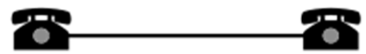
Lecture 2

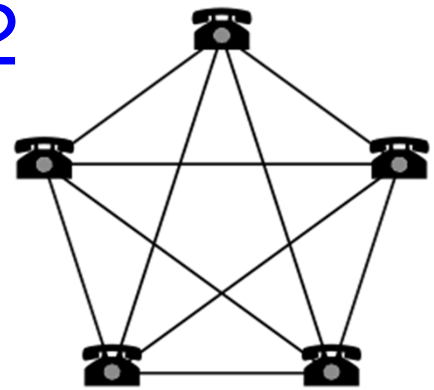
- Outline
  - Network value wrt. size
  - Multiplexing
  - Circuit vs. Packet Switching
  - Statistical Multiplexing
  - Network Software
  - the socket API
  - Protocols revisited

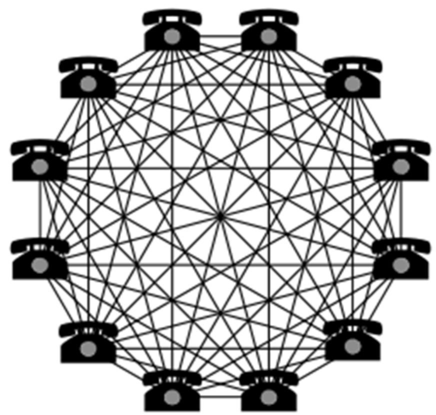
Metcalfe's law

- How many connections with ***n*** Telephones?
  - … or other directly connected end devices?
    - 2 can make only 1 connection
    - 5 can make 10 connections
    - 12 can make 66 connections

$$\# = n\,(n\text{-}1)\,/\,2$$



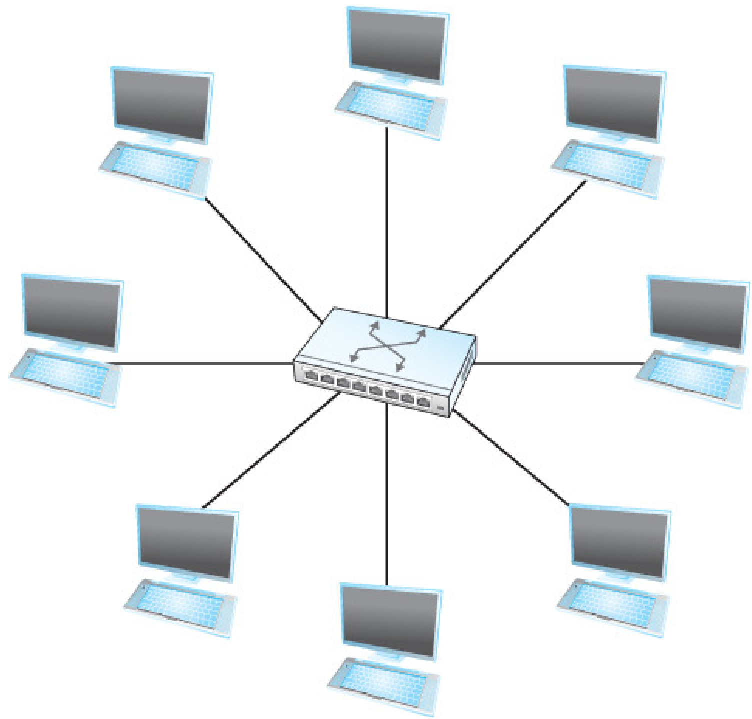




- **Metcalfe's law**
  - the **value** of a (telecommunications) **network** is proportional to the **square** of the number of connected users of the system (n^2)
    - First formulated in this form by George Gilder in 1993
    - attributed to **Robert Metcalfe**, **Ethernet designer**
  - Metcalfe's law was originally presented, circa 1980
    - not in terms of users, but rather
    - in terms of "compatible communicating devices“
      - for example, fax machines, telephones, etc.
- Related to economics
- exposition based on Wikipedia… (incl. graphic)

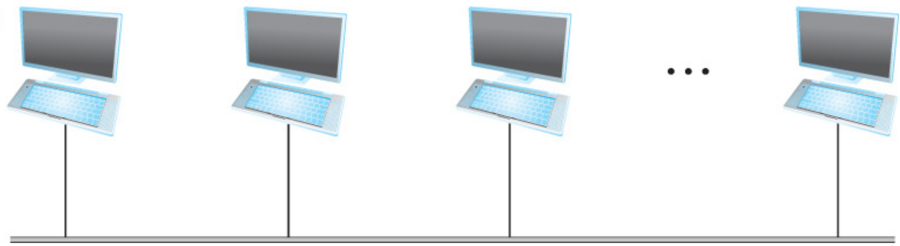
Full-connectivity Networks, Impractical

- n^2 physical connections
  - **Impractical!**



- Switch in the middle:
  - n^2 connections
  - + switch
    - Switch?
      - Non-blocking
      - Blocking
      - + resource allocation…
- or… take the switch away..

(b)

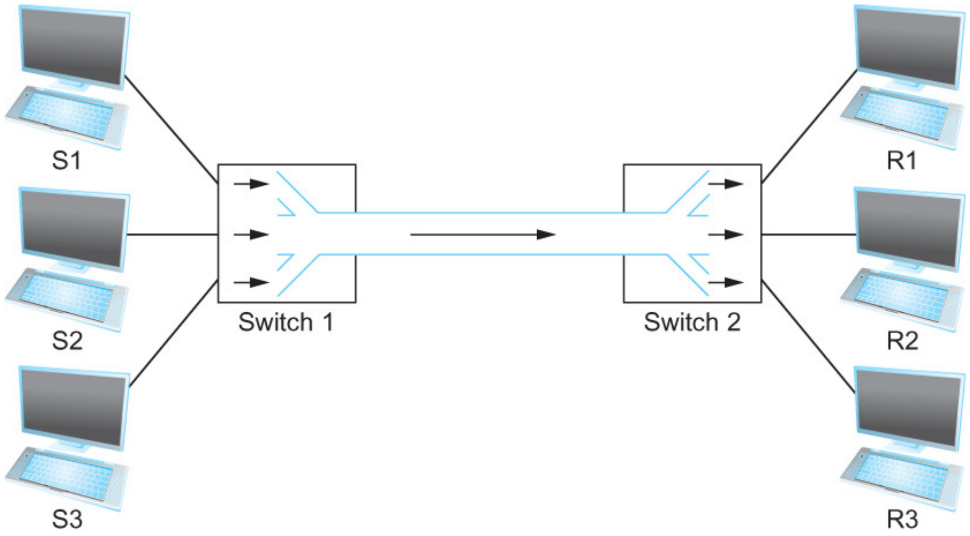


  - shared medium

    + resource allocation…

    = protocol…

Cost-Effective Resource Sharing

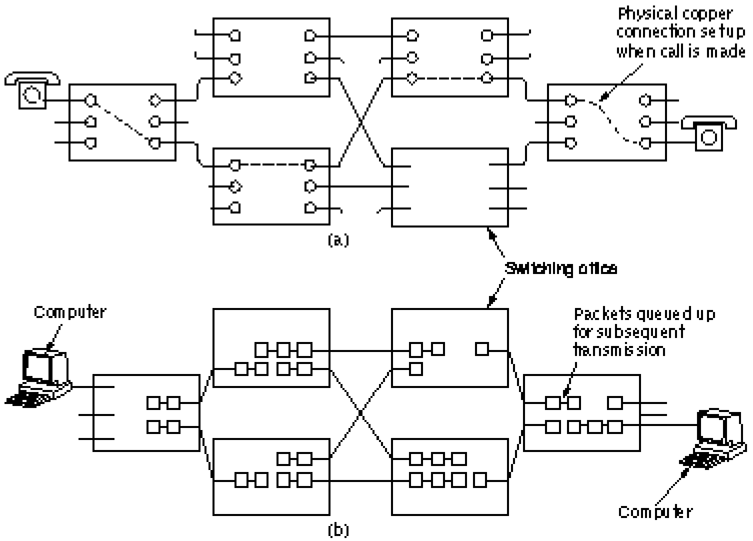


Multiplexing multiple logical flows over a single physical link

- Resource: links and nodes
- How to share a link?
  - Multiplexing
  - De-multiplexing
    - how?
  - *Synchronous* Time-division Multiplexing (sTDM)
    - Time slots
    - data transmitted in predetermined slots

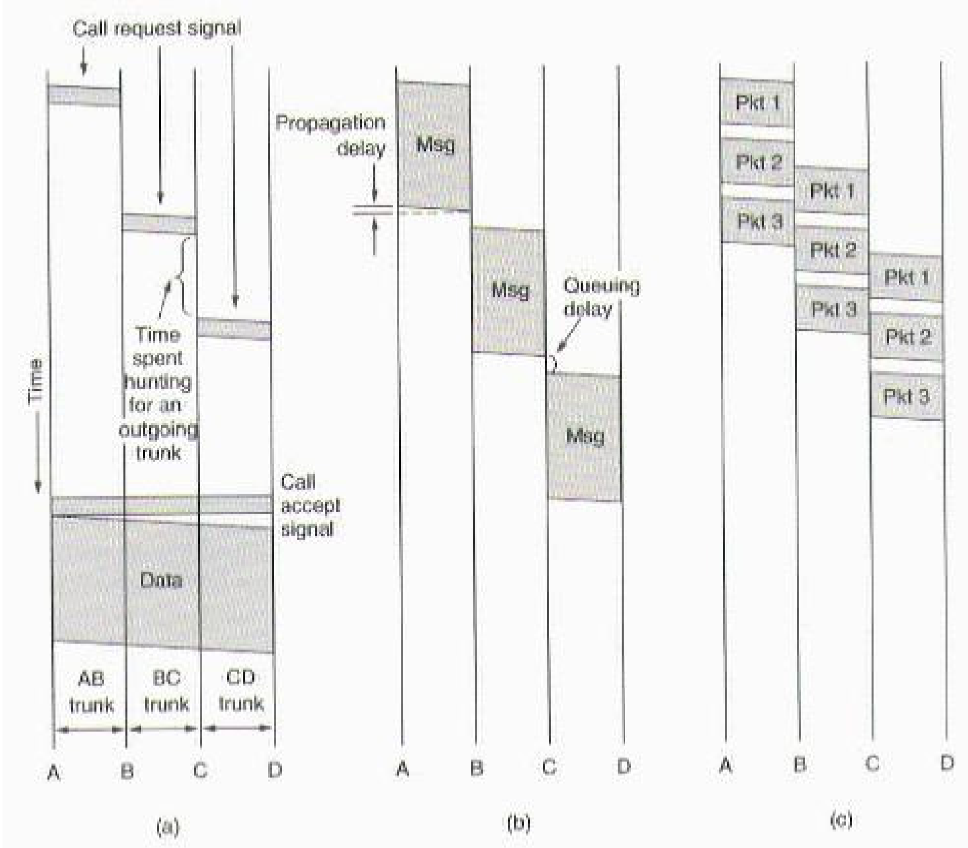
Circuit (vs. Message) vs. Packet Switching

- Circuit Switching
  - Resource allocated (statically)
  - Need set-up phase

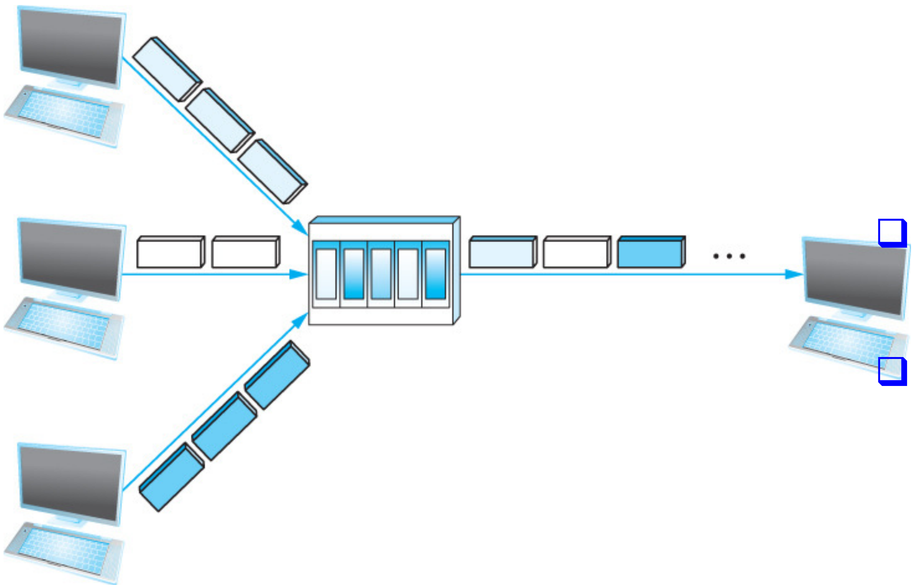


- Message/Packet Switching
  - Resource dynamically shared
  - Need address on every msg/pkt

- 4 nodes, 3 links example
  - A → B → C → D
  - circuit vs. msg vs. pkt switching



Multiplexing & Statistical Multiplexing



A switch multiplexing packets
from **multiple** sources
onto **one shared** link

- Mux = Static Resource Allocation
  - **TDM**: Time Division Multiplexing
  - **FDM**: Frequency Division Multiplexing
  - FDMA, TDMA, CDMA…
- Dynamic Resource Allocation
  - on demand…
- Statistical Multiplexing
  - Data transmitted based on the demand of each flow
  - What is a flow?
  - Packets vs. Messages
  - Service Disciplines
    - FIFO
    - Round-Robin
    - Priorities
  - Quality-of-Service (QoS)
  - Congested?
- LAN, MAN, WAN, PAN…

Application Programming Interface

- Interface exported by the network
  - most network protocols are implemented in software
  - *nearly all* computer systems implement their network protocols as part of the operating system
  - when we refer to the interface "*exported by the network*", we are generally referring to the interface that the OS provides to its applications (user level)
- This interface is called the…
  - ***network*** Application Programming Interface (API)

Application Programming Interface (Sockets)

- The **Socket** Interface
  - originally provided by the **Berkeley** distribution of **Unix**
  - now supported in virtually all operating systems
- each protocol provides a certain set of *services*
- the API provides a syntax by which those services can be invoked in this particular OS

Sockets

- What is a socket?
  - The point where a local application process attaches to the network
  - An interface between an application and the network
  - An application creates the socket

- The interface defines operations for
  - Creating a socket
  - Attaching a socket to the network
  - Sending and receiving messages through the socket
  - Closing the socket

Socket

- Socket Family
  - PF_INET denotes the Internet family
  - PF_UNIX denotes the Unix pipe facility
  - PF_PACKET denotes direct access to the network interface (i.e., it bypasses the TCP/IP protocol stack)

- Socket Type
  - SOCK_STREAM is used to denote a byte stream
  - SOCK_DGRAM is an alternative that denotes a message oriented service, such as that provided by UDP

Creating a Socket

```
int sockfd = socket(address_family, type, protocol);
```

- The socket number returned is the socket descriptor for the newly created socket

- `int sockfd = socket (PF_INET, SOCK_STREAM, 0);`
- `int sockfd = socket (PF_INET, SOCK_DGRAM, 0);`

The combination of PF_INET and SOCK_STREAM implies TCP

Client-Server Model with TCP

Server

- Passive open
- Prepares to accept connection, does not actually establish a connection

Server invokes

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

Client-Server Model with TCP

Bind

- Binds the newly created socket to the specified address
  - i.e. the network address of the local participant (the server)
- Address is a data structure which combines IP and port

Listen

- Defines how many connections can be pending on the specified socket

Accept

- Carries out the passive open
- Blocking operation
  - Does not return until a remote participant has established a connection
  - When it does, it returns a new socket that corresponds to the new established connection and the address argument contains the remote participant's address

Client-Server Model with TCP

Client

- Application performs active open
- It says who it wants to communicate with

Client invokes

```
int connect (int socket, struct sockaddr *address,
                                        int addr_len)
```

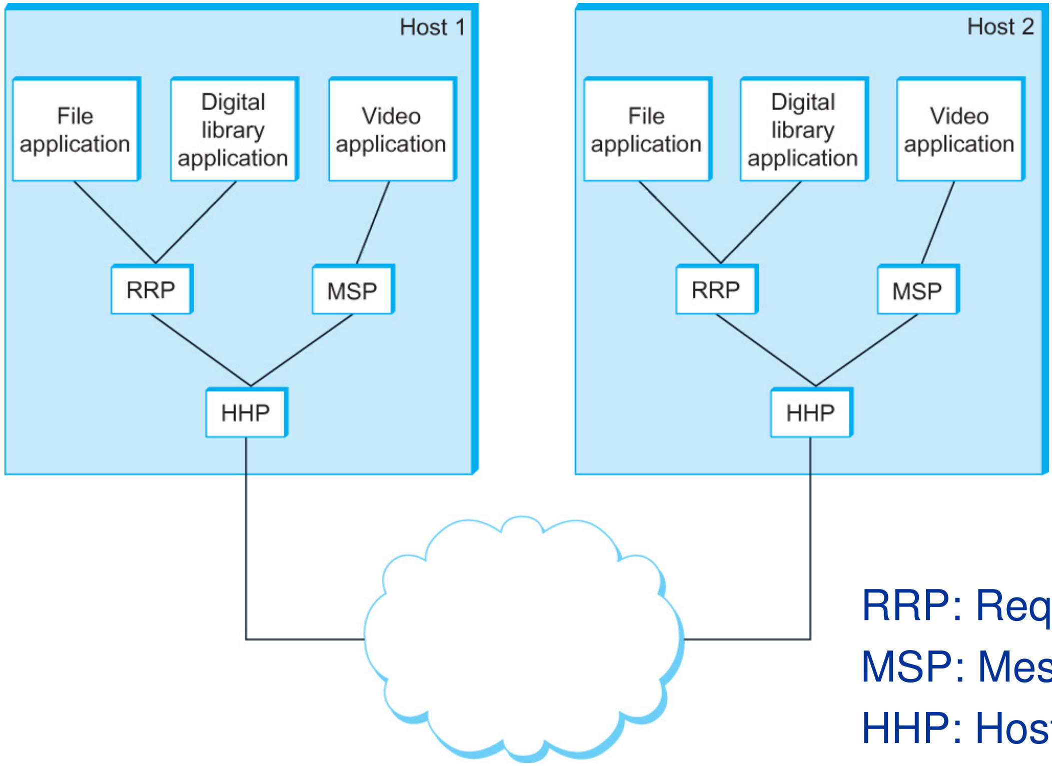
Connect

- Returns after TCP has successfully established a connection
  - at which point the application is free to begin sending data
- *address* contains remote machine's address

Protocols

- Protocol defines the interfaces
  - between the layers in the same system and
  - with the layers of the peer system
- 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

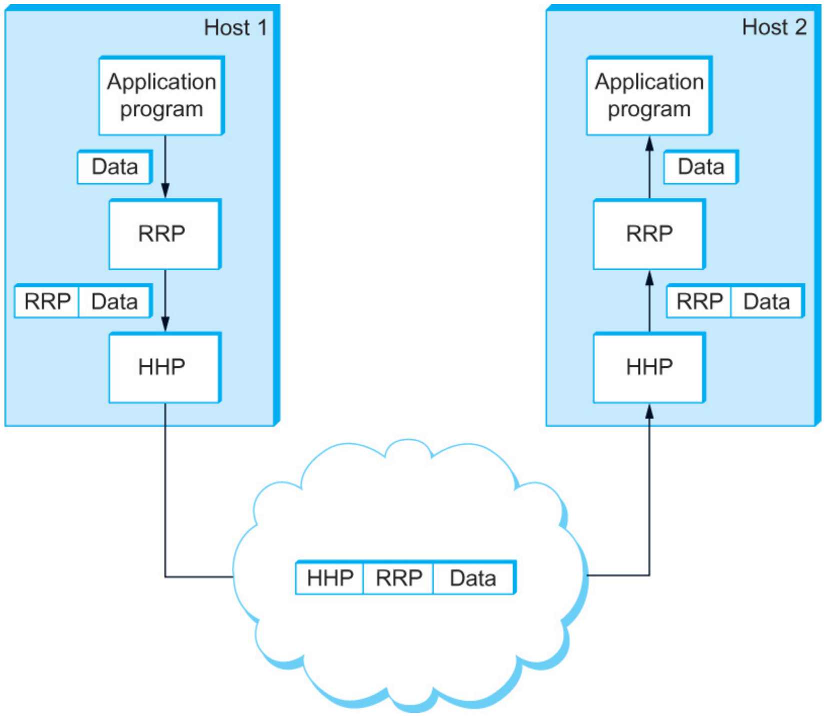
Protocol Graph



RRP: Request/Reply Protocol
MSP: MesagePassing Protocol
HHP: Host-Host Protocol

Example of a protocol graph:
nodes are the protocols and links are the "depends-on" relation

Encapsulation



High-level messages are **encapsulated** inside low-level messages

RRP: Request/Reply protocol

HHP: Host-Host Protocol

Application Protocol Example: HTTP

- HTTP: Hyper Text Transfer Protocol
- URL
  - Uniform resource locator
  - Ex.: http://cseweb.ucsd.edu/users/polyzos/
- uses TCP (Transmission Control Protocol)
- 17 messages for one URL request
  - 6 to find the IP (Internet Protocol) address (through the DNS)
  - 3 for connection establishment of TCP
  - 4 for HTTP request and acknowledgement
  - 4 messages for tearing down TCP connection

Summary

- The value of Networks and connectivity
- The need for Switching
- Efficient Resource Sharing
  - Packet vs. Circuit Switching
  - Multiplexing & Statistical Multiplexing
- We have defined a layered architecture for computer networks that will serve as a blueprint for the design
- We have discussed the socket interface which used by applications for invoking the services of the network