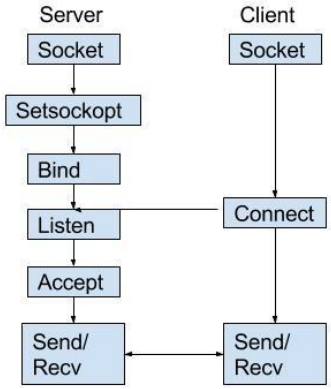
CS348: Computer Networks

Socket Programming



Dr. Manas Khatua

Assistant Professor

Dept. of CSE, IIT Guwahati

E-mail: manaskhatua@iitg.ac.in

Socket Programming

Goal: Learn how to build client-server application that communicate using sockets

- typical network application consists of
  - a client program and a server program
  - Those programs resides in two different end systems.

- There are two types of network applications
  - Open, i.e. operation rules are known to all and published as RFC
    - Two different organizations can develop two programs -- client and server
  - Proprietary, i.e. operation rules has not been published
    - One organization must develop both the programs -- client and server
    - Other independent developers will not be able to develop code that interoperates with this application

- Developer decides - whether the application is to run over TCP or UDP

- Proprietary should not use well known port for their applications

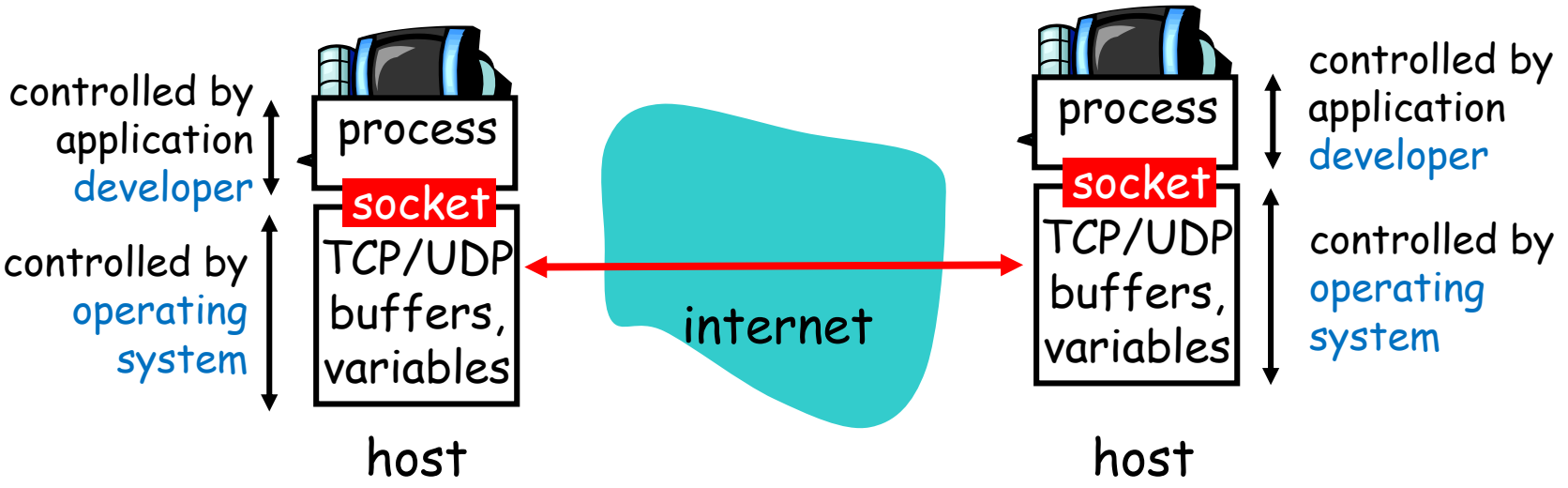
Socket API

Socket API

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram (use UDP)
  - reliable, byte stream-oriented (use TCP)

socket

a *host-local*, *application-created*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process



Types of Internet Sockets

- Stream Sockets (SOCK_STREAM)
  - Connection oriented
  - Rely on TCP to provide reliable two-way connected communication
- Datagram Sockets (SOCK_DGRAM)
  - Rely on UDP
  - Connection is unreliable

Socket Programming

- Application developer has
  - control of everything on the application-layer side of the socket;
  - But, it has little control of the transport-layer side.

- When a socket is created, an identifier, called a port number, is assigned to it.

- The sending process attaches to the packet
  - a destination address which consists of the destination host's IP address and
  - the destination socket's port number.

- These are also attached to the packet
  - The sender's address consisting of the IP address of the source host,
  - the port number of the source socket

Let a simple client-server application

1. The client reads a line of characters (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts the characters to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the line on its screen.

Socket programming with UDP

UDP: no "connection" between client and server

- no handshaking

- Sender (i.e., client) explicitly attaches IP address and port of destination to each packet

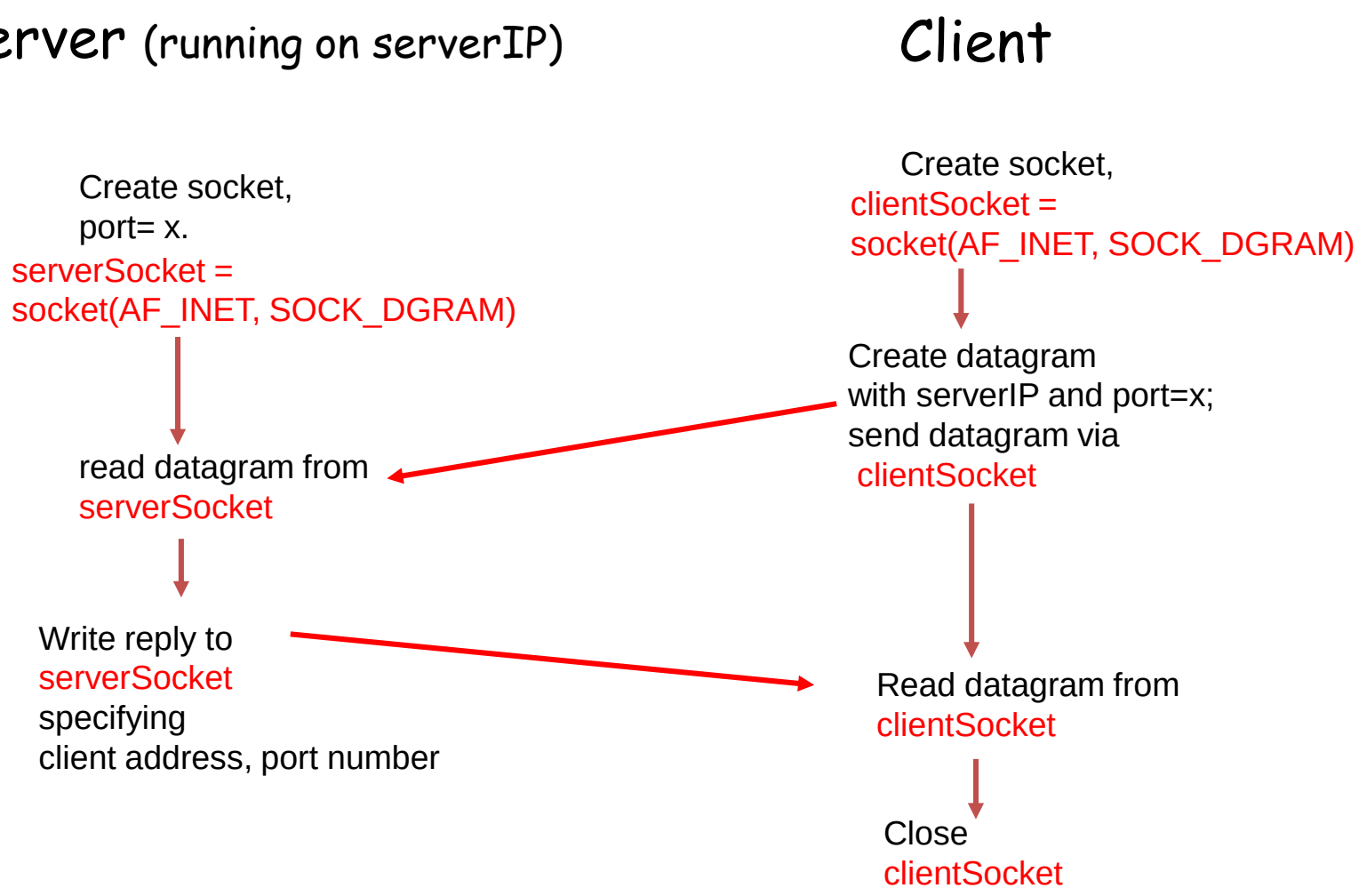
- server must extract IP address, port of sender from received packet

UDP: transmitted data may be received out of order, or lost

application viewpoint

UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

Client/Server socket interaction: UDP



Socket Programming (in Python)

- **UDPClient.py**

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

- **UDPServer.py**

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
        message, clientAddress = serverSocket.recvfrom(2048)
        modifiedMessage = message.upper()
        serverSocket.sendto(modifiedMessage, clientAddress)
```

UDPClient.py

from socket import *

- Invoke socket library; will be able to create sockets within our program

serverName = 'hostname' ; serverPort = 12000

- sets the IP address of the server (e.g., "128.138.32.126") OR
- sets the hostname of the server (e.g., "cis.poly.edu").
- sets the integer variable *serverPort* to 12000.

clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)

- creates the client's socket
- Family: defines the address family (AF). The common values are AF_INET (for IPv4),
- Type: defines four types of sockets
  - SOCK_STREAM (for TCP); SOCK_DGRAM (for UDP),
  - SOCK_SEQPACKET (for SCTP); SOCK_RAW (for directly use the IP)
- Note: we are not specifying the port number of the client socket when we create it; we are instead letting the operating system do this for us.

clientSocket.bind((", 19157))

- associate a port number (say, 19157) to this UDP client socket. *bind*() is implicitly called by *socket*()

message = raw_input('Input lowercase sentence:')

- It is a built-in function used to take inputs from the user using keyboard.

Cont…

clientSocket.sendto(message, (serverName, serverPort))

- attaches the destination address (*serverName*, *serverPort*) to the *message*, and
- sends the resulting packet into the process's socket, *clientSocket*.

- After sending the packet, the client waits to receive data from the server.

modifiedMessage, serverAddress = clientSocket.recvfrom(2048)

- when a packet arrives from the Internet at the client's socket :
- the packet's data is put into the variable *modifiedMessage,* and
- the packet's source address is put into the variable *serverAddress*.
- method *recvfrom* also takes the buffer size 2048 as input

print modifiedMessage

- prints out *modifiedMessage* on the user's display
- Note: It should be the original line that the user typed, but now capitalized by the server

clientSocket.close()

- This line closes the socket. The process then terminates.

UDPServer.py

from socket import *

- Invoke socket library; will be able to create sockets within our program

serverPort = 12000

- sets the integer variable *serverPort* to 12000.

serverSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)

- creates the server's socket

serverSocket.bind(('', serverPort))

- The above line binds (i.e., assigns) the port number 12000 to the server's socket.

print "The server is ready to receive"

while 1:

- UDPServer is ready and waits for a packet to arrive.

Cont…

message, clientAddress = serverSocket.recvfrom(2048)

- This line is similar to what we saw in UDPClient.
- *UDPServer* will make use of this address information (*clientAddress*)

modifiedMessage = message.upper()

- use the method *upper*() to capitalize it.

serverSocket.sendto(modifiedMessage, clientAddress)

- attaches the client's address (IP address and port number) to the capitalized message,
- sends the resulting packet into the server's socket (*serverSocket*)

- After the server sends the packet, it remains in the while loop, waiting for another UDP packet to arrive

Socket Programming with TCP

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts Server by:

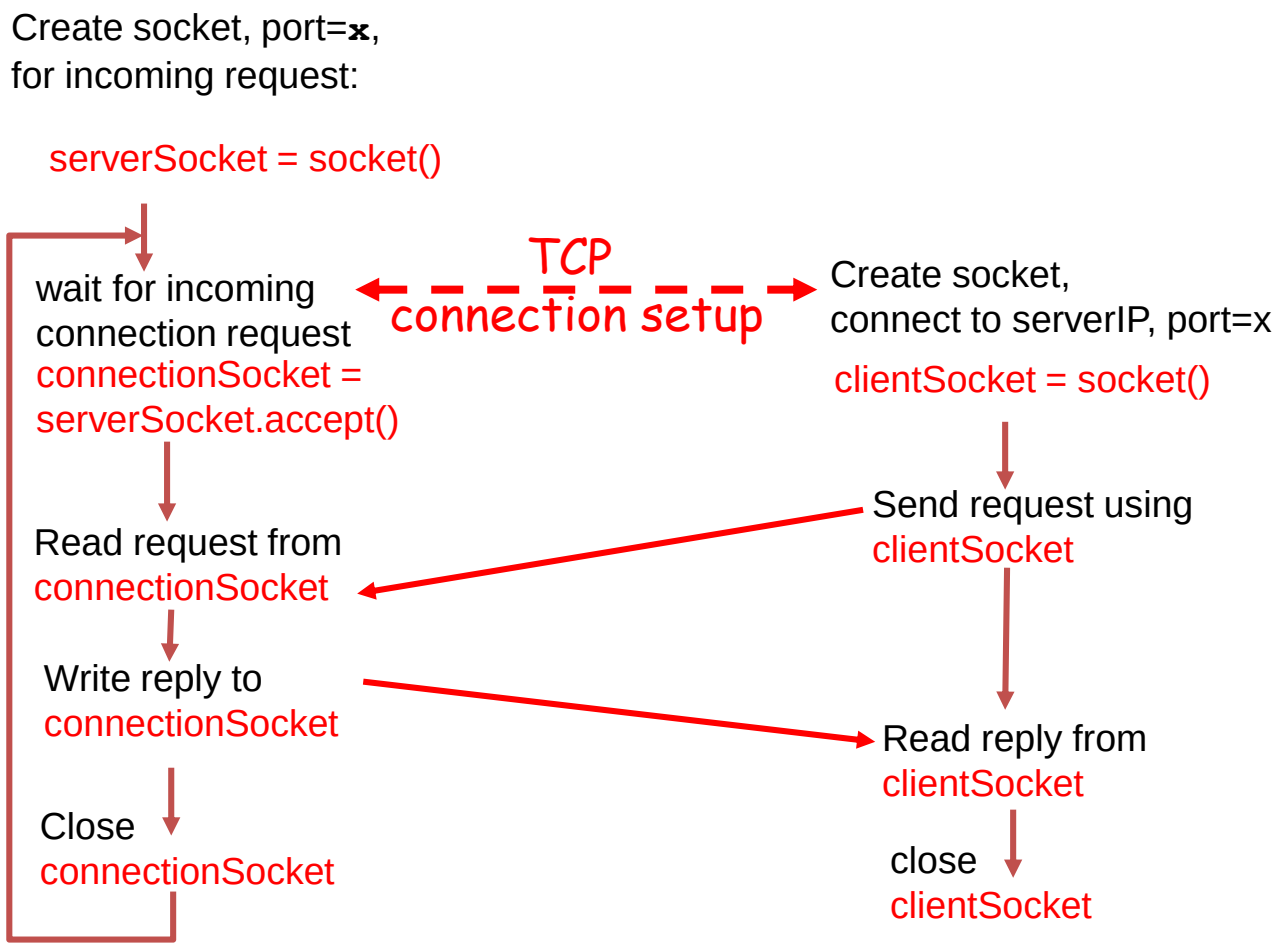
- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients

application viewpoint

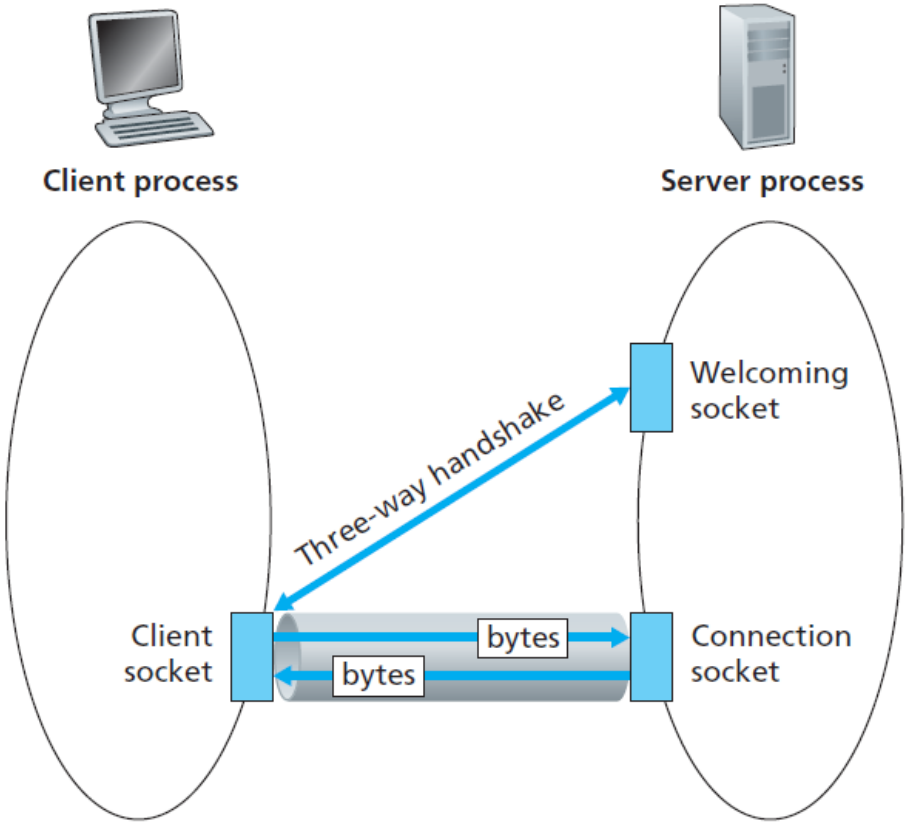
TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

Client/Server socket interaction: TCP



Cont…

- Unlike UDP, TCP is a connection-oriented protocol
  - before the client and server can start to send data to each other, they first need to handshake and establish a TCP connection.
  - When creating the TCP connection,
    - we associate with it the client socket address and server socket address
  - After TCP connection is established,
    - it just drops the data into the TCP connection via its socket.
    - This is different from UDP, for which the server must attach a destination address to the packet before dropping it into the socket.



- The client has the job of initiating contact with the server.

Socket Programming (in Python)

- **TCPClient.py**

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

- **TCPServer.py**

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

Cont…

- Lines of code that differ significantly from the UDP implementation

clientSocket = socket(AF_INET, SOCK_STREAM)

- The second parameter indicates that the socket is of type SOCK_STREAM, which means it is a TCP socket

clientSocket.connect((serverName,serverPort))

- a TCP connection must first be established between the client and server.

clientSocket.send(sentence)

- sends the string sentence through the client's socket and into the TCP connection.
- Note: this is not packet, and did not attach the destination address to the packet

clientSocket.close()

- closes the socket, and, hence, closes the TCP connection

Cont…

serverSocket.bind((‘’,serverPort))

- with TCP, *serverSocket* will be our welcoming socket.
- we will wait and listen for some client to knock on the door.

serverSocket.listen(1)

- server listen for TCP connection requests from the client.
- The parameter of *listen*() specifies the maximum number of queued connections (at least 1)

connectionSocket, addr = serverSocket.accept()

- When a client knocks on this door, the program invokes the *accept*() method for *serverSocket*, which creates a new socket in the server, called *connectionSocket*, dedicated to this particular client
- The client and server then complete the handshaking, creating a TCP connection between the client’s *clientSocket* and the server’s *connectionSocket*.

connectionSocket.close()

- after sending the modified sentence to the client, we close the connection socket.
- But *serverSocket* remains open, another client can now knock on the door

Thanks!