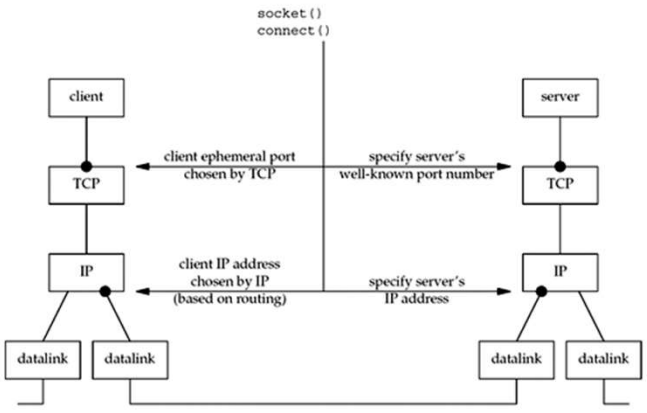
School of Engineering and Applied Science
Department of Computer Science & Information Technology

PROJECT 4

CSIT at UDC
CSCI 351-Computer Networks

1

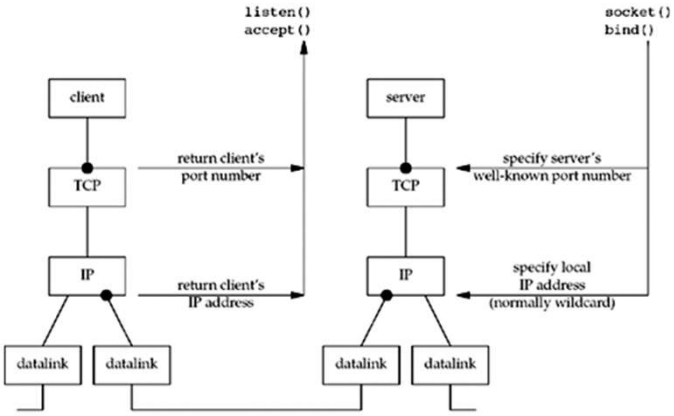
TCP Client



Client's Perspective

2

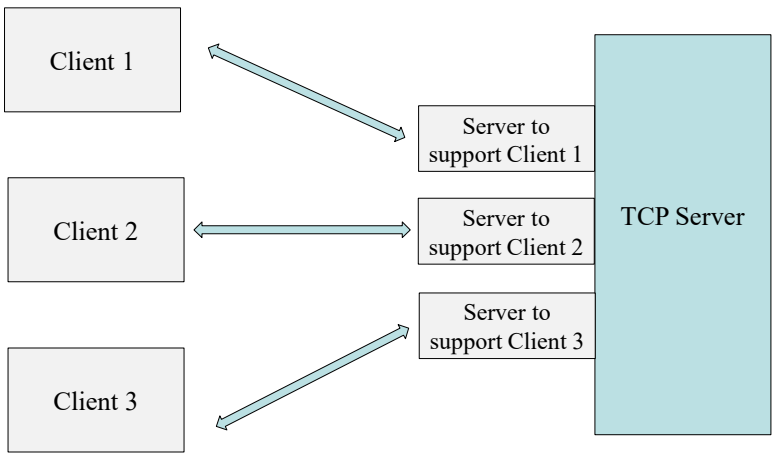
TCP Server



Server's Perspective

3

Supporting Multiple Clients



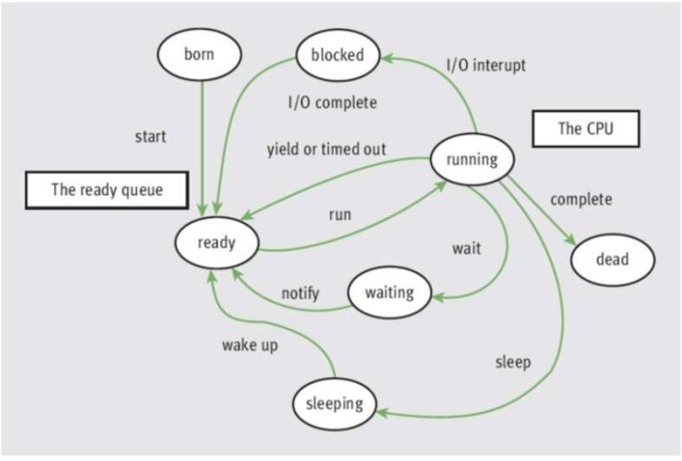
4

Threads in Python

- In Python, a thread is an object like any other in that it can hold data, be run with methods, be stored in data structures, and be passed as parameters to methods
- A thread can also be executed as a process
  - Before it can execute, a thread's class must implement a **`run`** method
- During its lifetime, a thread can be in various states

5

Threads



Source: Fundamentals of Python: From First Programs Through Data Structures

6

Threads (continued)

- A thread remains inactive until **`start`** method runs
  - Thread is placed in the **ready queue**
  - Newly started thread's **`run`** method is also activated
- A thread can lose access to the CPU:
  - Time-out (process also known as **time slicing**)
  - Sleep
  - Block
  - Wait
- Process of saving/restoring a thread's state is called a **context switch**

Slide #7

7

TCP Server in Python

```
server_address = (“”, PORT)
serversock = socket(AF_INET, SOCK_STREAM)
serversock.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
serversock.listen(5)

while True:
        print 'waiting for connection... listening on port', PORT
        clientsock, client_address = serversock.accept()
        print 'client connected: address ', client_address
        x = threading.Thread(target=handler, args=(clientsock, client_address,))
        x.start()
```

8

Handler function for a Thread

```
def handler(clientsock, addr):
    while True:
        data = clientsock.recv(1024)
        if not data:
            break
        sys.stderr.write('received "%s"\n' % data.decode())
        if "exit" == data.decode():
            break
    clientsock.close()
```

9

Question 1 (30 Points)

- Design a TCP server capable of handling simultaneous connections from multiple clients, with a maximum limit of five clients.
- The server's architecture is engineered to enable broadcasting to all connected clients through the TCP server.
- It is essential that each client can identify the sender of a message. Therefore, assign a unique ID to each client.
- This design ensures that when a message is broadcasted from one client, all other clients can identify which specific client sent the message.

Slide #10

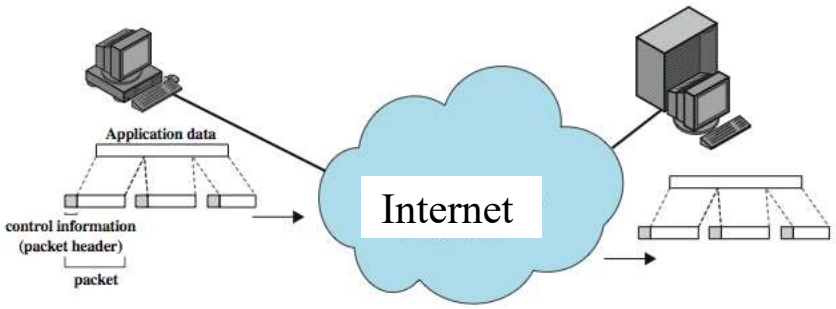
10

Review Codes and Demo

Slide #11

11

Packet Forwarder



12

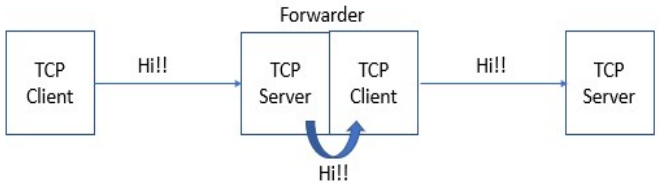
Question 2 (30 Points)

- A TCP server must start first and wait for a forwarder. Write a python code to establish a connection to the TCP server in the forwarder.
- Once your forwarder is connected to the TCP server, a server in the forwarder will start. And then, a TCP client must start.
- If you type a word on the TCP client, the TCP client will send this word to the forwarder. Such a forwarder can be added more. Write a python code for the forwarder to send this word to the TCP server.

Slide #13

13

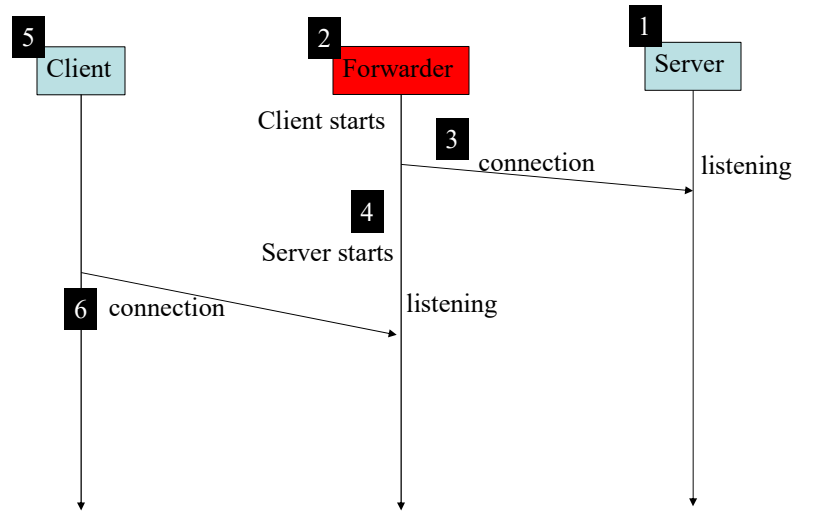
Architecture for Question 2



Slide #14

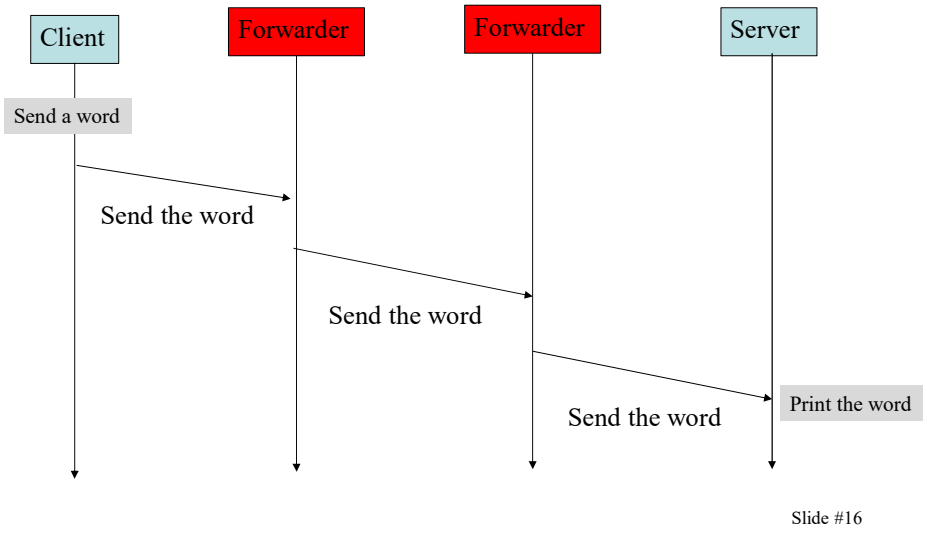
14

Execution Step of Question 2



15

Another Execution Steps



16

Note

- Do not modify the TCP client and server. Your forwarder.py must be working with the TCP client and server given in project 3.

Slide #17

17

Demo

Slide #18

18

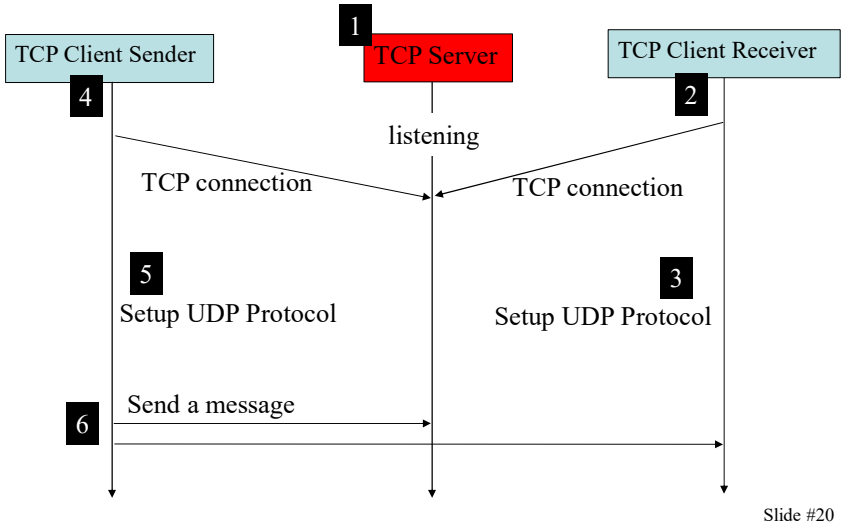
Question 3 (40 Points)

- Multiple TCP clients can connect to a TCP server. You can use a TCP client given in Project 3.
- Write two TCP client python codes to communicate with each other using UDP.
  - Two TCP clients must join the TCP server given in Project 3 and set up UDP communication.
  - A TCP client should send a message to another TCP client and the TCP server
  - If the message is "exit", two TCP clients must disconnect their TCP connection and terminate UDP-based communication.

Slide #19

19

Execution Step



Slide #20

20

Demo

Slide #21

21

Optional Question

Slide #22

22

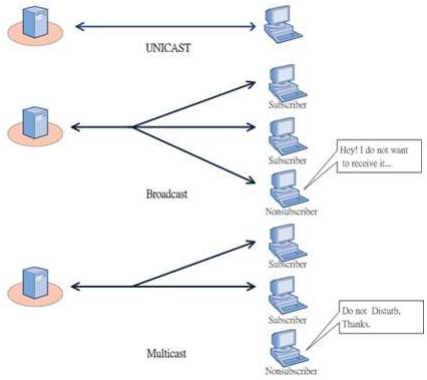
Design Issues

- Store-and-Forward Packet Switching
- Implementation of Connectionless Service
- Implementation of Connection-Oriented Service

23

Types of IP addresses

- Unicast
  - Identify one host
- Anycast
  - Identify one host in a set of hosts
- Broadcast
  - Identify all hosts
- Multicast
  - Identify a set of hosts



24

Broadcasting and Multicasting

– An area that is not well standardized – systems from different vendors may behave differently when dealing with broadcast and multicast packets.

- May not work on WAN (some equipment along the path may not support broadcasting/multicasting).
- Usually work on LAN (e.g. Ethernet) – no router in between.

25

Broadcasting

- IP broadcast address:
  - IP address can logically be viewed as three components: <netid, subnetid, hostid>, e.g.
  - When the hostid contains all one bits 111111111…111, it is a broadcast address, let us denote the all one bits as –1.
  - two kinds of broadcast addresses:
    - Subnet-directed broadcast address: <subnetid, -1>
    - Limited broadcast address < -1, -1> or 255.255.255.255, broadcast to all machines in the local network.
    - Router typically does not forward subnet-directed broadcast address
    - Router should not forward limited broadcast address

26

Broadcasting

- Ethernet broadcast address:
  - ff:ff:ff:ff:ff:ff
  - All Ethernet cards recognize this address
- What happens when a broadcast packet is sent in a LAN?
  - The packet will go up all the way to the IP layer on ALL machines!!
    - Implication?
  - Ethernet switches must support broadcast
    - Many applications are built on top of it. ARP, BOOTP

27

Broadcasting

- Sending a broadcast message:
  - Set the SO_BROADCAST option.
  - Set destination ip address to 255.255.255.255
  - Different system may behavior differently.
- Receiving a broadcast message:
  - Nothing extra

28

Multicasting

- In between unicast (send to one) and broadcast (send to all).
- IP Multicast address
  - 1110 xxxx.xxxxxxxx.xxxxxxxx.xxxxxxxx (224.0.0.0 to 239.255.255.255)
  - These addresses are associated with a group of interfaces.
    - A host must explicitly join and leave a group.
- Ethernet multicast address:

  1110 xxxx.x xxxxxxx.xxxxxxxx.xxxxxxxx
- ---------- 23bits

29

Multicasting

- Some special multicast addresses:
  - 224.0.0.1 -- all hosts group
  - 224.0.0.2 – all routers group
  - 224.0.0.0 – 224.0.0.255 are reserved.

30

Multicasting

- Sending multicast messages:
  - Use *sendto*, just treat a multicast address as a regular IP address.
  - Can control the number of hops for multicast packets by setting IP_MULTICAST_TTL
  - Can avoid loopback by turning off IP_MULTICAST_LOOP.
- Receiving multicast messages:
  - Use *recvfrom()*

31

Multicasting in Python

UDP Sender:

- # Create a UDP socket
- client_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
- multicast_group = '224.1.1.1'
- udp_server_address = (multicast_group, 11000)
- client_sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, 2)

Slide #32

32

Multicasting in Python

UDP Receiver:

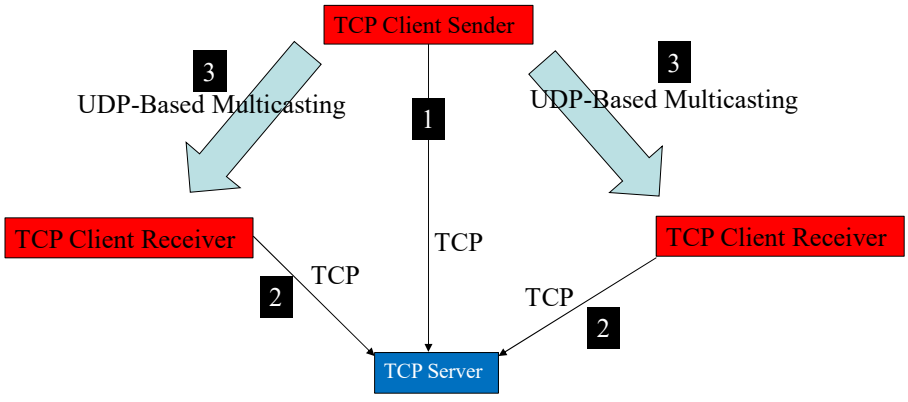
- # Create a UDP socket
- multicast_group = '224.1.1.1'
- client_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
- client_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
- udp_server_address = (multicast_group, 11000)
- client_sock.bind(udp_server_address)

- group = socket.inet_aton(multicast_group)
- mreq = struct.pack('4sL', group, socket.INADDR_ANY)
- client_sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)

Slide #33

33

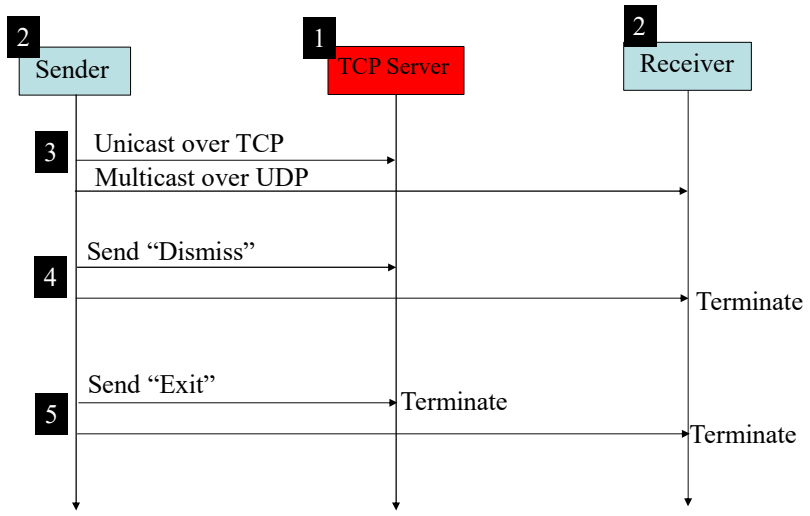
Architecture



You need at least 4 terminals to test this.

34

Execution Steps



Slide #35

35

Submission

- Submit your python codes and screenshots
- Deadline: 11:59 PM on April 26
- Late Submission: 10% penalty per day

Slide #36

36