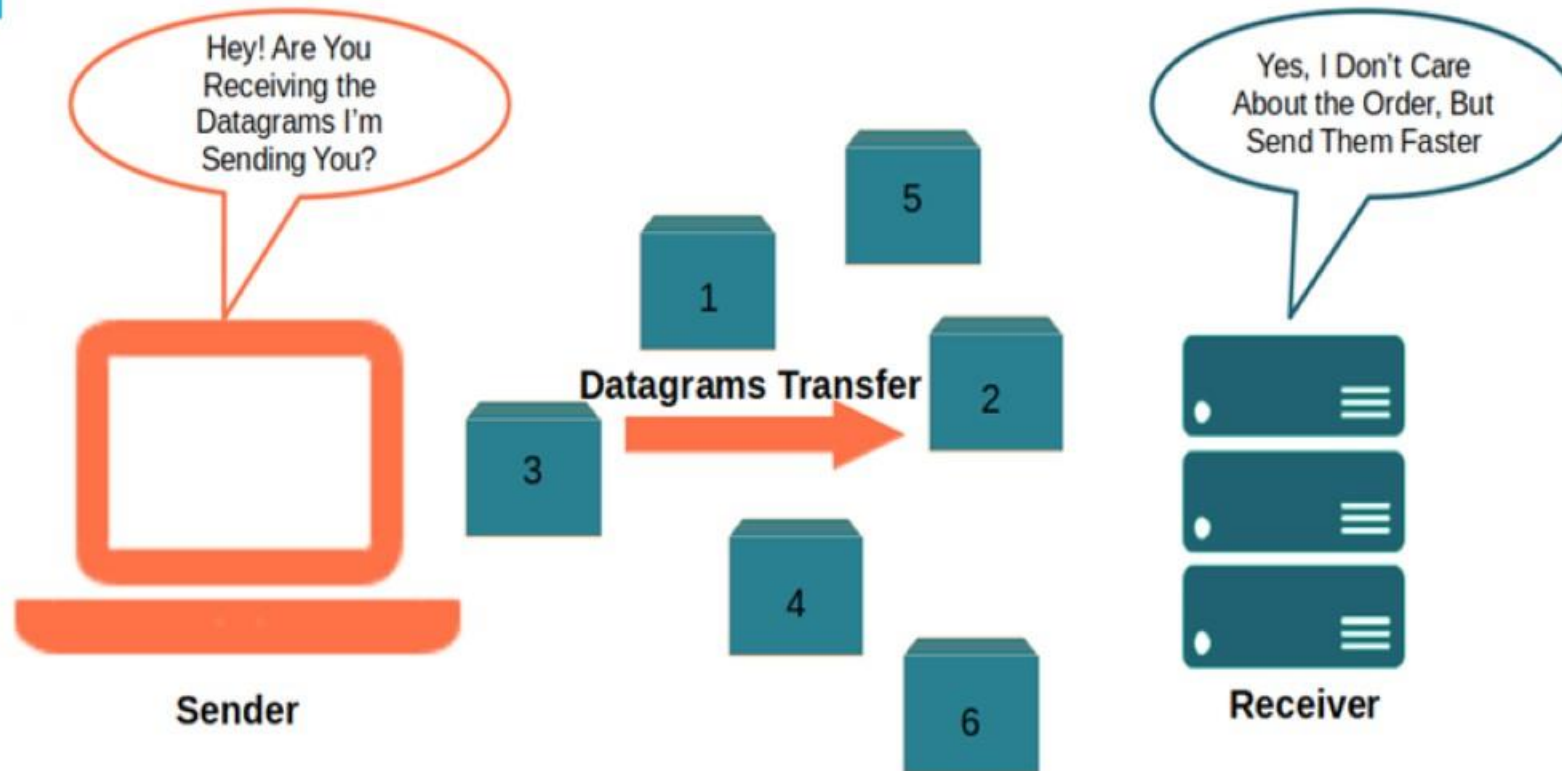


Lecture

04

THE USER DATAGRAM PROTOCOL (UDP)

Cont'd



Lecture

04

THE USER DATAGRAM PROTOCOL (UDP)

Cont'd

- UDP features:
 - Faster since no overhead of setting up a connection.
 - Small packet sizes.
 - No compensation for lost packets.
 - Packets can arrive out of order.
 - No error recovery.
 - No congestion control.

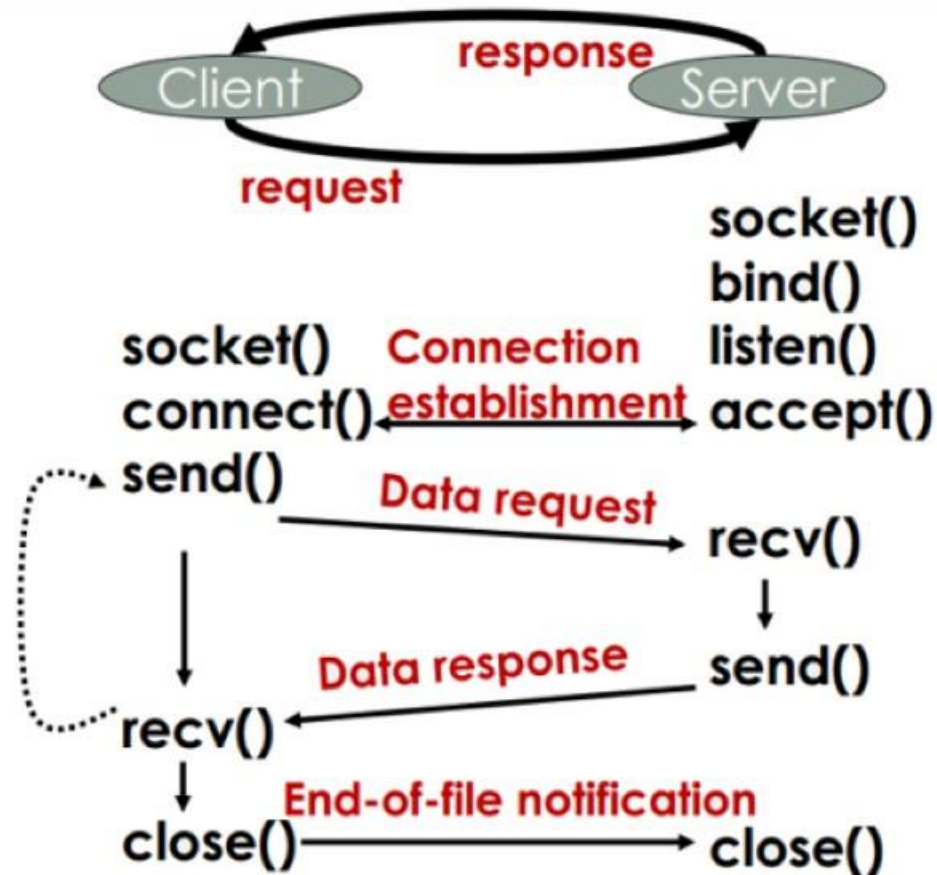
Lecture

04

**THE TCP AND UDP IN SOCKET
MODULE**

- Both protocols (TCP and UDP) are supported by socket module.
 - A socket is a file-like object.
 - Allows data to be sent and received across the network like a file.
 - But it also includes functions to accept and establish connections.
- Before two machines can establish a connection, both must create a socket object.

THE SYSTEM CALLS IN TCP



Lecture

04

THE SERVER SOCKET CREATION STEPS

- To create a socket server, follow the following steps:
 1. Create a socket object.
 2. Get the current machine name and the port number for connection.
 3. Bind the socket object with the address.
 4. Listen for any connections.
 5. Accept requests from clients.
 6. **Send & receive messages.**
 7. Close the connection.

Lecture

04

THE CLIENT SOCKET CREATION STEPS

- To create a socket client, follow the following steps:
 1. Create a socket object.
 2. Identify the server machine name (i.e., hostname) and the port number where the client can find the server.
 3. Connect to the server.
 4. **Send & receive messages.**
 5. Close the connection.

SOME SOCKET METHODS

- **gethostname():** returns the host name of the current system under which the Python interpreter is executed.

```
print( socket.gethostname() )
```

- **gethostbyname():** returns the IP address of the host.

```
print( socket.gethostbyname("www.github.com") )
```

SOME SOCKET METHODS

Cont'd

- **gethostbyaddr():** returns a tuple containing (Host Name, Alias list for the IP address if any, IP address of the host)

```
ipAddress = "192.0.43.8"  
hostName = socket.gethostbyaddr(ipAddress)  
print( hostName )
```

- **getprotobyname():** takes a protocol string like TCP, UDP, or FTP and returns the associated constant for the protocol as defined by the socket module.

```
print( socket.getprotobyname("TCP") )
```

SOME SOCKET METHODS

Cont'd

- **getservbyname():** takes a network service name and an underlying protocol name and returns the port number on which the service is defined.

```
print(socket.getservbyname("https", "tcp"))
```

- **getservbyport():** returns the name of the service for a given port number.

```
print(socket.getservbyport(21, "tcp"))
```

SOME SOCKET METHODS

Cont'd

- **recv(bufsize):** receives the TCP message from socket. The argument **bufsize** stands for buffer size and defines the maximum data.
 - You should use the **decode()** method when receiving data from the socket to convert it to a string.
- **send(bytes):** sends data to the socket which is connected to the remote machine. The argument **bytes** will give the bytes sent to the socket.
 - You should use the **encode()** method to convert the string to bytes.

Lecture

04

A SIMPLE CLIENT/SERVER APPLICATION USING TCP

- Code: 1- the server

```
import socket
soc = socket.socket()
print("socket created")
soc.bind( (socket.gethostname(), 5485) )
soc.listen()
print("Waiting for connections .....")
```

A SIMPLE CLIENT/SERVER APPLICATION USING TCP

Cont'd

- Code: 1- the server
 while True:
 #return client socket and address
 clientSoc, clientAddre = soc.accept()
 name = clientSoc.recv(1024).decode()
 print("Connected with", clientAddre, name)
 clientSoc.send(("Welcome we are connected :) " + name).encode())
 clientSoc.close()

A SIMPLE CLIENT/SERVER APPLICATION USING TCP

Cont'd

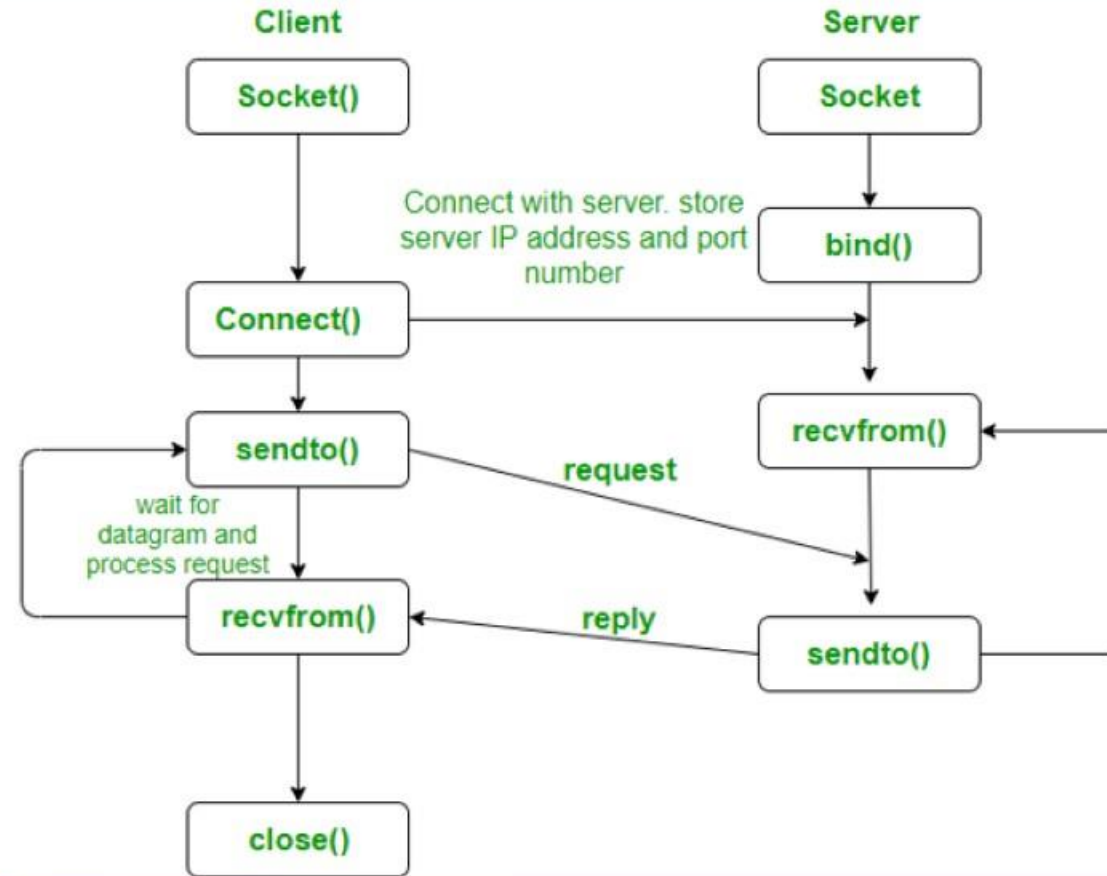
- Code: 2- the client

```
import socket
soc = socket.socket()
# the address is the same as in the server
soc.connect( (socket.gethostname(), 5485) )
name = input("Enter your name? ")
soc.send(name.encode())
print(soc.recv(1024).decode())
soc.close()
```

Lecture

04

THE SYSTEM CALLS IN UDP



Lecture

04

SOME SOCKET METHODS

- **recvfrom(bufsize):** receives the maximum data size, bufsize. Two pair (data, address) value is returned by this method.
 - Data defines the received data and address specifies the address of socket sending the data..
 - You should use the `decode()` method when receiving data from the socket to convert it to a string.
- **sendto(message, address):** sends message as bytes to the socket specified with the given address.
 - You should use the `encode()` method to convert the string to bytes.

Lecture

04

A SIMPLE CLIENT/SERVER APPLICATION USING UDP

- Code: 1- the server

```
import socket
localIP = "127.0.0.1"
localPort = 20001
UDPsoc = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
UDPsoc.bind((localIP, localPort))
print("UDP server up and listening")
```

A SIMPLE CLIENT/SERVER APPLICATION USING UDP

Cont'd

- Code: 1- the server

```
while True:
    bytesAddressPair = UDPServerSocket.recvfrom(1024)
    message = bytesAddressPair[0].decode()
    address = bytesAddressPair[1]
    print("The client IP address:", address)
    print("The message from client:", message)
    message = message.upper().encode()
    UDPSoc.sendto(message, address)
```

A SIMPLE CLIENT/SERVER APPLICATION USING UDP

Cont'd

- Code: 2- the client

```
serverAddress = ("127.0.0.1", 20001)
UDPcplt = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while True:
    data = input("Enter a word: ")
    data = data.encode()
    UDPcplt.sendto(data, serverAddress)
    msgFromServer = UDPcplt.recvfrom(1024)
    msg = "Message from Server"+ msgFromServer[0].decode()
    print(msg)
```

Lecture

04

QUESTIONS FOR ONLINE DISCUSSION

- Write a program to create a client/server application that pretends to be a calculator. It begins by asking the client about his name and age, then ask him to enter two numbers and the operation he want to perform. The server should calculate the result of the required operation and send it to the client.
- Write a program to create a UDP client/server program that calculate and send the factorial ($n!$) of a given number send by a client.



Thank You

