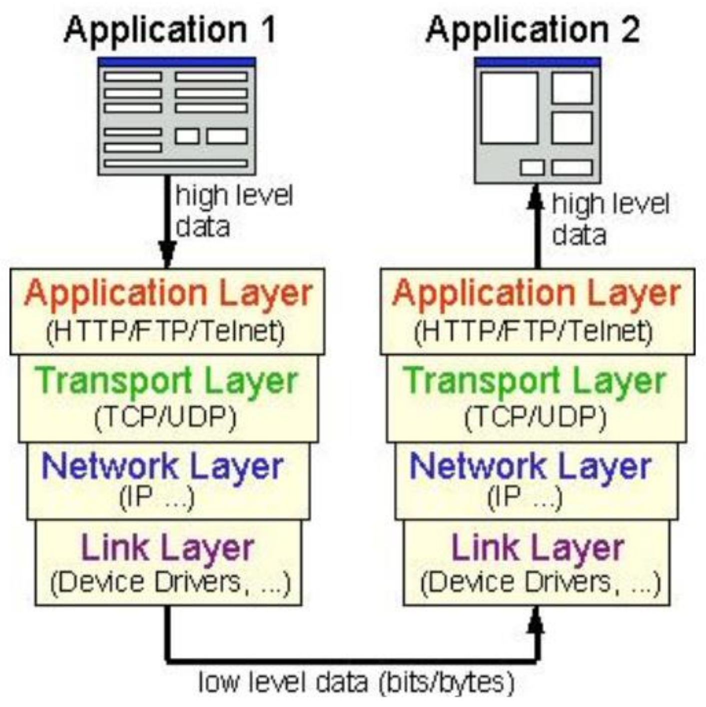
Introduction

What is Network Programming

- ***Network Programming*** *involves writing programs that communicate with other programs across a computer network*
- distributed applications or client server applications

Protocol

- *A* ***Protocol*** *is a standard pattern of exchanging information*


Application 1
Application 2
high level data
high level data
Application Layer (HTTP/FTP/Telnet)
Application Layer (HTTP/FTP/Telnet)
Transport Layer (TCP/UDP)
Transport Layer (TCP/UDP)
Network Layer (IP ...)
Network Layer (IP ...)
Link Layer (Device Drivers, ...)
Link Layer (Device Drivers, ...)
low level data (bits/bytes)

Protocol

- **Application Layer** protocols to allow applications to communicate:
  o **Hyper Text Transfer Protocol** (**HTTP**)
  o **File Transfer Protocol** (**FTP**)
  o **Telnet**

Protocol

- In a lower **Transport Layer** of communication, there is a separate protocol which is used to
determine how the data is to be transported from one machine to another:
o **Transport Control Protocol** (TCP)
o **User Datagram Protocol** (UDP)

UDP

- a protocol that sends independent packets of data, called ***datagrams***, from one computer to another.
- No guarantees about arrival. UDP is not connection based like TCP.
- Sending packets is like sending a letter through the postal service
- the order of delivery is not important and not guaranteed
- **faster** since no overhead of setting up end-to-end connection

TCP

- a ***connection-based*** protocol that provides a reliable flow of data between two computers.
- Guarantees that data sent from one end of the connection actually gets to the other end and in the same order
- similar to a phone call. Your words come out in the order that you say them.
- provides a point-to-point channel for applications that require **reliable communications**.
- **slow overhead time** of setting up an end-to-end connection.

Basics of Sockets

- **Sockets** are the endpoints of a bidirectional, point-to-point communication channel. Given an internet connection, say between client(a browser) and the server(say studytonight.com), we will have two sockets. A Client Socket and a Server Socket.
- **Socket acts on two parts**: IP Address + Port Number
- Server socket requires a standard or well defined port for connection like**: Port 80 for Normal HTTP Connection, Port 23 for Telnet** etc.

Socket Module in Python

- To create a socket, we must use **socket.socket** function available in the Python socket module, which has the general syntax as follows:
- **S = socket.socket(socket_family, socket_type, protocol=0)**
- **socket_family**: This is either AF_UNIX or AF_INET. We are only going to talk about INET sockets in this tutorial, as they account for at least 99% of the sockets in use.
- **socket_type:** This is either **SOCK_STREAM**(for TCP) or **SOCK_DGRAM**(for UDP).
- **Protocol:** This is usually left out, defaulting to 0.

Python Sockets

- There are two type of sockets: **SOCK_STREAM and SOCK_DGRAM.**

SOCK_STREAM	SOCK_DGRAM
For TCP protocols	For UDP protocols
Reliable delivery	Unrelible delivery
Guaranteed correct ordering of packets	No order guaranteed
Connection-oriented	No notion of connection(UDP)
Bidirectional	Not Bidirectional

Socket Functions

- **gethostname():**returns a string containing the **hostname** of the machine where the python interpreter is currently executing
- **Gethostbyname(hostname):**returns the IP address of the host.
- **getservbyname (name):** returns the port number on which the service is defined
- **Getservbypor(portno):** returns the name of the service for a given port number.

Get host name and IP for host name

- **gethostname() :**
- **Function Signature**: socket.gethostname()

```
import socket
hostname=socket.gethostname()
print(hostname)
```

Gethostbyname function

- **gethostbyname(): to get IP of host**
- **Function Signature:**
  - gethostbyname(hostname)
- **Return Value:**
  - The IPv4 address of the host name provided.
- **Example**

```
ip_add=socket.gethostbyname(hostname)
print(ip_add)
```

Turning a Hostname into an IP Address

```
import socket
hostname = 'www.python.org'

addr = socket.gethostbyname(hostname)

print('The IP address of {} is
{}'.format(hostname, addr))
```

Get port by service name and via vers

- Get port :**getservbyname** used to get port for specific service
- Get service by port : **getservbypor**

```
port=socket.getservbyname('domain')
print (port)

serv=socket.getservbyport(53)
print(serv)
```

Socket Object Methods

Server Socket Methods

Method	Description
s.bind()	This method binds address (hostname, port number pair) to socket.
s.listen()	This method sets up and start TCP listener.
s.accept()	This passively accept TCP client connection, waiting until connection arrives (blocking).

Client Socket Methods

- **s.connect()**
- This method actively initiates TCP server connection.

General Socket Methods

Method	Description
s.recv()	This method receives TCP message
s.send()	This method transmits TCP message
s.recvfrom()	This method receives UDP message, packet size as argument Return data, add
s.sendto()	This method transmits UDP message
s.close()	This method closes socket

bind function

- The **bind()** method of Python's socket class assigns an **IP address** and **a port number** to a socket instance.
- The **bind()** method is used when a socket needs to be made a server socket.
- As server programs listen on published ports, it is required that a **port** and the **IP address** to be assigned explicitly to a server socket.
- For client programs, it is not required to bind the socket explicitly to a port. The **kernel of the operating system** takes care of assigning the source IP and a **temporary port number**.
- Example
  - **s.bind(("127.0.0.1", 32007));**

recvfrom() function

- The **recvfrom**() method Python's socket class, reads a number of bytes sent from an **UDP socket.**
- The **recvfrom**() method can be used with an UDP server to receive data from a UDP client or it can be used with an UDP client to receive data from a UDP server.
- **Method Signature/Syntax:**
  - socket.recvfrom(bufsize[, flags])
- **Parameters:**
  - bufsize - The number of bytes to be read from the **UDP socket.**
  - flags - This is an optional parameter. As supported by the operating system. Multiple values combined together using bitwise OR. The default value is zero.
- **Return Value:**
  - Returns a **bytes** object read from an UDP socket and the address of the client socket as a tuple.

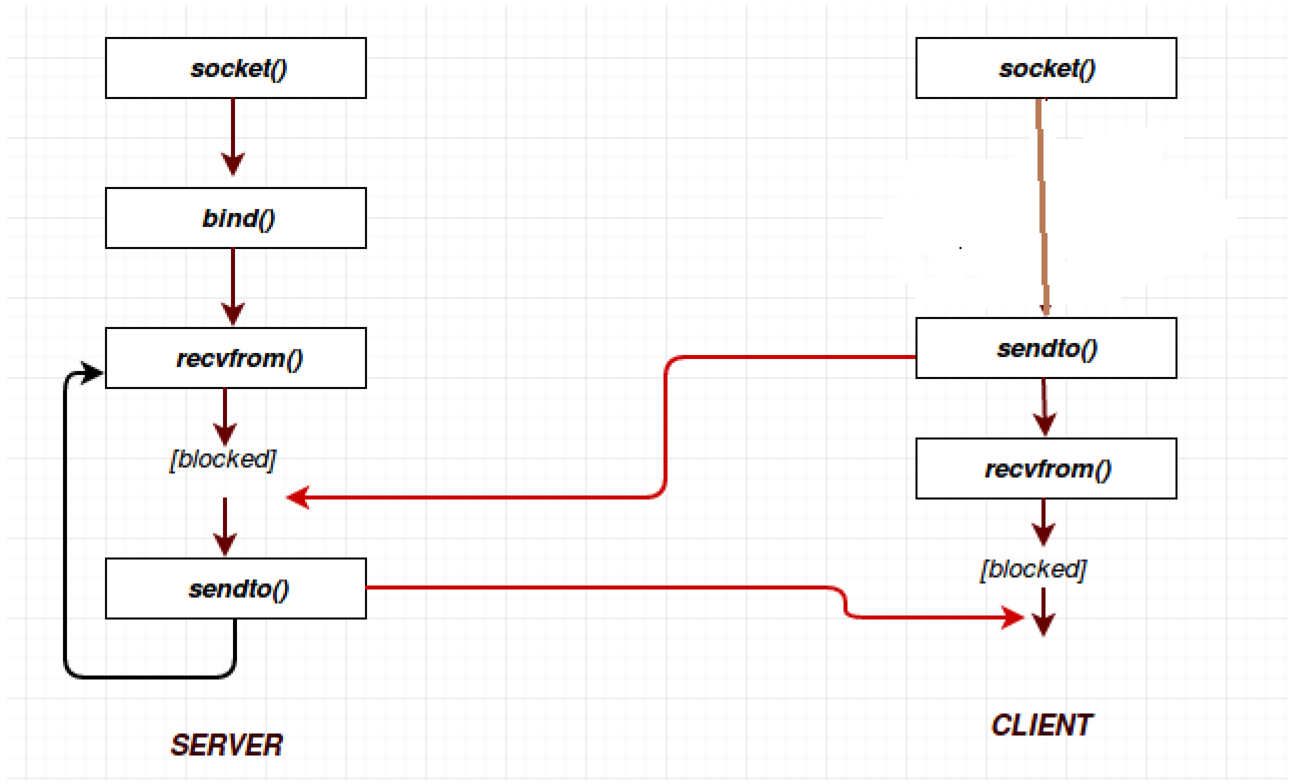
Sendto() function

- The method sendto() of the Python's socket class, is used to send datagrams to a UDP socket.
- The communication could be from either side. It could be from client to server or from the server to client.
- **Method Signature/Syntax:**
  - sendto(bytes, flags, address)
- **Parameters:**
  - bytes - The data to be sent in bytes format. If the data is in string format, str.encode() method can be used to convert the strings to bytes.
  - flags - As supported by the operating system, multiple values can be combined using bitwise OR. This optional parameter has a default value of 0.
  - **address** - A tuple consisting of IP address and port number.
- **Return Value:**
  - Returns the number of bytes sent.

User Datagram Protocol (UDP)

- **User Datagram Protocol (UDP)** is a Transport Layer protocol.
- Unlike TCP, it is an **unreliable and connectionless protocol.** So, there is no need to establish a connection prior to data transfer.
- Used for real-time services like computer gaming, voice or video communication, live conferences;

UDP client server application



UDP server

```
import socket
def Server():
    #Create Socket object, for UDP use  SOCK_DGRAM
    sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
    #get local machine name
    host=socket.gethostname()
    port=12345
    server_add=(host,port)
    #bind server add(ip,port) to socket
    sock.bind(server_add)
    while True:
        print('waiting for clients')
        #recive message from client recvfrom  method take max data size as argement
        #and return message and address of sender
        data,add= sock.recvfrom(1024)
        #converte the recieved bytes to string
        text = data.decode('ascii')
        print('recive data from',add,'data=',text)
        #send data to client sendto take the message in bytes and the address that i will send  the message to  it
        sock.sendto('replay from server'.encode('ascii'),add)
Server()
```

UDP Client

```
import socket
#Create Socket object , SOCK_DGRAM for UDP
sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
#get the local machine name
host=socket.gethostname()
port=12345
server_add=(host,port)
msg='hello my first udp program'
#convert string message to bytes
msg=msg.encode('ascii')
#send the message to server
sock.sendto(msg,server_add)
#recieve message from the server, recvfrom return message and sender address
data,add= sock.recvfrom(1024)
#convert the recived bytes message to string
text = data.decode('ascii')
print('recive data from',add,'data=',text)
```

Encoding and Decoding

- ***Decoding*** is what happens when bytes are on their way *into* your application and you need to figure out what they mean.
- ***Encoding*** is the process of taking character strings that you are ready to present to the outside world and turning
  them into bytes

Broadcast

- Use setsockopt() method to turn on broadcast.

```
sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
```

- Instead of using the ip for host we use "<broadcast>",
- The *level* argument specifies the protocol level at which the option resides. To set options at the socket level, specify the *level* argument as **SOL_SOCKET.** To set options at other levels, supply the appropriate *level* identifier for the protocol controlling the option. For example, to indicate that an option is interpreted by the TCP, set *level* to IPPROTO_TCP

UDP client broadcast

```python
import socket

def client(ip,port):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
    text = 'Broadcast datagram!'
    sock.sendto(text.encode('ascii'), (ip, port))
    msg, add = sock.recvfrom(1024)
    msg = msg.decode('ascii')
    print(add,msg)

client("<broadcast>",12345)
```

Task1

- Create UPD broad cast server to send message to any client

Thank You