

Chapter 1 – Foundation

Problem: Building a Network

Outline

- Applications
- Requirements
- Network Architecture
- Implementation Network Software (Skipped to later)
- Performance
- Open Issue: Ubiquitous Networking

CS 158A

1

1.1 Applications

Figure 1.1



1.2 Requirements

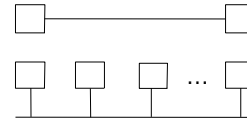
- Set of constraints and requirements influencing network design:
 - An application programmer
 - Specify services needed. Eg. error and delay requirements
 - Network designer
 - Properties of a cost-effective design. Eg. Efficient utilization and fair allocation of network resources
 - Network provider
 - System characteristics for the ease of administration and management

CS 158A

3

1.2.1 Connectivity: Building Blocks

- Nodes: PC, special-purpose hardware...
 - hosts
 - switches
- Links: coax cable, optical fiber...
 - point-to-point
 - multiple access

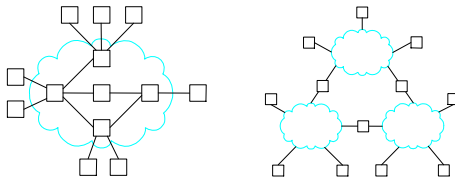


CS 158A

4

Switched Networks

- A network can be defined recursively as...
 - two or more nodes connected by a link, or
 - two or more networks connected by a node



CS 158A

5

Strategies

- Circuit switching: carry bit streams
 - original telephone network
- Packet switching: store-and-forward messages
 - Internet

CS 158A

6

Addressing and Routing

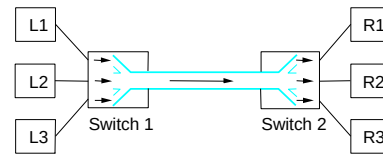
- Address: byte-string that identifies a node
 - usually unique
- Routing: process of forwarding messages to the destination node based on its address
 - broadcast: all nodes on the network
- Types of addresses
 - unicast: node-specific
 - broadcast: all nodes on the network
 - multicast: some subset of nodes on the network

CS 158A

7

Cost-Effective Resource Sharing Multiplexing

- Multiplexing and demultiplexing
- Synchronous Time-Division Multiplexing (STDM)
- Frequency-Division Multiplexing (FDM)

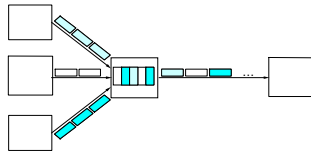


CS 158A

8

Statistical Multiplexing

- On-demand time-division
- Schedule link on a per-packet basis
- Packets from different sources interleaved on link
- Buffer packets that are *contending* for the link
- Buffer (queue) overflow is called *congestion*



CS 158A

9

1.2.3 Support for Common Services

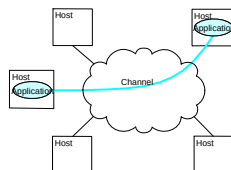
- Network as
 - Connecting application programs running on the hosts
 - Allow the programs to communicate meaningfully
- Applications may be complex, diversified
 - Build common services
 - View network as providing logical channels
 - Channels provide set of services
 - Application-level process may communicate over channels by using these services

CS 158A

10

Inter-Process Communication (IPC)

- Turn host-to-host connectivity into process-to-process communication.
- Fill gap between what applications expect and what the underlying technology provides.



CS 158A

11

Identifying Common Communication Patterns - IPC Abstractions

- Two types of channels
- 1) Request/Reply
 - distributed file systems
 - digital libraries (web)
- 2) Stream-Based
 - video applications
 - on-demand video
 - video conferencing
 - video: sequence of frames
 - Problem 1-28
 - 640 x 480 pixels
 - 3 bytes/pixel
 - 30 frames per second
 - Calculate bandwidth needed

- Functions of a channel:
 - What?
 - Where?
 - Switches/routers?
 - End-hosts?

CS 158A

12

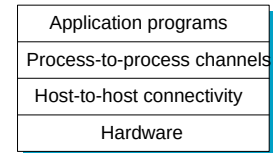
Reliability – What Goes Wrong in the Network?

- Bit-level errors (electrical interference)
- Packet-level errors (congestion)
- Link and node failures
- Packets are delayed
- Packets are deliver out-of-order
- Third parties eavesdrop

CS 158A

13

1.3 Network Architecture – Layering and Protocols (Fig. 1.8)

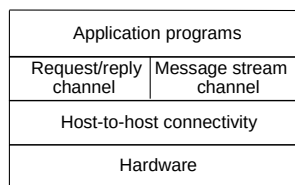


CS 158A

14

Network Architecture - Layering

- Use abstractions to hide complexity
- Abstraction naturally lead to layering
- Alternative abstractions at each layer



CS 158A

15

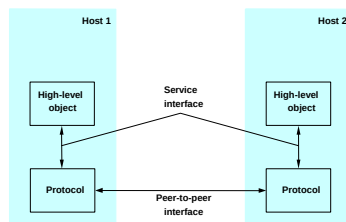
Protocols

- Building blocks of a network architecture
- Each protocol object has two different interfaces
 - *service interface*: operations on this protocol
 - *peer-to-peer interface*: messages exchanged with peer
- Term “protocol” is overloaded
 - specification of peer-to-peer interface
 - module that implements this interface

CS 158A

16

Interfaces

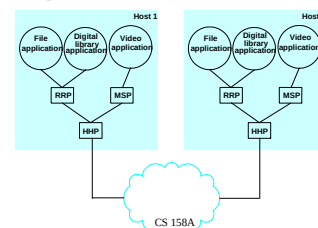


CS 158A

17

Protocol Machinery

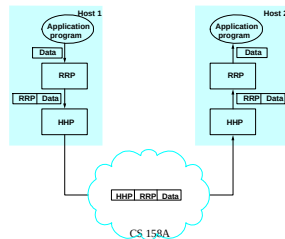
- Protocol Graph
 - most peer-to-peer communication is indirect
 - peer-to-peer is direct only at hardware level



18

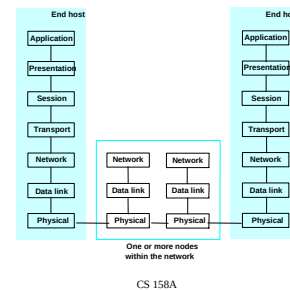
Encapsulation & Multiplexing/Dem

- Encapsulation (header/body)
- Multiplexing and Demultiplexing (demux key)



19

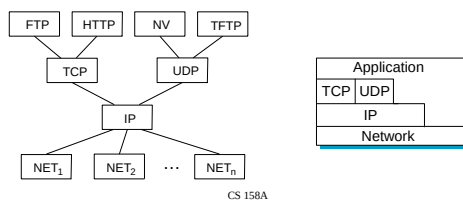
1.3.2 ISO/OSI Architecture



20

1.3.3 Internet Architecture

- Defined by Internet Engineering Task Force (IETF)
- Hourglass Design
- Application vs Application Protocol (FTP, HTTP)



CS 158A

21

1.5 Performance Performance Metrics

- Bandwidth (throughput)
 - data transmitted per time unit
 - link versus end-to-end
 - notation
 - KB = 2^{10} bytes
 - Mbps = 10^6 bits per second
- Latency (delay)
 - time to send message from point A to point B
 - one-way versus round-trip time (RTT)
 - components
 - Latency = Propagation + Transmit + Queue
 - Propagation = Distance / c
 - Transmit = Size / Bandwidth

CS 158A

22

Bandwidth versus Latency

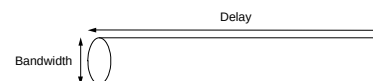
- Relative importance
 - 1-byte: 1ms vs 100ms dominates 1Mbps vs 100Mbps
 - 25MB: 1Mbps vs 100Mbps dominates 1ms vs 100ms
- Infinite bandwidth
 - RTT dominates
 - Throughput = TransferSize / TransferTime
 - TransferTime = RTT + 1/Bandwidth x TransferSize
 - 1-MB file to 1-Gbps link as 1-KB packet to 1-Mbps link

CS 158A

23

Delay x Bandwidth Product

- Amount of data “in flight” or “in the pipe”
- Usually relative to RTT
- Example: 100ms x 45Mbps = 560KB



CS 158A

24

Fig 1-19 - Bits transmitted at a particular bandwidth can be regarded as having some width (each bit 1 microsecond wide): (a) bits transmitted at 1 Mbps (b) at 2 Mbps

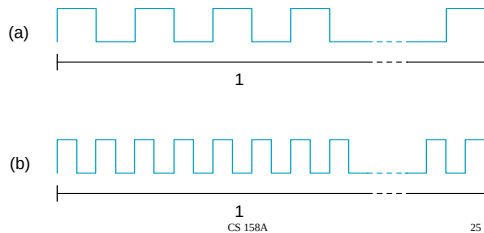


Fig 1-20
Perceived latency (response time) versus round-trip time for various object sizes and link speeds

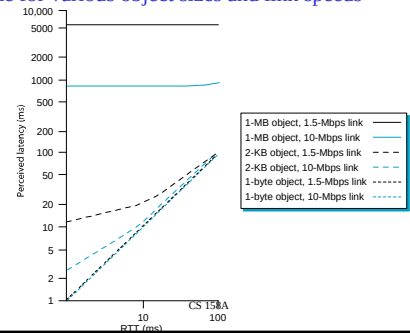


Fig 1-21
Network as a pipe

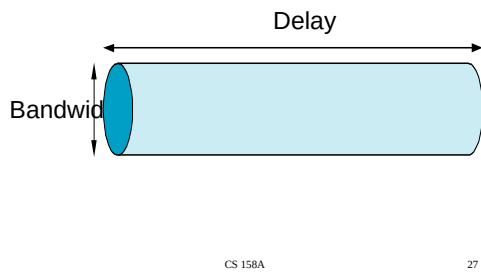


Fig 1-22
Relationship between bandwidth and latency. With a 10-MB file, (a) the 1-Mbps link has 80 pipes full of data; (b) the 1-Gbps link has 1/12 of one pipe full of data.

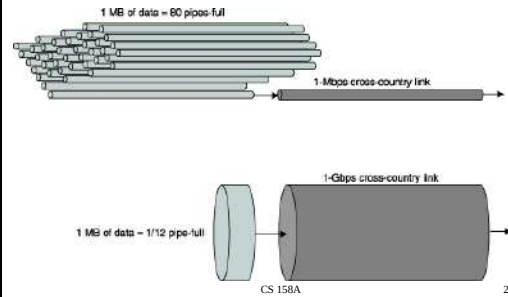
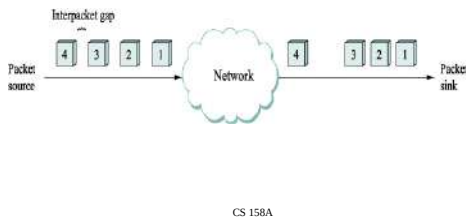


Fig 1-23
Network introduced Jitter.



1.4 Implementing Network Software

- Phenomenal success of the Internet:
 - Computer # connected doubled every year since 1981, now approaching 200 M
 - People # well over 600 M
 - Bits transmitted over Internet surpass voice phone system in 2001 (but I believe it was in 2nd half of 1998)
- Reasons:
 - Most network functionalities are provided by software running in general-purpose computers
 - Massive increase in computing power
 - Important to know how to implement network software!!

1.4.1 Socket API (Application Programming Interface)

- Most network protocols implemented in software
- Nearly all computer systems implement network protocols as part of the OS
- “Interface exported by the network” referred to the interface that the OS provided to its networking system.
- This interface is called the network API
- Socket interface have been supported in virtually all popular OS.

CS 158A

31

API

- Socket API originally provided by the Berkeley dist. of Unix.
- API typically interact with many parts of the OS than the network: read and write files, for concurrent processes, output to graphical display.
- Each protocol provides a certain set of services
- API provides a syntax by which those services can be invoked in the OS
- API implementation – mapping {operations and objects defined by API} to {services defined by the protocol}

CS 158A

32

Socket API

- The main abstraction of the socket interface is the socket.
 - A socket is the point where a local application process attaches to the network
- The socket interface defines operations for
 - Creating a socket
 - Attaching the socket to the network
 - Sending/receiving messages through the socket
 - Closing the socket

CS 158A

33

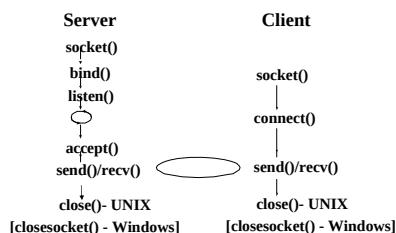
Socket API

- The book shows the implementation of a simple client/server program (in C/C++)
 - Uses socket interface to send messages over a TCP connection
 - The program also uses other Unix networking utilities.
 - Allows a user on one machine to type in and send text to a user on another machine
 - A simplified version of the Unix talk program (similar to chat room programs)

CS 158A

34

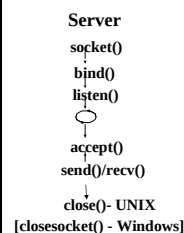
Flowchart of Stream Sockets Communication (Connection-oriented, TCP)



CS 158A

35

Flowchart of Stream Sockets Communication - Server side



- socket: the point where a local application process attaches to the network.
- bind: binds the newly created socket to the specified address (network address of the local participant – the server).
 - Address includes IP address and a TCP port number.
- Listen: defines how many connections can be pending on the specified socket
- Accept: carries out the passive open; does not return until a remote participant has established a connection. When completes, it returns a new socket corresponding to the new connection.

CS 158A

36

Socket API (Server)

- Creating a socket


```
int socket (int domain, int type, int protocol)
```

 - domain = PF_INET, PF_UNIX
 - type = SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
 - Returns a handle for the newly created socket; i.e., an identifier by which one can refer to the socket in the future.
- Passive Open (on server)

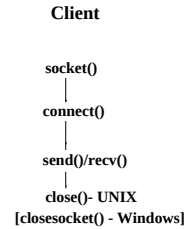

```
int bind (int socket, struct sockaddr *addr, int addr_len)
int listen (int socket, int backlog)
int accept (int socket, struct sockaddr *addr, int addr_len)
```

CS 158A

37

Flowchart of Stream Sockets Communication - Client side

- The application process performs an active open.
- **connect** – it does not return until TCP has successfully established a connection, at this point the application is free to begin sending data.



CS 158A

38

Sockets (Client)

- Active Open (on client)

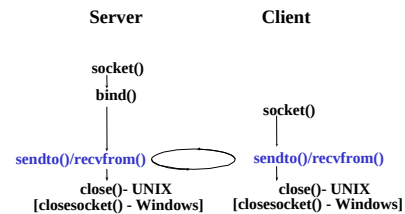

```
int connect (int socket, struct sockaddr *addr, int addr_len)
```
- Sending/Receiving Messages


```
int send(int socket, char *msg, int mlen, int flags)
int recv(int socket, char *buf, int blen, int flags)
```

CS 158A

39

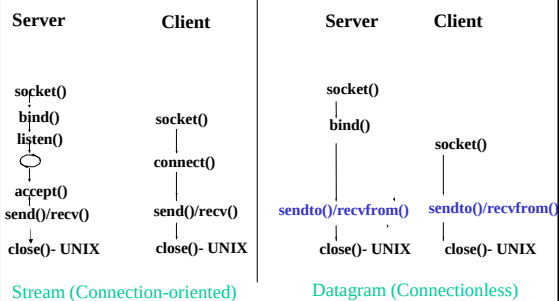
Flowchart of Datagram Sockets Communication (Connectionless, UDP)



CS 158A

40

Flowcharts of Stream and Datagram Sockets Communication



CS 158A

41