**2/6/2018**

**Part 1: Introduction to Application Layer**

Network applications are programs that run on multiple end-points and communicate with each other

- IM, Email, Web, Search, VoIP, Gaming, Social Networking, AR/VR
- Run on end-points: Web client (Browser) Web server (Apache, IIS, Nginx)
- Can't write applications on routers: allows innovation and flexibility

**Network Application**

**- Protocols**

- Protocol defines the types of exchanges messages, their syntax
- Open protocols with RFC (HTTP RFC 2616)
- Proprietary protocols (Skype)

**- Architectures**

- Client-server (Netflix, YouTube)
  - Always-on server with fixed address that serves multiple clients
  - Clients don't communicate with each other
  - To serve millions of users, need thousands of servers (data center/cloud; own/rent)
- Peer-to-peer (P2P applications like BitTorrent, Skype, Bitcoin)
  - Self-scalable: no servers, clients talk to each other
  - Security, Reliability, Incentives

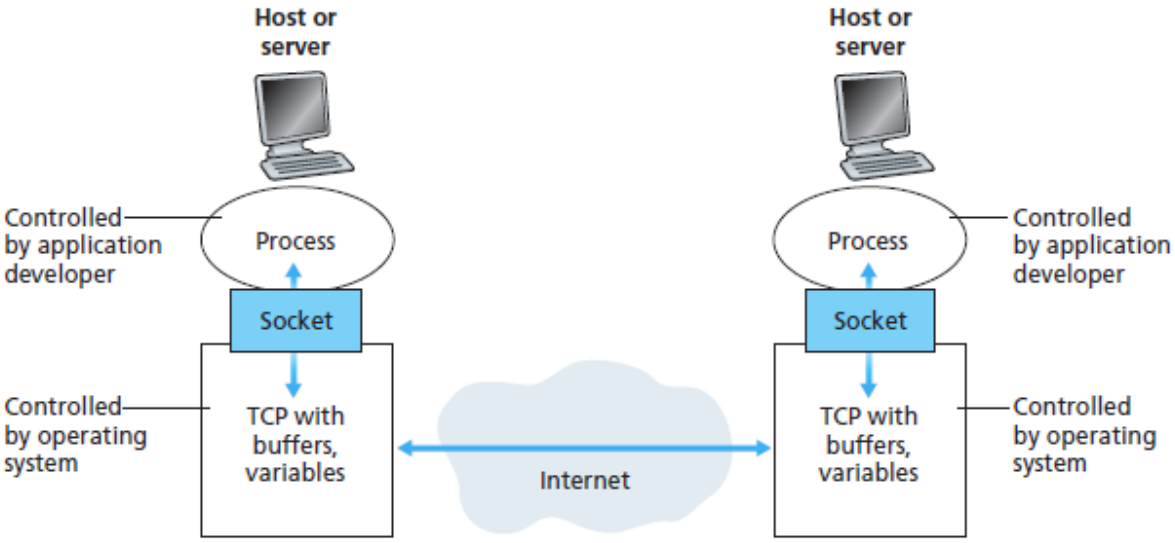
**Part 2: Network Applications**

Processor communication basics

- Programs -> processes; can on same end system
- Client process: downloading, initiates communication
- Server process: uploading, waits to be contacted
- In P2P: Process can be both client/server
- In Web: client process is web browser; server process is web server

Process communication interface

- Client and server processes need to exchange messages
- Interface between process and network: transport **sockets**
- Socket: Application Programming Interface (API) between application and network
- Application developer can only control: (1) choice of transport protocol used by socket (2) set a few parameters of transport layer (e.g., buffer size)
- Addressing in socket API:
  - Address of the host (IP address)
  - Identifier of the process/socket (port number)




**Part 3: Transport services available to applications**

**TCP**

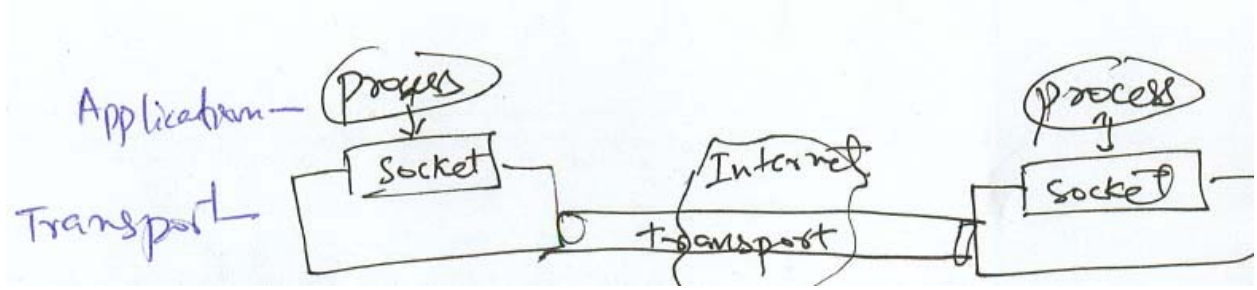
- Reliable data transfer (no lost/out-of-order/duplicate packets)
  - Packets can be lost in the network
  - Duplicate packets if you retransmit “lost packets”
  - Some applications may not want reliable data transfer
- Connection-oriented service
  - Exchange transport layer control information
  - Using handshake
- Congestion control
  - Adjust packet sending rate based on congestion

**UDP**

- No-frills, minimal
- Unreliable data transfer, no handshake, no congestion control

**2/8/2018**

**Part 1: Network Programming using socket API**




Processor communication

- What is a process?
- What is the difference between client and server?
- Client and server processes need to exchange messages
- Interface between process and network: transport **sockets**
- Socket: Application Programming Interface (API) between application and network

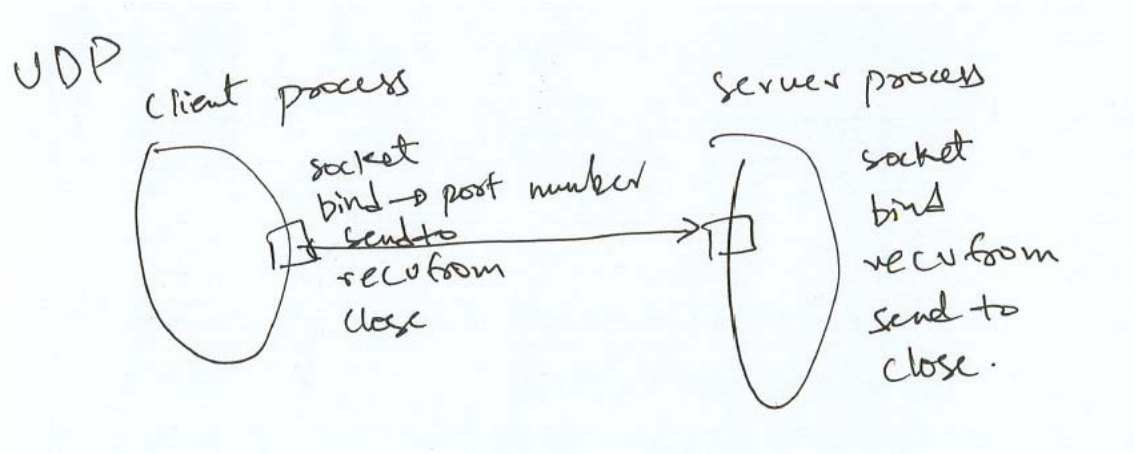
Addressing

- Hosts are addressed using IP address
- Multiple sockets on the same host are identified using **port numbers** (0-65535)
- Port numbers for popular applications: HTTP (80), HTTPS (443), SMTP (25)
- Manually assign port numbers or automatically assigned by the operating system

Types of sockets (choice of transport protocol)

UDP (User Datagram Protocol)
Connection-less
Unreliable
No overheads

TCP (Transmission Control Protocol)
Connection-oriented
Reliable + Flow control
Overheads



**Part 2: UDP Socket**

**UDP Client**

```
from socket import *
serverPort = 50008

clientSocket = socket(AF_INET, SOCK_DGRAM)

clientSocket.sendto('testing'.encode(),('', serverPort))

modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode())
clientSocket.close()
```

**UDP Server**

```
from socket import *
serverPort = 50008

serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))

print("The server is ready to receive")
message, clientAddress = serverSocket.recvfrom(2048)
print(message.decode())

modifiedMessage = message.decode().upper()
serverSocket.sendto(modifiedMessage.encode(), clientAddress)
serverSocket.close()
```

**Useful commands:**

ps (list of processes)
pkill Python (kill all processes named Python)
kill pid (kill the process with id of pid)
IP = gethostbyname('www.google.com')

- Run server in an infinite loop
- Wireshark DEMO

**Misc.**

- No need to bind port in client socket
- In UDP sockets, packets are assigned based on (serverIP, serverPort)
    - meaning two UDP sockets cannot bind to the same port