Computer Networks and Internets

By: Douglas E. Comer

http://www.eg.bucknell.edu/~cs363/lecture_notes/lecture_notes.html

Prof. M. Anvari, OS and Networking

Chapter 1 - Introduction

Section	Title
1	How do Computer Networks and Internets Operate?
2	Explosive growth
3	Internet
4	Economic impact
5	Complexity
6	Abstractions and concepts
7	On-line resources

How do Computer Networks and Internets Operate?

***Network*: system for connecting computer using a single transmission technology**
***Internet*: set of networks connected by routers that are configured to pass traffic among any computers attached to networks in the set**

- **Data transmission - media, data encoding**
- **Packet transmission - data exchange over a network**
- **Internetworking - universal service over a collection of networks**
- **Network applications - programs that use an internet**

Explosive growth

- **New phenomenon - now, networks are an important part of everyday activities**
  - **Business**
  - **Home**
  - **Government**
  - **Education**
- **Global Internet growing exponentially**
  - **Initially a research project with a few dozen sites**
  - **Today, millions of computers and thousands of networks world-wide**

Internet

- **Roots in military network called Arpanet**
  - **Fundamental changes from centralized to distributed computing**
  - **Incorporated features for reliability and robustness**
    - **Multiple links**
    - **Distributed routing**
- **Ethernet made local networking feasible**
- **TCP/IP protocol made *internetworking* possible**
  - **Developed *after* Arpanet**
  - **Switchover occurred in 1983**
- **Exponential growth - doubling every 18 months**

Economic impact

- **Large industry has grown around:**
  - **Networking hardware**
  - **Computers**
  - **Software**
- **Companies must integrate planning, implementation, management and upgrade**

Complexity

- **Computer networking is *complex***
  - **Many different hardware technologies**
  - **Many different software technologies**
  - **All can be interconnected in an internet**
- **No underlying theory**
- **Terminology can be confusing**
  - **TLAs**
  - **Industry redefines or changes terminology from academia**
  - **New terms invented all the time**

Abstractions and concepts

- **Will concentrate on abstractions and concepts to unravel complexity**
- **Examples:**
  - **Types of LAN wiring, rather than details of LAN data transmission**
  - **Definition and concept of congestion, rather than specific congestion control mechanisms**

Chapter 2 - Motivation and Tools

Introduction

- **Motivation**
- **Service**
- **Tools for exploration**

Historic motivation

- **Early computers were *expensive***
  - **Large footprint**
  - **Centralized**
- **Programs took a long time to run**
- **Couldn't afford to put computers everywhere**

ARPA

- ***Advanced Research Projects Agency* initiated project to connect researchers with computers**
- **Adopted new technology:**
  - **Packet switching**
  - **Internetworking**
- **Resulted in system for remote access to expensive resources**

Packet switching

- **Data transmitted in small, independent pieces**
  - **Source divides outgoing messages into *packets***
  - **Destination recovers original data**
- **Each packet travels independently**
  - **Includes enough information for delivery**
  - **May follow different paths**
  - **Can be retransmitted if lost**

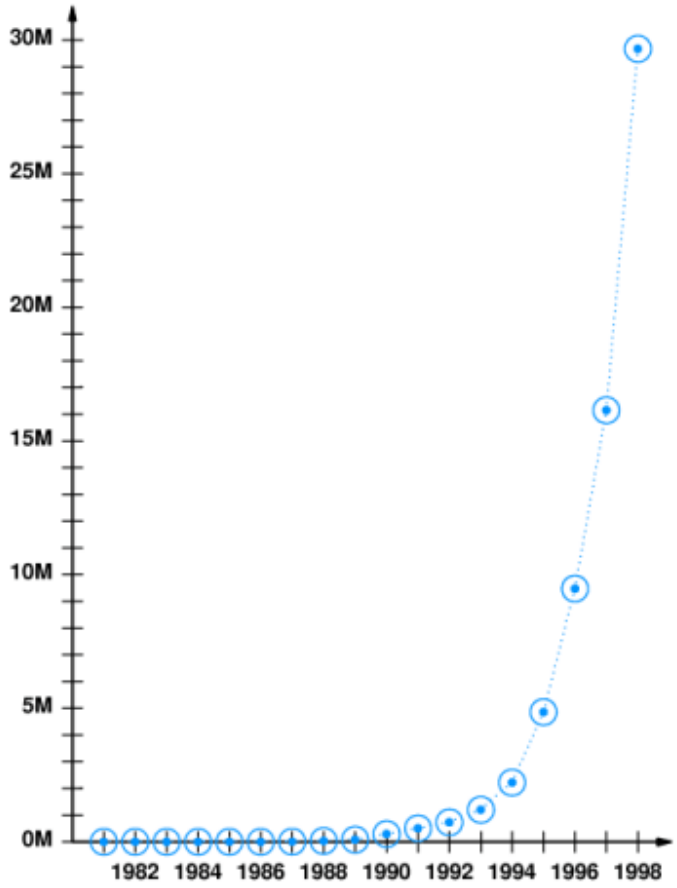
Internetworking

- ***Many* (mutually incompatible) network technologies**
- **No one technology appropriate for every situation**
- ***Internetworking* glues together networks of dissimilar technologies with *routers***
- **Result is *virtual network* whose details are invisible**

History and growth

- **ARPAnet began in late 1960s (not using TCP/IP)**
- **TCP/IP developed in late 1970s**
- **ARPAnet switched to TCP/IP in early 80s**
- **Start of Internet**
  - **Few hundred computers**
  - **Few tens of networks**

Growth since 1981



Probing the Internet

- **Two tools:**
  - **ping - sends message that is echoed by remote computer**
  - **traceroute - reports path to remote computer**

ping

- **Sends packet to remote computer**
- **Remote computer replies with echo packet**
- **Local computer reports receipt of reply**
- % ping merlin.cs.purdue.edu
- merlin.cs.purdue.edu is alive

ping (Example)

- **Can arrange to send multiple packets**
- **Reports *round trip time***
- % ping -s www.bucknell.edu 100 5
- PING web.bucknell.edu: 100 data bytes
- 108 bytes from web.bucknell.edu (134.82.6.6): icmp_seq=0. time=8. ms
- 108 bytes from web.bucknell.edu (134.82.6.6): icmp_seq=1. time=5. ms
- 108 bytes from web.bucknell.edu (134.82.6.6): icmp_seq=2. time=3. ms
- 108 bytes from web.bucknell.edu (134.82.6.6): icmp_seq=3. time=2. ms
- 108 bytes from web.bucknell.edu (134.82.6.6): icmp_seq=4. time=2. ms
-
- ----web.bucknell.edu PING Statistics----
- 5 packets transmitted, 5 packets received, 0% packet loss
- round-trip (ms) min/avg/max = 2/4/8

ping (Example)

% PING merlin.cs.purdue.edu: 100 data bytes
108 bytes from merlin.cs.purdue.edu (128.10.2.3): icmp_seq=0. time=64. ms
108 bytes from merlin.cs.purdue.edu (128.10.2.3): icmp_seq=1. time=58. ms
108 bytes from merlin.cs.purdue.edu (128.10.2.3): icmp_seq=2. time=55. ms
108 bytes from merlin.cs.purdue.edu (128.10.2.3): icmp_seq=3. time=63. ms
108 bytes from merlin.cs.purdue.edu (128.10.2.3): icmp_seq=4. time=54. ms

----merlin.cs.purdue.edu PING Statistics----
5 packets transmitted, 5 packets received, 0% packet loss
round-trip (ms) min/avg/max = 54/58/64

traceroute

- **Sends series of packets along path to destination**
  - **Each successive packet identifies next router along path**
  - **Uses *expanding ring* search**
- **Reports list of packets**

Traceroute (Example)

```
% traceroute www.bucknell.edu
traceroute to web.bucknell.edu (134.82.6.6), 30 hops max, 40 byte packets
 1  DanaRout-p13-s56.eg.bucknell.edu (134.82.56.254)  8 ms  5 ms  5 ms
 2  CCSServB-p1p17-s254.bucknell.edu (134.82.254.3)  4 ms  7 ms  4 ms
 3  web.bucknell.edu (134.82.6.6)  3 ms  3 ms  3 ms
```

traceroute (Example)

```
traceroute merlin.cs.purdue.edu
traceroute to merlin.cs.purdue.edu (128.10.2.3), 30 hops max, 40 byte packets
 1  CCSServC (134.82.7.254)  2 ms  1 ms  1 ms
 2  134.82.254.253 (134.82.254.253)  2 ms  2 ms  3 ms
 3  12.127.210.89 (12.127.210.89)  22 ms  20 ms  20 ms
 4  gr1-a3100s5.wswdc.ip.att.net (192.205.34.9)  20 ms  20 ms  20 ms
 5  Hssi2-1-0.GW1.DCA1.ALTER.NET (157.130.32.21)  20 ms  20 ms  20 ms
 6  104.ATM2-0.XR2.DCA1.ALTER.NET (146.188.161.30)  21 ms  39 ms  20 ms
 7  194.ATM2-0.TR2.DCA1.ALTER.NET (146.188.161.146)  20 ms  20 ms  20
ms
 8  101.ATM6-0.TR2.CHI4.ALTER.NET (146.188.136.109)  40 ms  41 ms  56 ms
 9  198.ATM7-0.XR2.CHI4.ALTER.NET (146.188.208.229)  41 ms  41 ms  41 ms
10  194.ATM8-0-0.GW1.IND1.ALTER.NET (146.188.208.165)  63 ms  66 ms  51
ms
11  purdue-gw.customer.alter.net (157.130.101.106)  56 ms  54 ms  54 ms
12  cisco-cs-atm.gw.purdue.edu (128.210.252.21)  66 ms  65 ms  63 ms
13  merlin.cs.purdue.edu (128.10.2.3)  68 ms  84 ms  63 ms
```

Web access to tools

ping/traceroute

http://www.net.cmu.edu/cgi-bin/netops.cgi

Ping/Traceroute Gateway
Ping/Traceroute Gateway
This application allows you to ping or traceroute to hosts on other networks. Carnegie Mellon has redundant internet connections described here. We also have a connection to the VBNS.

Host (IP or hostname):

Operation: **traceroute** **ping**

Submit Query

Chapter 3 - Transmission Media

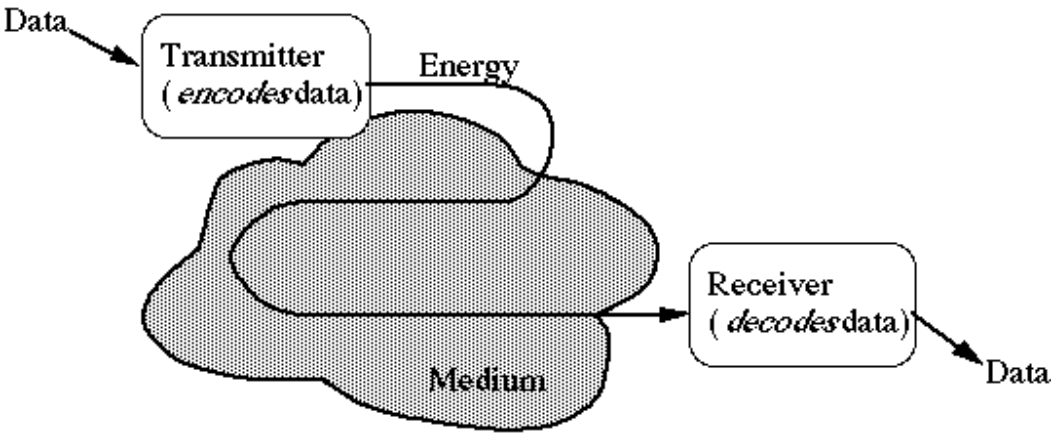
Section	Title
1	Basic Idea
2	Transmission media
3	Copper wires
4	Glass fibers
5	Radio
6	Wireless Example
7	Wireless Exmaple
8	Microwave
9	Infrared
10	Laser
11	Choosing a medium
12	Media in use at Bucknell

Basic Idea

- **Encode *data* as *energy* and transmit energy**
- **Decode energy at destination back into data**
- **Energy can be electrical, light, radio, sound, ...**
- **Each form of energy has different properties and requirements for transmission**

Transmission media

- **Transmitted energy is carried through some sort of *medium***
- **Transmitter encodes data as energy and transmits energy through medium**
  - **Requires special hardware for data encoding**
  - **Requires hardware connection to transmission medium**

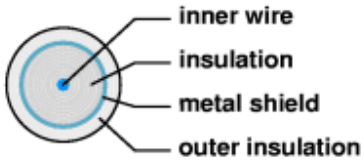


- **Media can be copper, glass, air, ...**

Copper wires

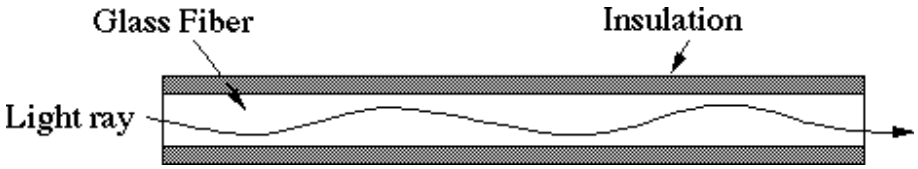
- **Twisted pair uses two wires**

- **Coaxial cable includes shield for improved performance**



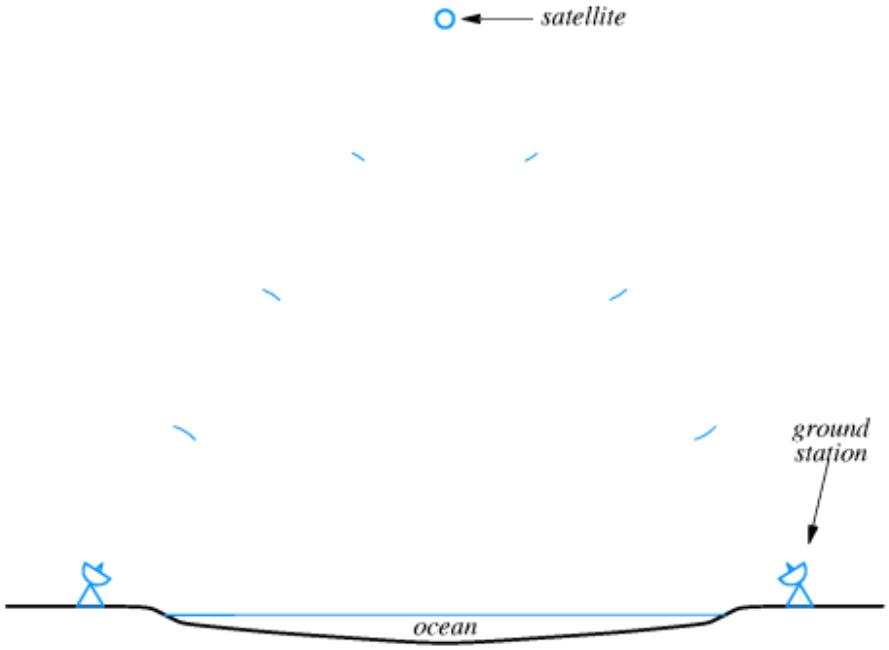
Glass fibers

- **Thin glass fiber carries light with encoded data**
- **Plastic jacket allows fiber to bend (some!) without breaking**
- **Fiber is very clear and designed to reflect light internally for efficient transmission**
- ***Light emitting diode* (LED) or *laser* injects light into fiber**
- **Light sensitive receiver at other end translates light back into data**



Radio

- **Data transmitted using radio waves**
- **Energy travels through the air rather than copper or glass**
- **Conceptually similar to radio, TV, cellular phones**
- **Can travel through walls and through an entire building**
- **Can be long distance or short distance**
    - **Long distance with satellite relay**



- **Short distance - wireless computer network**

Wireless Example

- **Wireless bridge and antenna**

Wireless Example

- **Remote station (laptop) interface**

Microwave

- **High frequency radio waves**
- **Unidirectional, for point-to-point communication**
- **Antennas mounted on towers relay transmitted data**

Infrared

- **Infrared light transmits data through the air**
- **Similar to technology used in TV remote control**
- **Can propagate throughout a room (bouncing off surfaces), but will not penetrate walls**
- **Becoming common in *personal digital assistants***

Laser

- **Unidirectional, like microwave**
- **Higher speed than microwave**
- **Uses laser transmitter and photo-sensitive receiver at each end**
- **Point-to-point, typically between buildings**
- **Can be adversely affected by weather**

Choosing a medium

- **Copper wire is mature technology, rugged and inexpensive; maximum transmission speed is limited**
- **Glass fiber:**
  - **Higher speed**
  - **More resistant to electro-magnetic interference**
  - **Spans longer distances**
  - **Requires only single fiber**
  - **More expensive; less rugged**
- **Radio and microwave don't require physical connection**
- **Radio and infrared can be used for mobile connections**
- **Laser also does not need physical connection and supports higher speeds**

Media in use at Bucknell

- **Copper/fiber for long-distance connection to Internet**
- **Fiber between buildings**

- **Copper within buildings**

Chapter 4 - Local Asynchronous Communication

Last modified: Mon Jan 18 08:51:15 2000

Bit-wise data transmission

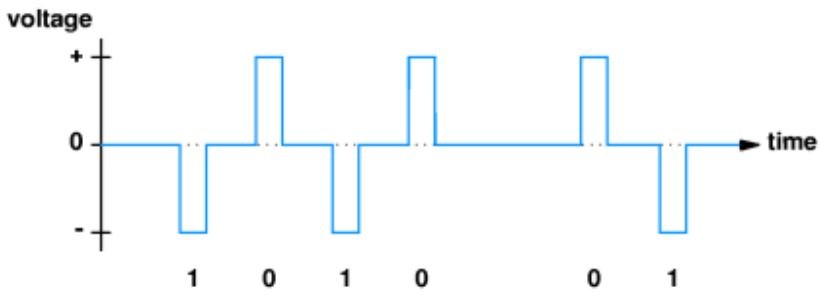
- **Data transmission requires:**
  - ***Encoding* bits as energy**
  - ***Transmitting* energy through medium**
  - ***Decoding* energy back into bits**
- **Energy can be electric current, radio, infrared, light**
- **Transmitter and receiver must agree on encoding scheme and transmission timing**

Asynchronous communication

- **One definition of *asynchronous*: transmitter and receiver do not explicitly coordinate each data transmission**
  - **Transmitter can wait arbitrarily long between transmissions**
  - **Used, for example, when transmitter such as a keyboard may not always have data ready to send**
- **Asynchronous may also mean no explicit information about where data bits begin and end**

Using electric current to send bits

- **Simple idea - use varying voltages to represent 1s and 0s**
- **One common encoding use negative voltage for 1 and positive voltage for 0**
- **In following figure, transmitter puts positive voltage on line for 0 and negative voltage on line for 1**



Transmission timing

- **Encoding scheme leaves several questions unanswered:**
  - **How long will voltage last for each bit?**
  - **How soon will next bit start?**
  - **How will the transmitter and receiver agree on timing?**
- ***Standards* specify operation of communication systems**
  - **Devices from different vendors that adhere to the standard can *interoperate***
  - **Example organizations:**
    - **International Telecommunications Union (ITU)**
    - **Electronic Industries Association (EIA)**
    - **Institute for Electrical and Electronics Engineers (IEEE)**

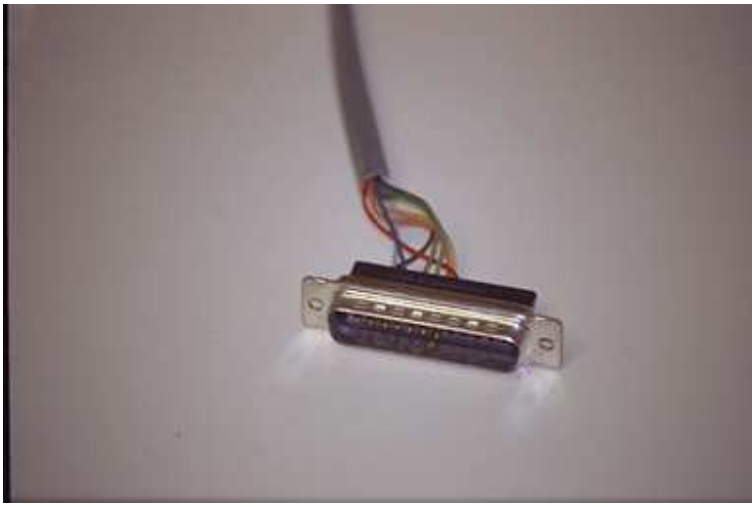
RS-232

- **Standard for transfer of characters across copper wire**
- **Produced by EIA**
- **Full name is *RS-232-C***
- **RS-232 defines *serial*, *asynchronous* communication**
  - **Serial - bits are encoded and transmitted one at a time (as opposed to *parallel* transmission)**
  - **Asynchronous - characters can be sent at any time and bits are not individually synchronized**

Details of RS-232

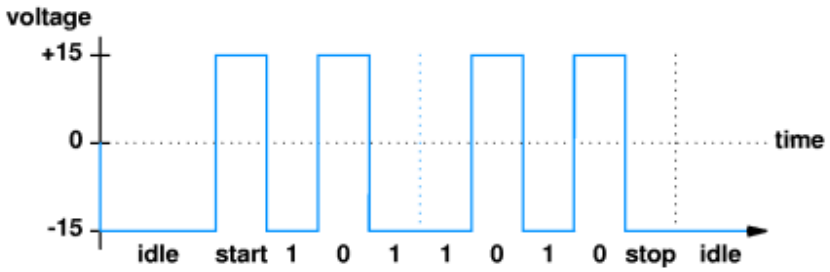
- **Components of standard:**
  - **Connection must be less than 50 feet**
  - **Data represented by voltages between +15v and -15v**
  - **25-pin connector, with specific signals such as data, ground and control assigned to designated pins**
  - **Specifies transmission of characters between, e.g., a terminal and a modem**
- **Transmitter never leaves wire at 0v; when idle, transmitter puts negative voltage (a 1) on the wire**

RS-232 wiring and connectors



Identifying asynchronous characters

- **Transmitter indiciates start of next character by transmitting a zero**
  - **Receiver can detect transition as start of character**
  - **Extra zero called the *start bit***
- **Transmitter must leave wire idle so receiver can detect transition marking beginning of next character**
  - **Transmitter sends a one after each character**
  - **Extra one call the *stop bit***
- **Thus, character represented by 7 data bits requires transmission of 9 bits across the wire**



- **RS-232 terminology:**
  - ***MARK* is a negative voltage (== 1)**

***SPACE* is a positive voltage (== 0)**

Timing

- **Transmitter and receiver must agree on timing of each bit**
- **Agreement accomplished by choosing *transmission rate***
  - **Measured in *bits per second***
  - **Detection of start bit indicates to receiver when subsequent bits will arrive**
- **Hardware can usually be *configured* to select matching bit rates**
  - **Switch settings**
  - **Software**
  - **Auto detection**

Measures of transmission rates

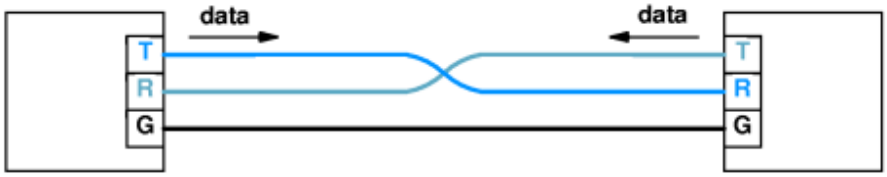
- ***Baud* rate measures number of signal changes per second**
- ***Bits per second* measures number of bits transmitted per second**
- **In RS-232, each signal change represents one bit, so baud rate and bits per second are equal**
- **If each signal change represents more than one bit, bits per second may be greater than baud rate**

Framing

- **Start and stop bits represent *framing* of each character**
- **If transmitter and reciver are using different speeds, stop bit will not be received at the expected time**
- **Problem is called a *framing error***
- **RS-232 devices may send an intentional framing error called a *BREAK***

Full-duplex communication

- **Two endpoints may send data simultaneously - *full-duplex* communication**
- **Requires an electrical path in each direction**

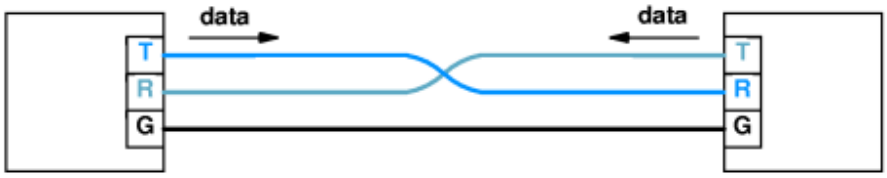


RS-232 connection standards

- **RS-232 specifies use of 25 pin connector (*DB-25*)**
- **Pins are assigned for use as data, ground and control:**
  - **Pin 2 - Receive (RxD)**
  - **Pin 3 - Transmit (TxD)**
  - **Pin 4 - Ready to send (RTS)**
  - **Pin 5 - Clear to send (CTS)**
  - **Pin 7 - Ground**

2-3 swap

- **Cable must cross-over wires to connect pins 2 and 3 on receiver and transmitter**



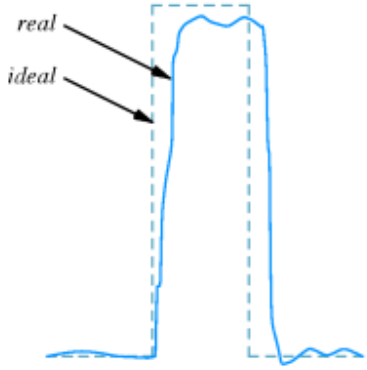
- **To avoid *2-3 swap*, RS-232 specifies that modems transmit on pin 2 and receive on pin 3, while computers transmit on pin 3 and receive on pin 2**
- **However, RS-232 cables between two computers must have 2-3 swap**

RS-232 cable breakout-box

- **May need to test RS-232 connections**
- ***Breakout-box* gives access to signals**

Limitations of real hardware

- **Effects of wire mean waveforms look like:**



- **Longer wire, external interference may make signal look even worse**
- **RS-232 standard specifies how precise a waveform the transmitter must generate, and how tolerant the receiver must be of imprecise waveform**

Hardware bandwidth

- **Limitations on time to change voltages imposes upper limit on number of changes per second**
- **Theoretical upper limit is called the *bandwidth***
- **Measured in *cycles per second* or *Hertz***

Bandwidth and data transmission

- ***Nyquist sampling theorem* expresses relationship between bandwidth and maximum data transmission speed**
- **For RS-232, using two voltages, maximum speed over medium with bandwidth *B* is *2B***
- **In general, for system using *K* different states, maximum is $2B\log_2 K$**
- **In practice, *noise* limits maximum data transmission rate to less than maximum allowed by Nyquist sampling theorem; Claude Shannon**

Summary

- **Asynchronous communication - data can start at any time; individual bits not delineated**
- **RS-232 - EIA standard for asynchronous character transmission**
- **Characters per second and baud rate**
- **Bandwidth limits maximum data transmission rate**

Chapter 5 - Long-Distance Communication

Last modified: Wed Jan 20 07:28:26 2000

Long-distance communication

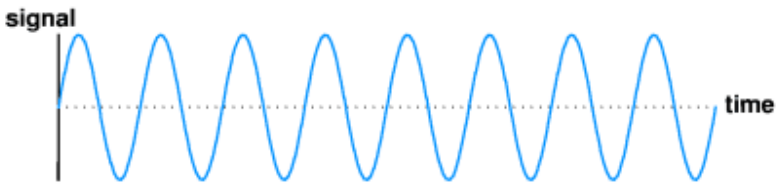
- **Encoding used by RS-232 cannot work in all situations**
  - **Over long distances**
  - **Using existing systems like telephone**
- **Different encoding strategies needed**

Sending signals long distances

- **Electric current becomes weaker as it travels on wire**
- **Resulting *signal loss* may prevent accurate decoding of data**
- **Signal loss prevents use of RS-232 over long distances**

Oscillating signals

- **Continuous, oscillating signal will propagate farther than electric current**
- **Long distance communication uses such a signal, called a *carrier***
- **Waveform for carrier looks like:**



- **Carrier can be detected over much longer distances than RS-232 signal**

Encoding data with a carrier

- **Modifications to basic carrier encode data for transmission**
- **Technique called *modulation***
- **Same idea as in radio, television transmission**

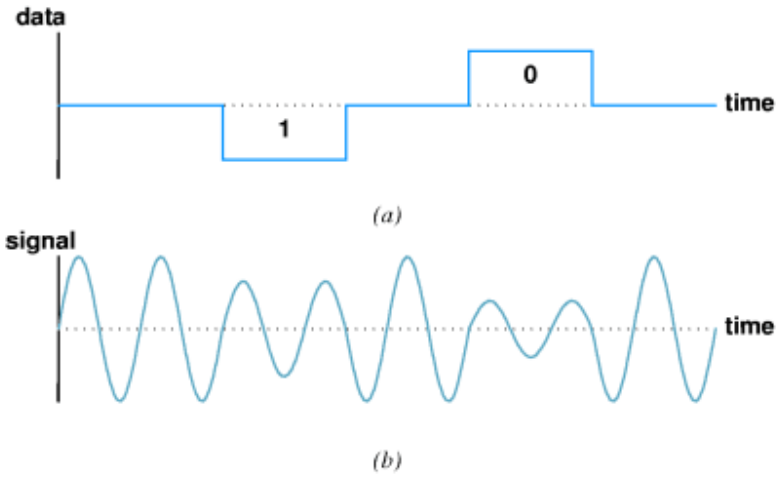
- **Carrier modulation used with all types of media - copper, fiber, radio, infrared, laser**

Types of modulation

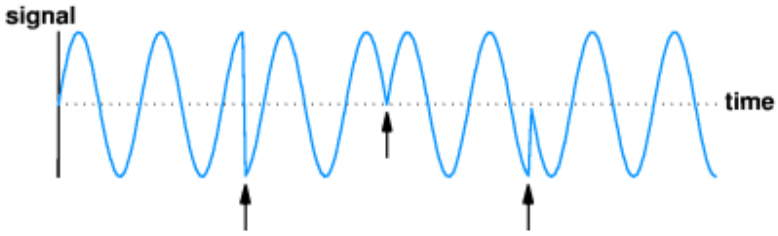
- ***Amplitude modulation*** **- strength, or *amplitude* of carrier is modulated to encode data**
- ***Frequency modulation*** **- frequency of carrier is modulated to encode data**
- ***Phase shift modulation*** **- changes in timing, or *phase shifts* encode data**

Examples of modulation techniques

- **Amplitude modulation:**

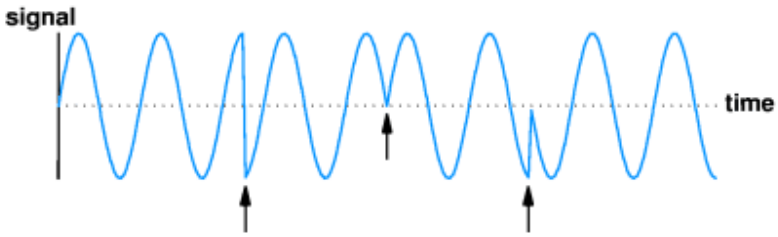


- **Phase shift modulation:**



Encoding data with phase shift modulation

- ***Amount* of phase shift can be precisely measured**
  - **Measures how much of sine wave is "skipped"**
  - **Example shows 1/2 and 3/4 cycle**



- **Each phase shift can be used to carry more than one bit; in example, four possible phase shifts encode 2 bits:**
  - **00 - no shift**
  - **01 - 1/4 phase**
  - **10 - 1/2 phase**
  - **11 - 3/4 phase**
- **Thus, each phase shift carries 2 bits**
- **Data rate is twice the baud rate**

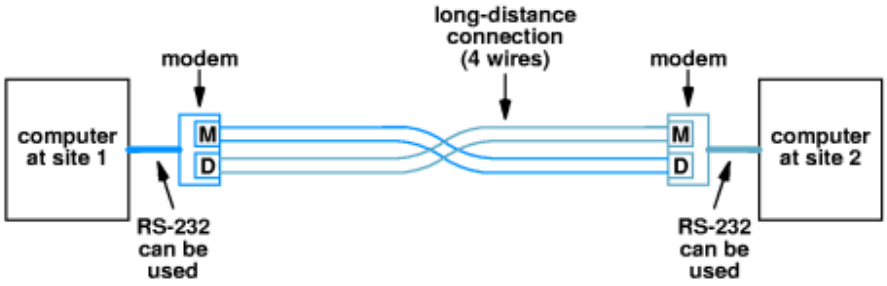
Hardware for data transmission

- ***Modulator* encodes data bits as modulated carrier**
- ***Demodulator* decodes bits from carrier**

- **Data transmission requires modulator at source and demodulator at destination**

Full duplex communication

- **Most systems provide for simultaneous bidirectional, or *full duplex*, transmission**
- **Requires modulator and demodulator at both endpoints:**



- **Long-distance connection is called *4-wire circuit***
- **Modulator and demodulator typically in single device called a *modem* (*mod*ulator/*dem*odulator)**

Modems

- **If external to computer, RS-232 can be used between modem and computer**

- **If internal, direct bus connection used**
- **Can also be rack-mounted**

Other types of modems

- **ISDN modem**

- **Cable modem (front)**

- **Cable modem (rear) with coax connector for cable and 10Base-T connector**

Leased serial data circuits

- **Organizations often include 4-wire circuits in network**
- **Within a site - on a campus - organization can install its own 4-wire circuits**
- **Telephone company supplies off-campus wires**
  - **Telephone cables have extra wires (*circuits*) for expansion**
  - **Telephone company lease right to use wires to organization**
  - **Organization uses modems for data transfer**
- **Called *serial data circuit* or *serial line***
- **Operates in parallel with (but not connected to) telephone circuits**

Optical, radio and dialup modems

- **Modems used with other media in addition to dedicated data circuits**
- **Special form of encoding/decoding transducers that use modulation for data encoding**
  - **Glass - data encoded as modulated light beam**
  - **Radio - data encoded as modulated radio signal**
  - **Dialup - data encoded as modulated sound**

- ***Dialup modem* connects to ordinary phone line**

Dialup modems

- **Circuitry for sending data**
- **Circuitry to mimic telephone operation**
  - **Lifting handset**
  - **Dialing**
  - **Replacing handset (hanging up)**
  - **Detect dial tone**
- **Full duplex on one voice channel**
  - **Different carrier frequencies for each direction**
  - **Filters eliminate interference**

Operation of dialup modems

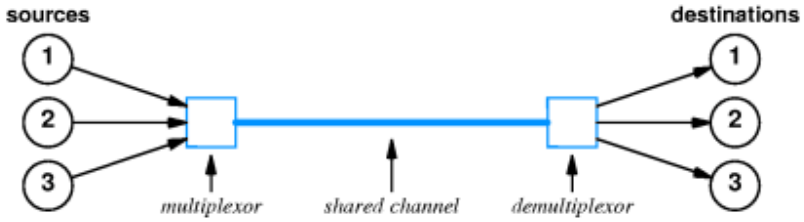
- **Receiving modem waits for call in *answer mode***
- **Other modem, in *call mode*:**
  - **Simulates lifting handset**
  - **Listens for dial tone**
  - **Sends tones (or pulses) to dial number**
- **Answering modem:**
  - **Detects ringing**
  - **Simulates lifting handset**
  - **Sends carrier**
- **Calling modem:**
  - **Sends carrier**
- **Data exchanged**

Carrier frequencies and multiplexing

- **Multiple signals with data can be carried on same medium without interference**
  - **Allows multiple simultaneous data streams**
  - **Dialup modems can carry full-duplex data on one voice channel**
- **Example - multiple TV stations in air medium**
- **Each separate signal is called a *channel***

Multiplexing

- **Carrying multiple signals on one medium is called *multiplexing***



- ***Frequency division multiplexing* (FDM) achieves multiplexing by using different carrier frequencies**
- **Receiver can "tune" to specific frequency and extract modulation for that one channel**
  - **Frequencies must be separated to avoid interference**
  - **Only useful in media that can carry multiple signals with different frequencies - high-bandwidth required**

Spread spectrum multiplexing

- ***Spread spectrum* uses multiple carriers**
- **Single data stream divided up and sent across different carriers**
- **Can be used to bypass interference or avoid wiretapping**

Time division multiplexing

- ***Time division multiplexing* uses a single carrier and sends data streams sequentially**
- **Transmitter/receiver pairs share single channel**
- **Basis for most computer networks used shared media - will give details in later chapters**

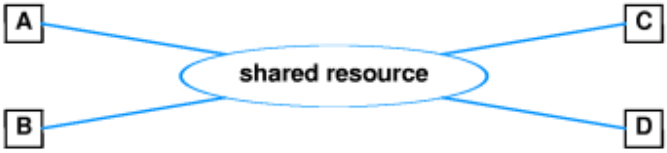
Summary

- **Long-distance communications use *carrier* and *modulation* for reliable communication**
- ***Modulator* encodes data and *demodulator* decodes data**
- **Can use *amplitude*, *frequency* or *phase shift* modulation**
- **Multiple transmitter/receiver pairs can use *multiplexing* to share a single medium**

Chapter 6 - Packets, Frames and Error Detection

Shared communication media

- **Most network use shared media which interconnect all computers**
- However **- only one source can transmit data at a time**



Packets

- **Most networks divide into small blocks called *packets* for transmission**
- **Each packet sent individually**
- **Such networks are called *packet networks* or *packet switching networks***

Motivation

- **Coordination - helps transmitter and receiver determine which data have been received correctly and which have not**
- **Resource sharing - allows multiple computers to share network infrastructure**
- **Networks enforce *fair use* - each computer can only send one packet at a time**

Dedicated network access

- **5MB file transferred across network with 56Kbps capacity will require *12 minutes*:**

$$\frac{5\text{x}10^6 \text{ bytes} * 8 \text{ bits/byte}}{60 \text{ secs/minute} * 56\text{x}10^3 \text{ bits/second}} = 11.9 \text{ minutes}$$

- **All other computers will be forced to wait 12 minutes before initiating other transfers**

Packet switching access

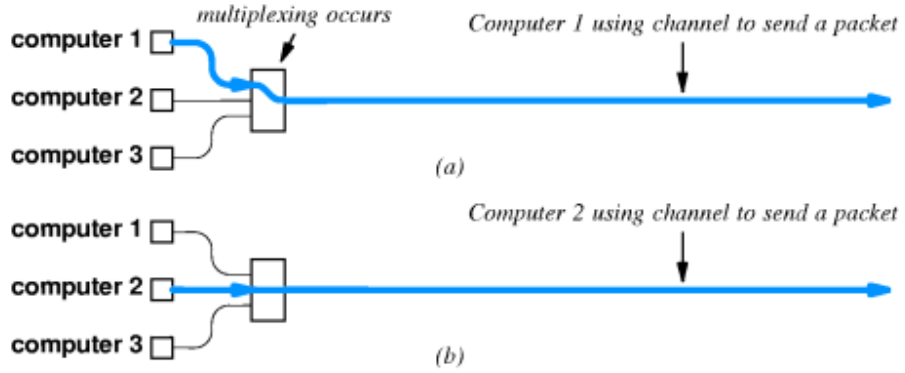
- **If file is broken into packets, other computers must only wait until packet (not entire file) has been sent**
- **From previous example, suppose file is broken into 1000 byte packets**
- **Each packet takes less than .2 seconds to transmit:**

$$\frac{\text{1000 bytes * 8 bits/byte}}{\text{56x10}^3\text{ bits/second}} = .143 \text{ seconds}$$

- **Other computer must only wait .143 seconds before beginning to transmit**
- **Note:**
  - **If both files are both 5MB long, each now takes 24 minutes to transmit**
  - BUT **if second file is only 10KB long, it will be transmitted in only 2.8 seconds, while 5MB file still takes roughly 12 minutes**

Time-division multiplexing

- **Dividing data into small packets allows *time-division multiplexing***
- **Each packet leaves the source and is switched onto the shared communication channel through a *multiplexor***
- **At the destination, the packet is switched through a *demultiplexor* to the destination**



Time-division multiplexing - example

---

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap06/chap06_7.html

Shockwave

Packets and frames

- ***Packet*** **is ``generic'' term that refers to a *small block of data***
- **Each hardware technology uses different packet format**
- ***Frame*** **or *hardware frame* denotes a packet of a specific format on a specific hardware technology**

Frame formats

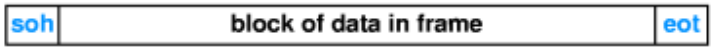
- **Need to define a standard format for data to indicate the beginning and end of the frame**
- ***Header* and *trailer* used to ``frame'' the data**

Defining the framing standard

- **Can choose two unused data values for framing**
- **E.g., if data is limited to printable ASCII, can use**
  - **``*Start of header*'' (soh)**

  - ***``end of text''*** **(eot)**
- **Sending computer sends soh first, then data, finally eot**
- **Receiving computer interprets and discards soh, stores data in buffer and interprets and discards eot**

Frame format



Packet framing

---

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap06/chap06_12.html

Shockwave

Framing in practice

- **Incurs extra overhead - soh and eot take time to transmit, but carry no data**
- **Accommodates transmission problems:**
  - **Missing eot indicates sending computer crashed**
  - **Missing soh indicates receiving computer missed beginning of message**
  - **Bad frame is discarded**

Transmitting arbitrary data

- **Suppose system can't afford to reserve two special characters for framing**
- **E.g., transmitting arbitrary 8-bit binary data**
- **soh and eot as part of data will be misinterpreted as framing data**
- **Sender and receiver must agree to encode special characters for unambiguous transmission**

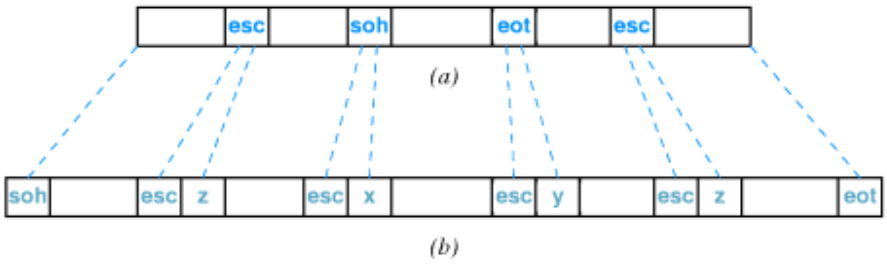
Data stuffing

- ***Bit stuffing* and *byte stuffing* are two techniques for inserting extra data to encode reserved bytes**
- **Byte stuffing translates each reserved byte into two unreserved bytes**
- **For example, can use esc as prefix, followed by x for soh, y for eot and z for esc:**

Character In Data	Characters Sent
soh	esc x
eot	esc y
esc	esc z

Byte stuffing

- **Sender translates each reserved byte into the appropriate encoding pair of bytes**
- **Receiver interprets pairs of bytes and stores encoded byte in buffer**
- **Data still framed by soh and eot**



Byte stuffing example

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap06/chap06_17.html

Shockwave

Transmission errors

- **External electromagnetic signals can cause incorrect delivery of data**
  - **Data can be received incorrectly**
  - **Data can be lost**
  - **Unwanted data can be generated**
- **Any of these problems are called *transmission errors***

Error detection and correction

- **Error detection - send additional information so incorrect data can be detected and rejected**
- **Error correction - send additional information so incorrect data can be corrected and accepted**

Parity checking

- ***Parity* refers to the number of bits set to 1 in the data item**
  - ***Even parity* - an even number of bits are 1**
  - ***Odd parity* - an odd number of bits are 1**
- **A *parity bit* is an extra bit transmitted with a data item, chose to give the resulting bits even or odd parity**
  - **Even parity - data: 10010001, parity bit 1**
  - **Odd parity - data: 10010111, parity bit 0**

Parity and error detection

- **If noise or other interference introduces an error, one of the bits in the data will be changed from a 1 to a 0 or from a 0 to a 1**
- **Parity of resulting bits will be wrong**
  - **Original data and parity: 10010001+1 (even parity)**
  - **Incorrect data: 10110001+1 (odd parity)**
- **Transmitter and receiver agree on which parity to use**
- **Receiver detects error in data with incorrect parity**

Limitations to parity checking

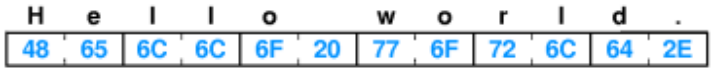
- **Parity can only detect errors that change an *odd* number of bits**
  - **Original data and parity: 10010001+1 (even parity)**
  - **Incorrect data: 10110011+1 (even parity!)**
- **Parity usually used to catch one-bit errors**

Alternative error detection schemes

- ***Many* alternative schemes exist**
  - **Detect multi-bit errors**
  - **Correct errors through redundant information**
- ***Checksum* and *CRC* are two widely used techniques**

Checksums

- **Sum of data in message treated as array of integers**
- **Can be 8-, 16- or 32-bit integers**
- **Typically use *1s-complement* arithmetic**
- **Example - 16-bit checksum with 1s complement arithmetic**



4865 + 6C6C + 6F20 + 776F + 726C + 642E + carry = 71FC

Implementing checksum computation

- **Easy to do - uses only addition**
- **Fastest implementations of 16-bit checksum use 32-bit arithmetic and add carries in at end**
- **Can also speed computation by *unrolling loop* and similar optimizations**

Limitations to checksums

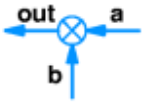
- **May not catch all errors**

Data Item In Binary	Checksum Value	Data Item In Binary	Checksum Value
0001	1	0011	3
0010	2	0000	0
0011	3	0001	1
0001	1	0011	3
totals	7		7

Cyclic redundancy checks

- **Consider data in message as coefficients of a polynomial**
- **Divide that coefficient set by a known polynomial**
- **Transmit remainder as *CRC***
  - **Good error detection properties**
  - **Easy to implement in hardware**

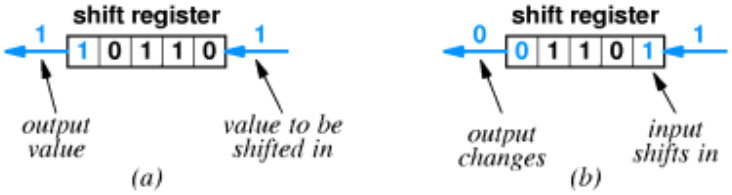
Hardware components



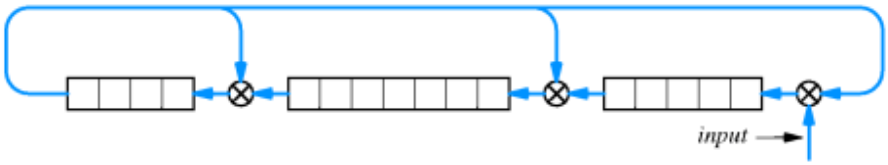
(a)

a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

(b)

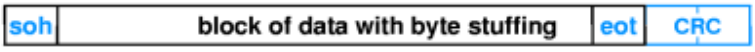


CRC hardware



Error detection and frames

- **Error detection typically done for each frame**
- **Error in frame typically causes receiver to discard frame**
- **Example - CRC sent after end of frame computed on data in frame**



Summary

- **Computer networks divide data into packets**
  - **Resource sharing**
  - **Fair allocation**
- **Hardware frames are specific to a particular hardware network technology**
- **Each frame has a specific format that identifies the beginning and end of the frame**
- **Error detection and correction is used to identify and isolate transmission errors**

Chapter 7 - LAN Technologies and Network Topology

Introduction

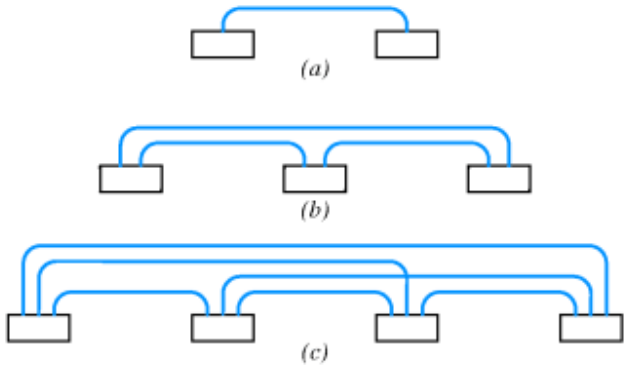
- **Sending packets across shared networks**
- **Network wiring topologies**
- **Details of *Local Area Network* (LAN) technologies**

Direct point-to-point communication

- **Computers connected by communication channels that each connect exactly two computers**
- **Forms *mesh* or *point-to-point* network**
- **Allows flexibility in communication hardware, packet formats, etc.**
- **Provides security and privacy because communication channel is not shared**

Connections in a point-to-point network

- **Number of wires grows as square of number of computers**

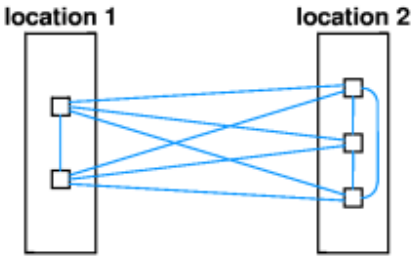


- **For N computers:**

$$\text{Connections} = \frac{(n^2 - n)}{2}$$

Connections in a point-to-point network

- **Connections between buildings can be prohibitive:**



- **Adding a new computer requires *N - 1* new connections**

Reducing the number of communication channels

- **LANs developed in late 1960s and early 1970s**
- **Key idea - reduce number of connections by *sharing* connections among many computers**
  - **Computers take turns - TDM**
  - **Must include techniques for synchronizing use**

Growth of LAN technologies

- **LAN technologies reduce cost by reducing number of connections**
- But **... attached computers compete for use of shared connection**
- **Local communication almost exclusively LAN**
- **Long distance almost exclusively point-to-point**
  - **SMDS**
  - **ATM**

Locality of reference

- **Principle of *locality of reference* helps predict computer communication patterns:**
  - ***Spatial* (or *physical*) locality of reference - computers likely to communicate with other computers that are located nearby**

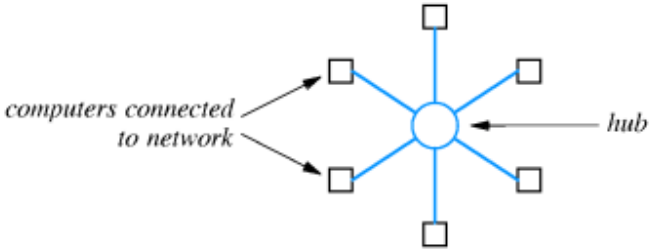
  - ***Temporal* locality of reference - computers are likely to communicate with the same computers repeatedly**
- **Thus - LANs are effective because of spatial locality of reference, and temporal locality of reference may give insight into which computers should be on a LAN**

LAN topologies

- **Networks may be classified by shape**
- **Three most popular:**
  - **Star**
  - **Ring**
  - **Bus**

Star topology

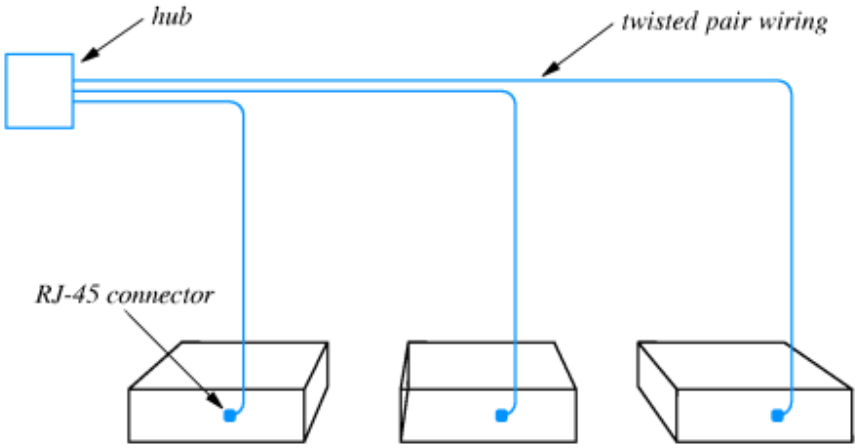
- **All computers attach to a central point:**



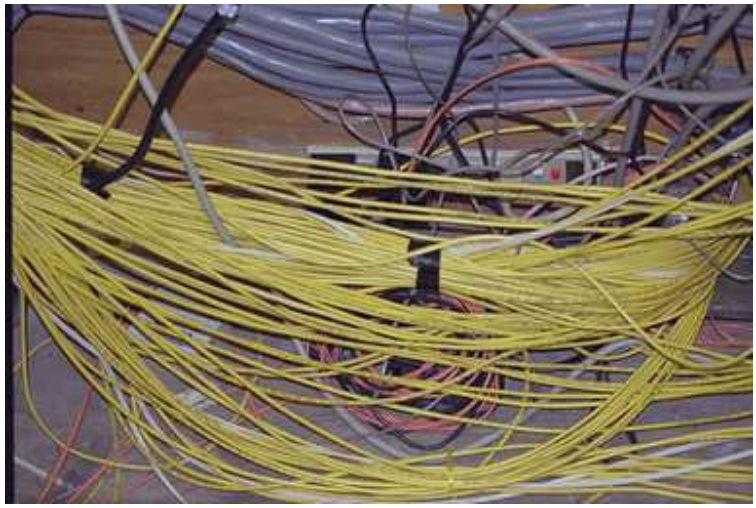
- **Center of star is sometimes called a *hub***

Star topology in practice

- **Previous diagram is idealized; usually, connecting cables run in parallel to computers:**

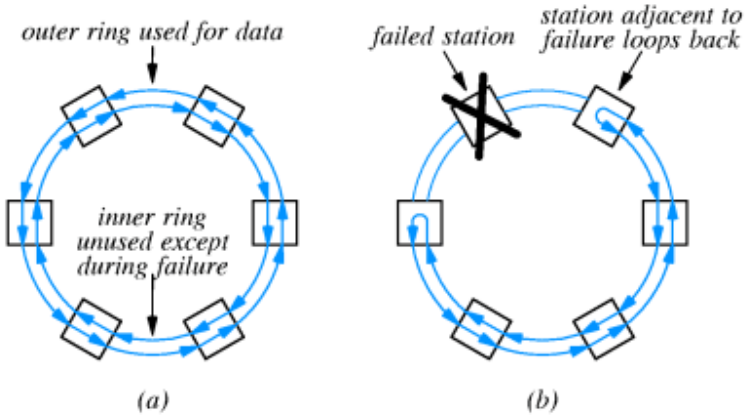


- **Result is:**



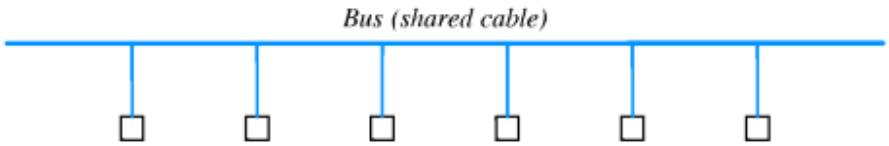
Ring topology

- **Computers connected in a closed loop**
- **First passes data to second, second passes data to third, and so on**
- **In practice, there is a short connector cable from the computer to the ring**
- **Ring connections may run past offices with connector cable to socket in the office:**



Bus topology

- **Single cable connects all computers**
- **Each computer has connector to shared cable**
- **Computers must synchronize and allow only one computer to transmit at a time**



Why multiple topologies?

- **Each has advantages and disadvantages:**
  - **Ring ease synchronization; may be disabled if any cable is cut**
  - **Star easier to manage and more robust; requires more cables**
  - **Bus requires fewer cables; may be disable if cable is cut**
- **Bucknell has used all three; now almost entirely star topology**
- **Industry is settling on star topology as most widely used**

Ethernet

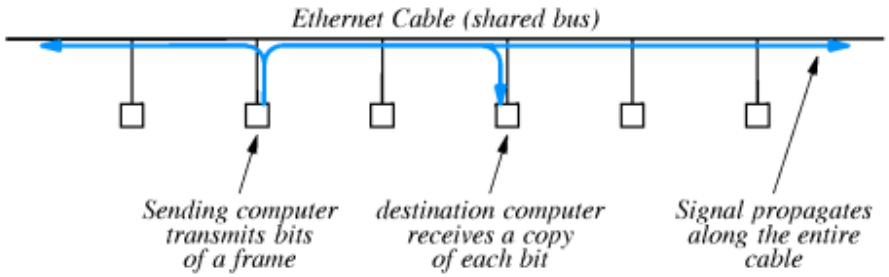
- **Widely used LAN technology**
  - **Invented at Xerox PARC (Palo Alto Research Center) in 1970s**
  - **Defined in a standard by Xerox, Intel and Digital - *DIX* standard**
  - **Standard now managed by IEEE - defines formats, voltages, cable lengths, ...**
- **Uses bus topology**
  - **Single coax cable - the *ether***
  - **Multiple computers connect**
- **One Ethernet cable is sometimes called a *segment***
  - **Limited to 500 meters in length**
  - **Minimum separation between connections is 3 meters**

Ethernet speeds

- **Originally 3Mbps**
- **Current standard is 10Mbps**
- ***Fast Ethernet* operates at 100Mbps**

Ethernet operation

- **One computer transmits at a time**
- **Signal is a modulated carrier which propagates from transmitter in both directions along length of segment**



Ethernet example

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap07/chap07_17.html

Shockwave

CSMA

- **No central control managing when computers transmit on ether**
- **Ethernet employs CSMA to coordinate transmission among multiple attached computers**
- ***Carrier Sense with Multiple Access***
  - **Multiple access - multiple computers are attached and any can be transmitter**
  - **Carrier sense - computer wanting to transmit tests ether for carrier before transmitting**

CSMA example

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap07/chap07_19.html

Shockwave

Collision detection - CD

- **Even with CSMA, two computers may transmit simultaneously**
  - **Both check ether at same time, find it idle, and begin transmitting**
  - **Window for transmission depends on speed of propagation in ether**
- **Signals from two computers will interfere with each other**
- **Overlapping frames is called a *collision***
  - **No harm to hardware**
  - **Data from both frames is garbled**

Collision example

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap07/chap07_21.html

Shockwave

Ethernet CD

- **Ethernet interfaces include hardware to detect transmission**
  - **Monitor outgoing signal**
  - **Garbled signal is interpreted as a collision**
- **After collision is detected, computer stops transmitting**
- **So, Ethernet uses CSMA/CD to coordinate transmissions**

Recovery from collision

- **Computer that detects a collision sends special signal to force all other interfaces to detect collision**
- **Computer then waits for ether to be idle before transmitting**
  - **If both computers wait same length of time, frames will collide again**
  - **Standard specifies maximum delay, and both computers choose random delay less than maximum**
- **After waiting, computers use carrier sense to avoid subsequent collision**
  - **Computer with shorter delay will go first**
  - **Other computers may transmit first**

Exponential back-off

- **Even with random delays, collisions may occur**
- **Especially likely with busy segments**
- **Computers double delay with each subsequent collision**
- **Reduces likelihood of sequence of collisions**

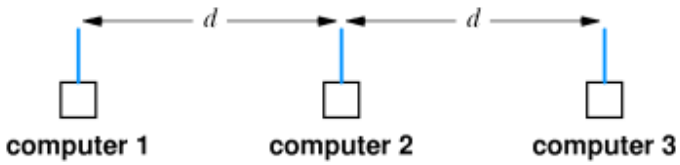
Wireless LAN

- **Use radio signals at 900MHz**
- **Data rate of 2Mbps**
- **Shared medium - radio instead of coax**

Limited connectivity with wireless

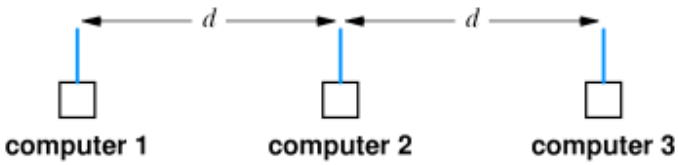
- **In contrast with wired LAN, not all participants may be able to reach each other**
  - **Low signal strength**
  - **Propagation blocked by walls, etc.**

- **Can't depend on CD; not all participants may hear**



CSMA/CA

- **Wireless uses *collision avoidance* rather than collision detection**
  - **Transmitting computer sends very short message to receiver**
  - **Receiver responds with short message reserving slot for transmitter**
- **Response from receiver is *broadcast* so all potential transmitters receive reservation**



Collisions

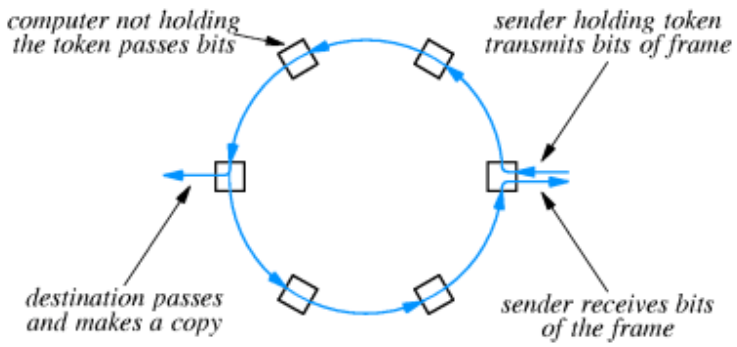
- **Receiver may receive simultaneous requests**
  - **Results in collision at receiver**
  - Both **requests are lost**
  - **Neither transmitter receives reservation; both use back-off and retry**
- **Receiver may receive closely spaced requests**
  - **Selects one**
  - **Selected transmitter sends message**
  - **Transmitter not selected uses back-off and retries**

LocalTalk

- **LAN technology that uses bus topology**
- **Interface included with all Macintosh computers**
- **Relatively low speed - 230.4Kbps**
- **Low cost (``free'' with a Macintosh); easy to install and connect**
- **Uses CSMA/CD**

Token ring

- **Many LAN technologies that use ring topology use *token passing* for synchronized access to the ring**
- **Ring itself is treated as a single, shared communication medium**
- **Bits pass from transmitter, past other computers and are copied by destination**
- **Hardware must be designed to pass token even if attached computer is powered down**



Transmission around a token ring

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap07/chap07_31.html

Shockwave

Using the token

- **When a computer wants to transmit, it waits for the *token***
- **After transmission, computer transmits token on ring**
- **Next computer ready to transmit receives token and then transmits**

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap07/chap07_32.html

Shockwave

Token and synchronization

- **Because there is only one token, only one computer will transmit at a time**
  - **Token is short, reserved frame that cannot appear in data**
  - **Hardware must regenerate token if lost**
- **Token gives computer permission to send one frame**
  - **If all ready to transmit, enforces ``round-robin'' access**
  - **If none ready to transmit, token circulates around ring**

IBM token ring

- **Very widely used**
- **Originally 4mbps, now 16Mbps**
- **Uses special connector cable between computer and ring interface**

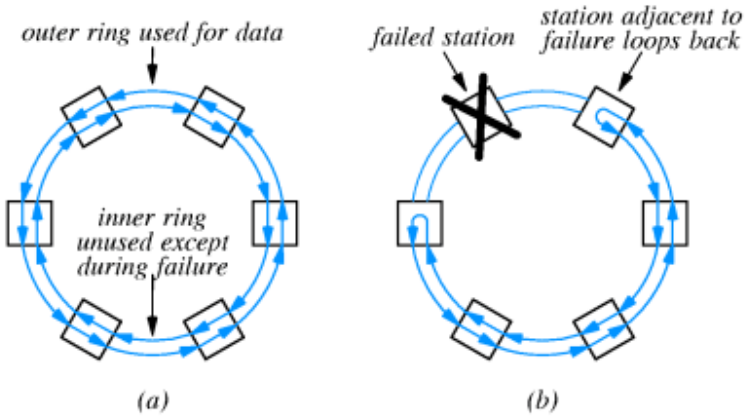
FDDI

- ***Fiber Distributed Data Interconnect* (FDDI) is another ring technology**
  - **Uses fiber optics between stations**
  - **Transmits data at 100Mbps**
- **Uses pairs of fibers to form two concentric rings**

FDDI and reliability

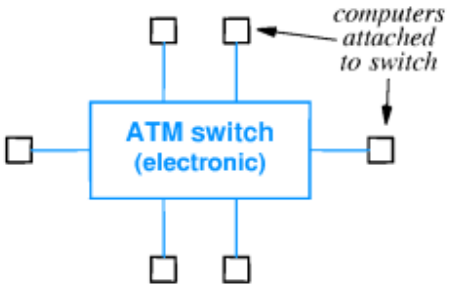
- **FDDI uses *counter-rotating* rings in which data flows in opposite directions**
- **In case of fiber or station failure, remaining stations *loop back* and reroute data through spare ring**

- **All stations automatically configure loop back by monitoring data ring**



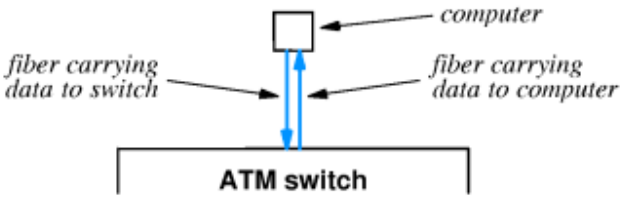
ATM - Star network

- ***Asynchronous Transfer Mode* technology consists of electronic packet switches to which computers can connect**
- **ATM switches form *hub* into which computers connect in a *star* topology**
- **Computers get point-to-point connections - data from transmitter is routed directly through hub switches to destination**

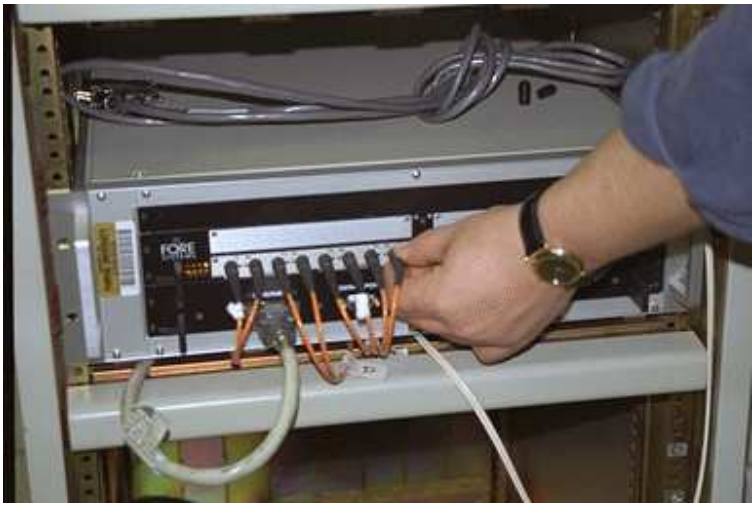


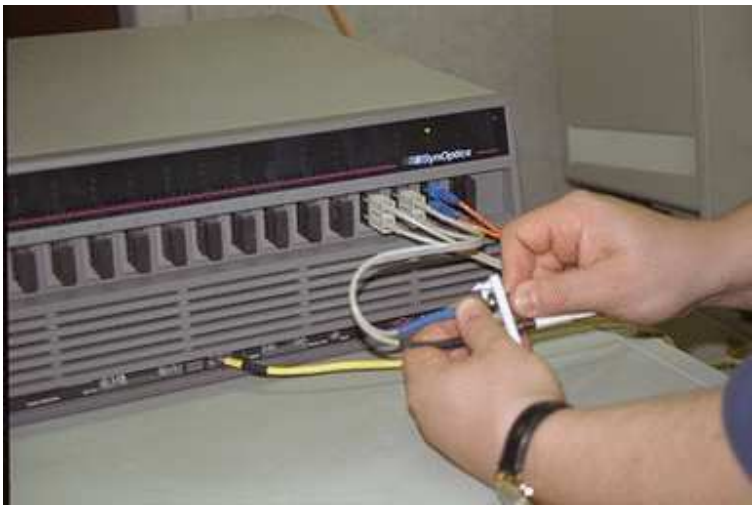
ATM details

- **Transmits data at over 100Mbps**
- **Uses fiber optics to connect computer to switch**
- **Each connection includes two fibers**



ATM switches





Summary

- **LAN technologies use shared communication media to interconnect multiple computers over short distances**
- **Transmitting computer has exclusive use of communication medium; computers must synchronize transmission and share available capacity**
- **LAN topologies:**
  - **Star**
  - **Ring**
  - **Bus**
- **LAN technologies**

- **Ethernet**
- **Wireless**
- **LocalTalk**
- **IBM Token Ring**
- **FDDI**
- **ATM**

Chapter 8 - Hardware Addressing and Frame Type Identification

---

Introduction

- **Previous chapter on LAN technology described techniques for providing connectivity between computers**
- **Need to devise technique for delivering message through LAN medium to single, specific destination computer**
- **Sending computer uses a *hardware address* to identify the intended destination of a frame**
- **Sending computer also identifies** type **of data carried in the frame**

Specifying a destination

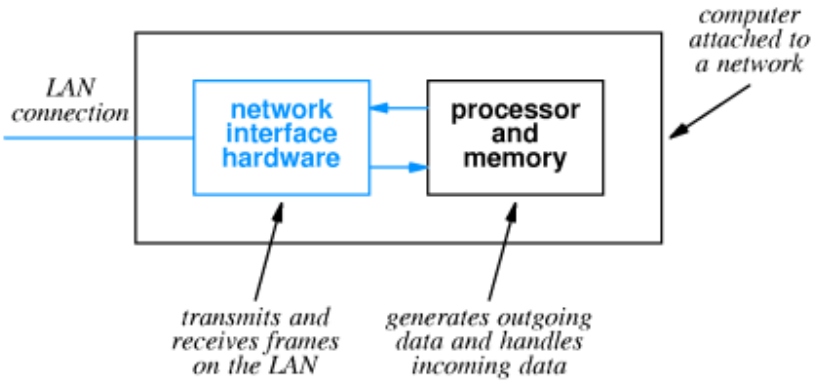
- **Data sent across a shared network reaches all attached stations - for all LAN topologies**
- **Interface hardware detects delivery of frame and extracts frame from medium**
- **But ... most applications want data to be delivered to one specific application on another computer - not** all **computers**

Hardware addressing

- **Most network technologies have a hardware addressing scheme that identifies stations on the network**
- **Each station is assigned a numeric *hardware address* or *physical address***
- **Sender includes hardware address in each transmitted frame**
- **Only station identified in frame receives copy of frame**
- **Most LAN technologies include sender's hardware address in frame, too**

LAN hardware and packet filtering

- **A little detail about organization of LAN hardware and computer:**



LAN hardware and packet filtering

- **LAN interface handles all details of frame transmission and reception**
  - **Adds hardware addresses, error detection codes, etc. to outgoing frames**
  - **May use DMA to copy frame data directly from main memory**
  - **Obeys access rules (e.g., CSMA/CD) when transmitting**
  - **Checks error detection codes on incoming frames**
  - **May use DMA to copy data directly into main memory**
  - **Checks destination address on incoming frames**
- **If destination address on incoming frame matches the local station's address, a copy of the frame is passed to the attached computer**
- **Frames** not **addressed to the local computer are ignored and don't affect the local computer in any way**

Format of hardware addresses

- **Numeric value**
- **Size selected for specific network technology**
- **Length is one to six bytes**

Assigning hardware addresses

- **Hardware addresses must be *unique* on a LAN**
- **How can those addresses be assigned and who is responsible for uniqueness?**

Static:	**Hardware manufacturer assigns permanent address to each interface**	**Manufacturer must ensure *every* interface has a unique address**
Dynamic:	**Address can be set by end user, either through switches or jumpers on the interface or through software**	**System administrators must coordinate to avoid conflict**
Automatic:	**Interface automatically assigns hardware address each time it is powered up**	**Automatic scheme must be reliable to prevent conflicts**

Broadcasting

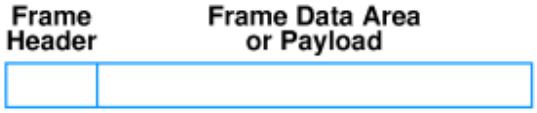
- **Some applications want to *broadcast* messages to all stations on the LAN**
- **Shared communication channel can make broadcast efficient - message is delivered to all stations**
- **Special *broadcast address* used to identify broadcast messages, which are captured by *all* stations**

Identifying packet contents

- **Destination must get some clue about how to interpret frame data**
- **Can use:**
  - ***Explicit frame type* - identifying value included with frame describes type of included data**
  - ***Implicit frame type* - receiver must infer type from frame data**

Headers and frame formats

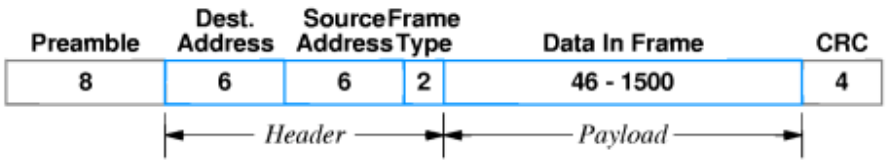
- **LAN technology standards define frame format for each technology**
- **All contemporary standards use the following general format:**



- **Frame header has address and other identifying information**
- **Information typically in *fields* with fixed size and location**
- **Data area may vary in size**

Example frame format

- **Ethernet frame format:**



- **Details:**

Field	Purpose
Preamble	**Receiver synchronization**
Dest. addr.	**Identifies intended receiver**
Source addr.	**Hardware address of sender**
Frame type	**Type of data carried in frame**
Data	**Frame payload**
CRC	**32-bit CRC code**

Ethernet fields

- **Preamble and CRC often not shown**
- **Destination address of all 1s is the broadcast address**

- **Special values are reserved for frame type field:**

Value	Meaning
0000-05DC	Reserved for use with IEEE 802.3
0800	Internet IP Version 4
0805	CCITT X.25
0900	Ungermann-Bass Corporation network debugger
0BAD	Banyan Systems Corporation VINES
1000-100F	Berkeley UNIX Trailer encapsulation
6004	Digital Equipment Corporation LAT
6559	Frame Relay
8005	Hewlett Packard Corporation network probe
8008	AT&T Corporation
8014	Silicon Graphics Corporation network games
8035	Internet Reverse ARP
8038	Digital Equipment Corporation LANBridge
805C	Stanford University V Kernel
809B	Apple Computer Corporation AppleTalk
80C4-80C5	Banyan Systems Corporation
80D5	IBM Corporation SNA
80FF-8103	Wellfleet Communications
8137-8138	Novell Corporation IPX
818D	Motorola Corporation
FFFF	Reserved

Frames without type fields

- **Some LAN technologies do not include a type field**
- **Sender and receiver can agree on interpretation:**
  - **Agree on a single data format and use only that format**
    - **Limits LAN to one type of data**
    - **All computers on LAN must use one format**
  - **Agree to encode the data format in the first few bytes of the data field**

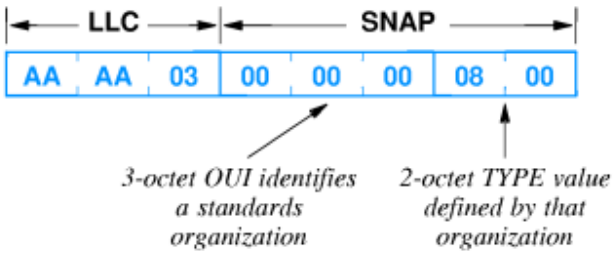
Encoding the data type

- **Illustration of using data area to encode data type:**

- **To ensure interoperability, format of encoding area must be universally agreed upon**
- **Format typically set by standards body**

IEEE 802.2 LLC

- **IEEE 802.2 standard includes *Logical Link Control* (LLC) *SubNetwork Attachment Point* (SNAP) header**
- **SNAP/LLC format widely used; e.g., by Ethernet**



- **LLC portion indicates SNAP field to follow**
- **OUI (*Organizationally Unique Identifier*) identifies Ethernet specification organization**
- **TYPE field interpreted as in Ethernet (in this case, IP)**

Unknown types

- **For either encoding format, some computers may not be prepared to accept frames of some types**
  - **Protocol type not installed**
  - **Newly defined type**
- **Receiving computer examines type field and discards any frames with unknown type**

Network analyzers

- **A *network analyzer* or *network monitor* or ``*network sniffer*'' is used to examine the performance of or debug a network**
- **Can report statistics such as capacity utilization, distribution of frame size, collision rate or token circulation time**
- **Can record and display specific frames, to understand and debug packet transmissions and exchanges**

Operation of a network analyzer

- **Basic idea is a computer with a network interface that receives** all **frames**
- **Sometimes called *promiscuous mode***
- **Many desktop computers have interface that can be configured for promiscuous mode**
  - **Combined with software, computer can examine *any* frame on LAN**
  - **Communication across a LAN is** not **guaranteed to be private!**
- **Computer receives and displays (but does not respond to) frames on the LAN**

Filtering incoming frames

- **Analyzer can be configured to filter and process frames**
  - **Count frames of a specific type or size**
  - **Display only frames from or to specific computers**
  - **In general, can be configured to match value of any field and capture only those frames meeting the filter specification**
- **Analyzer can display real-time performance by computing running totals over specific time periods**

Summary

- **LAN technologies use *hardware addresses* to identify destination for frames sent across shared communication channel**
- **Each LAN technology defines its own hardware format**
- **Addresses may be *statically* assigned, *configurable* or *automatically* assigned**
- **Each station must have a unique address on the LAN segment**
- **Frames include a *header* with fields for destination, source and other information such as frame type**
- ***Frame type* defines how to interpret frame data**
- ***Network analyzer* can receive** all **frames and display statistics or aid in debugging problems**

Chapter 9 - LAN Wiring, Physical Topology and Interface Hardware

Section	Title

Introduction

- **Interface cards**
  - **Why a separate card**
  - **How to connect the interface to the computer**
  - **What is a ``transceiver''?**
- **LAN wiring schemes**
- **Logical and physical topology**

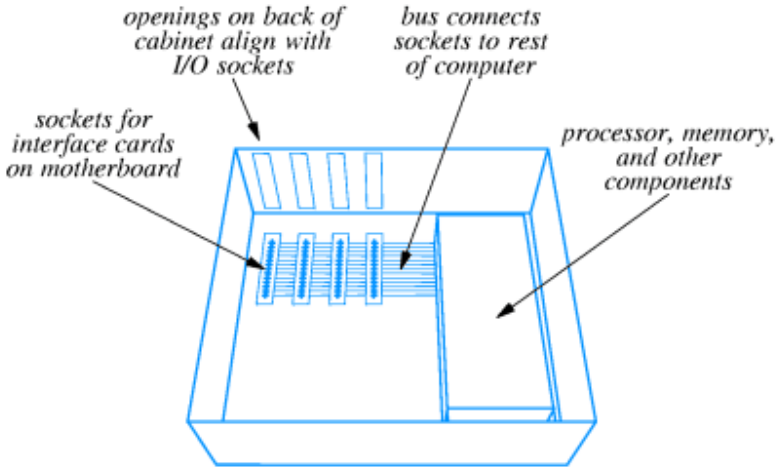
Speeds of LANs and computers

- **LAN data transmission speeds are typically ``fast'' relative to CPU speeds**
- **100MHz CPU could execute only one instruction for each bit on a 100Mhz Ethernet**
- **LAN speeds are defined independent of any specific processor speeds**
  - **Allows for mix of attached systems**
  - **New computers can be attached without affecting LAN speeds**

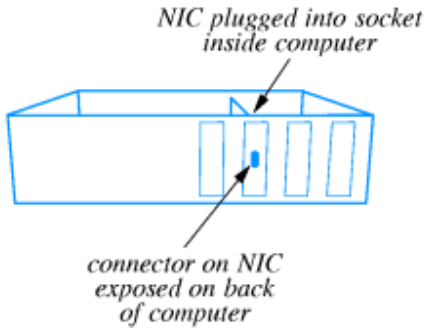
Network interface hardware

- **CPU can't process data at network speeds**
- **Computer systems use special purpose hardware for network connection**
  - **Typically a separate card in the backplane**
  - ***Network adapter card* or *network interface card (NIC)***
- **Connector at back of computer then accepts cable to physical network**

I/O interfaces



Network connector



NICs and network hardware

- **NIC is built for one kind of physical network**
  - **Ethernet interface can't be used with token ring**
  - **ATM interface can't be used with FDDI**
- **Some NICs can be used with different, similar hardware**
  - **Thick, thin and 10Base-T Ethernet**
  - **10Mbps and 100Mbps Ethernet**

NIC and CPU processing

- **NIC contains sufficient hardware to process data independent of system CPU**
  - **Some NICs contain separate microprocessor**
  - **Includes analog circuitry, interface to system bus, buffering and processing**
- **Looks like any other I/O device to system CPU**
  - **System CPU forms message request**
  - **Sends instructions to NIC to transmit data**
  - **Receives interrupt on arrival of incoming data**

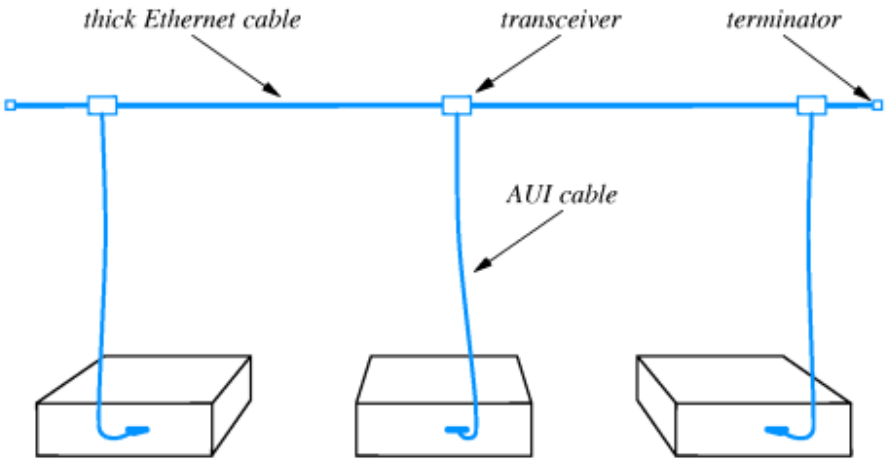
Connection between NIC and physical network

- **Two alternatives:**
  - **NIC contains all circuitry and connects directly to network medium**
  - **Cable from NIC connects to additional circuitry that then attaches to the network medium**
- **Thin Ethernet vs. 10Base-T**
- **Both are Ethernet; network technology not limited to one style of connection**

Thick Ethernet wiring

- **Uses thick coax cable**
- ***AUI* cable (or *transceiver* or *drop* cable connects from NIC to *transceiver***
- **AUI cable carries digital signal from NIC to transceiver**
- **Transceiver generates analog signal on coax**
- **Wires in AUI cable carry digital signals, power and other control signals**

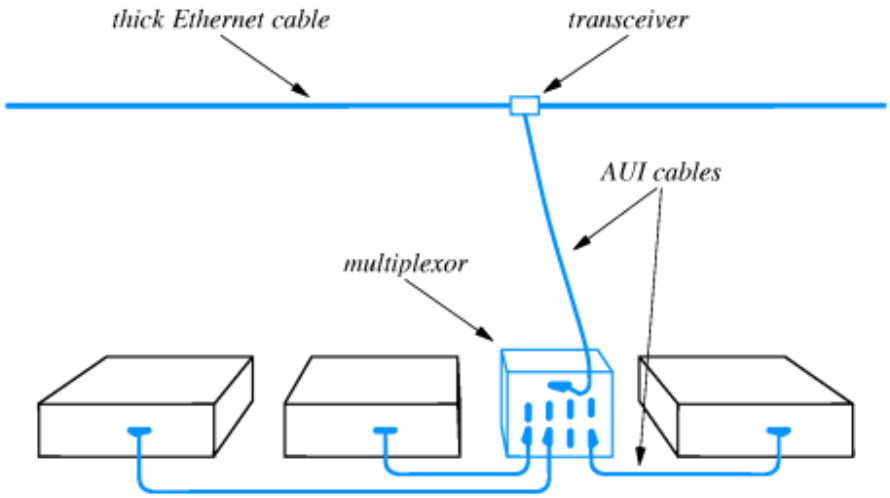
Thick Ethernet example



- **Thick Ethernet also requires *termination* to avoid signal reflectance**

Connection multiplexing

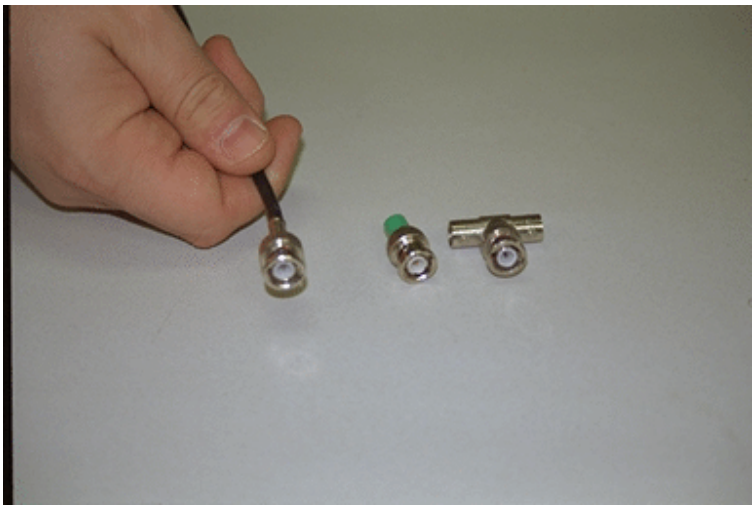
- **In some circumstances, transceivers may be inconvenient; e.g., workstations in a lab**
- ***Connection multiplexor* connects multiple computers to a single transceiver**
  - **Each computer's AUI cable connects to connection multiplexor**
  - **One AUI from multiplexor to Ethernet coax**



- **Connection multiplexor completely invisible to attached computers**

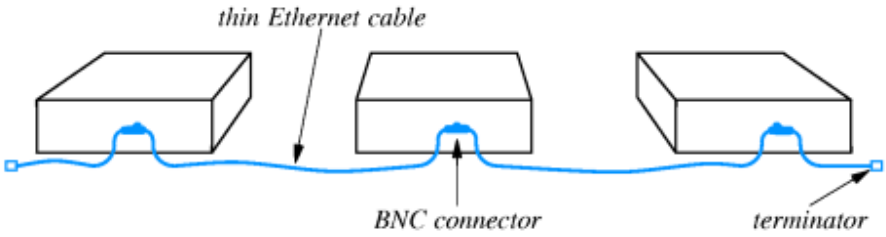
Thin Ethernet wiring

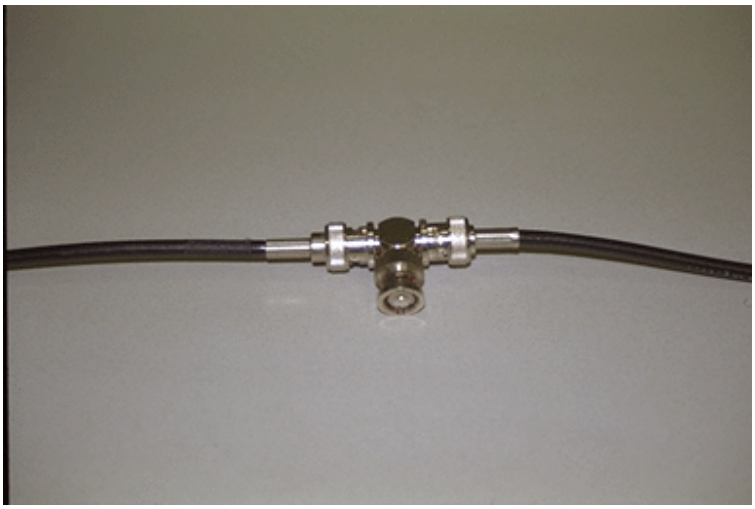
- **Uses thin coax that is cheaper and easier to install than thick Ethernet coax**
- **Transceiver electronics built into NIC; NIC connects directly to network medium**
- **Coax cable uses *BNC* connector**



Thin Ethernet wiring (continued)

- **Coax runs directly to back of each connected computer**
- ***T connector* attaches directly to NIC**



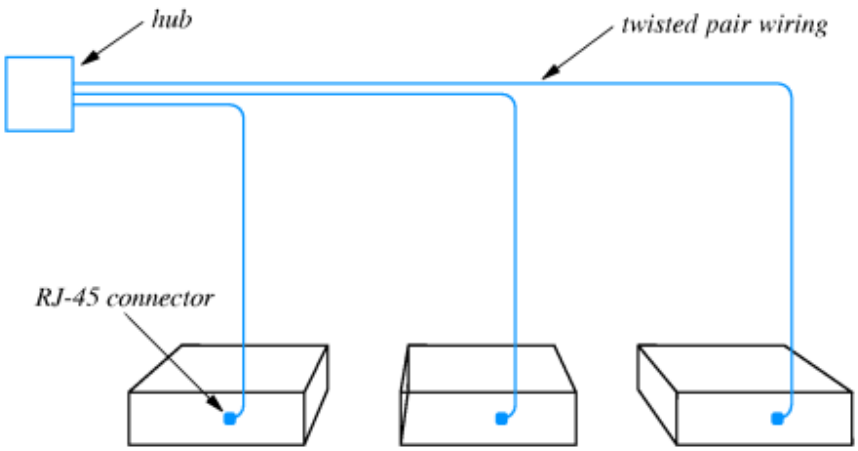


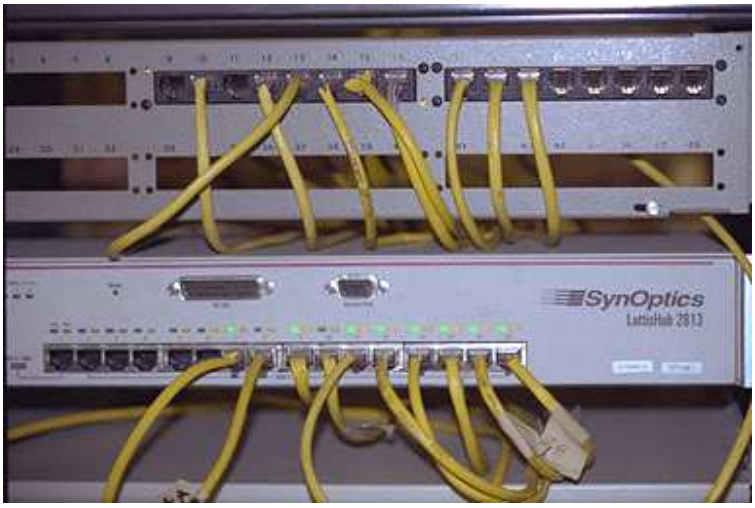
Thin Ethernet wiring (continued)

- **Useful when many computers are located close to each other**
- **May be unreliable - any disconnection disrupts entire net**

10Base-T

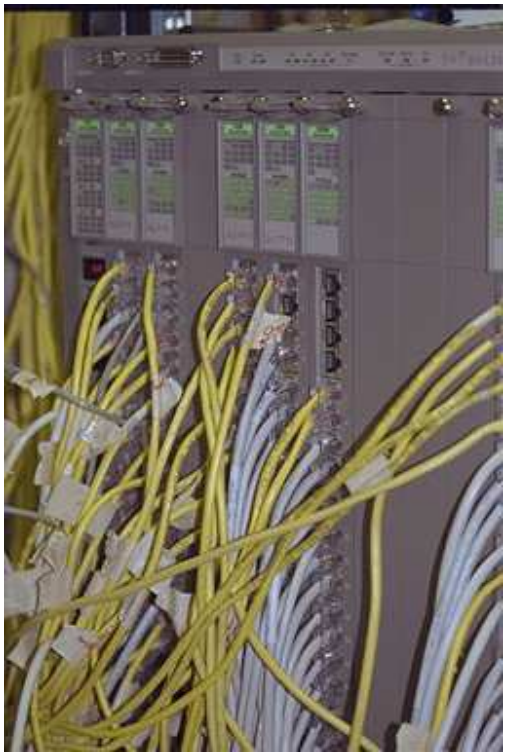
- **Variously called *10Base-T*, *twisted pair* or *TP Ethernet***
- **Replaces AUI cable with twisted pair cable**
- **Replaces thick coax with *hub***





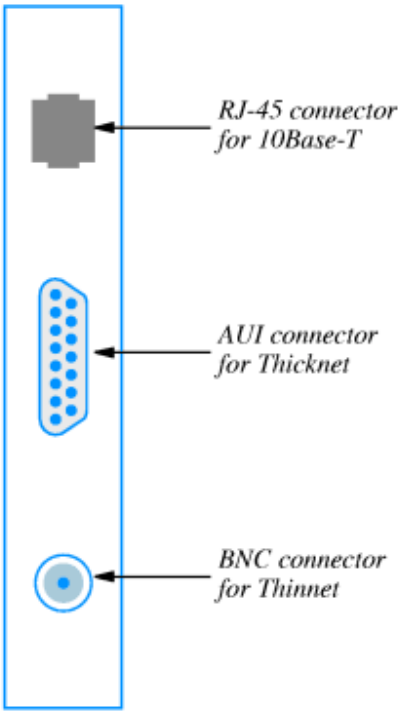
Hubs

- **Extension of connection multiplexing concept**
- **Sometimes called ``Ethernet-in-a-box''**
- **Effectively a very short Ethernet with very long AUI cables**
- **Can be connected into larger Ethernets**



Protocol software and Ethernet wiring

- **All wiring technologies use identical Ethernet specification**
  - **Same frame format**
  - **Same CSMA/CD algorithms**
- **Can mix different technologies in one Ethernet**
- **NICs can provide all three connection technologies**



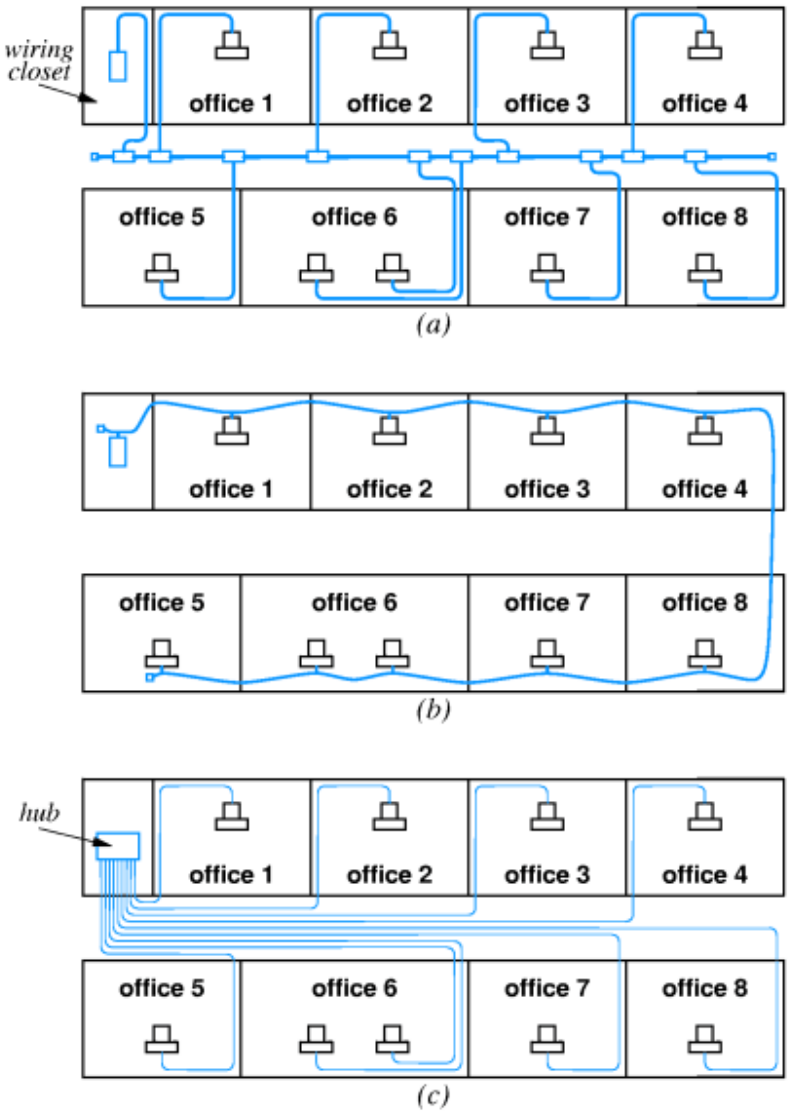
- **Protocol software can't differentiate among wiring technologies**

Comparison of wiring schemes

- **Separate transceiver allows computer to be powered off or disconnected from network without disrupting other communication**
- **Transceiver may be located in an inconvenient place**
- **Finding malfunctioning transceiver can be hard**
- **Thin coax takes minimum of cable**
- **Disconnecting one computer (or one loose connection) can disrupt entire network**

- **Hub wiring centralizes electronics and connections, making management easier**
- **Bottom line - 10Base-T most popular because of cost**

Comparison of wiring schemes (continued)

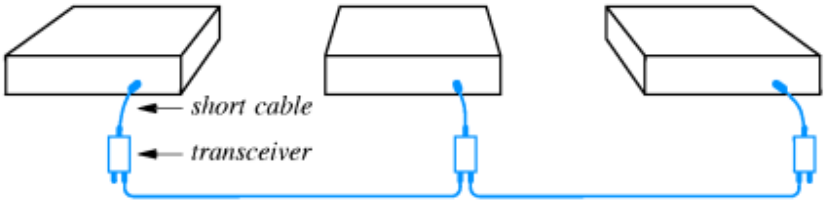


Topologies and network technologies

- **10Base-T *network topology* is a bus; *wiring topology* is a star**
- **Token ring *network topology* is a ring; *wiring topology* is a star**
- **Remember to distinguish between *logical* and *physical* topologies**

Other technologies

- **AppleTalk uses bus wiring with coax cable between transceivers**



- **AppleTalk can also use hub technology or spare wires in 4-wire phone cable**

Technology translation

- ***Adapters* can translate between some network technologies**
  - **Ethernet AUI-to-thinnet**

- **Ethernet AUI-to-10Base-T adapters**

- **AppleTalk to phone wire**

Summary

- ***Network interface card*** **(NIC) connects computer system to network**
  - **NIC operates independently; is fast enough to keep up with network**
  - **Typically uses interrupts to interact with CPU**
- **Many *physical* wiring schemes are available for *logical* network topology**
- **10Base-T is a *logical bus* and a *physical star***

Chapter 10 Extending LANs: Fiber Modems, Repeaters, Bridges and Switches

Introduction

- **LAN technologies are designed with constraints of speed, distance and costs**
- **Typical LAN technology can span, at most, a few hundred meters**
- **How can a network be extended to cover longer distances; e.g., the Bucknell campus?**

LAN design for distance

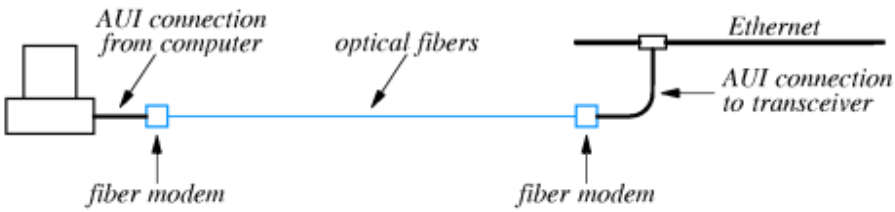
- **Many LANs use shared medium - Ethernet, token ring**
- **Length of medium affects fair, shared access to medium**
  - **CSMA/CD - delay between frames, minimum frame length**
  - **Token passing - circulation time for token**
- **Length of medium affects strength of electrical signals and noise immunity**

LAN extensions

- **Several techniques extend diameter of LAN medium**
- **Most techniques use additional hardware**
- **LAN signals relayed between LAN segments**
- **Resulting mixed technology stays within original engineering constraints while spanning greater distance**

Fiber optic extensions

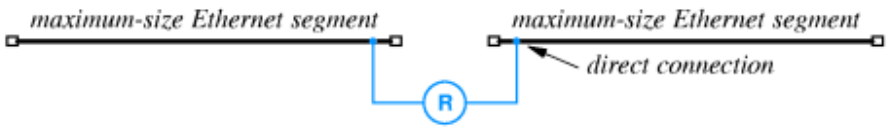
- **Can extend connection to a computer using fiber optic cable**
- **Insert *fiber modems* and fiber optic cable into AUI cable**



- **Fiber modems:**
  - **Convert AUI signals to digital signal**
  - **Transmit digital signals via fiber optic cable to other modem**
- **Most often used to connect two LANs - typically through a bridge - different buildings**

Repeaters

- **May want to extend LAN medium**
  - **Ethernet - timing constraints allow longer medium**
  - **Signal strength constraints limit length**
- ***Repeater* - bidirectional, analog amplifier that retransmits analog signals**



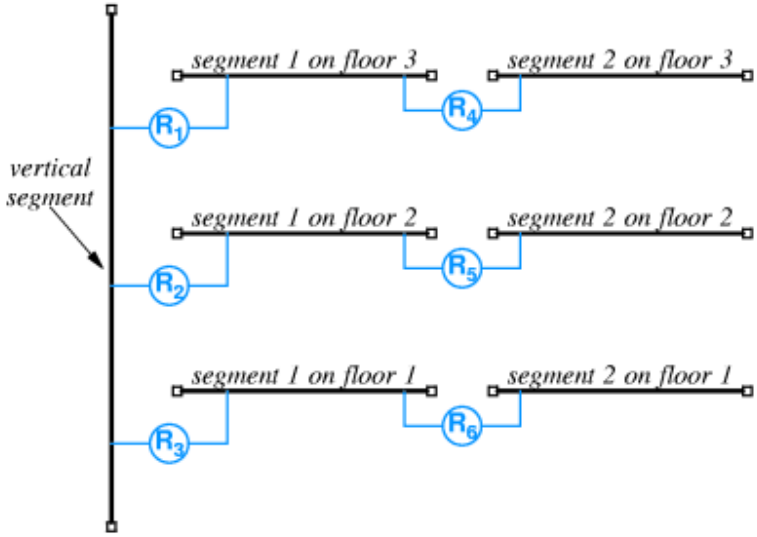
- **One repeater can effectively double the length of an LAN segment**

Ethernet repeaters

- **Simply copy signals between segments**
  - **Do not understand frame formats**
  - **Do not have hardware addresses**
- **Any Ethernet segment is limited to 500 meters**
- **Repeater can double to 1,000 meters**

Limits on repeaters

- **Can't extend Ethernet with repeaters indefinitely**
- **CSMA/CD requires low delay; if medium is too long, CSMA/CD won't work**
- **Ethernet standard includes limit of 4 repeaters between any two Ethernet stations**



Repeater architecture

- **With four repeaters, can extend Ethernet through a building**

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap10/chap10_8.html
Shockwave

- ***FOIRL* - Fiber Optic Intra-Repeater Link - can be used to connect bridges**

Characteristics of repeaters

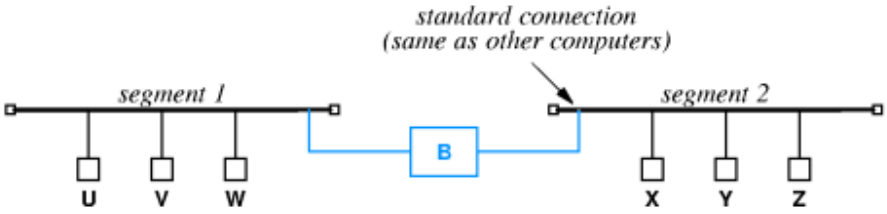
- Very **easy to use - just plug in**
- **Repeaters simply re-transmit analog signals**
  - **Collisions affect entire network**
  - **Transient problems - noise - propagates throughout network**

Bridges

- **Also connect two LAN segments**
- **Retransmits frames from one segment on other segment(s)**
- **Handles *complete frame***

  - **Uses NIC like any other station**
  - **Performs some processing on frame**
- **Invisible to other attached computers**

Bridged LAN segments



Characteristics of bridges

- Relatively **easy to use - just plug in**
- **Isolate collisions, noise**

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap10/chap10_12.html

Shockwave

Filtering bridges

- **Bridges can do additional processing**
  - **Don't forward collisions, noise**
  - **Only forward frames where necessary**
- **Bridge performs *frame filtering* and forwards frames along LAN segments to destination**
  - **Learns location of stations by watching frames**
  - **Forwards all broadcast and multicast packets**

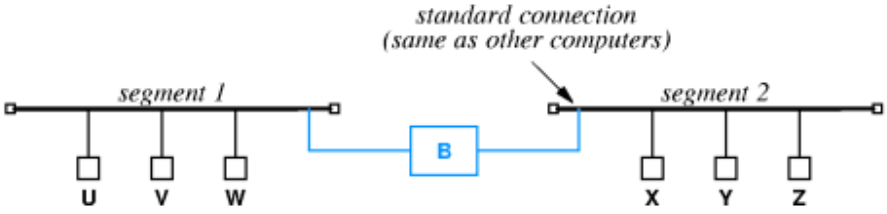
Frame filtering

- **Bridge checks destination of each incoming frame**
- **Looks up destination in list of known stations**
  - **Forwards frame to next interface on path to destination**
  - Doesn't **forward frame if destination on LAN segment from which frame was received**

How does bridge set up table?

- **Bridge examines *source* address in each frame**
- **Adds entry to list for LAN segment from which frame was received**
- **Must forward any frame whose *destination* is not in the list on *every* interface**

Filtering example



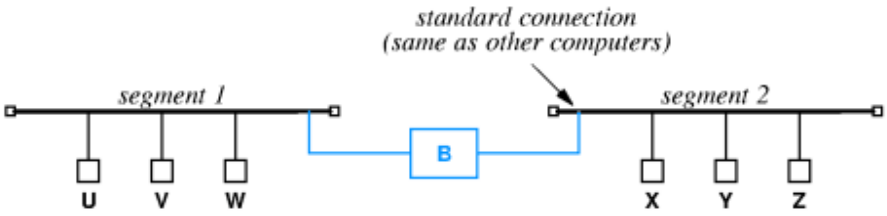
Event	Segment 1 List	Segment 2 List
Bridge boots	–	–
U sends to V	U	–
V sends to U	U, V	–
Z broadcasts	U, V	Z
Y sends to V	U, V	Z, Y
Y sends to X	U, V	Z, Y
X sends to W	U, V	Z, Y, X
W sends to Z	U, V, W	Z, Y, X

Startup behavior of filtering bridges

- **Initially, the forwarding tables in all bridges are empty**
- **First frame from each station on LAN is forwarded to *all* LAN segments**
- **After all stations have been identified, frames are only forwarded as needed**
- **May result in burst of traffic after, e.g., power failure**

Designing with filtering bridges

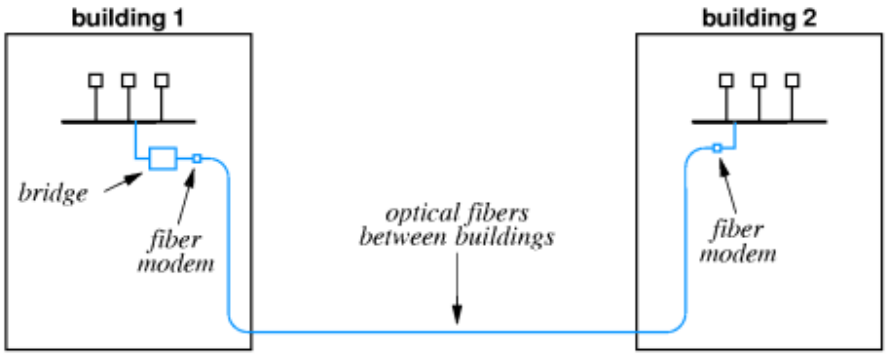
- **Filtering bridge allows concurrent use of different LAN segments** if **traffic is local**
- U **and** V **can exchange frames at the same time** X **and** Y **exchange frames**



- **Designers identify patterns of local communication and isolate groups of communicating computers with bridges**

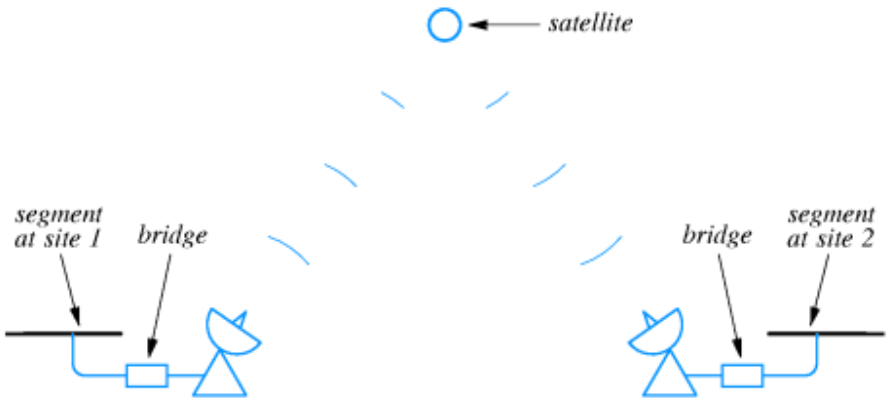
Bridging between buildings

- **Similar to extending AUI with fiber modems**
- **Can put bridge in one building with long connection to LAN segment in different building**
- **Avoids extended AUI connection for each computer in remote building**



Bridging across longer distances

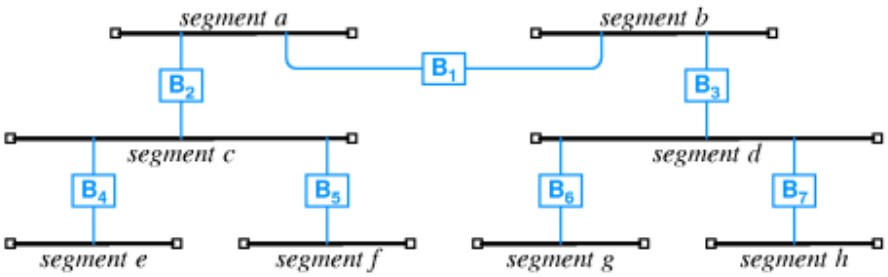
- **Can use leased line, microwave, laser or satellite to connect two bridges and LAN segments**



- **Using two bridges instead of one:**
  - **Filters at *both* ends, reducing traffic across slow link**
  - **Provides buffering at both ends, matching dissimilar transmission speeds**

Bridges and cycles

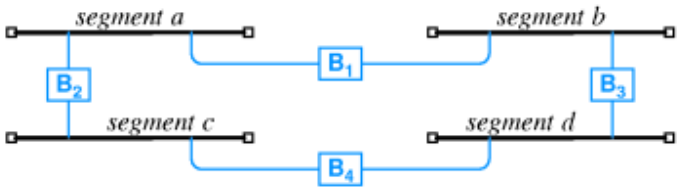
- **Can use multiple bridges to interconnect many LAN segments**



- **Station of segment *c* sends frames to station on segment *g* through** B_2**,** B_1**,** B_3 **and** B_6
- **Broadcasts are forwarded through all bridges**
- **Suppose another bridge connects *g* and *f*?**

Cycles of bridges

- **A circular path through bridged networks is called a *cycle***
- **Adding** B_4 **creates a cycle**



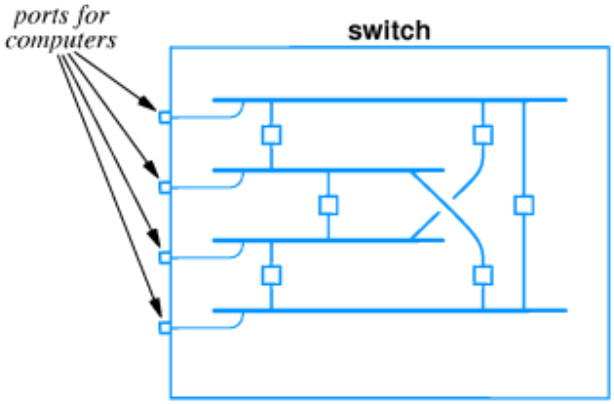
Eliminating broadcast cycles

- **Bridges must cooperate to broadcast frames exactly once on each segment**

- **Solution from graph theory - *spanning tree* - used to determine which bridges will forward broadcasts**
- **As each bridge joins the network, it communicates with other bridges on special hardware (typically multicast) address**
  - **Learns network topology**
  - **Performs spanning tree computation**
  - **Determines if bridge will form a cycle**

Switching

- **Effectively a separate LAN segment for each port**
- **Similar to hub - hub shares single segment among all ports**



- **With switching, multiple stations can transmit simultaneously**
- **Provides much higher aggregate bandwidth**

Switches and hubs

- **Switches are more expensive per port**
- **May make more sense economically to use hubs for some stations and switches for others**

Summary

- **Optical fiber and modems can be used to extend AUI for single station**
- **Repeater acts as amplifier and retransmits analog signals**
- **Bridge accepts entire incoming frame and retransmits**
  - **Doesn't forward collisions**
  - **Avoids collisions on destination segments**
- **Filtering bridge forwards frames only as needed**
  - **Allows simultaneous use of LAN segments for local transmission**
  - **Forwards all broadcast and multicast packets**
- **Switches provide full LAN speed to each port by simulating separate LAN segments**

Chapter 11 Long-Distance Digital Connection Technologies

Introduction

- **Previous technologies cover "short" distances**
- **Can extend over short distances**
- **Need to cover longer distances - e.g., Bucknell to New York**
- **Will call this technology *WAN* - *W*ide *A*rea *N*etwork**
- **Two categories:**
  - **Long distance between networks**
  - **"Local loop"**

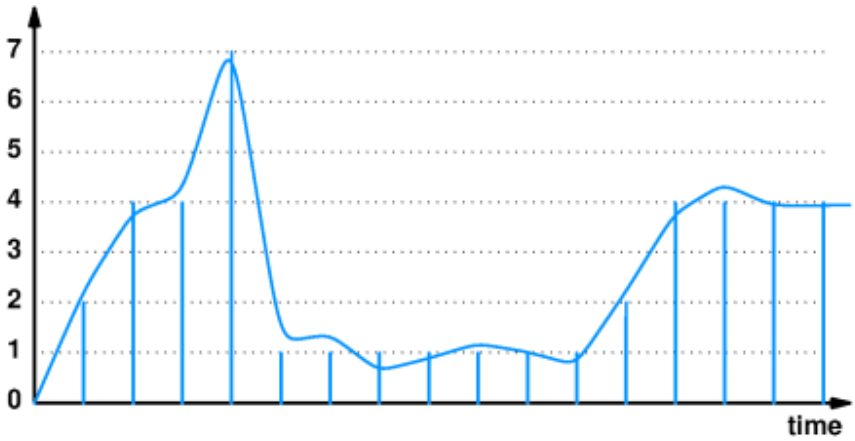
Digital telephony

- **Telephone system spans long distances**
- **Digital telephony improved long distance service:**
  - **Better quality**
  - **More connections in wire**

Digitizing voice

- **Problem: encode *analog* audio signal as *digital* data**
- **Solution:**
  - ***Sample* audio signal at periodic intervals**
  - **Convert to digital using *A-to-D converter***
  - **Send data over wire**
  - **Reconvert to audio using *D-to-A converter***

Example



Sampling parameters

- **Want to carry signals up to 4000Hz**
- **Select sample rate of 8000Hz**
- **Each sample is in range 0-255 (8 bits)**
- **Standard called *Pulse Code Modulation* (PCS**

Synchronous communication

- **Converting back to audio requires data be available "on time"**
- **Digital telephony systems use *clocking* for *synchronous* data delivery**
- **Samples not delayed as traffic increases**

Using digital telephony for data delivery

- **So, digital telephony can handle synchronous data delivery**
- **Can we use that for data delivery?**
- **Ethernet frame != 8-bit PCM synchronous**
- **Need to convert formats...**

Conversion for digital circuits

- **To use digital telephony for data delivery:**
  - **Lease *point-to-point digital circuit* between sites**
  - **Convert between local and PCM formats at each end**
- **Use a *Data Service Unit/Channel Service Unit* (DSU/CSU) at each end**
  - **CSU - manages control functions**
  - **DSU - converts data**

Using DSU/CSU

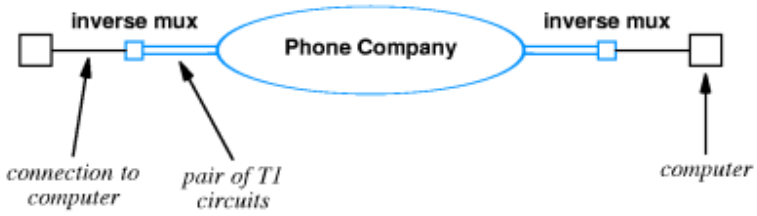
Telephone standards

- **Several standards exist for data transmission rates**
- **Called *T-series standards***

Name	Bit Rate	Voice Circuits
-	0.064 Mbps	1
T1	1.544 Mbps	24
T2	6.312 Mbps	96
T3	44.736 Mbps	672

Intermediate capacity

- **Price does not go up linearly with speed**
- **$$ for T3 < $$ for 28 * T1**
  ...however, if all you need is 9 Mbps, $$ for T3 > $$ for 6 * T1
- **Solution: combine multiple T1 lines with *inverse multiplexor***



Higher capacity circuits

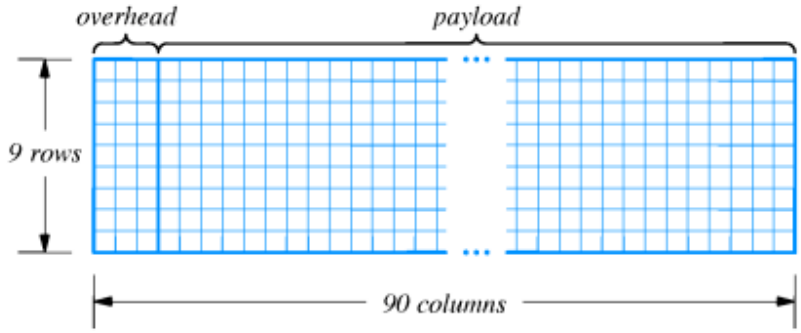
Standard name	Optical name	Bit rate	Voice circuits
STS-1	**OC-1**	**51.840 Mbps**	**810**
STS-3	**OC-3**	**155.520 Mbps**	**2,430**
STS-12	**OC-12**	**622.080 Mbps**	**9,720**
STS-24	**OC-24**	**1,244.160 Mbps**	**19,440**
STS-48	**OC-48**	**2,488.320 Mbps**	**38,880**

About the terminology

- **T-standards define underlying bit rate, *Digital Signal Level standards* (DS standards) define:**
  - **how to multiplex calls**
  - **effective bit rates**
  - **T1 line transmit data at DS-1 rate**
- ***Synchronous Transport Signal* (STS) standards define high speed connections over copper, *Optical Carrier* (OC) standard are for fiber**
- ***C* suffix indicates *concatenated*:**
  - **OC-3 == three OC-1 circuits at 51.84 Mbps**
  - **OC-3C == one 155.52 Mbps circuit**

SONET

- ***Synchronous Optical Network* (SONET) defines how to use high-speed connections**
  - **Framing: STS-1 uses 810 bytes per frame**
  - **Encoding: Each sample travels as one octet in payload**



Getting to your home

- ***Local loop*** **describes connection from telephone office to your home**
- **Sometimes called *POTS* (Plain Old Telephone Service)**
- **Legacy infrastructure is copper**
- **Other available connections include cable TV, wireless, electric power**

ISDN

- **Provides digital service (like T-series) on existing local loop copper**
- **Three separate circuits, or *channels***
  - **Two *B* channels, 64 Kbps each; == 2 voice circuits**
  - **One *D* channel, 16 Kbps; control**
- **Often written as *2B+D*; called *Basic Rate Interface* (BRI)**
- **Slow to catch on**
  - **Expensive**
  - **Charged by time used**
  - **(Almost) equaled by analog modems**

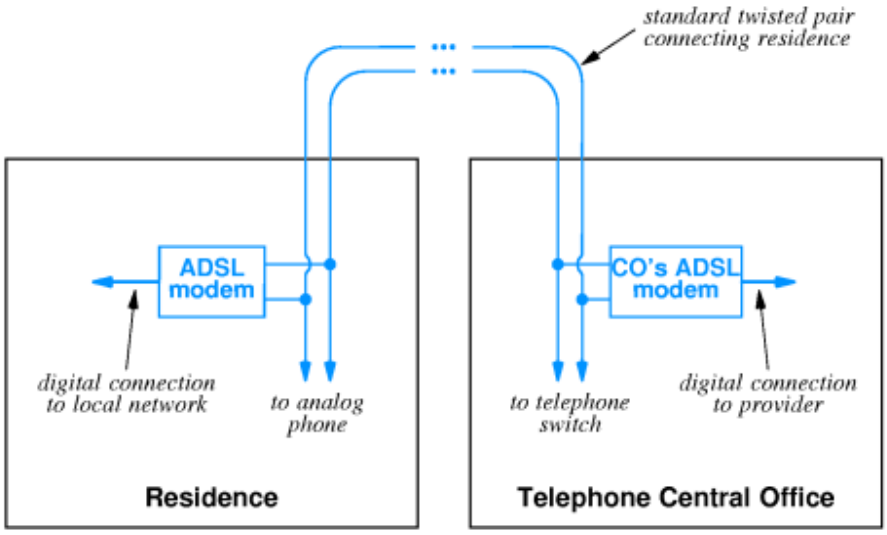
DSL

- ***DSL*** **(Digital Subscriber Line) is a family of technologies**
  - **Sometimes called *xDSL***
  - **Provides high-speed digital service over existing local loop**
- **One common form is *ADSL* (Asymmetric DSL)**
  - **Higher speed into home than out of home**
  - **More bits flow in ("downstream") than out ("upstream")**
- **ADSL maximum speeds:**

- **6.144 Mbps downstream**
- **640 Kbps upstream**

ADSL technology

- **Uses existing local loop copper**
- **Takes advantage of higher frequencies on most local loops**
- **Can be used simultaneously for POTS**



Adaptive transmission

- **Individual local loops have different transmission characteristics**
  - **Different maximum frequencies**
  - **Different interference frequencies**
- **ADSL uses FDM**
  - **286 frequencies**
    - **255 downstream**
    - **31 upstream**
    - **2 control**
- **Each frequency carries data independently**
  - **All frequencies out of audio range**
  - **Bit rate adapts to quality in each frequency**

Other DSL technologies

- ***SDSL* (Symmetric DSL) provides divides frequencies evenly**
- ***HDSL* (High-rate DSL) provides DS1 bit rate both directions**
  - **Short distances**
  - **Four wires**
- ***VDSL*(Very high bit rate DSL) provides up to 52 Mbps**
  - **Very short distance**
  - **Requires *Optical Network Unit* (ONU) as a relay**

Cable modem technologies

- **Cable TV already brings high bandwidth coax into your house**
- ***Cable modems* encode and decode data from cable TV coax**
  - **One in cable TV center connects to network**
  - **One in home connects to computer**

Features of cable modems

- **Bandwidth multiplexed among all users**
- **Multiple access medium; your neighbor can see your data!**
- **Not all cable TV coax plants are bi-directional**

Upstream communication

- **Cable TV is one direction**
  - **Signal broadcast at central location**
  - **Amplifiers boost signal through network**
- **Amplifiers are unidirectional!**
- **Solutions:**
  - **Retrofit bi-directional amplifiers**
  - **Alternate upstream path - e.g., dialup**

Alternatives

- **Satellite**
- **"Fiber to the curb'**

Summary

- **WAN links between sites use digital telephony**
  - **Based on digitized voice service**
  - **Several standard rates**
  - **Requires conversion vis DSU/CSU**
- **Local loop technologies**
  - **ISDN**
  - **xDSL**
  - **Cable modem**
  - **Satellite**
  - **Fiber to the curb**

Chapter 12 - WAN Technologies and Routing

Introduction

- **LANs can be extended using techniques in previous chapter**
- **Can not be extended arbitrarily far or to handle arbitrarily many computers**
  - **Distance limitations even with extensions**
  - **Broadcast a problem**
- **Need other technologies for larger networks**

Characterizations of networks

- ***Local Area Network* (WAN) - single building**
- ***Metropolitan Area Network* (MAN) - single city**
- ***Wide Area network* (WAN) - country, continent, planet**

Differences between LAN and WAN

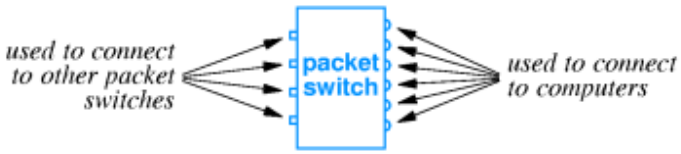
- **Satellite bridge can extend LAN across large distances**
- **Still cannot accommodate arbitrarily many computers**
- **WAN must be *scalable* to long distances** and **many computers**

Packet switches

- **To span long distances or many computers, network must replace *shared medium* with *packet switches***
  - **Each switch moves an *entire packet* from one connection to another**
  - **A small computer with network interfaces, memory and program dedicated to packet switching function**

Connections to packet switches

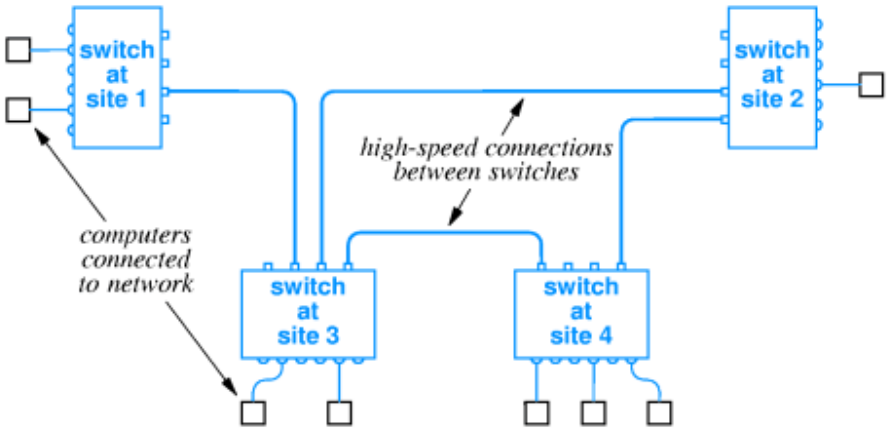
- **Packets switches may connect to computers and to other packet switches**



- **Typically high speed connections to other packets switches, lower speed to computers**
- **Technology details depend on desired speed**

Packet switches as building blocks

- **Packet switches can be linked together to form WANs**



- **WANs need not be symmetric or have regular connections**
- **Each switch may connect to one or more other switches and one or more computers**

Store and forward

- **Data delivery from one computer to another is accomplished through *store-and-forward* technology**
  - **Packet switch *stores* incoming packet**
  - **... and *forwards* the packet to another switch or computer**
- **Packet switch has internal memory**
  - **Can hold packet if outgoing connection is busy**
  - **Packets for each connection held on queue**

Store and forward example

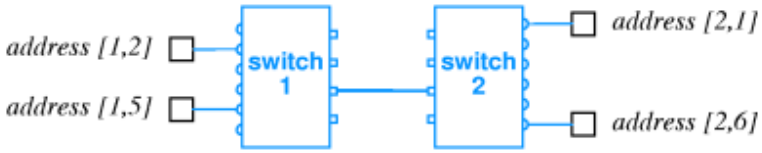
---

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap12/chap12_8.html

Shockwave

Physical addressing in a WAN

- **Similar to LAN**
  - **Data transmitted in *packets* (equivalent to frames)**
  - **Each packet has format with header**
  - **Packet header includes destination and source addresses**
- **Many WANs use *hierarchical addressing* for efficiency**
  - **One part of address identifies destination switch**
  - **Other part of address identifies port on switch**

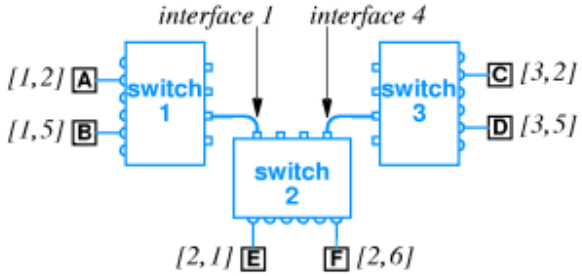


Next-hop forwarding

- **Packet switch must choose outgoing connection for forwarding**
  - **If destination is local computer, packet switch delivers computer port**
  - **If destination is attached another switch, this packet switch forwards to *next hop* through connection to another switch**
- **Choice based on destination address in packet**

Choosing next hop

- **Packet switch doesn't keep complete information about all possible destination**
- **Just keeps next hop**
- **So, for each packet, packet switch looks up destination in table and forwards through connection to next hop**

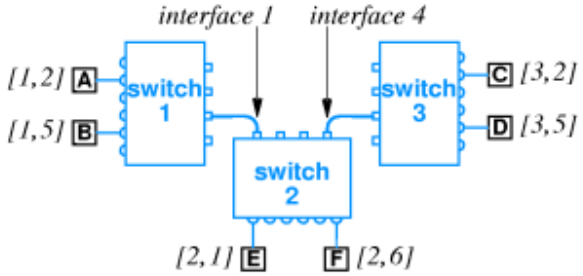


Source independence

- **Next hop to destination does not depend on source of packet**
- **Called *source independence***
- **Allows fast, efficient routing**
- **Packet switch need not have complete information, just next hop**
  - **Reduces total information**
  - **Increases dynamic robustness - network can continue to function even if topology changes *without* notifying entire network**

Hierarchical address and routing

- **Process of forwarding is called *routing***
- **Information is kept in *routing table***
- **Note that many entries have same next hop**



(a)

destination	next hop
[1,2]	interface 1
[1,5]	interface 1
[3,2]	interface 4
[3,5]	interface 4
[2,1]	computer E
[2,6]	computer F

(b)

- **In particular, all destinations on same switch have same next hop**
- **Thus, routing table can be collapsed:**

Destination	Next Hop
(1, anything)	interface 1
(3, anything)	interface 4
(2, anything)	local computer

WAN architecture and capacity

- **More computers == more traffic**
- **Can add capacity to WAN by adding more links and packet switches**
- **Packet switches need not have computers attached**
- ***Interior switch* - no attached computers**
- ***Exterior switch* - attached computers**

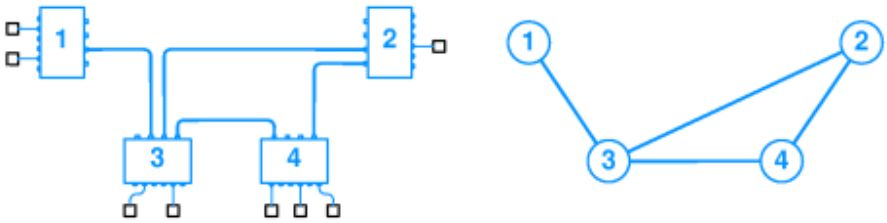
Routing in a WAN

- **Both interior and exterior switches:**
  - **Forward packets**
  - **Need routing tables**
- **Must have:**
  - ***Universal routing* - next hop for each possible destination**

- ***Optimal routes* - next hop in table must be on shortest path to destination**

Modeling a WAN

- **Use a graph:**
  - **Nodes model switches**
  - **Edges model direct connections between switches**
- **Captures essence of network, ignoring attached computers**



Route computation with a graph

- **Can represent routing table with edges:**

destin-ation	next hop
1	-
2	(1,3)
3	(1,3)
4	(1,3)

node 1

destin-ation	next hop
1	(2,3)
2	-
3	(2,3)
4	(2,4)

node 2

destin-ation	next hop
1	(3,1)
2	(3,2)
3	-
4	(3,4)

node 3

destin-ation	next hop
1	(4,3)
2	(4,2)
3	(4,3)
4	-

node 4

- **Graph algorithms can be applied to find routes**

Redundant routing information

- **Notice duplication of information in routing table for node 1:**

destin-ation	next hop
1	-
2	(1,3)
3	(1,3)
4	(1,3)

node 1

destin-ation	next hop
1	(2,3)
2	-
3	(2,3)
4	(2,4)

node 2

destin-ation	next hop
1	(3,1)
2	(3,2)
3	-
4	(3,4)

node 3

destin-ation	next hop
1	(4,3)
2	(4,2)
3	(4,3)
4	-

node 4

- **Switch 1 has only one outgoing connection; all traffic must traverse that connection**
- **Extrapolate to Bucknell and Internet**

Default routes

- **Can collapse routing table entries with a *default route***
- **If destination does not have an explicit routing table entry, use use the default route:**

destin-ation	next hop
1	-
*	(1,3)

node 1

destin-ation	next hop
2	-
4	(2,4)
*	(2,3)

node 2

destin-ation	next hop
1	(3,1)
2	(3,2)
3	-
4	(3,4)

node 3

destin-ation	next hop
2	(4,2)
4	-
*	(4,3)

node 4

- **Use of default route is optional (see node 3)**
- **Extrapolate to Bucknell**

Building routing tables

- **How to enter information into routing tables:**
  - ***Manual entry* - initialization file**
  - ***Dynamically* - through runtime interface**

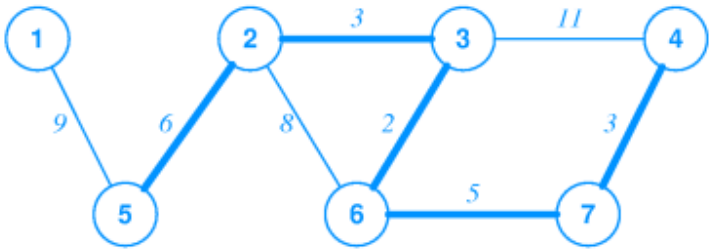
- **How to compute routing table information:**
  - ***Static routing* - at boot time**
  - ***Dynamic routing* - allow automatic updates by a program**
- **Static is simpler; doesn't accommodate changes to network topology**
- **Dynamic requires additional protocol(s); can work around network failures**

Computation of shortest path in a graph

- **Assume graph representation of network at each node**
- **Use *Djikstra's algorithm* to compute shortest path from each node to every other node**
- **Extract next-hop information from resulting path information**
- **Insert next-hop information into routing tables**

Weighted graph

- **Djikstra's algorithm can accommodate *weights* on edges in graph**
- **Shortest path is then the path with lowest total weight (sum of weights of all edges)**
- **Shortest path not necessarily fewest edges (or hops)**



Synopsis of Djikstra's algorithm

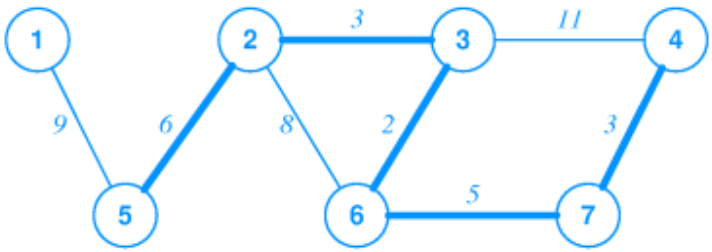
- **Keep data structure with list of nodes and weights of paths to those nodes**
- **Use *infinity* to represent a node in the set *S* of nodes for which a path has not yet been computed**
- **At each iteration, find a node in *S*, compute the path to that node, and delete the node from *S***

Synopsis of Djikstra's algorithm

- **Keep data structure with list of nodes and weights of paths to those nodes**
- **Use *infinity* to represent a node in the set *S* of nodes for which a path has not yet been computed**
- **At each iteration, find a node in *S*, compute the path to that node, and delete the node from *S***

Distance metrics

- **Weights on graph edges reflect "cost" of traversing edge**
  - **Time**
  - **Dollars**
  - **Hop count (weight == 1)**
- **Resulting shortest path may not have fewest hops**



Dynamic route computation

- **Network topology may change dynamically**
  - **Switches may be added**
  - **Connections may fail**
  - **Costs for connections may change**
- **Switches must update routing tables based on topology changes**

Distributed route computation

- **Pass information about network topology between nodes**
- **Update information periodically**

- **Each node recomputes shortest paths and next hops**
- **Inject changes into routing tables**

Vector-distance algorithm

- **Local information is next-hop routing table and distance from each switch**
- **Switches periodically broadcast topology information**
- **Other switches update routing table based on received information**

Vector-distance algorithm (continued)

- **In more detail:**
- Wait for next update message
- Iterate through entries in message
- If entry has shorter path to destination:
-
- Insert source as next hop to destination
- Record distance as distance from next hop to destination PLUS
- Distance from this switch to next hop

Link-state routing

- **Separates network topology from route computation**
- **Switches send *link-state* information about local connections**
- **Each switch builds own routing tables**
  - **Uses link-state information to update global topology**
  - **Runs Djikstra's algorithm**

Comparison

- **Vector-distance algorithm**
  - **Very simple to implement**
  - **May have convergence problems**
  - **Used in *RIP***
- **Link-state algorithm**
  - **Much more complex**
  - **Switches perform independent computations**
  - **Used in *OSPF***

Examples of WAN technology

- **ARPANET**
  - **Began in 1960s**
  - **Funded by *Advanced Research Projects Agency*, an organization of the US Defense Department**
  - **Incubator for many of current ideas, algorithms and internet technologies**
  - **See *Where Wizards Stay Up Late***
- **X.25**
  - **Early standard for connection-oriented networking**
  - **From *ITU*, which was originally *CCITT***
  - **Predates computer connections, used for terminal/timesharing connection**
- **Frame Relay**
  - **Telco service for delivering blocks of data**
  - **Connection-based service; must contract with telco for *circuit* between two endpoints**
  - **Typically 56Kbps or 1.5Mbps; can run to 100Mbps**
- **SMDS - *Switched Multi-megabit Data Service***
  - **Also a Telco service**
  - **Connectionless service; any SMDS station can send a frame to any other station on the same SMDS "cloud"**
  - **Typically 1.5-100Mbps**
- **ATM - *Asynchronous Transfer Mode***
  - **Designed as single technology for voice, video, data, ...**
  - **Low *jitter* (variance in delivery time) and high capacity**
  - **Uses fixed size, small *cells* - 48 octets data, 5 octets header**
  - **Can connect multiple ATM switches into a network**

Summary

- **WAN can span arbitrary distances and interconnect arbitrarily many computers**
- **Uses packet switches and point-to-point connections**
- **Packets switches use store-and-forward and routing tables to deliver packets to destination**
- **WANs use hierarchical addressing**
- **Graph algorithms can be used to compute routing tables**
- **Many LAN technologies exist**

Chapter 14 - Protocols and Layering

Introduction

- **LAN/WAN hardware can't solve all computer communication problems**
- **Software for LAN and WAN systems is large and complicated**
- ***Layering* is a structuring technique to organize networking software design and implementation**

Why network software?

- **Sending data through raw hardware is awkward and inconvenient - doesn't match programming paradigms well**
- **Equivalent to accessing files by making calls to disk controller to position read/write head and accessing individual sectors**
- **May not be able to send data to every destination of interest without other assistance**
- **Network software provides high-level interface to applications**

Why protocols?

- **Name is derived from the Greek *protokollen*, the index to a scroll**
- **Diplomats use rules, called protocols, as guides to formal interactions**
- **A *network protocol* or *computer communication protocol* is a set of rules that specify the format and meaning of messages exchanged between computers across a network**
  - **Format is sometimes called *syntax***
  - **Meaning is sometimes called *semantics***
- **Protocols are implemented by *protocol software***

One or many protocols?

- **Computer communication across a network is a very hard problem**
- **Complexity requires multiple protocols, each of which manages a part of the problem**
- **May be simple or complex; must all work together**

Protocol suites

- **A set of related protocols that are designed for compatibility is called a *protocol suite***

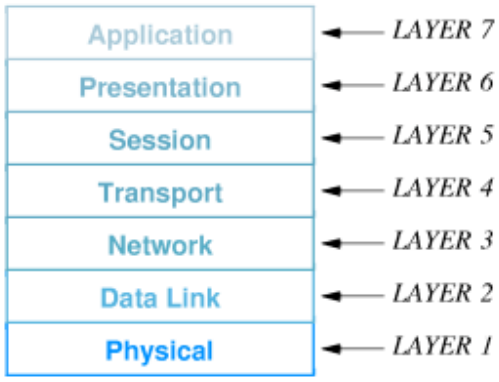
- **Protocol suite designers:**
  - **Analyze communication problem**
  - **Divide problems into subproblems**
  - **Design a protocol for each subproblem**
- **A well-designed protocol suite**
  - **Is efficient and effective - solves the problem without redundancy and makes best use of network capacity**
  - **Allows replacement of individual protocols without changes to other protocols**

Layered protocol design

- ***Layering model* is a solution to the problem of complexity in network protocols**
- **Model suggests dividing the network protocol into *layers*, each of which solves part of the network communication problem**
- **These layers have several constraints, which ease the design problem**
- **Network protocol designed to have a protocol or protocols for each layer**

The ISO 7-layer reference model

- ***International Organization for Standards* (ISO) defined a *7-layer reference model* as a guide to the design of a network protocol suite**



- **Layers are named and numbered; reference to ``layer *n*'' often means the n^{th} layer of the ISO 7-layer reference model**

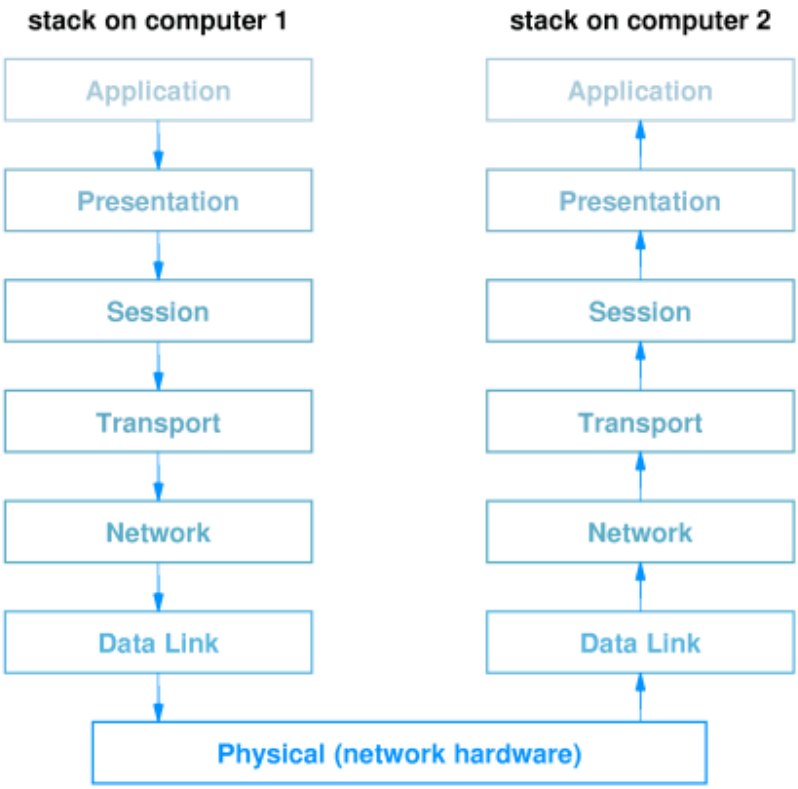
The layers in the ISO model

- **Caveat - many modern protocols do not exactly fit the ISO model, and the ISO protocol suite is mostly of historic interest**
- **Concepts are still largely useful and terminology persists**

Layer 7: Application
Application-specific protocols such as FTP and SMTP (electronic mail)
Layer 6: Presentation
Common formats for representation of data
Layer 5: Session
Management of sessions such as login to a remote computer
Layer 4: Transport
Reliable delivery of data between computers
Layer 3: Network
Address assignment and data delivery across a physical network
Layer 2: Data Link
Format of data in frames and delivery of frames through network interface
Layer 1: Physical
Basic network hardware - such as RS-232 or Ethernet

Layered software implementation

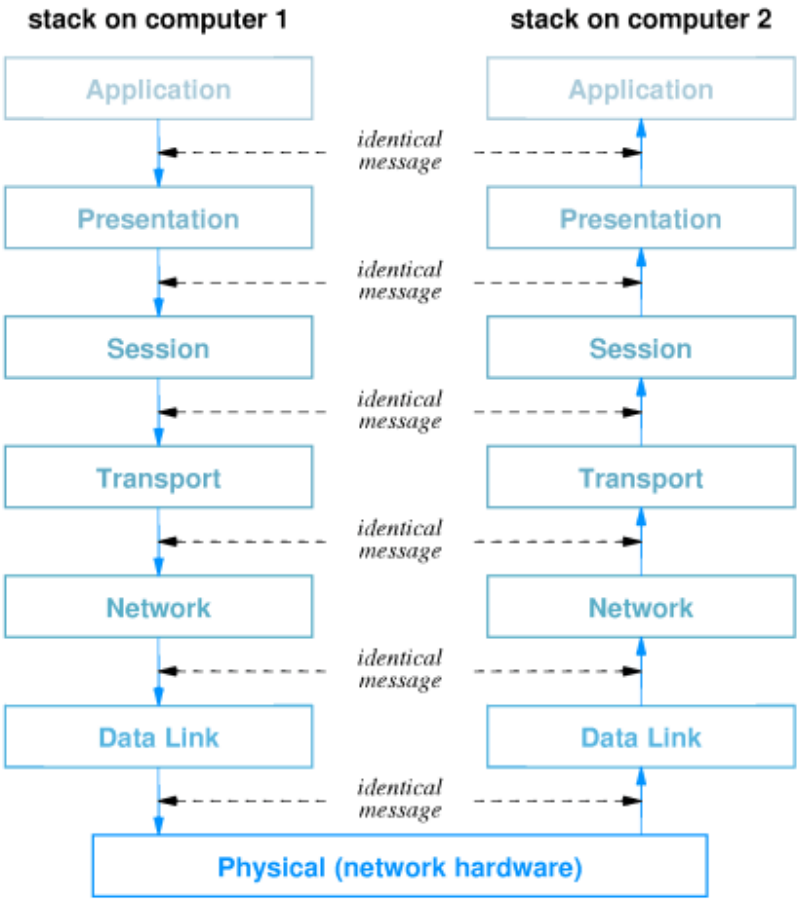
- **Software implemented from layered design has layered organization**
- **Software modules can be viewed as:**



Layered software and stacks

- **Related modules from previous figure are called a *protocol stack* or simply a *stack***
- **Two constraints:**
  - **The software for each layer depends only on the services of the software provided by lower layers**
  - **The software at layer *n* at the destination receives exactly the same protocol message sent by layer *n* at the sender**
- **These constraints mean that protocols can be tested independently and can be replaced within a protocol stack**

Layering principle



Messages and protocol stacks

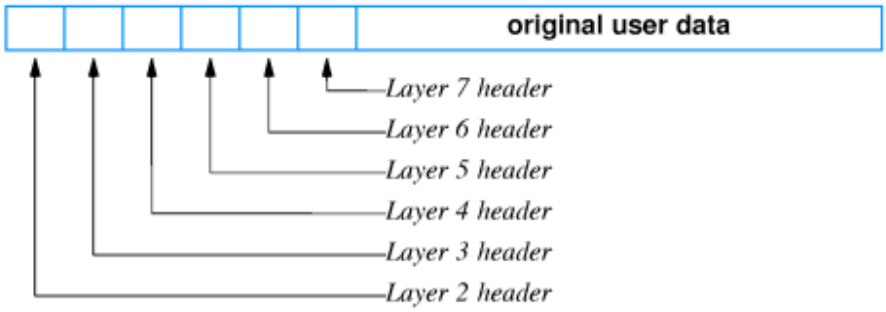
- **On the sender, each layer:**
  - **Accepts an outgoing message from the layer above**
  - **Adds a header and other processing**
  - **Passes resulting message to next lower layer**
- **On the receiver, each layer:**
  - **Receives an incoming message from the layer below**
  - **Removes the header for that layer and performs other processing**
  - **Passes the resulting message to the next higher layer**

Commercial stacks

Vendor	Stack
Novell Corporation	Netware
Banyan System Corporation	VINES
Apple Computer Corporation	AppleTalk
Digital Equipment Corporation	DECNET
IBM	SNA
(many vendors)	TCP/IP

Protocol headers

- **The software at each layer communicates with the corresponding layer through information stored in headers**
- **Each layer adds its header to the front of the message from the next higher layer**
- **Headers are nested at the front of the message as the message traverses the network**



Control packets

- **Protocol layers often need to communicate directly without exchanging data**
  - **Acknowledge incoming data**
  - **Request next data packet**
- **Layers use *control packets***

- **Generated by layer *n* on sender**
- **Interpreted by layer *n* on receiver**
- **Transmitted like any other packet by layers *n-1* and below**

Techniques for reliable network communication

- **Model - *reliable* delivery of a block of data from one computer to another**
  - **Data values unchanged**
  - **Data in order**
  - **No missing data**
  - **No duplicated data**
- **Example - parity bit, checksum and CRC used to ensure data is unchanged**

Out-of-order delivery

- **Packets may be delivered out of order - especially in systems that include multiple networks**
- **Out of order delivery can be detected and corrected through *sequencing***
  - **Sender attaches sequence number to each outgoing packet**
  - **Received uses sequence numbers to put packets in order and detect missing packets**

Duplicate delivery

- **Packets may be duplicated during transmission**
- **Sequencing can be used to...**
  - **Detect duplicate packets with duplicated sequence numbers**
  - **Discard those duplicate packets**

Lost packets

- **Perhaps the most widespread problem is lost packets**
- **Any error - bit error, incorrect length - causes receiver to discard packet**
- **Tough problem to solve - how does the receiver decide when a packet has been lost?**

Retransmission

- **Protocols use *positive acknowledgment with retransmission* to detect and correct lost packets**
  - **Receiver sends short message acknowledging receipt of packets**
  - **Sender infers lost packets from missing acknowledgments**
  - **Sender *retransmits* lost packets**
- **Sender sets timer for each outgoing packet**
  - **Saves copy of packet**
  - **If timer expires before acknowledgment is received, sender can retransmit saved copy**
- **Protocols define upper bound on retransmission to detect unrecoverable network failure**

Replay

- **Sufficiently delayed packets may be inserted into later sessions**
- **Suppose two computers exchange data with packets numbered 1 to 5**
- **Packet 4 encounters an extraordinary delay; computers use retransmission to deliver valid copy of packet 4**
- **Two computers exchange data later on with packets numbered 1 to 10**
- **Initial `packet 4' can arrive during second session, so that the data from that old packet (rather than the current `packet 4' is inserted into the data**
- **Protocols attach *session number* to each packet in a protocol session to differentiate packets from different sessions**

Flow control

- ***Data overrun* can occur when sender transmits data faster than receiver can process incoming data**
- **Protocols use *flow control* mechanisms through which receiver can control rate of data transmission**
  - ***Stop-and-go***
  - ***Sliding window***

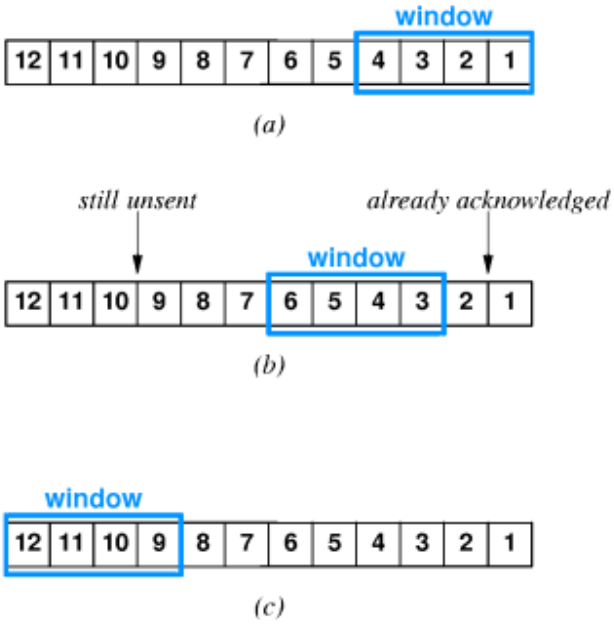
Stop-and-go flow control

- **Receiver sends small control packet when it is ready for next packet**
- **Sender waits for control packet before sending next packet**
- **Can be very inefficient of network bandwidth if delivery time is large**

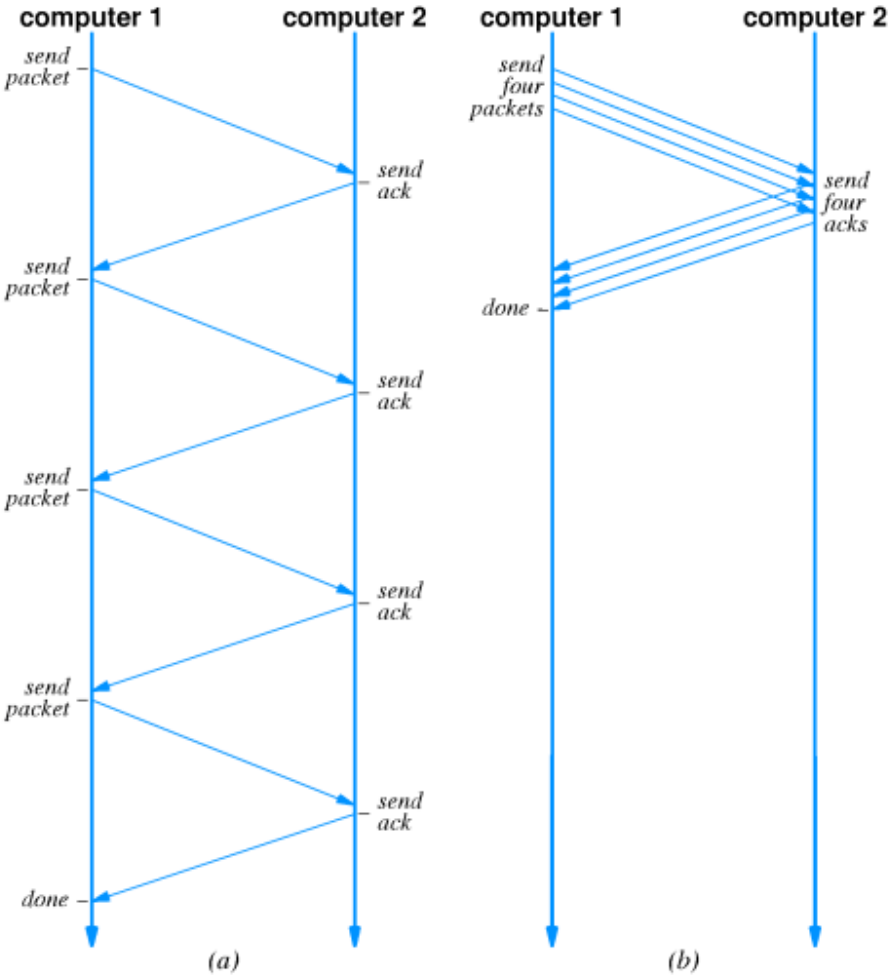
Sliding window

- **Allows sender to transmit multiple packets before receiving an acknowledgment**
- **Number of packets that can be sent is defined by the protocol and called the *window***
- **As acknowledgments arrive from the receiver, the window is moved along the data packets; hence ``sliding window''**

Example of sliding window



Comparison of stop-and-go and sliding window



Transmission times

- **For stop-and-go, each packet takes *2L* time to deliver (where *L* is the *latency*, or network delivery time)**
- **Sliding window can improve by number of packets in window:**

$$T_w = T_g * W$$

(T_w is sliding window throughput, T_g is stop-and-go throughput

- **Transmission time also limited by network transmission rate:**

$$T_w = \min(B, T_g * W)$$

(B is maximum network bandwidth)

Network congestion

- **Network congestion arises in network systems that include multiple links**
- **If input to some link exceeds maximum bandwidth, packets will queue up at connection to that link**

- **Eventually, packets will be discarded and packets will be retransmitted**
- **Ultimately, network will experience *congestion collapse***
- **Problem related to, but not identical to, data overrun**

Avoiding and recovering from network congestion

- **Protocols attempt to avoid congestion and recover from network collapse by monitoring the state of the network and taking appropriate action**
- **Can use two techniques:**

  - **Notification from packet switches**
  - **Infer congestion from packet loss**
- **Packet loss can be used to detect congestion because modern networks are reliable and rarely lose packets through hardware failure**
- **Sender can infer congestion from packet loss through missing acknowledgments**
- **Rate or percentage of lost packets can be used to gauge degree of congestion**

Art, engineering and protocol design

- **Protocol design mixes engineering and art**
  - **There are well-known techniques for solving specific problems**
  - **Those techniques interact in subtle ways**
  - **Resulting protocol suite must account for interaction**
- **Efficiency, effectiveness, economy must all be balanced**

Summary

- **Layering is a technique for guiding protocol design and implementation**
- **Protocols are grouped together into related protocol suites**
- **A collection of layered protocols is called a protocol stack**
- **Protocols use a variety of techniques for reliable delivery of data**

Chapter 15 - Internetworking

Motivation

- **There are many different LAN and WAN technologies**
- **In real world, computers are connected by many different technologies**
- **Any system that spans a large organization must accommodate multiple technologies**

Universal service

- **Telephones are useful because any telephone can reach any other telephone**
- ***Universal service* among computers greatly increases the usefulness of each computer**
- **Providing universal service requires interconnecting networks employing different technologies**

Internetworking

- ***Internetworking* is a scheme for interconnecting multiple networks of dissimilar technologies**
- **Uses both hardware and software**

  - **Extra hardware positioned between networks**
  - **Software on each attached computer**
- **System of interconnected networks is called an *internetwork* or an *internet***

Routers

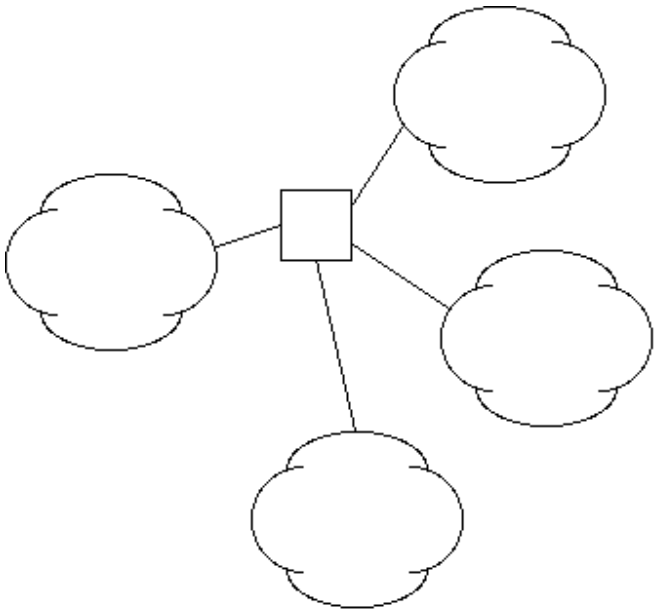
- **A *router* is a hardware component used to interconnect networks**
- **A router has interfaces on multiple networks**

- **Networks can use different technologies**
- **Router forwards packets between networks**
- **Transforms packets as necessary to meet standards for each network**

Internet architecture

- **An internetwork is composed of arbitrarily many networks interconnected by routers**

- **Routers can have more than two interfaces**

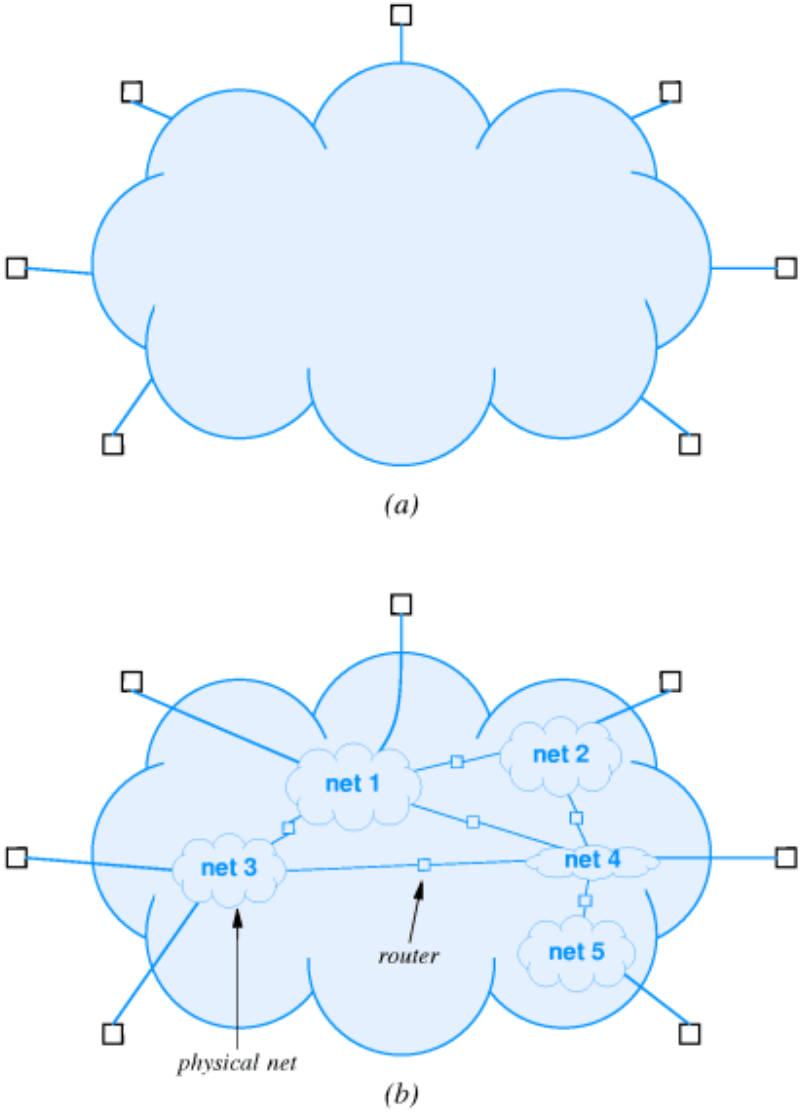


Routers in an organization

- **Would be possible to interconnect all networks in an organization with a single router**
- **Most organizations use multiple routers**
  - **Each router has finite capacity; single router would have to handle *all* traffic across entire organization**
  - **Because internetworking technology can automatically route around failed components, using multiple routers increases reliability**

A virtual network

- **Internetworking software builds a single, seamless *virtual network* out of multiple physical networks**
  - **Universal addressing scheme**
  - **Universal service**
- **All details of physical networks hidden from users and application programs**



A protocol suite for internetworking

- **The *TCP/IP Internet Protocols* or, simply, *TCP/IP* is the mostly widely used internetworking protocol suite**
- **First internetworking protocol suite**
- **Internet concept (originally called *catenet* developed in conjunction with TCP/IP**
- **Initially funded through ARPA**
- **Picked up by NSF**

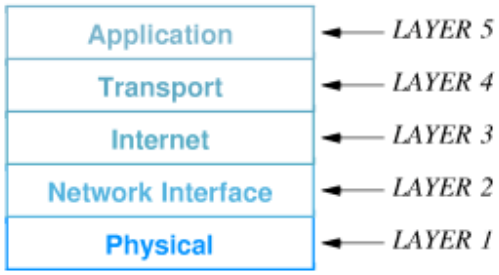
- **Described in *Where Wizards Stay Up Late***

Internetworking protocols

- **Others include IPX, VINES, AppleTalk**
- **TCP/IP is by far the most widely used**
- **Vendor and platform independent**
- **Used in the *Internet* - 20 million computers in 82 countries**

TCP/IP layering

- **OSI 7-layer model does not include internetworking**
- **TCP/IP layering model includes five layers**



Layer 5: Application
Corresponds to ISO model layers 6 and 7; used for communication among applications
Layer 4: Transport
Corresponds to layer 4 in the ISO model; provides reliable delivery of data
Layer 3: Internet
Defines uniform format of packets forwarded across networks of different technologies and rules for forwarding packets in routers
Layer 2: Network
Corresponds to layer 2 in the ISO model; defines formats for carrying packets in hardware frames
Layer 1: Hardware
Corresponds to layer 1 in the ISO model; defines basic networking hardware

Hosts, routers and protocol layers

- **A *host computer* or *host* is any system attached to an internet that runs applications**
- **Hosts may be supercomputers or toasters**
- **TCP/IP allows any pair of hosts on an internet communicate directly**
- **Both hosts and routers have TCP/IP stacks**
  - **Hosts typically have one interface and don't forward packets**
  - **Routers don't need layers 4 and 5 for packet forwarding**

Summary

- **An *internet* is a collection of physical networks interconnected into a single *virtual network***
- ***Routers* provide the physical interconnection and forward packets between networks**
- ***Hosts* communicate across multiple network through packets forwarded by routers**
- **TCP/IP is the most widely used internetworking protocol suite**

Chapter 16 - IP Protocol Addresses

Motivation

- **One key aspect of virtual network is single, uniform address format**
- **Can't use hardware addresses because different technologies have different address formats**
- **Address format must be independent of any particular hardware address format**
- **Sending host puts destination internet address in packet**
- **Destination address can be interpreted by any intermediate router**
- **Routers examine address and forward packet on to the destination**

TCP/IP addresses

- **Addressing in TCP/IP is specified by the *Internet Protocol* (IP)**
- **Each host is assigned a 32-bit number**
- **Called the *IP address* or *Internet address***
- **Unique across entire Internet**

IP address hierarchy

- **Each IP address is divided into a prefix and a suffix**
  - **Prefix identifies network to which computer is attached**
  - **Suffix identifies computer within that network**
- **Address format makes routing efficient**

Network and host numbers

- **Every network in a TCP/IP internet is assigned a unique *network number***
- **Each host on a specific network is assigned a *host number* or *host address* that is unique *within that network***
- **Host's IP address is the combination of the network number (prefix) and host address (suffix)**

Porperties of IP addresses

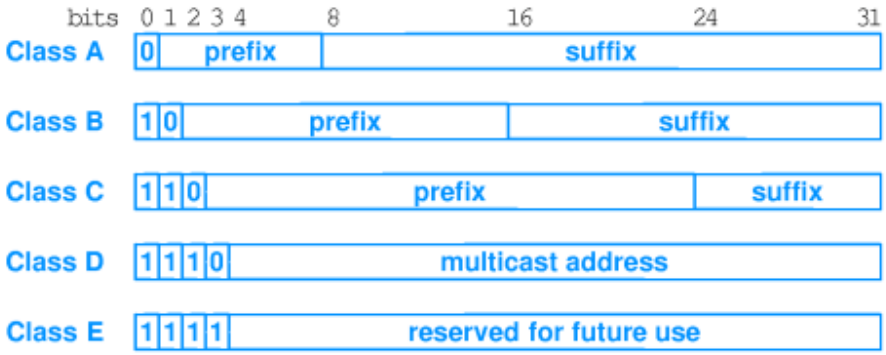
- **Network numbers are unique**
- **Host addresses may be reused on different networks; combination of network number prefix and host address suffix will be unique**
- **Assignment of network numbers must be coordinated globally; assignment of host addresses can be managed locally**

Designing the format of IP addresses

- **IP designers chose 32-bit addresses**
- **Allocate some bits for prefix, some for suffix**
  - **Large prefix, small suffix - many networks, few hosts per network**
  - **Small prefix, large suffix - few networks, many hosts per network**
- **Because of variety of technologies, need to allow for both large and small networks**

Classes of addresses

- **Designers chose a compromise - multiple address formats that allow both large and small prefixes**
- **Each format is called an address *class***
- **Class of an address is identified by first four bits**



Using IP address classes

- **Class A, B and C are *primary classes***
- **Used for ordinary host addressing**
- **Class D is used for multicast, a limited form of broadcast**
  - **Internet hosts join a *multicast group***
  - **Packets are delivered to all members of group**
  - **Routers manage delivery of single packet from source to all members of multicast group**
  - **Used for *mbone* (multicast backbone)**

- **Class E is reserved**

Dotted decimal notation

- **Class A, B and C all break between prefix and suffix on byte boundary**
- ***Dotted decimal notation* is a convention for representing 32-bit internet addresses in decimal**
- **Convert each byte of address into decimal; display separated by periods (``dots'')**

32-bit Binary Number	Equivalent Dotted Decimal
10000001 00110100 00000110 00000000	129 . 52 . 6 . 0
11000000 00000101 00110000 00000011	192 . 5 . 48 . 3
00001010 00000010 00000000 00100101	10 . 2 . 0 . 37
10000000 00001010 00000010 00000011	128 . 10 . 2 . 3
10000000 10000000 11111111 00000000	128 . 128 . 255 . 0

Bucknell's IP addresses

- **Bucknell has a single Class B network: 134.82.0.0**
- **All hosts at Bucknell have 134.82 prefix:**
  - **134.82.7.4 - coral**
  - **134.82.56.108 - leo**
  - **134.82.131.3 - charcoal**
- **Suffix bytes are used to determine local network and host through *sub-netting***

Address classes at a glance

- **While dotted decimal makes separating network address from host address easier, determining class is not so obvious**
- **Look at first dotted decimal number, and use this table:**

Class	Range of Values
A	0 through 127
B	128 through 191
C	192 through 223
D	224 through 239
E	240 through 255

Networks and hosts in each class

- **Classing scheme does not yield equal number of networks in each class**
- **Class A:**
  - **First bit must be 0**
  - **7 remaining bits identify Class A net**
  - **2^7 (= 128) possible class A nets**

Address Class	Bits In Prefix	Maximum Number of Networks	Bits In Suffix	Maximum Number Of Hosts Per Network
A	7	128	24	16777216
B	14	16384	16	65536
C	21	2097152	8	256

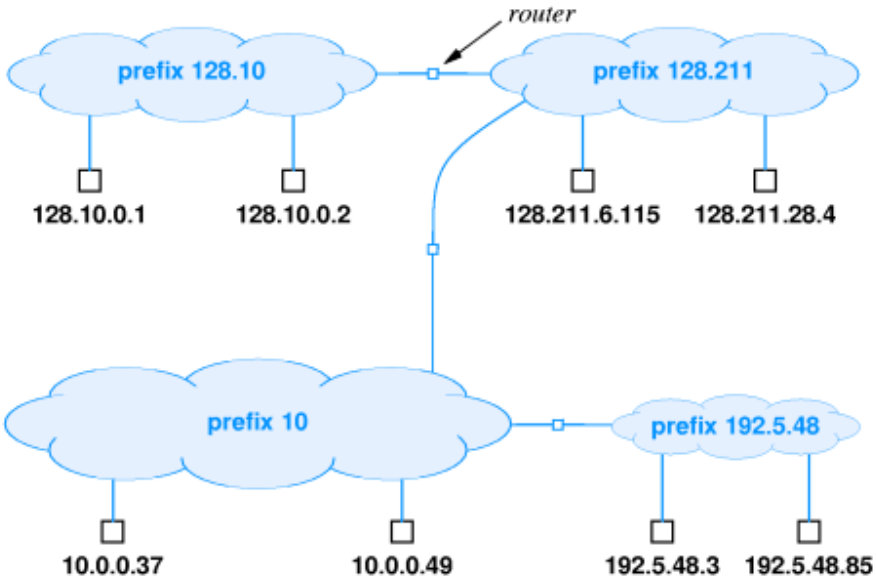
Internet address allocation

- **Addresses in the Internet are not used efficiently**
- **Bucknell is typical, using 2,000-3,000 out of possible 2^16**
- **Large organizations may not be able to get as many addresses in the Internet as they need**
- **Example - UPS needs addresses for** millions **of computers**
- **Solution - set up *private internet* and allocate addresses from entire 32-bit address space**

Example

- **Select address class for each network depending on expected number of hosts**
- **Assign network numbers from appropriate classes**

- **Assign host suffixes to form internet addresses for all hosts**



Special IP addresses

Prefix	Suffix	Type Of Address	Purpose
all-0s	all-0s	this computer	used during bootstrap
network	all-0s	network	identifies a network
network	all-1s	directed broadcast	broadcast on specified net
all-1s	all-1s	limited broadcast	broadcast on local net
127	any	loopback	testing

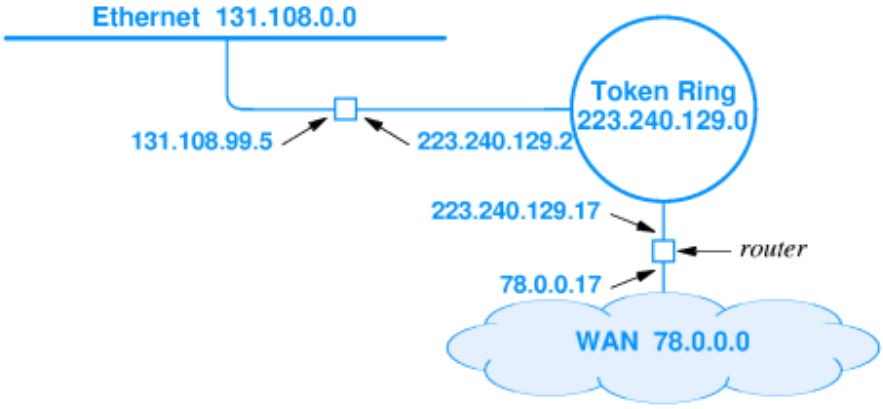
Berkeley broadcast address

- **First BSD implementation (Berkeley Software Distribution) of UNIX used all 0s for broadcast instead of all 1s**
- **This non-standard implementation spread with BSD UNIX**
- **Still in common use today**

``There are two major developments that have come out of Berkeley: BSD UNIX and LSD. This is not a coincidence.'' - *Anon.*

Routers and IP addressing

- **IP address depends on network address**
- **What about routers - connected to two networks?**
- **IP address specifies an *interface*, or network attachment point, *not* a computer**
- **Router has multiple IP addresses - one for each interface**



Multi-homed hosts

- **Hosts (that do not forward packets) can also be connected to multiple networks**
- **Can increase reliability and performance**
- **Multi-homed hosts also have one address for each interface**

Summary

- **Virtual network needs uniform addressing scheme, independent of hardware**
- **IP address is a 32-bit address; each interface gets a unique IP address**
- **IP address is composed of a network address and a host address**

- **Network addresses are divided into three primary classes: A, B and C**
- **Dotted decimal notation is a standard format for Internet addresses: 134.82.11.70**
- **Routers have multiple addresses - one for each interface**

Chapter 17 - Binding Protocol Addresses

Introduction

- **Upper levels of protocol stack use *protocol addresses***
- **Network hardware must use *hardware address* for eventual delivery**
- **Protocol address must be translated into hardware address for delivery; will discuss three methods**

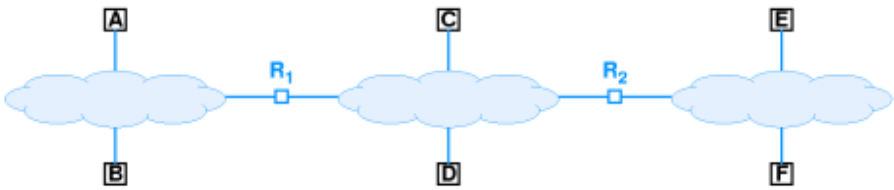
Address translation

- **Upper levels use only protocol addresses**
  - **"Virtual network" addressing scheme**
  - **Hides hardware details**
- **Translation occurs at *data link* layer**
  - **Upper layer hands down protocol address of destination**
  - **Data link layer translates into hardware address for use by hardware layer**

Address resolution

- **Finding hardware address for protocol address:**
  - ***address resolution***
  - **Data link layer *resolves* protocol address to hardware address**
- **Resolution is local to a network**
- **Network component only resolves address for other components on same network**

Address resolution (continued)



- ***A* resolves protocol address for *B* for protocol messages from an application on *A* sent to an application on *B***
- ***A* does *not* resolve a protocol address for *F***
  - **Through the internet layer, *A* delivers to *F* by routing through R_1 and R_2**
  - ***A* resolves R_1 hardware address**
- **Network layer on *A* passes packet containing destination protocol address *F* for delivery to R_1**

Address resolution techniques

- **Association between a protocol address and a hardware address is called a *binding***
- **Three techniques:**
  - **Table lookup**
    - **Bindings stored in memory with protocol address as key**
    - **Data link layer looks up protocol address to find hardware address**
  - **Closed-form computation**
    - **Protocol address based on hardware address**
    - **Data link layer derives hardware address from protocol address**
  - **Dynamic**
    - **Network messages used for "just-in-time" resolution**
    - **Data link layer sends message requesting hardware address; destination responds with its hardware address**

Table lookup

- **Use a simple list containing IP address and hardware address for each host on net**
- **Search on IP address and extract corresponding hardware address**

IP Address	Hardware Address
197.15.3.2	0A:07:4B:12:82:36
197.15.3.3	0A:9C:28:71:32:8D
197.15.3.4	0A:11:C3:68:01:99
197.15.3.5	0A:74:59:32:CC:1F
197.15.3.6	0A:04:BC:00:03:28
197.15.3.7	0A:77:81:0E:52:FA

- **Note that all IP addresses have same prefix; can save space by dropping prefix**

Table lookup (continued)

- **Sequential search may be prohibitively expensive (*O(n^2)*)**
- **Can use *indexing* or *hashing* for *O(n)* lookup**
  - **Indexing - use *hostid* part of IP address as list (array) index**

  - **Hashing - use hashing function on *hostid* to generate list index**

Closed-form computation

- **If hardware technology uses small, configurable hardware address, network administrator can choose hardware address based on IP address**
- **Example - hardware uses one octet address that can be configured**
- **Simply choose hardware address to be *hostid***

- **Now, any host can determine hardware address as:**

hardware_address = ip_address & 0xff

Dynamic resolution

- **Use *network* to resolve IP addresses**
- **Message exchange with other computer(s) returns hardware address to source**
- **Two designs:**
  - ***Server-based* - computer sends message to server to resolve address**
    - **List of servers**
    - **Broadcast to locate servers**
  - ***Distributed* - all computers participate; destination provides hardware address to host**

Dynamic resolution techniques

- **Server-based - centralized, easier to manage, used on non-broadcast media (e.g., ATM)**
- **Distributed - requires no dedicated computers, no administration**

Feature	Type Of Resolution
Useful with any hardware	T
Address change affects all hosts	T
Protocol address independent of hardware address	T, D
Hardware address must be smaller than protocol address	C
Protocol address determined by hardware address	C
Requires hardware broadcast	D
Adds traffic to a network	D
Produces resolution with minimum delay	T, C
Implementation is more difficult	D

ARP

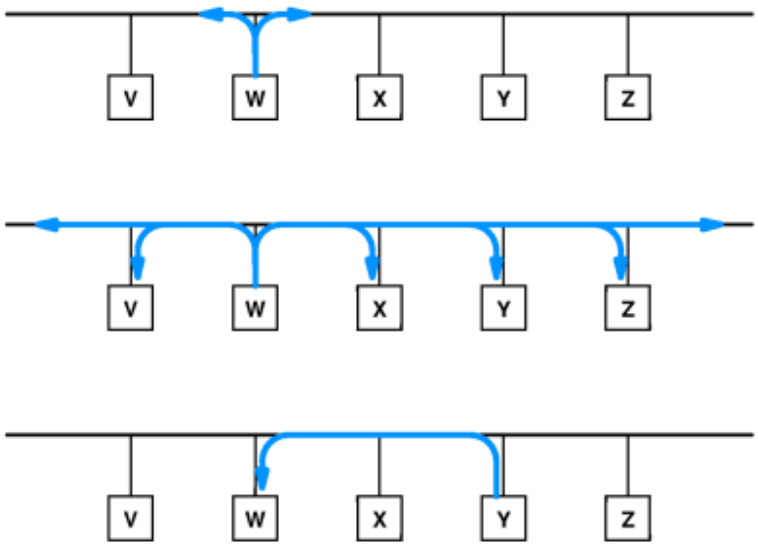
- **IP uses distributed resolution technique**
- ***Address Resolution Protocol* (ARP) - part of TCP/IP protocol suite**
- **Two-part protocol**
  - **Request from source asking for hardware address**

- **Reply from destination carrying hardware address**

ARP message exchange

- **ARP request message dropped into hardware frame and broadcast**
- **Uses separate protocol type in hardware frame (ethernet = 806)**
- **Sender inserts IP address into message and broadcast**
- **Every other computer examines request**
- **Computer whose IP address is in request responds**
  - **Puts hardware address in response**
  - **Unicasts to sender**
- **Original requester can then extract hardware address and send IP packet to destination**

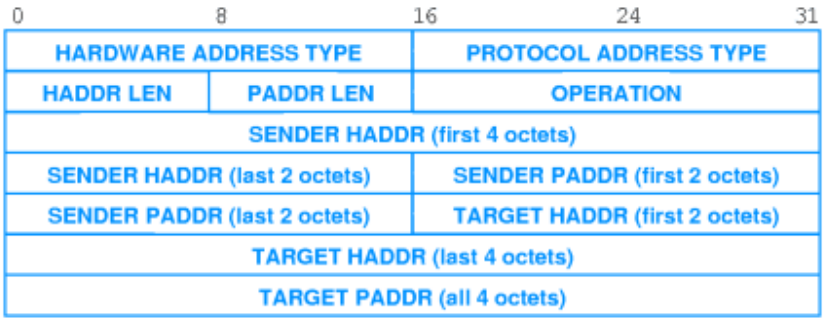
ARP example



ARP message contents

- **Maps *protocol address* to *hardware address***
- **Both protocol address and hardware address sizes are variable**
  - **Ethernet = 6 octets**
  - **IP = 4 octets**
- **Can be used for other protocols and hardware types**

ARP message format



- ***HARDWARE ADDRESS TYPE* = 1 for Ethernet**
- ***PROTOCOL ADDRESS TYPE* = 0x0800 for IP**
- ***OPERATION* = 1 for request, 2 for response**
- **Contains both *target* and *sender* mappings from protocol address to hardware address**
  - **Request sets hardware address of target to 0**
  - **Target can extract hardware address of sender (saving an ARP request)**
  - **Target exchanges sender/target in response**

Sending an ARP message

- **Sender constructs ARP message**
- **ARP message carried as data in hardware frame - *encapsulation***

Caching ARP responses

- **Using ARP for each IP packet adds two packets of overhead for each IP packet**
- **Computer caches ARP responses**
  - **Flushes cache at system startup**
  - **Entries discarded periodically**
- **Cache searched prior to sending ARP request**

Identifying ARP frames

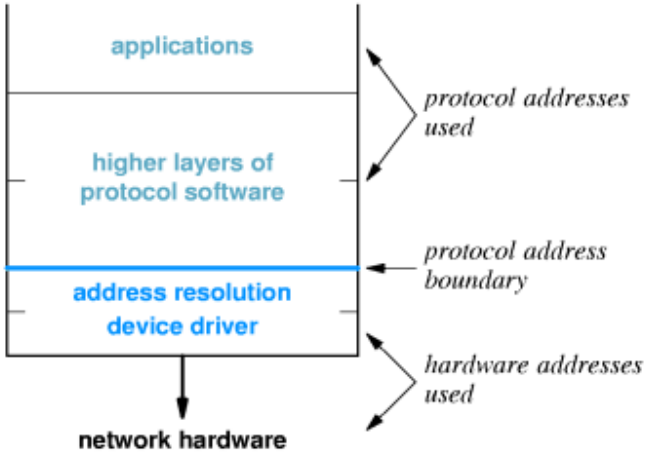
- **Uses separate frame type**
- **Ethernet uses type 0x0806**

Processing ARP messages

- **Receiver extracts sender's hardware address and updates local ARP table**
- **Receiver checks operation - request of response**
- **Response:**
  - **Adds sender's address to local cache**
  - **Sends pending IP packet(s)**
- **Request:**
  - **If receiver is target, forms response**
  - **Unicasts to sender**
  - **Adds sender's address to local cache**
- **Note:**
  - **Target likely to respond "soon"**
  - **Computers have finite storage for ARP cache**
  - **Only target *adds* sender to cache; others only update if target already in cache**

Layering and address resolution

- **Address resolution (ARP) is a *network interface* layer function**
- **Protocol addresses used in *all* higher layers**
- **Hides ugly details and allows generality in upper layers**



Summary

- **Address resolution - translates protocol address to hardware address**
  - **Static - table lookup**
  - **Computation - extract hardware address from protocol address**
  - **Dynamic - use network messages to resolve protocol address**
- **ARP - TCP/IP protocol for address resolution**

Chapter 18 - IP Datagrams and Datagram Forwarding

Introduction

- **Fundamental Internet communication service**
- **Format of packets**
- **Processing of packets by routers**
  - **Forwarding**
  - **Delivery**

Connectionless service

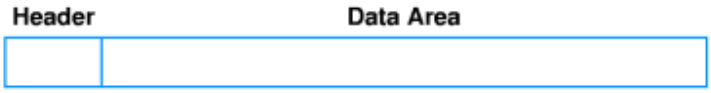
- **End-to-end delivery service is *connectionless***
- **Extension of LAN abstraction**
  - **Universal addressing**
  - **Data delivered in packets (frames), each with a header**
- **Combines collection of physical networks into single, virtual network**
- **Transport protocols use this connectionless service to provide connectionless data delivery (UDP) and connection-oriented data delivery (TCP)**

Virtual packets

- ***Packets* server same purpose in internet as frames on LAN**
- **Each has a header**
- ***Routers* (formerly *gateways*) forward between physical networks**
- **Packets have a uniform, hardware-independent format**
  - **Includes header and data**
  - **Can't use format from any particular hardware**
- **Encapsulated in hardware frames for delivery across each physical network**

IP datagram format

- **Formally, the unit of IP data delivery is called a *datagram***
- **Includes header area and data area**



- **Datagrams can have different sizes**
  - **Header area usually fixed (20 octets) but can have options**
  - **Data area can contain between 1 octet and 65,535 octets (2^{16} - 1)**
  - **Usually, data area much larger than header**

Forwarding datagrams

- **Header contains all information needed to deliver datagram to destination *computer***
  - **Destination address**
  - **Source address**
  - **Identifier**
  - **Other delivery information**
- **Router examines header of each datagram and forwards datagram along path to destination**

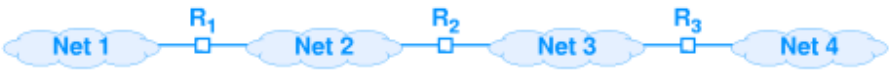
Routing example

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap18/chap18_6.html

Shockwave

Routing table

- **For efficiency, information about forwarding is stored in a *routing table***
  - **Initialized at system initialization**
  - **Must be updated as network topology changes**
- **Contains list of destination networks and *next hop* for each destination**



(a)

Destination	Next Hop
net 1	R_1
net 2	deliver direct
net 3	deliver direct
net 4	R_3

(b)

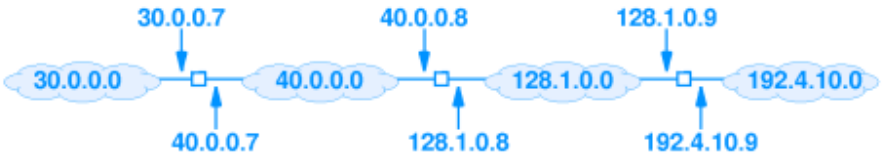
Default routes

- **Routing table kept small by listing destination networks rather than hosts**
- **Can be further reduced through *default route***
  - **Entry used if destination network not explicitly listed in routing table**
  - **E.g., Bucknell uses default routes for all off-campus networks**

Routing tables and address masks

- **In practice, additional information is kept in routing table**

- **Destination stored as *network address***
- **Next hop stored as IP address of router**
- ***Address mask* defines how many bits of address are in prefix**
  - **Prefix defines how much of address used to identify network**
  - **E.g., class A mask is 255.0.0.0**
  - **Used for subnetting**



(a)

Destination	Mask	Next Hop
30.0.0.0	255.0.0.0	40.0.0.7
40.0.0.0	255.0.0.0	deliver direct
128.1.0.0	255.255.0.0	deliver direct
192.4.10.0	255.255.255.0	128.1.0.9

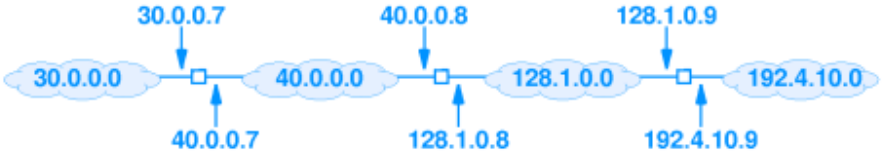
(b)

Address masks

- **To identify destination network, apply *address mask* to *destination address* and compare to *network address* in routing table**
- **Can use Boolean and**

if ((Mask[i] & D) == Dest[i]) forward to NextHop[i]

- **Consider 128.1.15.26:**



(a)

Destination	Mask	Next Hop
30.0.0.0	255.0.0.0	40.0.0.7
40.0.0.0	255.0.0.0	deliver direct
128.1.0.0	255.255.0.0	deliver direct
192.4.10.0	255.255.255.0	128.1.0.9

(b)

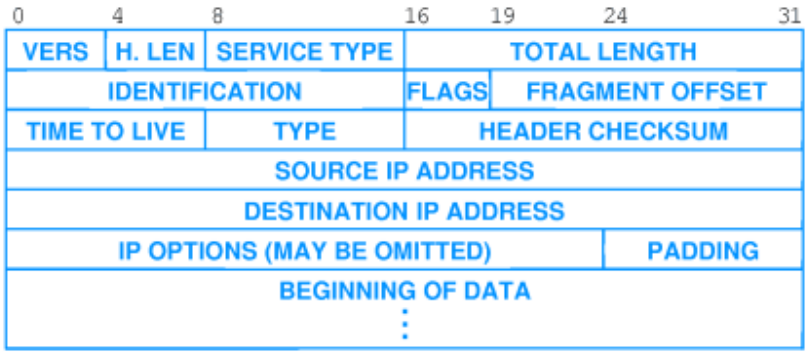
Forwarding, destination address and next-hop

- ***Destination address* in IP datagram is always ultimate destination**
- **Router looks up *next-hop address* and forwards datagram**
- ***Network interface layer* takes two parameters:**
  - **IP datagram**
  - **Next-hop address**
- **Next-hop address *never* appears in IP datagram**

Best-effort delivery

- **IP provides service equivalent to LAN**
- **Does *not* guarantee to prevent**
  - **Duplicate datagrams**
  - **Delayed or out-of-order delivery**
  - **Corruption of data**
  - **Datagram loss**
- ***Reliable delivery* provided by *transport layer***
- ***Network layer* - IP - can *detect* and *report* errors without actually *fixing* them**
  - **Network layer focuses on datagram delivery**
  - **Application layer not interested in differentiating among delivery problems at intermediate routers**

IP datagram header format



IP datagram header fields

- **VERS - version of IP (currently 4)**
- **H. LEN - header length (in units of 32 bits)**
- **SERVICE TYPE - sender's preference for low latency, high reliability (rarely used)**
- **TOTAL LENGTH - total octets in datagram**
- **IDENT, FLAGS, FRAGMENT OFFSET - used with fragmentation**
- **TTL - *time to live*; decremented in each router; datagram discarded when TTL = 0**
- **TYPE - type of protocol carried in datagram; e.g., TCP, UDP**
- **HEADER CHECKSUM - 1s complement of 1s complement sum**
- **SOURCE, DEST IP ADDRESS - IP addresses of *original* source and *ultimate* destination**

IP datagram options

- **Several options can be added to IP header:**
  - **Record route**
  - **Source route**
  - **Timestamp**
- **Header with no options has H. LEN field value 5; data begins immediately after DESTINATION IP ADDRESS**
- **Options added between DESTINATION IP ADDRESS and data in multiples of 32 bits**
- **Header with 96 bits of options has H. LEN field value 8**

Summary

- **Basic unit of delivery in TCP/IP is *IP datagram***
- **Routers use *destination address* in IP datagram header to determine *next-hop***
- **Forwarding information stored in *routing table***
- **IP datagram header has 40 octets of fixed field information and (possibly) options**

Chapter 19 - IP Encapsulation, Fragmentation and Reassembly

Datagram transmission and frames

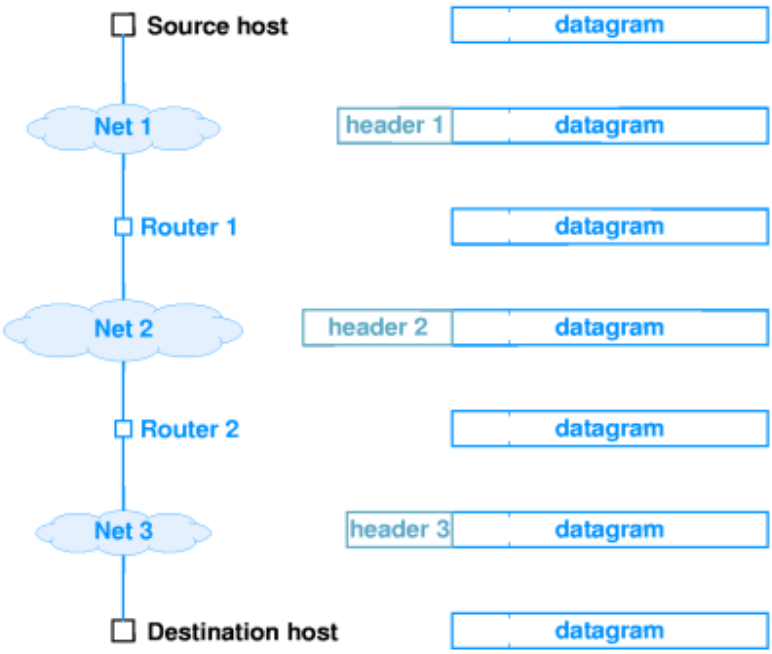
- **IP internet layer**
  - **Constructs datagram**
  - **Determines next hop**
  - **Hands to network interface layer**
- **Network interface layer**
  - **Binds next hop address to hardware address**
  - **Prepares datagram for transmission**
- **But ... hardware frame doesn't understand IP; how is datagram transmitted?**

Encapsulation

- **Network interface layer *encapsulates* IP datagram as data area in hardware frame**
  - **Hardware ignores IP datagram format**
  - **Standards for encapsulation describe details**
- **Standard defines data type for IP datagram, as well as others (e.g., ARP)**
- **Receiving protocol stack interprets data area based on frame type**

Encapsulation across multiple hops

- **Each router in the path from the source to the destination:**
  - ***Unencapsulates* incoming datagram from frame**
  - **Processes datagram - determines next hop**
  - ***Encapsulates* datagram in outgoing frame**



- **Datagram may be encapsulated in different hardware format at each hop**
- **Datagram itself is (almost!) unchanged**

Internet encapsulation (example)

---

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap19/chap19_4.html

Shockwave

MTU

- **Every hardware technology specification includes the definition of the maximum size of the frame data area**
- **Called the *maximum transmission unit* (MTU)**
- **Any datagram encapsulated in a hardware frame must be smaller than the MTU for that hardware**

MTU and datagram transmission

- **IP datagrams can be larger than most hardware MTUs**
  - **IP: $2^{16 - 1}$**
  - **Ethernet: 1500**
  - **Token ring: 2048 or 4096**
- **Source can simply limit IP datagram size to be smaller than local MTU**
  - **Must pass local MTU up to TCP for TCP segments**
  - **What about UDP?**

MTU and heterogeneous networks

- **An internet *may* have networks with different MTUs**
- **Suppose downstream network has *smaller* MTU than local network?**

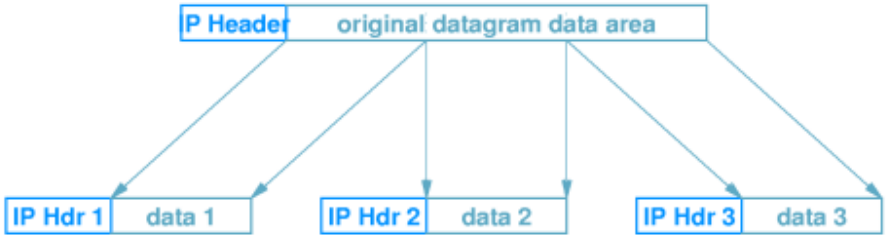
Fragmentation

- **One technique - limit datagram size to** smallest **MTU of *any* network**
- **IP uses *fragmentation* - datagrams can be split into pieces to fit in network with small MTU**
- **Router detects datagram larger than network MTU**
  - **Splits into pieces**
  - **Each piece *smaller* than outbound network MTU**

Fragmentation (details)

- **Each fragment is an independent datagram**
  - **Includes all header fields**
  - **Bit in header indicates datagram is a fragment**
  - **Other fields have information for reconstructing original datagram**
  - **FRAGMENT OFFSET gives original location of fragment**
- **Router uses local MTU to compute size of each fragment**
- **Puts part of data from original datagram in each fragment**

- **Puts other information into header**



Datagram reassembly

- **Reconstruction of original datagram is call *reassembly***
- **Ultimate destination performs reassembly**

- **Fragments may arrive out of order; header bit identifies fragment containing end of data from original datagram**
- **Fragment 3 identified as last fragment**

Fragment identification

- **How are fragments associated with original datagram?**
- **IDENT field in each fragment matches IDENT field in original datagram**
- **Fragments from different datagrams can arrive out of order and still be sorted out**

Fragment loss

- **IP may drop fragment**
- **What happens to original datagram?**
  - **Destination drops *entire* original datagram**
- **How does destination identify lost fragment?**
  - **Sets timer with each fragment**
  - **If timer expires before all fragments arrive, fragment assumed lost**
  - **Datagram dropped**
- ***Source* (application layer protocol) assumed to retransmit**

Fragmenting a fragment

- **Fragment may encounter subsequent network with even smaller MTU**
- **Router fragments the fragment to fit**
- **Resulting (sub)fragments look just like original fragments (except for size)**
- **No need to reassemble hierarchically; (sub)fragments include position in *original* datagram**

Summary

- **IP uses *encapsulation* to transmit datagrams in hardware frames**
- **Network technologies have an *MTU***
- **IP uses *fragmentation* to carry datagrams larger than network MTU**

Chapter 20 - IPv6

Section Title

Introduction - the future of IP

- **Current version of IP - version *4* - is 20 years old**
- **IPv4 has shown remarkable ability to move to new technologies**
- **IETF has proposed entirely new version to address some specific problems**

Success of IP

- **IP has accommodated dramatic changes since original design**
  - **Basic principles still appropriate today**
  - **Many new types of hardware**
  - **Scale**
- **Scaling**
  - **Size - from a few tens to a few tens of millions of computers**
  - **Speed - from 56Kbps to 1Gbps**
  - **Increased frame size in hardware**

Motivation for change

- **Address space**
  - **32 bit address space allows for over a million networks**
  - **But...most are Class C and too small for many organizations**
  - **2^{14} Class B network addresses already almost exhausted (and exhaustion was first predicted to occur a couple of years ago)**
- **Type of service**
  - **Different applications have different requirements for delivery reliability and speed**
  - **Current IP has type of service that's not often implemented**
- **Multicast**

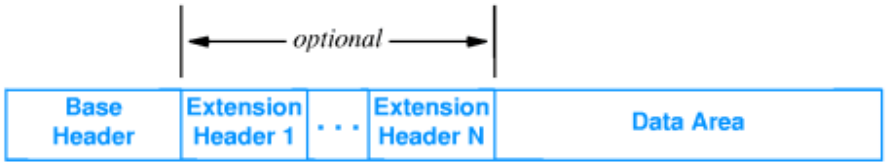
Name and versions number

- **Preliminary versions called *IP - Next Generation* (IPng)**
- **Several proposals all called IPng**
- **One was selected and uses next available version number (6)**
- **Result is *IP version 6* (IPv6)**

New features

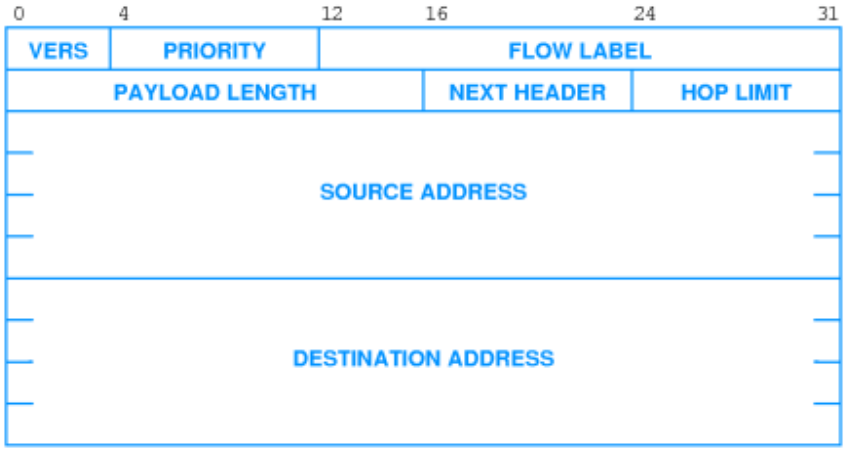
- **Address size - IPv6 addresses are 128bits**
- **Header format - entirely different**
- **Extension headers - Additional information stored in optional extension headers, followed by data**
- **Support for audio and video - flow labels and quality of service allow audio and video applications to establish appropriate connections**
- **Extensible - new features can be added more easily**

IPv6 datagram format



IPv6 base header format

- **Contains less information than IPv4 header**
- **NEXT HEADER points to first extension header**
- **FLOW LABEL used to associate datagrams belonging to a *flow* or communication between two applications**
  - ***Traffic class***
  - **Specific path**
  - **Routers use FLOW LABEL to forward datagrams along prearranged path**



IPv6 NEXT HEADER

(a)

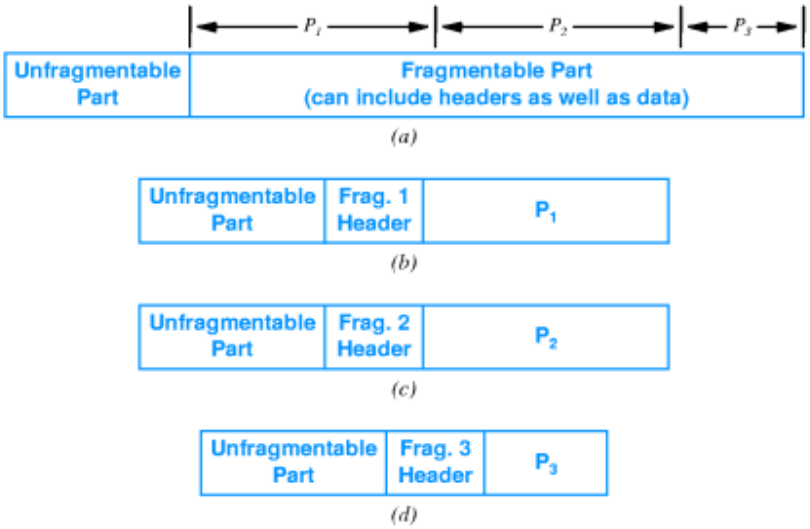
(b)

Parsing IPv6 headers

- **Base header is fixed size - 40 octets**
  - **NEXT HEADER field in base header defines type of header**
  - **Appears at end of fixed-size base header**
- **Some extensions headers are variable sized**
  - **NEXT HEADER field in extension header defines type**
  - **HEADER LEN field gives size of extension header**

Fragmentation

- **Fragmentation information kept in separate extension header**
- **Each fragment has base header and (inserted) fragmentation header**
- **Entire datagram, including original header may be fragmented**



Fragmentation and path MTU

- **IPv6 *source* (not intermediate routers) responsible for fragmentation**
  - **Routers simply drop datagrams larger than network MTU**
  - **Source must fragment datagram to reach destination**
- **Source determines *path MTU***
  - **Smallest MTU on any network between source and destination**
  - **Fragments datagram to fit within that MTU**
- **Uses *path MTU discovery***
  - **Source sends probe message of various sizes until destination reached**
  - **Must be dynamic - path may change durin**

Use of multiple headers

- **Efficiency - header only as large as necessary**
- **Flexibility - can add new headers for new features**
- **Incremental development - can add processing for new features to testbed; other routers will skip those headers**

IPv6 addressing

- **128-bit addresses**
- **Includes network prefix and host suffix**
- **No address classes - prefix/suffix boundary can fall anywhere**
- **Special types of addresses:**
  - ***unicast* - single destination computer**
  - ***multicast* - multiple destinations; possibly not at same site**
  - ***cluster* - collection of computers with same prefix; datagram is delivered to one out of cluster**
- **IPv4 broadcast flavors are subsets of multicast**
- **Cluster addressing allows for duplication of services**

IPv6 address notation

- **128-bit addresses unwieldy in dotted decimal; requires 16 numbers**

105.220.136.100.255.255.255.255.0.0.18.128.140.10.255.255

- **Groups of 16-bit numbers in hex separated by colons - *colon hexadecimal* (or *colon hex*)**

69DC:8864:FFFF:FFFF:0:1280:8C0A:FFFF

- **Zero-compression - series of zeroes indicated by two colons**

FF0C:0:0:0:0:0:0:B1

FF0C::B1

- **IPv6 address with 96 leading zeros is interpreted to hold an IPv4 address**

Summary

- **IPv4 basic abstractions have been *very* successful**
- **IPv6 carries forward many of those abstraction... but, all the details are changed**
  - **128-bit addresses**
  - **Base and extension headers**
  - **Source does fragmentation**
  - **New types of addresses**
  - **Address notation**

Chapter 21 - ICMP

Section	Title
1	Introduction
2	Error detection
3	Error reporting
4	Types of messages
5	ICMP message transport
6	ICMP and reachability
7	ICMP and internet routes
8	ICMP and path MTU discovery
9	ICMP and router discovery
10	ICMP redirect
11	Summary

Introduction

- **IP provides *best-effort delivery***
- **Delivery problems can be ignored; datagrams can be "dropped on the floor"**
- ***Internet Control Message Protocol* (ICMP) provides error-reporting mechanism**

Error detection

- **Internet layer can detect a variety of errors:**
  - **Checksum (header only!)**
  - **TTL expires**
  - **No route to destination network**
  - **Can't deliver to destination host (e.g., no ARP reply)**
- **Internet layer discards datagrams with problems**
- **Some - e.g., checksum error - can't trigger error messages**

Error reporting

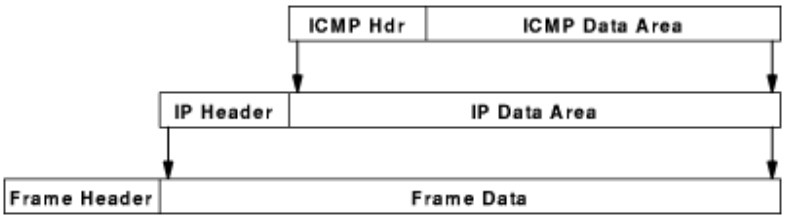
- **Some errors can be reported**
  - **Router sends message back to source in datagram**
  - **Message contains information about problem**
- **Encapsulated in IP datagram**

Types of messages

- ***Internet Control Message Protocol* (ICMP) defines error and informational messages**
- **Error messages:**
  - **Source quench**
  - **Time exceeded**
  - **Destination unreachable**
  - **Redirect**
  - **Fragmentation required**
- **Informational messages:**
  - **Echo request/reply**
  - **Address mask request/reply**
  - **Router discovery**

ICMP message transport

- **ICMP encapsulated in IP**



- **But ... how can that work?**
- **ICMP messages sent in response to incoming datagrams with problems**
- **ICMP message** not **sent for ICMP message**

ICMP and reachability

- **An internet host, *A*, is *reachable* from another host, *B*, if datagrams can be delivered from *A* to *B***
- ***ping* program tests reachability - sends datagram from *B* to *A* that *A echoes* back to B**
- **Uses ICMP *echo request* and *echo reply* messages**
- **Internet layer includes code to reply to incoming *ICMP echo request* messages**

ICMP and internet routes

- **List of all routers on path from *A* to *B* is called the *route* from *A* to *B***
- ***traceroute* uses UDP to non-existent port and TTL field to find route via *expanding ring* search**
- **Sends ICMP echo messages with increasing TTL**
  - **Router that decrements TTL to 0 sends *ICMP time exceeded* message, with router's address as source address**
  - **First, with TTL 1, gets to first router, which discards and sends time exceeded message**
  - **Next, with TTL 1, gets through first router to second router**
  - **Continue until message from destination received**

- ***traceroute* must accommodate varying network delays**
- **Must also accommodate dynamically changing routes**

ICMP and path MTU discovery

- **Fragmentation should be avoided**
- **How can source configure outgoing datagrams to avoid fragmentation?**
- **Source determines *path MTU* - smallest network MTU on path from source to destination**
- **Source *probes* path using IP datagrams with *don't fragment* flag**
- **Router responds with *ICMP fragmentation required* message**
- **Source sends smaller probes until destination reached**

ICMP and router discovery

- **Router can fail, causing "black-hole" or isolating host from internet**
- ***ICMP router discovery* used to find new router**
- **Host can broadcast request for router announcements to auto-configure default route**
- **Host can broadcast request if router fails**
- **Router can broadcast advertisement of existence when first connected**

ICMP redirect

- **Default route may cause *extra hop***
- **Router that forwards datagram on same interface sends *ICMP redirect***
- **Host installs new route with correct router as next hop**

Summary

- **Internet layer provides *best-effort delivery* service**
- May **choose to report errors for some problems**
- ***ICMP* provides error message service**

Chapter 22 - Transport Protocols

Introduction

- ***Internet Protocol*** **(IP) provides ``unreliable datagram service'' between *hosts***
- ***Transport protocols*** **provide end-to-end delivery between endpoints of a connection; e.g., processes or programs**
- ***User Datagram Protocol*** **(UDP) provides datagram service**
- ***Transmission Control Protocol*** **(TCP) provides reliable data delivery**

User Datagram Protocol

- **UDP delivers independent messages, called *datagrams* between applications or processes on host computers**
  - **``Best effort'' delivery - datagrams may be lost, delivered out of order, etc.**
  - **Checksum (optionally) guarantees integrity of data**
- **For generality, endpoints of UDP are called *protocol ports* or *ports***
- **Each UDP data transmission identifies the internet address and port number of the destination and the source of the message**
- ***Destination port*** **and *source port* may be different**

UDP and TCP/IP layering

- **Transport protocols use IP to provide data delivery for application protocols**

Application
Transport UDP, TCP
Internet
Network interface
Hardware

UDP headers

- **UDP datagrams have a header that follows the hardware and IP headers:**

Hardware	**IP**	**UDP**	**Application**	**Data**

- **UDP header is very simple:**
  - **Port numbers**
  - **Message length**
  - **Checksum**

UDP source port	**UDP destination port**
UDP message length	**UDP checksum**
Data	

Selecting UDP port numbers

- **Communicating computers must agree on a port number**
  - **``Server'' opens selected port and waits for incoming messages**
  - **``Client'' selects local port and sends message to selected port**
- **Services provided by many computers use reserved, *well-known port numbers*:**
  - **ECHO**
  - **DISCARD**
  - **NTP**
- **Other services use *dynamically assigned* port numbers**

Well-known port numbers

Port	Name	Description
7	**echo**	**Echo input back to sender**
9	**discard**	**Discard input**
11	**systat**	**System statistics**
13	**daytime**	**Time of day (ASCII)**
17	**quote**	**Quote of the day**
19	**chargen**	**Character generator**
37	**time**	**System time (seconds since 1970)**
53	**domain**	**DNS**
69	**tftp**	**Trivial File Transfer Protocol (TFTP)**
123	**ntp**	**Network Time Protocol (NTP)**
161	**snmp**	**Simple Network Management Protocol (SNMP)**

TCP

- ***Transmission Control Protocol* (TCP) is most widely used transport protocol**
- **Provides *reliable data delivery* by using IP *unreliable datagram delivery***
- **Compensates for loss, delay, duplication and similar problems in Internet components**
- **Reliable delivery is high-level, familiar model for construction of applications**

Features of TCP

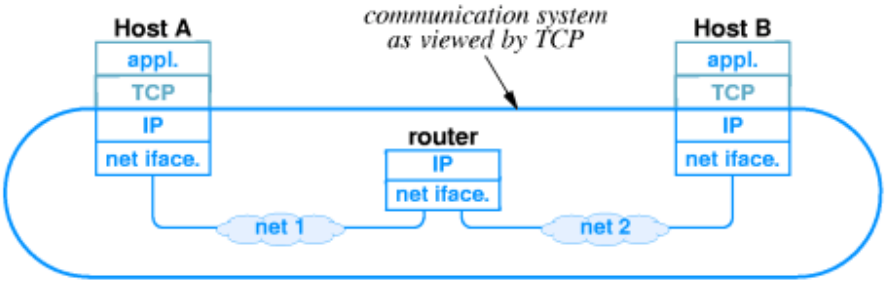
- ***Connection oriented*: Application requests connection to destination and then uses connection to deliver data to transfer data**
- ***Point-to-point*: A TCP connection has two endpoints**
- ***Reliability*: TCP guarantees data will be delivered without loss, duplication or transmission errors**
- ***Full duplex*: The endpoints of a TCP connection can exchange data in both directions simultaneously**
- ***Stream interface*: Application delivers data to TCP as a continuous *stream*, with no record boundaries; TCP makes no guarantees that data will be received in same blocks as transmitted**
- ***Reliable connection startup*: *Three-way handshake* guarantees reliable, synchronized startup between endpoints**
- ***Graceful connection shutdown*: TCP guarantees delivery of all data after endpoint shutdown by application**

Using IP for data delivery

- **TCP uses IP for data delivery (like UDP)**
- **Endpoints are identified by ports (like UDP)**
  - **Allows multiple connections on each host**
  - **Ports may be associated with an application or a process**
- **IP treats TCP like data and does not interpret any contents of the TCP message**

Delvering TCP

- **TCP travels in IP datagrams**
- **Internet routers only look at IP header to forward datagrams**
- **TCP at destination interprets TCP messages**

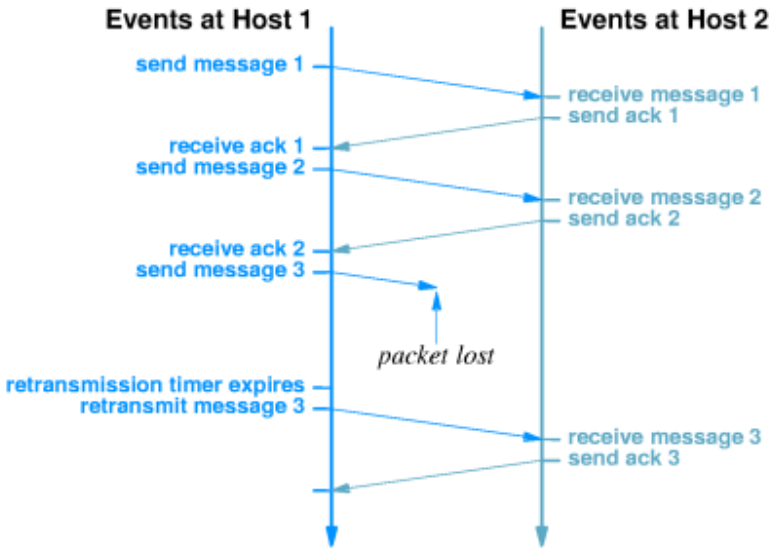


TCP and reliable delivery

- **TCP uses many techniques described earlier to provide reliable delivery**
- **Recovers from**
  - **Lost packets**
  - **Duplicate packets**
  - **Delayed packets**
  - **Corrupted data**
  - **Transmission speed mismatches**
  - **Congestion**
  - **System reboots**

Lost packets

- **TCP uses *positive acknowledgment with retransmission* to achieve reliable data delivery**
- **Recipient sends *acknowledgment* control messages (ACK) to sender to verify successful receipt of data**
- **Sender sets timer when data transmitted; if timer expires before acknowledgment arrives, sender retransmits (with new timer)**



TCP segments and sequence numbers

- **Application delivers arbitrarily large chunks of data to TCP as a``stream''**
- **TCP breaks this data into *segments*, each of which fits into an IP datagram**
- **Original stream is numbered by bytes**
- **Segment contains *sequence number* of data bytes**

Acknowledgments

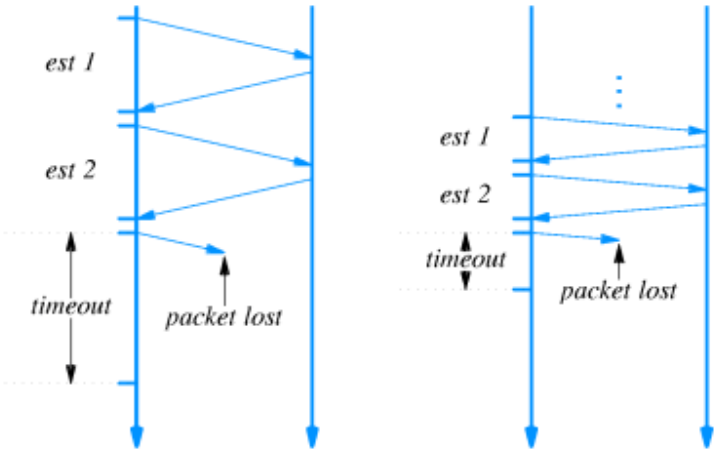
- **Receiver sends segment with sequence number of acknowledged *data* (not segments)**
- **One ACK can acknowledge many segments**

Setting the timeout

- **Inappropriate timeout can cause poor performance:**
  - **Too long - sender waits longer than necessary before retransmitting**
  - **Too short - sender generates unnecessary traffic**
- **Timeout must be different for each connection and set dynamically**

- **Host on same LAN should have shorter timeout than host 20 hops away**
- **Delivery time across internet may change over time; timeout must accommodate changes**

RTOs for different network delays



Picking a timeout value

- **Timeout should be based on *round trip time* (RTT)**
- **Sender can't know RTT of any packet before transmission**
- **Sender picks *retransmission timeout* (RTO) based on *previous* RTTs**
- **Specific method is call *adaptive retransmission algorithm***

Computing RTT and RTO

- **Weighted average for RTT:**

$$RTT_{new} = (alpha*RTT_{old}) + ((1 - alpha)*RTT_{sample}))$$

- **Computation of RTO:**

$$RTO = beta*RTT_{new}$$

Measuring RTT

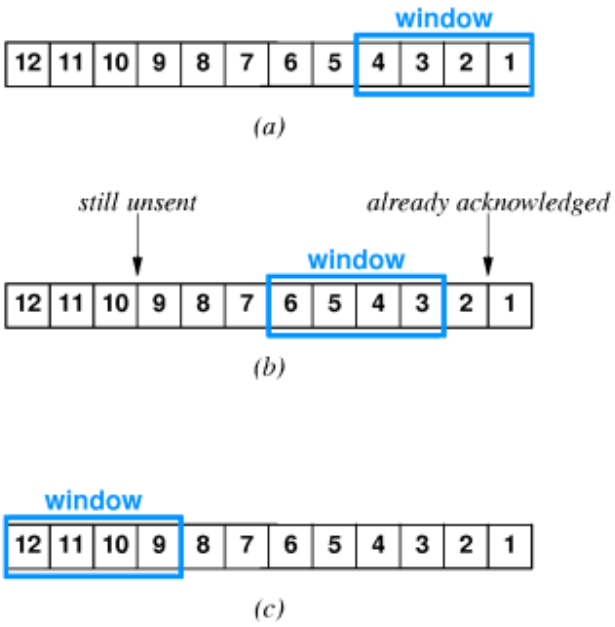
- **RTT measured by observing difference between time of transmission and arrival of acknowledgment**
- ***However* - acknowledgments carry no information about** which **packet is acknowledged**
- **Sender cannot determine whether acknowledgment is from original transmission or retransmission**
  - **Choosing original transmission *overestimates* RTT**
  - **Choosing retransmission *underestimates* RTT**

Karn's algorithm

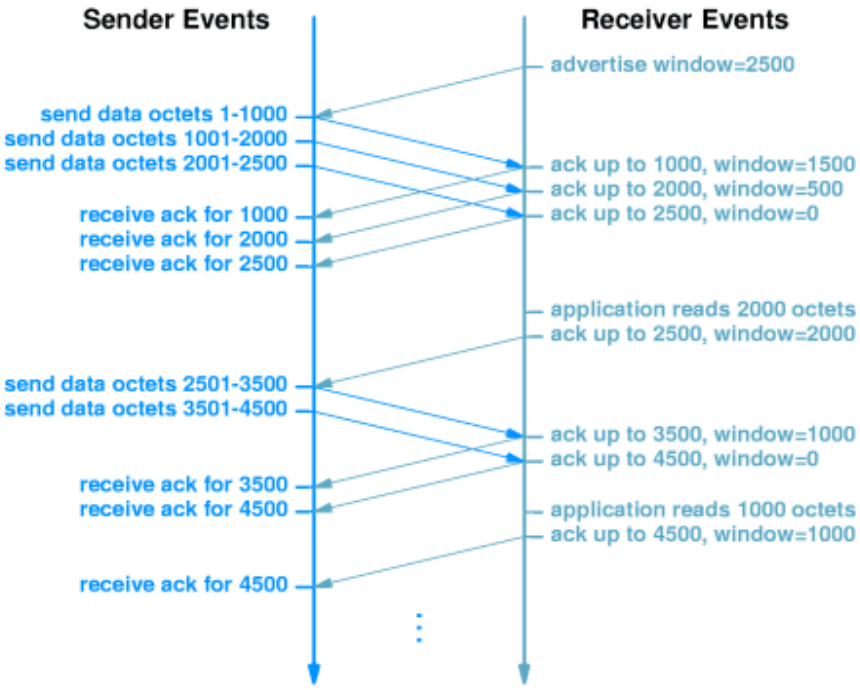
- **How to choose between original and retransmission? Answer: choose neither!**
- ***Karn's algorithm* specifies that sender ignores RTTs for retransmitted segments**
- **How will RTT get updated if internet round trip time increases?**
- **Karn's algorithm specifies that RTO is separated from RTT when retransmission occurs**
- **RTO *doubles* for each new message until ACK arrives with no retransmission**

TCP sliding window

- **TCP uses sliding window for flow control**
- **Receiver specifies window**
  - **Called *window advertisement***
  - **Specifies which *bytes* in the data stream can be sent**
  - **Carried in segment along with ACK**
- **Sender can transmit any bytes, in any size segment, between last acknowledged byte and within window size**



Sliding window with acknowledgments



Sliding window example

---

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap22/chap22_23.html

Shockwave

Sliding window with lost segment

---

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap22/chap22_24.html

Shockwave

Flow control with sliding window

---

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap22/chap22_25.html

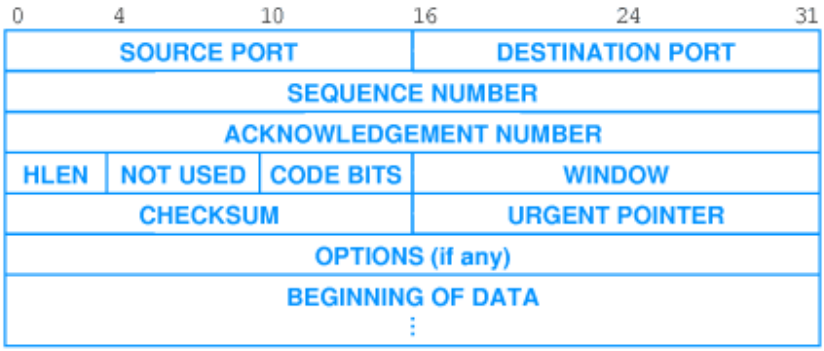
Shockwave

Silly window syndrome

- **Under some circumstances, sliding window can result in transmission of many small segments**
- **If receiver window full, and receiving application consumes a few data bytes, receiver will advertise small window**
- **Sender will immediately send small segment to fill window**
- **Inefficient in processing time and network bandwidth**
- **Solutions:**
  - **Receiver delays advertising new window**
  - **Sender delays sending data when window is small**

TCP segment format

- **TCP segment has header containing:**



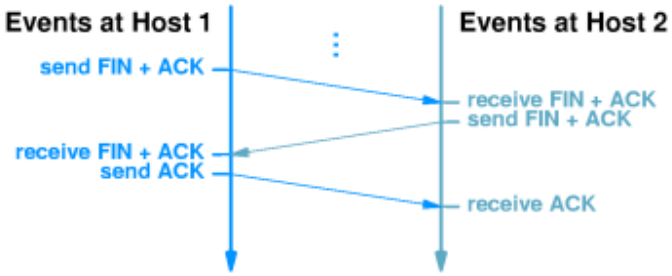
- **Same header format used in both directions**
- **Segment can carry both data and acknowledgment**

Three-way handshake

- **TCP uses *three-way handshake* for reliable connection establishment and termination**
  - **Host 1 sends segment with SYN bit set and random sequence number**
  - **Host 2 responds with segment with SYN bit set, acknowledgment to Host 1 and random sequence number**

  - **Host 1 responds with acknowledgment**
- **TCP will retransmit lost segments**
- **Random sequence numbers ensure synchronization between endpoints**

Closing a connection



Opening a connection

---

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap22/chap22_30.html

Shockwave

Closing a connection

---

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap22/chap22_31.html

Shockwave

Congestion control

- **Excessive traffic can cause packet loss**
  - **Transport protocols respond with retransmission**
  - **Excessive retransmission can cause *congestion collapse***
- **TCP interprets packet loss as an indicator of congestion**
- **Sender uses TCP *congestion control* and slows transmission of packets**
  - **Sends single packet**
  - **If acknowledgment returns without loss, sends two packets**
  - **When TCP sends one-half window size, rate of increase slows**

Summary

- **UDP provides end-to-end best-effort message delivery**
  - **IP used for delivery to destination host**
  - **Protocol ports demultiplex to destination application**
- **TCP provides end-to-end reliable bytestream delivery**
  - **IP used for delivery to destination host**
  - **Protocol ports demultiplex to destination application**
  - **Additional techniques develop reliable delivery from IP messages**
- **Positive acknowledgment with retransmission**
- **Sequence numbers detect missing, duplicate and out-of-order data**
- **Sliding window flow control**
- **Three-way handshake**
- **Congestion control**

Chapter 23 - Client-Server Interaction

Introduction

- **Application-level protocols provide high-level services**
  - **DNS**
  - **Electronic mail**
  - **Remote login**
  - **FTP**
  - **World Wide Web**
- **All of these applications use *client-server* architecture**

Internet protocols and network applications

- **Internet protocols provide**
  - **General-purpose facility for reliable data transfer**
  - **Mechanism for contacting hosts**
- **Application programs**
  - **Use internet protocols to contact other applications**
  - **Provide *user-level* services**

Establising contact through internet protocols

- **Application must interact with protocol software *before* contact is made**
- ***Listening* application informs local protocol software that it is ready to accept incoming messages**
- ***Connecting* application uses internet protocol to contact listener**
- **Applications exchange messages through resulting connection**

Client-server paradigm

- ***Server* application is ``listener''**
  - **Waits for incoming message**
  - **Performs service**
  - **Returns results**
- ***Client* application establishes connection**
  - **Sends message to server**
  - **Waits for return message**

Characteristics of client

- **Arbitrary application program**
  - **Becomes client when network service is needed**
  - **Also performs other computations**
- **Invoked directly by user**
- **Runs locally on user's computer**
- **Initiates contact with server**
- **Can access multiple services (one at a time)**
- **Does not require special hardware or sophisticated operating system**

Characteristics of server

- **Special purpose application dedicated to providing network service**
- **Starts at system initialization time**
- **Runs on a remote computer (usually centralized, shared computer)**
- **Waits for service requests from clients; loops to wait for next request**
- **Will accept requests from arbitrary clients; provides one service to each client**
- **Requires powerful hardware and sophisticated operating system**

``Server-class'' computers

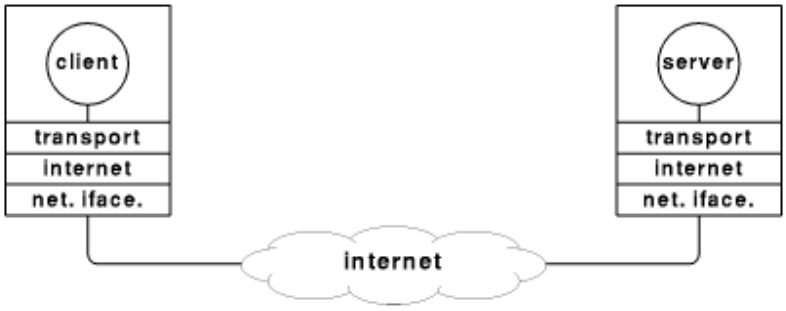
- **Shared, centralized computers that run many server applications are sometimes called ``servers''**
- **More precisely, the *applications* are the ``servers'' and the computer is a ``server-class computer''**
- **Servers *can* run on very simple computers...**

Message exchanges

- **Typically, client and server exchange messages:**
  - **Client sends request, perhaps with data**
  - **Server send response, perhaps with data**
- **Client may send multiple requests; server sends multiple responses**
- **Server may send multiple response - imagine video feed**

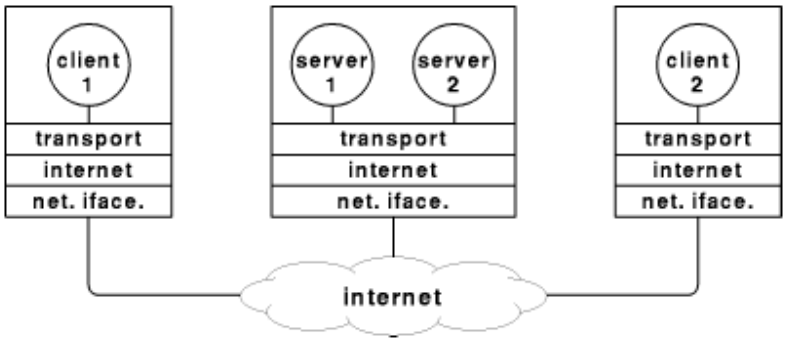
Transport protocols and client-server paradigm

- **Clients and servers exchange messages through transport protocols; e.g., TCP or UDP**
- **Both client and server must have same protocol stack and both interact with transport layer**



Multiple services on one computer

- **Sufficiently powerful computer - fast enough processor, multi-tasking OS - may run multiple servers**
- **Servers run as independent processes and can manage clients simultaneously**



Identifying a service

- **Each service gets a *unique identifier*; both client and**
- **Server use that identifier**
  - **Server registers with local protocol software under the identifier**
  - **Client contacts protocol software for session under that identifier**
- **Example - TCP uses *protocol port numbers* as identifiers**
  - **Server registers under port number for service**
  - **Client requests session with port number for service**

Multiple servers for one service

- **Responding to a client request may require significant time**
- **Other clients must wait while earlier requests are satisfied**
- **Multiple servers can handle requests *concurrently*, completing shorter requests without waiting for longer requests**

Master-slave servers

- **One way to run concurrent servers is to dynamically create server processes for each client**
- ***Master* server accepts incoming requests and starts *slave* server for each client**
- **Slave handles subsequent requests from its client**
- **Master server then waits for next request**

Selecting from multiple servers

- **How do incoming messages get delivered to the correct server?**
- **Each transport session has two unique identifiers**
  - **(IP address, port number) on server**
  - **(IP address, port number) on client**
- **No two clients on one computer can use same source port**
- **Thus, client endpoints are unique, and server computer protocol software can deliver messages to correct server process**

Connection-oriented and connectionless transport

- **Which to choose?**
- **TCP - connection-oriented**
  - **Client establishes connection to server**
  - **Client and server exchange multiple messages of arbitrary size**
  - **Client terminates connection**
- **UDP - connectionless**
  - **Client constructs message**
  - **Client sends message to server**
  - **Server responds**
  - **Message must fit in one UDP datagram**
- **Some services use both**
  - **DNS, chargen, motd**
  - **Can be provided by single server**

Client-server interactions

- **Clients can access multiple services sequentially**
- **Clients may access different servers for one service**
- **Servers may become clients of other servers**
- **Circular dependencies may arise...**

Summary

- ***Client-server* paradigm used in almost every distributed computation**
  - ***Client* requests service when needed**
  - ***Server* waits for client requests**
- **Servers usually run on *server-class computer***
- **Clients and servers use *transport protocols* to communicate**
- **Often, but not always, there is an *application protocol***

Chapter 24 - Socket Interface

Section Title

Introduction

- **The *socket* is one form of interface between application programs and protocol software**
- **Widely available - program portability**
- **Used by both clients and servers**
- **Extension to UNIX *file I/O* paradigm**

API

- **Application interactions with protocol software:**
  - **Passive listen or active open**
  - **Protocol to use**
  - **IP address and port number**
- **Interface to protocol is call *Application Program Interface* (API)**
  - **Defined by programming/operating system**
  - **Includes collection of procedures for application program**

The Socket API

- **Protocols do not typically specify API**
- **API defined by programming system**
- **Allows greatest flexibility - compatibility with different programming systems**
- ***Socket API* is a specific protocol API**
  - **Originated with Berkeley BSD UNIX**
  - **Now available on Windows 95 and Windows NT, Solaris, etc.**
- **Not defined as TCP/IP standard; *de facto*standard**

Sockets and socket libraries

- **BSD UNIX includes sockets as *system calls***
- **Other vendors (mostly UNIX) have followed suit**
- **Some systems have different API**
  - **Adding sockets would require changing OS**
  - **Added library procedures - *socket library* - instead**
- **Adds layer of software between application and operating system**
  - **Enhances portability**
  - **May hide native API altogether**

Sockets and UNIX I/O

- **Developed as extension to UNIX I/O system**
- **Uses same *file descriptor* address space (small integers)**
- **Based on *open-read-write-close* paradigm**
  - ***open* - prepare a file for access**
  - ***read/write* - access contents of file**
  - ***close* - gracefully terminate use of file**
- **Open returns a file descriptor, which is used to identify the file to read/write/close**

The socket API

- **Socket programming more complex than file I/O**
- **Requires more parameters**
  - **Addresses**
  - **Protocol port numbers**
  - **Type of protocol**
  - **New semantics**
- **Two techniques**
  - **Add parameters to existing I/O system calls**
  - **Create new system calls**
- **Sockets use a collection of new system calls**

Summary of socket system calls

- **socket - create a new socket**
- **close - terminate use of a socket**
- **bind - attach a network address to a socket**
- **listen - wait for incoming messages**
- **accept - begin using incoming connection**
- **connect - make connection to remote host**
- **send - transmit data through active connection**
- **recv - receive data through active connection**

socket

descriptor = socket(protofamily, type, protocol)

- **Returns socket descriptor used in subsequent calls**
- **protofamily selects protocol family; e.g.:**
  - **PF_INET - Internet protocols**
  - **PF_APPLETALK - AppleTalk protocols**
- **type selects type of communication**
  - **SOCK_DGRAM - connectionless**
  - **SOCK_STREAM - connection-oriented**
- **protocol specifies protocol within protocol family**
  - **IPPROTO_TCP - selects TCP**
  - **IPPROTO_UDP - selects UDP**

close

---

close(descriptor)

- **Terminates use of socket descriptor**
- **descriptor contains descriptor of socket to be closed**

bind

---

bind(socket, localaddr, address)

- **Initially, socket has no addresses attached**
- **bind selects either local, remote or both addresses**
  - ***server* binds local port number for incoming messages**
  - ***client* binds remote address and port number to contact server**

Socket address formats

- **Because sockets can be used for any protocols, address format is generic:**

```
struct sockaddr {
   u_char sa_len;          /* total length of address */
   u_char sa_family;       /* family of the address   */
   char   sa_data[14];     /* address                 */
}
```

- **For IP protocols, sa_data hold IP address and port number:**

```
struct sockaddr_in {
   u_char  sin_len;        /* total length of address */
   u_char  sin_family;     /* family of the address   */
   u_short sin_port;       /* protocol port number    */
   struct  in_addr sin_addr; /* IP address            */
   char    sin_zero[8]     /* unused                  */
}
```

- **First two fields match generic sockaddr structure**
- **Remainder are specific to IP protocols**
- **INADDR_ANY interpreted to mean "any" IP address**

listen

listen(socket, queuesize)

- **Server uses listen to wait for incoming connections**
- **socket identifies socket through which connections will arrive (address)**
- **New connection requests may arrive while server processes previous request**
- **Operating system can hold requests on queue**
- **queuesize sets upper limit on outstanding requests**

accept

accept(socket, caddress, caddresslen)

- **Server uses accept to accept the next connection request**
- **accept call blocks until connection request arrives**
- **Returns *new socket* with server's end of new connection**
- ***Old socket* remains unchanged and continues to field incoming requests**
- **caddress returns struct sockaddr client address; format depends on address family of socket**
- **caddresslen returns length of address**

connect

connect(socket, saddress, saddresslen)

- **Client uses connect to establish connection to server**
- **Blocks until connection completed (accepted)**
- **socket holds descriptor of socket to use**
- **saddress is a struct sockaddr that identifies server**
- **saddresslen gives length of saddress**
- **Usually used with connection-oriented transport protocol**
- **Can be used with connectionless protocol**
  - **Marks local socket with server address**
  - **Implicitly identifies server for subsequent messages**

send

send(socket, data, length, flags)

- **Used to send data through a connected socket**
- **socket identifies socket**
- **data points to data to be sent**
- **length gives length of data (in bytes)**
- **flags indicate special options**

sendto, sendmsg

sendto(socket, data, length, flags, destaddress, addresslen)
sendmsg(socket, msgstruct, flags)

- **Used for *unconnected* sockets by explicitly specifying destination**
- **sendto adds additional parameters:**
  - **destaddress - struct sockaddr destination address**
  - **addresslen - length of destaddress**
- **sendmsg combines list of parameters into single structure:**

```
struct msgstruct {
   struct sockaddr *m_addr;   /* ptr to destination address */
   struct datavec  *m_vec;    /* pointer to message vector  */
   int    m_dvlength;         /* num. of items in vector    */
   struct access   *m_rights; /* ptr to access rights list  */
   int    m_alength;          /* num. of items in list      */
}
```

recv

recv(socket, buffer, length, flags)

- **Used to receive incoming data through connected socket**
- **socket identifies the socket**
- **Data copied into buffer**
- **At most length bytes will be recved**
- **flags give special options**
- **Returns number of bytes actually recved**
  - **0 implies connection closed**
  - **-1 implies error**

recvfrom, recvmsg

recvfrom(socket, buffer, length, flags, sndraddress, addresslen)
recvmsg(socket, msgstruct, flags)

- **Like sendto and sendmsg (in reverse!)**
- **Address of source copied into sndraddress**
- **Length of address in addresslen**
- **recvmsg uses msgstruct for parameters**

Other procedures

- **getpeername - address of other end of connection**
- **getsockname - current address bound to socket**
- **setsockopt - set socket options**

Sockets and processes

- **Like file descriptors, sockets are *inherited* by child processes**
- **Socket disappears when *all* processes have closed it**
- **Servers use socket inheritance to pass incoming connections to slave server processes**

Summary

- **Socket API is *de facto* standard**
  - **Originally developed for BSD UNIX**
  - **Copied to many other systems**
- **Sockets are an extension of the UNIX file I/O system**
  - **Use same descriptor addresses**
  - **Can (but typically don't) use same system calls**
- **Many specific system calls for sockets**

Chapter 25 - Client/Server Example

Introduction

- **Will examine details of client and server programs**
- **Examples use socket API**
- **Will illustrate details of socket use**
- **Will also illustrate program architecture**

Connection-oriented communication

- **Client/server developer must choose between connectionless and connection-oriented service**
  - **Connectionless can be used at any time; does not provide reliability**
  - **Connection-oriented requires explicit connection; provides reliable data delivery**
- **This example will use connection-oriented transport**
  - **Server contacts local protocol software to accept incoming connections**
  - **Client establishes connection to server through client's local protocol software**
- **Client and server exchange data once connection is established**

An example service

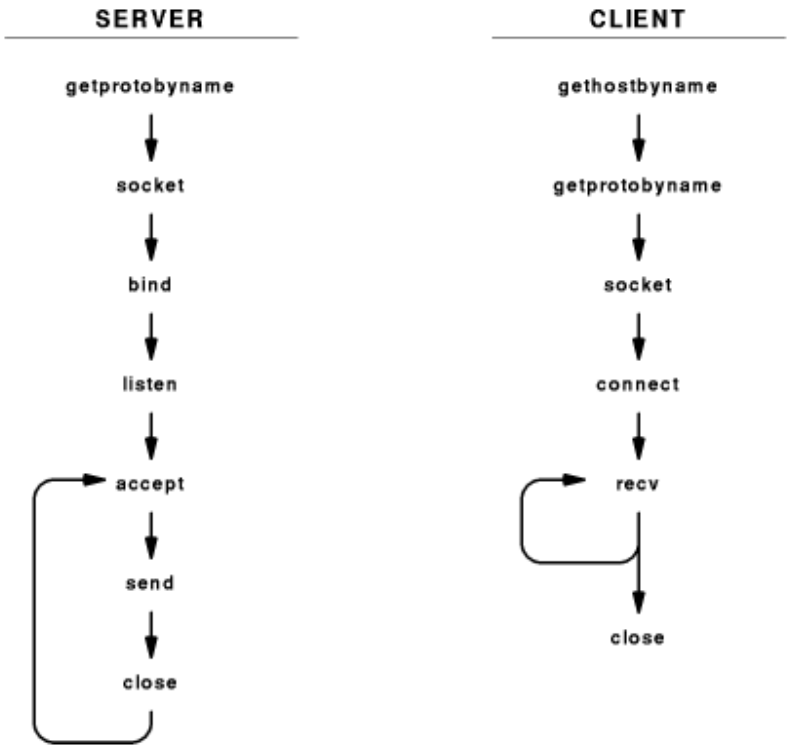
- **Server**
  - **Keeps track of how many times it has been contacted**
  - **Returns that count in an ASCII string**
- **Client**
  - **Establishes connection**
  - **Waits for message from server (ASCII string)**
  - **Prints data from message**
- **Application-level protocol:**
  - **Syntax:**
    - **No headers**
    - **Client sends null message body**
    - **Server message contains ASCII string**
  - **Semantics - Client establishes connection; server returns ASCII string**

Example programs

- **Example server in server.c**
  - **Runs as background program**
  - **Takes single (optional) command line argument: port number to listen on (default is 5193)**
- **Example client in client.c**
  - **Runs as user program**
  - **Takes two (optional) command line arguments: name of host to contact (default is localhost) and port number to contact (default is 5193)**

Program architecture

- **Sequence of socket calls in server and client**



Server

- **Initialization:**
  - **getprotobyname - looks up protocol number for TCP**
  - **socket - creates socket**
  - **listen - associates socket with incoming requests**
- **Loop:**
  - **accept - accepts incoming connection**
  - **send - send message to client**
  - **close - closes connection socket**

Client

- **Initialization:**
  - **gethostbyname - looks up server**
  - **getprotobyname - looks up protocol port number for TCP**
  - **socket - creates socket**
  - **connect - connects to server port**

- **Loop:**
  - **recv - receives message from server**
- **Termination:**
  - **close - closes socket**

Client calls to recv

- **The call to recv is in a loop in the client**
- **But, the server send only a single message - why would multiple calls be required?**
- **Protocol software does *not* guarantee to deliver data to receiver in the same blocks as generated by the server**
  - **May travel in different segments**
  - **recv need not return as much data as was requested**
- **Client must make repeated calls to recv until 0 returned**

Sockets and blocking

- **Most socket calls are *blocking***
- **Calling process is blocked (taken off *ready* queue) until socket operation completes**
- **Server blocks:**
  - **accept until new connection arrives**
  - **send until data delivered to protocol software**
- **Client blocks:**
  - **gethostbyname until DNS resolves server name**
  - **recv until message delivered from server**

Using client with another server

- **Other services use similar application protocol:**
  - **DAYTIME**
  - **CHARGEN**
  - **MOTD**
- **Specifying port connects to alternate services**
- **Example:**

```
$ client www.netbook.cs.purdue.edu 13
Thu Feb 27 15:19:06 1997
$ client leo.eg.bucknell.edu 13
Thu Feb 27 15:19:18 1997
$ client merlin.cs.purdue.edu 13
Thu Feb 27 15:19:26 1997
```

```
$ client cs.ucla.edu 13
Thu Feb 27 12:20:28 1997
$ client uran.informatik.uni-bonn.de 13
Thu Feb 27 21:22:37 1997
$ client yotaga.psu.ac.th 13
Fri Feb 28 03:24:20 1997
```

Using another client with server

- **Other client that uses same application protocol can test server**
- **Example: telnet**

```
$ telnet www.netbook.cs.purdue.edu 5193
Trying 134.82.11.70 ...
Connected to regulus.eg.bucknell.edu.
Escape character is '^]'.
This server has been contacted 5 times.
Connection closed by foreign host.
```

Summary

- **Example client and server**
  - **Connection-oriented transport**
  - **Very simple application protocol**
- **Demonstrates use of socket calls**
- **Can be used with other clients and servers**

Chapter 26 - the Domain Name System

Introduction

- **IP assigns 32-bit addresses to hosts (interfaces)**
  - **Binary addresses easy for computers to manage**
  - **All applications use IP addresses through the TCP/IP protocol software**
  - **Difficult for humans to remember:**

% telnet 134.82.11.70

- **The *Domain Name System* (DNS) provides translation between symbolic names and IP addresses**

Structure of DNS names

- **Each name consists of a sequence of alphanumeric components separated by periods**
- **Examples:**

  www.eg.bucknell.edu
  www.netbook.cs.purdue.edu
  charcoal.eg.bucknell.edu

- **Names are hierarchical, with most-significant component on the right**
- **Left-most component is computer name**

DNS naming structure

- ***Top level domains* (right-most components; also known as *TLD*s) defined by global authority**

com	**Commercial organization**
edu	**Educational institution**
gov	**Government organization**
mil	**Military organization**

- **Organizations apply for names in a top-level domain:**

  bucknell.edu
  macdonalds.com

- **Organizations determine own internal structure**

  eg.bucknell.edu
  cs.purdue.edu

Geographic structure

- **Top-level domains are US-centric**
- **Geographic TLDs used for organizations in other countries:**

TLD	Country
.uk	**United Kingdom**
.fr	**France**

.ch Switzerland
.in India

- **Countries define their own internal hierarchy: ac.uk and .edu.au are used for academic organizations in the United Kingdom and Australia**

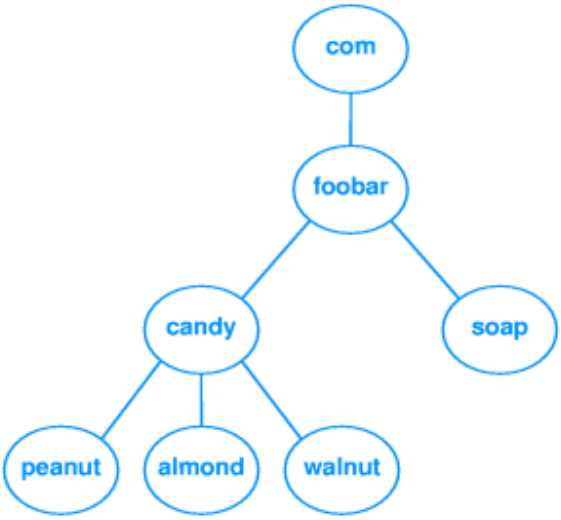
Domain names within an organization

- **Organizations can create any internal DNS hierarchy**
- **Uniqueness of TLD and organization name guarantee uniqueness of any internal name (much like file names in your directories)**
- **All but the left-most component of a domain name is called the *domain* for that name:**

Name	Domain
www.netbook.cs.purdue.edu	**netbook.cs.purdue.edu**
regulus.eg.bucknell.edu	**eg.bucknell.edu**
coral.bucknell.edu	**bucknell.edu**

- **Authority for creating new *subdomains* is delegated to each domain**
- **Administrator of bucknell.edu has authority to create eg.bucknell.edu and need not contact any central naming authority**

Example DNS hierarchy

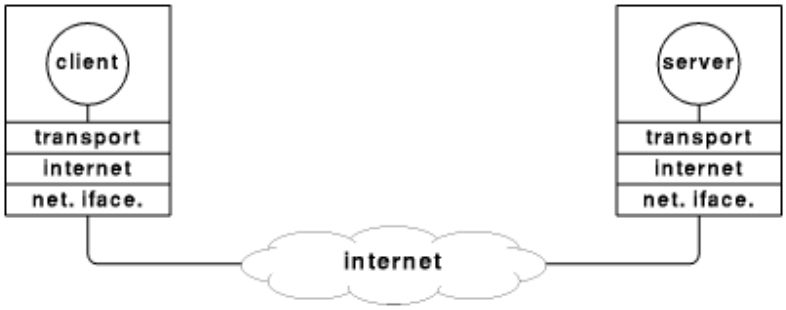


DNS names and physical location

- **DNS domains are logical concepts and need not correspond to physical location of organizations**
- **DNS domain for an organization can span multiple networks**
  - **bucknell.edu covers all networks at Bucknell**
  - **www.netbook.cs.purdue.edu is in 318 Dana**
  - **laptop.eg.bucknell.edu could be connected to a network in California**

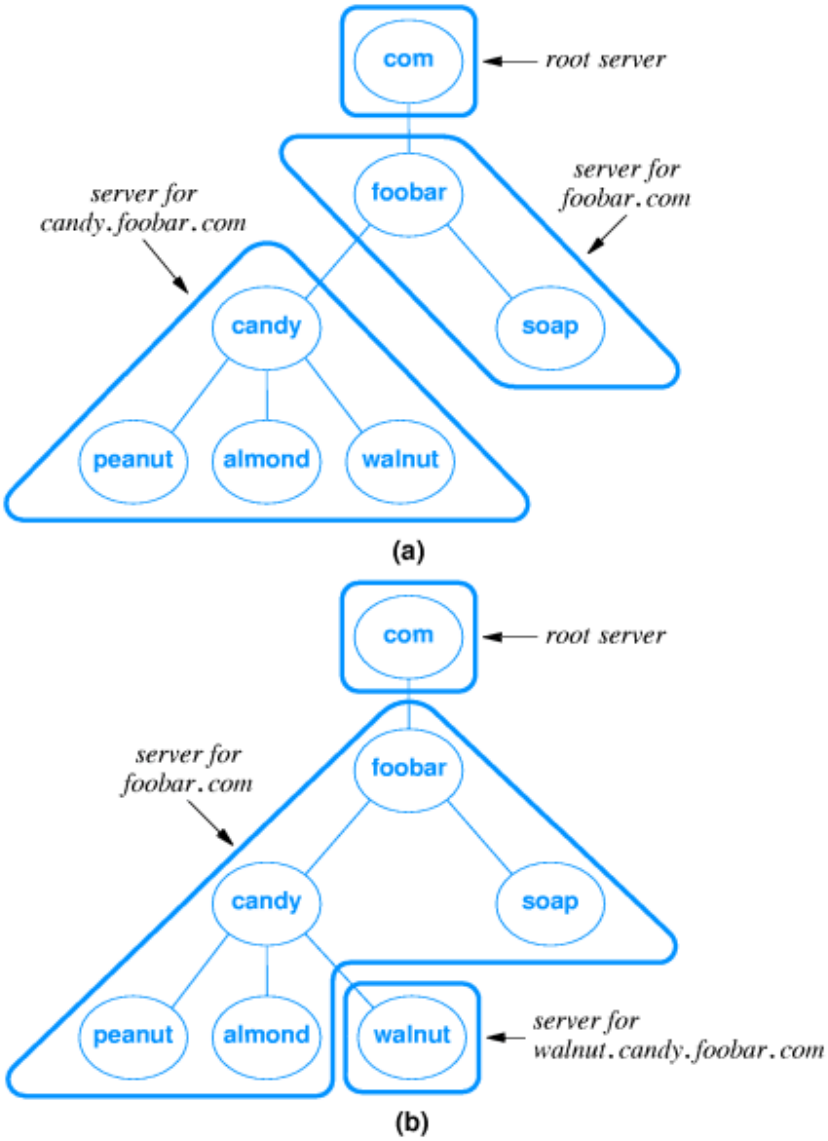
Client-server computing

- ***Clients* and *servers* communicate in distributed computing**
  - **Client initiates contact to request some remote computation**
  - **Server waits for clients and answers requests as received**
- **Clients are usually invoked by users as part of an end-user application**
- **Servers are usually run on central, shared computers**



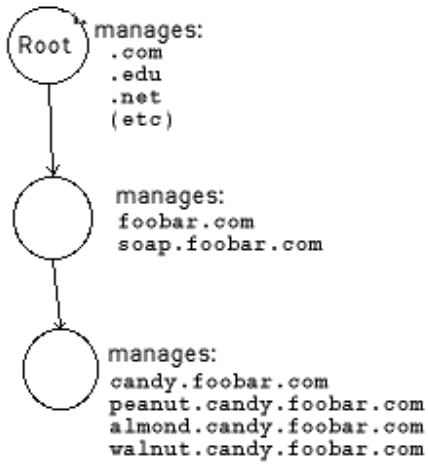
DNS and client-server computing

- **DNS names are managed by a hierarchy of DNS servers**
- **Hierarchy is related to DNS domain hierarchy**



- **Root server at top of tree knows about next level servers**
- **Next level servers, in turn, know about lower level servers**

DNS server hierarchy



Choosing DNS server architecture

- **Small organizations can use a single server**
  - **Easy to administer**
  - **Inexpensive**
- **Large organizations often use multiple servers**
  - **Reliability through redundancy**
  - **Improved response time through load-sharing**
  - **Delegation of naming authority**
- **Locality of reference applies - users will most often look up names of computers within same organization**

Name resolution

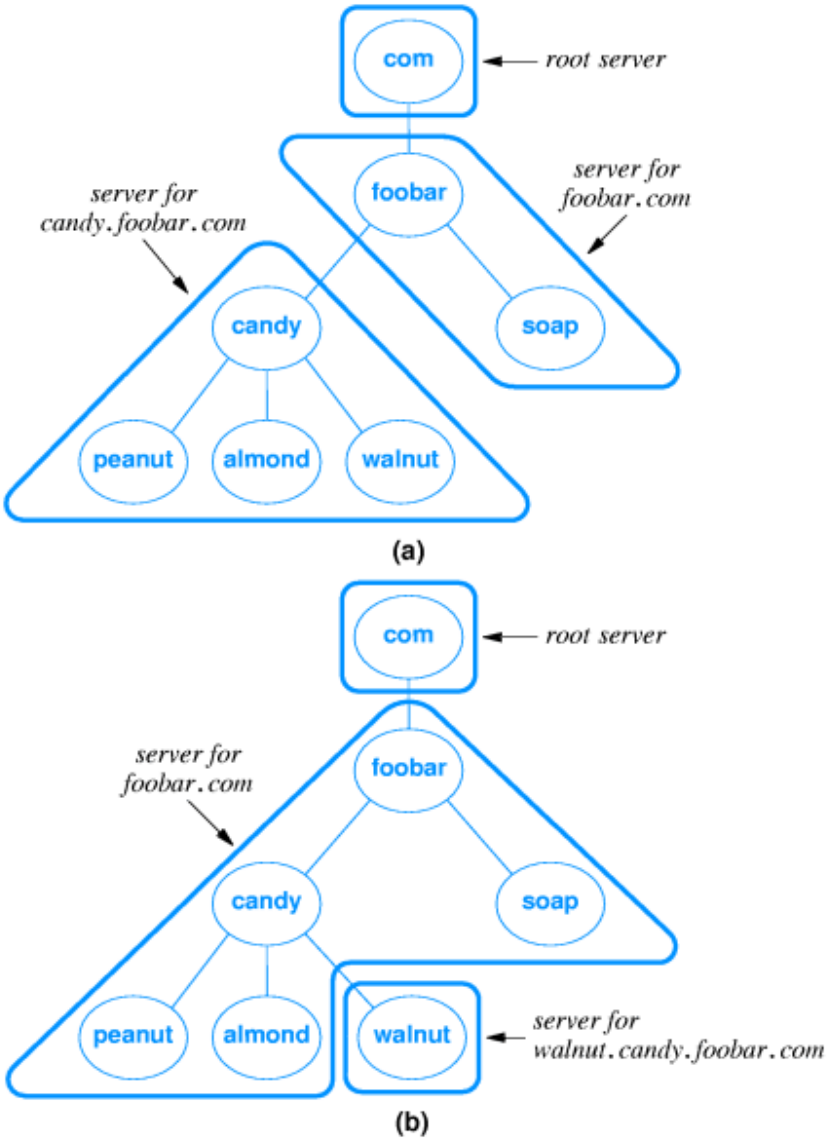
- **Resolver software typically available as library procedures**
  - **Implement DNS application protocol**
  - **Configured for local servers**
  - **Example - UNIX gethostbyname**
- **Calling program is *client***
  - **Constructs DNS protocol message - a *DNS request***
  - **Sends message to local DNS server**
- **DNS *server* resolves name**
  - **Constructs DNS protocol message - a *DNS reply***
  - **Sends message to client program and waits for next request**

DNS messages

- **DNS request contains name to be resolved**
- **DNS reply contains IP address for the name in the request**

DNS servers

- **Each DNS server is the *authoritative server* for the names it manages**
- **If request contains name managed by receiving server, that server replies directly**
- **Otherwise, request must be forwarded to the appropriate authoritative server**



Using DNS servers

- **DNS request is forwarded to root server, which points at next server to use**
- **Eventually, authoritative server is located and IP address is returned**
- **DNS server hierarchy traversal is called *iterative resolution***
- **Applications use *recursive iteration* and ask DNS server to handle traversal**

DNS caching

- **DNS resolution can be** very **inefficient**
  - **Every host referenced by name triggers a DNS request**
  - **Every DNS request for the address of a host in a different organization goes through the root server**
- **Servers and hosts use *caching* to reduce the number of DNS requests**
  - **Cache is a list of recently resolved names and IP addresses**
  - **Authoritative server include *time-to-live* with each reply**

Types of DNS entries

- **DNS can hold several types of records**
- **Each record includes**
  - **Domain name**
  - **Record type**
  - **Data value**
- ***A* records map from domain name to IP address**
  - **Domain name - regulus**
  - **Record type - A**
  - **Data value - 134.82.56.118**
- **Other types:**
  - ***MX* (Mail eXchanger) - maps domain name used as e-mail destination to IP address**
  - ***CNAME* - alias from one domain name to another**
- **Result - name that works with one application may not work with another!**

Abbreviations

- **May be convenient to use abbreviations for local computers; e.g. coral for coral.bucknell.edu**
- **Abbreviations are handled in the *resolver*; DNS servers only know *full-qualified domain names* (FQDNs)**
- **Local resolver is configured with list of suffixes to append**
- **Suffixes are tried sequentially until match found**

Summary

- **Domain Name System maps from computer names and IP addresses**
- **Important to hide 32-bit IP addresses from humans**
- **DNS names are hierarchical and allocated locally**
- **Replication and caching are important performance enhancements**
- **DNS provides several types of records**

Chapter 27 - Electronic Mail

Introduction

- **Many user applications use *client-server* architecture**
- **Electronic mail client accepts mail from user and delivers to server on destination computer**
- **Many variations and styles of delivery**

Electronic mail paradigm

- **Electronic version of paper-based office memo**
  - **Quick, low-overhead written communication**
  - **Dates back to time-sharing systems in 1960s**
- **Because e-mail is encoded in an electronic medium, new forms of interaction are possible**
  - **Fast**
  - **Automatic processing - sorting, reply**
  - **Can carry other content**

Electronic mailboxes

- **E-mail users have an *electronic mailbox* into which incoming mail is deposited**
- **User then accesses mail with a mail reader program**
- **Usually associated with computer account; one user may have a different electronic mailboxes**

E-mail addresses

- **Electronic mailbox is identified by an *e-mail address***
- **Typically user's account ID, although not always**
- **On non-networked multi-user computer, e-mail address is just account ID (no need to identify computer)**

Networked e-mail addresses

- **Mail delivery among networked computers is more complicated**
- **Must identify *computer* as well as *mailbox***
- **Syntactically, e-mail address is composed of computer name and mailbox name**
- **Common example - user@host**
- **Other:**

  - **host1!host2!host!user**
  - **host%user**

Internet mail addressing

- **User portion is site-specific**
  - **droms**
  - **Ralph_E._Droms**
  - **578.4309**
- **Host portion is domain name**
- **Source mail client**
  - **Resolves destination name using DNS (MX, if available)**
  - **Contacts mail delivery server at destination**
  - **Copies mail to server**
- **Destination mail server**
  - **Interprets user name according to local mailbox addresses**
  - **Places mail in appropriate mailbox**

E-mail message format

- **Simple two-part format:**
  - ***Header* includes delivery information**
  - ***Body* carries text of message**
- **Header and body separated by blank line**

E-mail headers

- **Lines of text in format *keyword: information***
- ***keyword* identifies information; information can appear in any order**
- **Essential information:**
  - ***To:* list of recipients**
  - ***From:* sender**
  - ***Cc:* list of copy recipients**
- **Useful information:**
  - ***Reply-to:* different address than *From:***
  - ***Received-by:* for debugging**
- **Frivolous information:**
  - ***Favorite-drink:* lemonade**
  - ***Phase-of-the-moon:* gibbous**

E-mail example

```
From: John_Q_Public@foobar.com
To: 912743.253843@nonexist.com
Date: Wed, 4 Sep 96 10:21:32 EDT
Subject: lunch with me?

Bob,

   Can we get together for lunch when you visit next
week?  I'm free on Tuesday or Wednesday -- just let me
know which day you would prefer.

John
```

E-mail headers

- **Mail software passes unknown headers unchanged**
- **Some software may interpret vendor-specific information**

Keyword	Meaning
From	Sender's address
To	Recipients' addresses
Cc	Addresses for carbon copies
Date	Date on which message was sent
Subject	Topic of the message
Reply-To	Address to which reply should go
X-Charset	Character set used (usually ASCII)
X-Mailer	Mail software used to send the message
X-Sender	Duplicate of sender's address
X-Face	Encoded image of the sender's face

Data in e-mail

- **Original Internet mail carried only 7-bit ASCII data**
- **Couldn't contain arbitrary binary values; e.g., executable program**
- **Techniques for *encoding* binary data allowed transport of binary data**
- **uuencode: 3 8-bit binary values as 4 ASCII characters (6 bits each)**

- **Also carries file name and protection information**
- **Incurs 33% overhead**
- **Requires manual intervention**

MIME

- **Extends and automates encoding mechanisms - *Multipart Internet Mail Extensions***
- **Allows inclusion of separate components - programs, pictures, audio clips - in a single mail message**
- **Sending program identifies the components so receiving program can automatically extract and inform mail recipient**
  - **Header includes:**

MIME-Version: 1.0
Content-Type: Multipart/Mixed; Boundary=Mime_separator

  - **Separator line gives information about specific encoding**
  - **Plain text includes:**

Content-type: text/plain

MIME (continued)

- **MIME is extensible - sender and receiver agree on encoding scheme**
- **MIME is compatible with existing mail systems**
  - **Everything encoded as ASCII**
  - **Headers and separators ignored by non-MIME mail systems**
- **MIME *encapsulates* binary data in ASCII mail envelope**

Programs as mail recipients

- **Can arrange for e-mailbox to be associated with a program rather than a user's mail reader**
- **Incoming mail automagically processed as input to program**
- **Example - mailing list subscription administration**
- **Can be used to implement client-server processing**
  - **Client request in incoming mail message**
  - **Server response in returned mail reply**

Mail transfer

- **E-mail communication is really a two-part process:**
  - **User composes mail with an *e-mail interface* program**
  - ***Mail transfer* program delivers mail to destination**
    - **Waits for mail to be placed in outgoing message queues**
    - **Picks up message and determines recipient(s)**
    - **Becomes *client* and contacts *server* on recipient's computer**
    - **Passes message to server for delivery**

SMTP

- ***Simple Mail Transfer Protocol* (SMTP) is standard application protocol for delivery of mail from source to destination**
- **Provides reliable delivery of messages**
- **Uses TCP and message exchange between client and server**
- **Other functions:**
  - **E-mail address lookup**
  - **E-mail address verification**

SMTP protocol exchange

220 coral.bucknell.edu Sendmail 5.65v3.0 (1.1.8.2/29Aug94-0956AM) Sat, 5 Apr 1997 06:47:12 -0500
HELO regulus.eg.bucknell.edu
250 coral.bucknell.edu Hello regulus.eg.bucknell.edu, pleased to meet you
MAIL FROM: droms
250 droms... Sender ok
RCPT TO: droms
250 droms... Recipient ok

DATA
354 Enter mail, end with "." on a line by itself
This is a test mail message.
.
250 Ok
QUIT
221 coral.bucknell.edu closing connection

Multiple recipients on one computer

- **Suppose droms@bucknell.edu, zaccone@bucknell.edu and hyde@bucknell.edu are all recipients of a mail message**
- **SMTP allows client to specify all three and delvier single copy of message**
- **Server makes three copies for delivery**

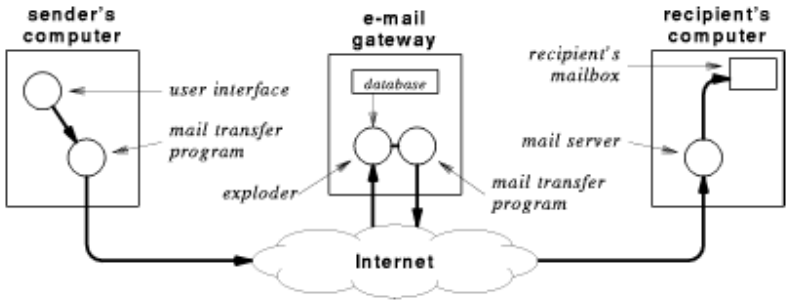
Mailing lists and forwarders

- **E-mail addresses can be attached to programs as well as electronic mailboxes**
- ***Mail exploder* or *mail forwarder* resends copies of message to e-mail addresses in *mailing list***
  - **UNIX mail program sendmail provides simple *mail aliases***
  - **Mailing list processor, e.g., listserv, can also interpret subscription management commands**

List	Contents
friends	Joe@foobar.com, Jill@bar.gov, Tim@StateU.edu, Mary@acollege.edu, Hank@nonexist.com,
customers	george@xyz.com, VP_Marketing@news.com
bball-interest	hank@nonexist.com, Linda_S_Smith@there.com, John_Q_Public@foobar.com, Connie@foo.edu,

Mail gateways

- **Mailing list processing may take significant resources in large organization**
- **May be segregated to a dedicated server computer: *mail gateway***
  - **Provides single mail destination point for all incoming mail**
  - **e.g., bucknell.edu**
  - **Can use MX records in DNS to cause all mail to be delivered to gateway**



Mail gateways and forwarding

- **Users within an organization may want to read mail on local or departmental computer**
- **Can arrange to have mail *forwarded* from mail gateway**
- **Message now makes multiple hops for delivery**
- **Hops may be recorded in header**
- **Forwarded mail may use proprietary (non-SMTP) mail system**

Mail gateways and e-mail addresses

- **Organization may want to use uniform naming for external mail**
- **Internally, may be delivered to many different systems with different naming conventions**
- **Mail gateways can translate e-mail addresses**

Ralph_Droms droms@bucknell.edu
Dan_Little dlittle@mail.bucknell.edu

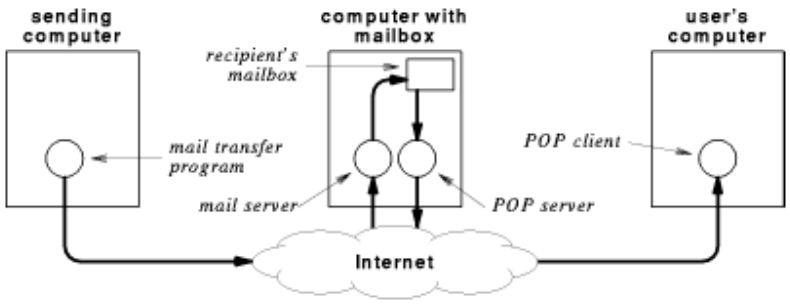
Ruth_Miller miller@charcoal.eg.bucknell.edu

Mailbox access

- **Where should mailbox be located?**
- **Users want to access mail from most commonly used computer**
- **Can't always use desktop computer as mail server**
  - **Not always running**
  - **Requires multitasking operating system**
  - **Requires local disk storage**
- **Can TELNET to remote comptuer with mail server**

Mail access protocols

- **Instead of TELNET, use prtocl that accesses mail on remote computer directly**
- **TCP/IP protocol suite includes *Post Office Protocol* (POP) for remote mailbox access**
  - **Computer with mailboxes runs POP server**
  - **User runs POP client on local computer**
  - **POP client can access and retrieve messages from mailbox**
  - **Requires authentication (password)**
  - **Local computer uses SMTP for outgoing mail**



POP and dialup access

- **POP useful for dialup connection**

- **Users computer not always connected**
- **Can download all mail at once and read off-line**
- **Can compose mail off-line and mail in one connection**

Summary

- ***Electronic mail* based on office memo paradigm**
- **Allows quick, asynchronous communication across entire Internet**
- **Can attach e-mail addresses to programs for processing**
  - **Mailing lists**
  - **Other client-server applications**
- ***Simple Mail Transfer Protocol* (SMTP) is Internet standard for mail delivery**
- **Mail gateways**
  - **Provide uniform user addressing outside organizations**
  - **Translate from Internet mail (e.g., SMTP) to proprietary systems**
- ***Post Office Protocol* (POP) allows remote access to electronic mailboxes**

Chapter 28 - File Transfer and Remote File Access

Introduction

- **Many programs written to use *disk file* paradigm for I/O**
- **Moving a file from one computer to another required removable medium and sneakernet**
- **Network allows direct communication**
  - ***File transfer* - equivalent of tape, floppy transfer**
  - ***Remote file system* - access to files on networked computer through same interface as local files**

Two problems

- **Coordinating scheduling of distributed computations**
- **Saving intermediate results**
- **File transfer paradigm - programs write intermediate results to disk file**
  - **Components of distributed application need not be run concurrently**
  - **Intermediate results can be used to restart failed computation**

Generalized file transfer

- **Allow transfer of arbitrary files**
- **Accommodate different file types**
- **Convert between heterogeneous systems**
  - **Data types**
  - **Word lengths**
  - **Rules for file names**
- **User login**

Interactive and batch transfer

- **Batch transfer**
  - **User creates list of files to be transferred through interface program**
  - **Request dropped in queue**
  - **Transfer program reads requests and performs transfers**
  - **Transfer program retries until successful**
  - **Good for slow or unreliable transfers**
- **Interactive transfer**
  - **User starts transfer program**
  - **Actions include listing contents of directories, transferring files**
  - **User can find and transfer files immediately**
  - **Quick feedback in case of, e.g., spelling errors**

File transfer Protocol

- **TCP/IP standard is *File Transfer Protocol* (FTP)**
- **General purpose protocol**
  - **Operating system and hardware independent**
  - **Transfers arbitrary files**

  - **Accommodates file ownership and access restrictions**
- **Predates TCP/IP; adapted to TCP/IP later**

Model and interface

- **Underlying protocol can run either *interactive* or *batch* mode**
  - **`ftp` client gives interactive interface**
  - **MIME, HTTP can use directly**
- **Protocol actions include:**
  - **List contents of directory**
  - **Change to a different working directory**
  - **Retrieve a file**
  - **Put a file**

`ftp` client commands

!	cr	macdef	proxy	send
$	delete	mdelete	sendport	status
account	debug	mdir	put	struct
append	dir	mget	pwd	sunique
ascii	disconnect	mkdir	quit	tenex
bell	form	mls	quote	trace
binary	get	mode	recv	type
bye	glob	mput	remotehelp	user
case	hash	nmap	rename	verbose
cd	help	ntrans	reset	?
cdup	lcd	open	rmdir	
close	ls	prompt	runique	

- **`ftp` client interface from BSD UNIX is *de facto* standard**
  - **Many commands archaic and no longer used: `tenex`, `carriage control`**
  - **Most often used: `cd`, `dir`, `ls`, `get`, `put`**
  - **Other useful: `cr`, `pwd`, `lcd`**
- **Two-step process:**
  - **Launch `ftp`**
  - **Connect to remote host**
    - **Connect involves using user account on remote host**
    - **Some FTP servers provide *anonymous FTP***

Two-way file transfer

- **`get`: *from* FTP server to local host**
- **`put`: *to* FTP server from local host**
- **Default uses same name on both hosts; `ftp` client allows specification of different names**
- **`mget`, `mput` transfer multiple files**
  - **UNIX-like wildcard expansion**
  - **`prompt` disables prompt for batch transfer**

File name translation

- **File name syntaxes may be incompatible**
- **UNIX - 128 character, mixed case; DOS - 8+3 character, upper case**
- **Some names may not be legal in all systems**
- **BSD `ftp` allows rules for filename translation**

File types and transfer modes

- **Many different styles of file typing**
  - **UNIX - untyped; may hold anything**
  - **MacOS - strongly typed**
- **`ftp` does two types of transfer:**
  - **Text - with appropriate translations to maintain integrity**
  - **Binary - *no* translation whatsoever**

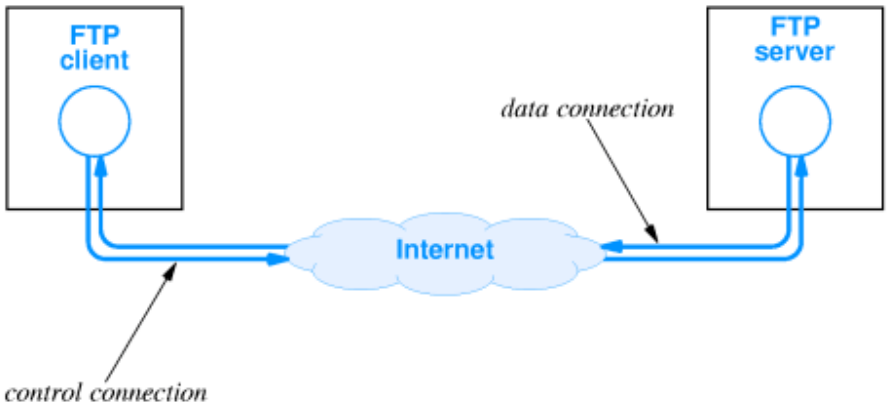
FTP messages

- **Each message from server includes a three-digit decimal number**
  - **`226 Transfer complete`**
  - **`221 Goodbye`**
- **Convenient for computer and human recognition**
- ***Verbose* mode shows messages; *quiet* mode suppresses messages**

FTP client-server model

- **Remote *server* accepts *control connection* from local *client***
  - **Client sends commands to server**
  - **Persists through entire session**
- ***Server* creates *data* connection for data transfer**
  - **One data connection for each transferred file**

- **Data transferred (either way)**



Using separate data connections

- **Separates commands from data**
- **Client can send commands during data transfer**
- **Closed connection indicates end of file**

TFTP

- ***Trivial File Transfer Protocol* (TFTP) - much simpler than FTP**
  - **Based on UDP**
  - **File transfer only; no directory listing**
  - **No authorization**
- **Can be used in UDP-only system**
- **Requires less code than FTP**
- **Often used for bootstrap; e.g., ROM-based diskless system**

NFS

- ***Network File System* gives random access to files across a network**
- **Many distributed file systems exist; NFS is TCP/IP standard**
  - **AppleShare**
  - **Microsoft/NETBIOS**
- **Based on client-server model**

- **Originally developed by Sun; implementations now available on most UNIX and many PC systems**

NFS function

- **Provides functions equivalent to OS access to local files**
  - **Open**
  - **Read**
  - **Write**
  - **Close**
- **Local file access operations mapped to network messages**
  - **Each message contains file system operation**
  - **Read/write carry one disk block in each message**
- **File naming integrated into local directory system**
  - **Remote file systems *mounted* onto local directory**
  - **Access through *mount point* mapped to server**

NFS implementation

- **NFS operations are UNIX-like, but not exactly equivalent**
  - **Local OS performs some operations - open, close**
  - **NFS provides block read/write; local OS does buffering**
- **Other OS file functions can also be mapped into NFS operations**
- **File naming, authorization/account identification, access rights all problematic**

Summary

- **FTP - whole file transfer using TCP between Internet hosts**
  - **Directory listing**
  - **Data translation**
- **TFTP - whole file transfer using UDP between Internet hosts**
  - **File transfer only**
  - **No authorization**
- **NFS - file-level access using UDP**

Chapter 29 - WWW

Introduction

- **Hypertext model**
- **Use of hypertext in *World Wide Web* (WWW)**
- **WWW client-server model**
- **Use of TCP/IP protocols in WWW**

Hypertext/hypermedia

- ***Hypermedia* system allows interactive access to collections of documents**
- **Document can hold:**
  - **Text (*hypertext*)**
  - **Graphics**
  - **Sound**
  - **Animations**
  - **Video**

- **Documents linked together**
  - **Nondistributed - all documents stored locally (like CD-ROM)**
  - **Distributed - documents stored on remote servers**

Hypermedia pointers

- **Each document contains *links* (pointers) to other documents**
  - **Link represented by "active area" on screen**
    - **Graphic - button**
    - **Text - highlighted**
  - **Selecting link fetches referenced document for display**
- **Links may become invalid**
  - **Link is simply a text name for a remote document**
  - **Remote document may be removed while name in link remains in place**

Browser interface

- **Interactive, "point-and-click" interface to hypermedia documents**
- **Each document is displayed in screen**
- **User can select and follow links - "point-and-click"**
- **Application is called a *browser* (infinite time sink)**

Document representation

- **Each WWW document is called a *page***
- **Initial page for individual or organization is called a *home page***
- **Page can contain many different types of information; page must specify**
  - **Content**
  - **Type of content**
  - **Location**
  - **Links**
- **Rather than fixed WYSIWYG representation (e.g., Word), pages are formatted with a *mark up language* (like TeX)**
  - **Allows browser to reformat to fit display**
  - **Allows text-only browser to discard graphics**
- **Standard is *HyperText Markup Language* (HTML)**

HTML

- **HTML specifies**
  - **Major structure of document**
  - **Formatting instructions**
  - **Hypermedia links**
  - **Additional information about document contents**
- **Two parts to document:**
  - ***Head* contains details about the document**
  - ***Body* contains information/content**
- **Page is represented in ASCII text with embedded HTML *tags* formatting instructions**
  - **Tags have format <TAGNAME>**
  - **End of formatted section is </TAGNAME>**

Example

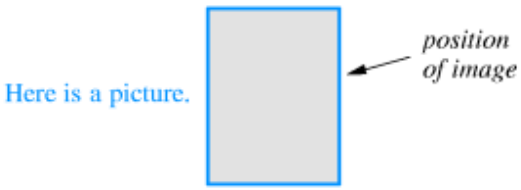
```
<HTML>
<HEAD>
  <TITLE>
    CS363 - Example Page
  </TITLE>
</HEAD>
<BODY>
<HR>
Lecture notes for today go here!
<HR>
<CENTER>
<TABLE BORDER=3>
  <TR>
  <TD><A HREF="./chap27_6.html">Previous</A>
  <TD><A HREF="./chap27_8.html">Next</A>
  <TD><A HREF="./chap27_0.html">Top of chapter</A>
  <TD><A HREF="../lecture_notes.html">Lecture notes</A>
  <TD><A HREF="../../index.html">Top of CS363</A>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

Other HTML Tags

- **Headings - <H1>, <H2>**
- **Lists**
  - **<OL> - Ordered (numbered) list**
  - **<UL> - Unordered (bulleted) list**
  - **<LI> - List item**
- **Tables**
  - **<TABLE>, </TABLE> - Define table**
  - **<TR> - Begin row**
  - **<TD> - Begin item in row**
- **Parameters**
  - **Keyword-value pairs in HTML tags**
  - **<TABLE BORDER=3>**

Embedded graphics

- **IMG tag specifies insertion of graphic**
- **Parameters:**
  - **SRC="filename"**
  - **ALIGN= - alignment relative to text**
- **Image must be in format known to browser, e.g., *Graphics Interchange Format* (GIF)**



Identifying a page

- **Page identified by:**
  - **Protocol used to access page**
  - **Computer on which page is stored**
  - **TCP port to access page**
  - **Pathname of file on server**

- **Specific syntax for *Uniform Resource Locator* (URL): *protocol://computer_name:port/document_name***
  - **Protocol can be http, ftp, file, mailto**
  - **Computer name is DNS name**
  - **(Optional) port is TCP port**
  - **document_name is path on computer to page**

Links between HTML documents

- **Each link is specified in HTML**
- **Item on page is associated with another HTML document**
- **Link is *passive*; no action taken until selected**
- **HTML tags are <A> and </A>**
  - **Linked document specified by parameter: HREF="document URL"**
  - **Whatever is between HTML tags is highlighted item**
- **<A HREF="http://www.bucknell.edu/~droms">Your obdt. svt.</A>**
- **Your obdt. svt.**

Client-server model

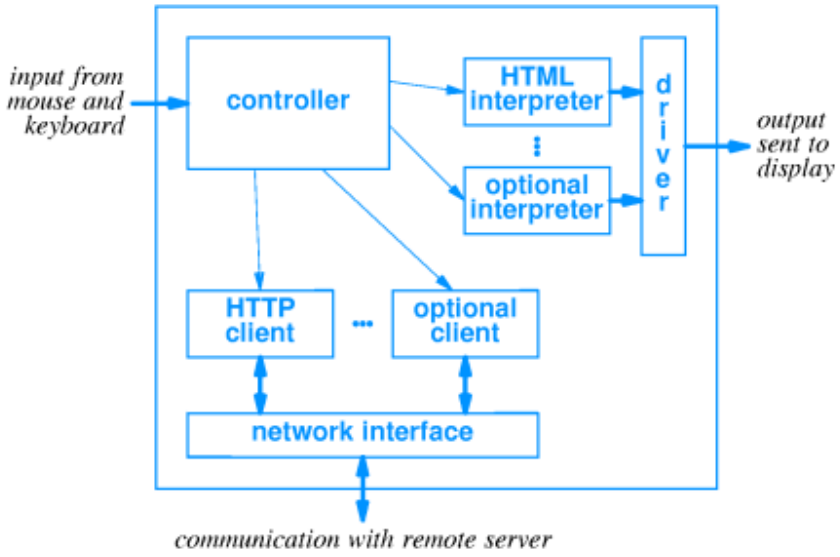
- ***Browser* is client, *WWW server* is server**
- **Browser:**
  - **Makes TCP connection**
  - **Sends request for page**
  - **Reads page**
- **Each different item - e.g., IMG - requires separate TCP connection**
- ***HyperText Transport Protocol* (HTTP) specifies commands and client-server interaction**

Server architecture

- **Much like robowar or ftp server**
  - **Waits for incoming connection**
  - **Accepts command from connection**
  - **Writes page to connection**
- **Performance is hard issue**

Browser architecture

- **Browser has more components:**
  - **Display driver for painting screen**
  - **HTML interpreter for HTML-formatted documents**
  - **Other interpreters (e.g., Shockwave) for other items**
  - **HTTP client to fetch HTML documents from WWW server**
  - **Other clients for other protocols (e.g., ftp)**
  - **Controller to accept input from user**
- **Must be multi-threaded**



Caching in browsers

- **Downloading HTML documents from servers may be** slow
  - **Internet congested**
  - **Dialup connection**
  - **Server busy**
- **Returning to previous HTML document requires reload from server**
- **Local *cache* can be used to hold copies of visited pages**
- **Also can implement organizational *HTTP proxy* that caches documents for multiple users**

Summary

- **WWW is based on *hypermedia***
- **HTML is *markup* language for WWW documents**
- **HTML can specify links to other documents**
- **WWW based on client-server model**
  - **Browser - client**
  - **WWW server - server**

Chapter 30 - CGI and Dynamic Web Documents

Introduction

- **Documents in previous section are *static***
  - **Defined in text file by page author**
  - **Remains unchanged until edited by author**
- ***Dynamic* documents are generated on demand by HTTP server**
- ***Active* execute code on the WWW browser host computer**

Document types

Document type	Description	Advantages/disadvantages
Static	• **Text file on server** • **Edited by author** • **Contents unchanged (until edited)**	• **Easy to generate** • **Fast to display** • **Inflexible**
Dynamic	• **Program on server** • **Output returned to client** • **Client displays as HTML page**	• **Can generate contents at time of access** • **Harder to author** • **Must be refreshed to update display**

Active	• **Consists of a program that is downloaded to browser**
	• **Browser executes program**
	  o **Interacts with user through keyboard**
	  o **Updates display**
	  o **Can read files and contact other Internet services**
		• **Browser must be able to run program**
		• **Program must be platform independent**
		• **Requires more complexity in browser**
		• **Standards not fixed (Java is *de facto* standard**
		• **Requires programming skill (typically more than dynamic)**

Dynamic documents and servers

- **Support for dynamic documents requires some changes on HTTP server**
- **Typically implemented as extensions to existing server**
- **Server must be able to execute program that generates dynamic document**
  - **Can be implemented in**
    - **Shell/csh script**
    - **C program (executable)**
    - **Others?**
  - **Output returned to browser (looks like contents of static page)**

- **Each dynamic document needs separate program**
- **Server must differentiate between static and active document references**

CGI standard

- ***Common Gateway Interface* (CGI) standard defines server-application interaction**
- **Program sometimes called *CGI program***
- **Guidelines for interaction**
  - **Language**
  - **Parameter passing**

Output from CGI program

- **Output from CGI program routed to WWW browser**
- **Can be in several formats, e.g., plain text or HTML**
- **Program must identify content to server for relay to browser**
- **Header - consists of lines of text followed by blank line**

```
Content-type: text/html
Content-type: text/plain
Location: http://other.server.com/new.txt
```

CGI example

- **Here's an example**

A simple CGI program

- **Try it again...**
- **Uses shell script language**
  - **`echo` prints its command line arguments**
  - **`date` returns date**
  - **`` `foo` `` returns output from command `foo` as command line argument**

```
#!/bin/sh

#
# CGI script that prints the date and time at which it was run
#

# Output the document header followed by a blank line

echo Content/type: text/plain
echo

# Output the date

echo This document was created on `date`
```

- **Linked to with:**

```
<a href="http://www.netbook.cs.purdue.edu/cgi-bin/netbook-
28-1">
A simple CGI program</a>
```

Inputs to CGI programs

- **Server passes inputs to CGI programs through parameters**
- **Browser may supply parameters (through server)**
  - **Browser extends URL with additional parameters**
  - **Separated by "?"**
- **Parameters passed as *environment variables***
  - **Essentially *keyword-value* pairs**
  - **Based on original UNIX servers**
- **Very easy to access in shell script**

Name of Variable	Meaning
SERVER_NAME	The domain name of the computer running the server.
GATEWAY_INTERFACE	The version of the CGI software the server is using.
SCRIPT_NAME	The path in the URL after the server name.
QUERY_STRING	Information following "?" in the URL.
REMOTE_ADDR	The IP address of the computer running the browser that sent the request.

State information

- **CGI program invoked identically at each reference**
- **Server maintains no history of previous invocations or any other state**
- **Long-term - store in file on server**
- **Short-term - return to browser in URL**

CGI program with long-term state

- **Keeps record of IP addresses**
- **Responds with initial message at first reference; subsequent message otherwise**
- **Try me!**
- **Try it again...**
- **Implementation details**
  - **List of known addresses kept in file `ipaddrs`**
  - **`grep -s` returns `TRUE` if argument string *not* found in file**
  - **Environment variables `$REMOTE_ADDR` contains string with IP address of browser host**

CGI program with short-term state

- **Trick is to encode state in URL, return URL in link**
- **When user selects link, state returned as parameter suffix in new URL**
- **Script can parse `$QUERY_STRING` to retrieve state**
- **Try me!**
- **Try it again...**
- **Implementation details**

- **N is internal variable**
- **case statement allows for null string**
- **;; is end of case**

```
#!/bin/sh

echo Content-type: text/html
echo

N=$QUERY_STRING
echo "<HTML>"

case "x$N" in

x)      N=1
        echo "This is the initial page.<BR><BR>"
        ;;

x[0-9]*) N=`expr $N + 1`
        echo "You have refreshed this page $N times.<BR><BR>"
        ;;

*)      echo "The URL you used is invalid.</HTML>"
        exit 0
        ;;

esac
echo "<A HREF=\"http://$SERVER_NAME$SCRIPT_NAME?$N\">"
echo "Click here to refresh the page.</A> </HTML>"
```

Forms and interactions

- **CGI includes support for *forms***
- **Server sends document to browser identified as a form**
- **Form includes items for information entry**
  - **Each form item has a name**
  - **Contents of that item encoded in query string:**

```
?ITEM1=4,ITEM2=Bucknell,ITEM3=CS363
```

Summary

- **Three types of WWW documents**
  - ***Static***
  - ***Dynamic***
  - ***Active***
- **Dynamic document**
  - **Generated by program on server at each reference**
  - **May display different contents over time**
  - **Must be *refreshed* to update display**
- **May include *long-term* and *short-term state***

Chapter 31 Java Technology For Active Web Documents

Section **Title**

Introduction

- ***Active* documents consist of code executed on computer running browser**
- ***Java* language allows development of active document programs**
  - **Programs called *applets***
  - **Platform independent**
  - **Secure**

Continuous update through server push

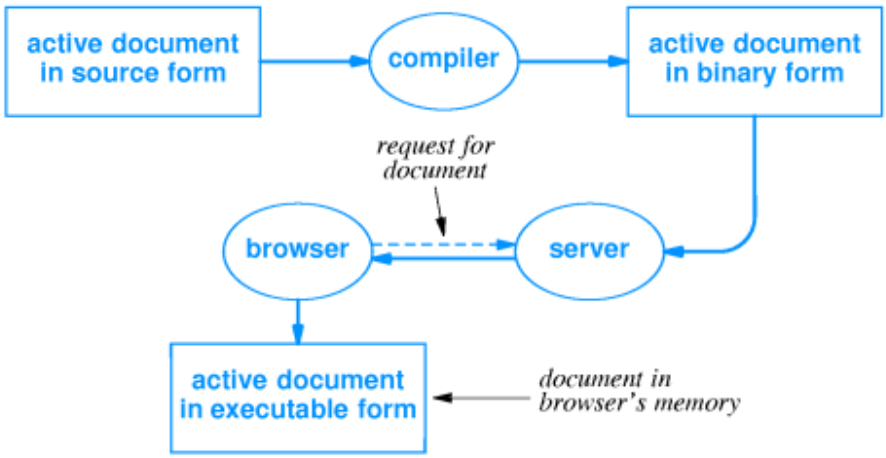
- **Some servers will send new versions of document**
- **Technique called *server push***
- **Each server push document requires dedicated server resources**
- **May scale with number of clients**
- **Also requires network bandwidth for each update**

Active documents

- **Delegates responsibility for updates to browser client**
- **Work scales with number of active documents on *client***
- **Active document can, in fact, incur less server overhead than dynamic document**

Representing and executing active documents

- **Requires programming language; what should that language look like?**
- **How should the language be represented for execution?**



- **Multiple implementations possible; standards necessary for coordination**

Java

- **Technique for writing, compiling, downloading and executing active documents**
- **Includes:**
  - **Programming language**
  - **Runtime environment**
  - **Class library**

Java language

- **Resembles C++**
  - **Object-oriented**
  - **Some C++ cruft excised or restricted**
- **Characteristics:**
  - **High level, general purpose, object oriented**
  - **Dynamic**
  - **Strongly typed, static type checking**
  - **Concurrent**

Java run-time environment

- **Interpretive execution - Java language compiled into *bytecodes***
- **Automatic garbage collection**
- **Multithreaded execution**
- **Internet access**
- **Graphics**

Portability

- **Java must be platform-independent**
- **Run-time environment clearly defined and has no implementation dependencies**
- **Bytecode representation is platform independent**

Java library

- **Library provides collection of common functions for Java applets**
- **Class definitions and methods**
- **Classes:**
  - **Graphics**
  - **Low-level network I/O (socket-level)**
  - **Web server interaction**
  - **Run-time system calls**
  - **File I/O**
  - **Data structures**
  - **Event capture - user interaction**
  - **Exception handling**

AWT graphics

- **Java graphics library called *Abstract Window Toolkit* (AWT)**
- **Includes high-level and low-level facilities**
  - **Windows with components - scrollbars, buttons**
  - **Blank rectangular area with object drawing**
- **Implements abstract functions; run-time environment implementation maps to window system-specific functions**

Java and browsers

- **Browser must include Java interpreter**
- **Java interpreter works through browser**
  - **Graphics**
  - **HTML**
- **Interpreter also works through native operating system**
  - **File I/O**
  - **Network operations**

Compiling a Java program

- **javac translates Java source code into bytecodes**
  - **Checks for correct syntax**
  - **Imports classes from library**
  - **Writes bytecode program to *filename*.class**
- **Other development environments exist**
  - **May include source code management**
  - **"Visual" systems provide "pluggable" modules**

An example applet

NerdClock

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap31/chap31_13.html

Shockwave

A Java example

- **Example code...**

```
import java.applet.*; import java.awt.*;

public class clickcount extends Applet {
 int count;
 TextField f;

 public void init() {
  count = 0;
  add(new Button("Click Here"));
  f = new TextField("The button has not been clicked at all.");
  f.setEditable(false);
  add(f);
 }

 public boolean action(Event e, Object arg) {
  if (((Button) e.target).getLabel() == "Click Here") {
    count += 1;
    f.setText("The button has been clicked " + count + " times.");
  }
  return true;
 } }
```

Running the example

<applet code="clickcount.class" height=50 width=400> </applet>

Shockwave

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap31/chap31_15.html

Interacting with the browser

```
import java.applet.*; import java.net.*; import java.awt.*;

public class buttons extends Applet {

 public void init() {
  add(new Button("Ying"));
  add(new Button("Yang"));
 }

 public boolean action(Event e, Object arg) {
  if (((Button) e.target).getLabel() == "Ying") {
   try {
    getAppletContext().showDocument(new
    URL("http://www.nonexist.com/ying"));

   }
   catch( Exception ex ) {
   // note: code to handle the exception goes here //
   }
  }
  else if (((Button) e.target).getLabel() == "Yang") {
   try {
    getAppletContext().showDocument(new
    URL("http://www.other.com/yang"));
   }
   catch( Exception ex ) {
   // note: code to handle the exception goes here //
   }
  }
  return true;
 }

}
```

Running the example

http://www.eg.bucknell.edu/~cs363/lecture_notes/chap31/chap31_17.html

Alternatives

- **JavaScript**
  - **Interpreted scripting language**
  - **Like csh for browser**
- **Other languages compiled into Java bytecodes**
- **Other programming technologies - Inferno**

JavaScript example

```
<SCRIPT LANGUAGE="JavaScript">
<!--
var loc = location.href.toString()
var ep = loc.lastIndexOf("/")
var dirloc = ""
if (ep > 0) {
            dirloc = loc.substring(0, ep)
            }
function doform(form)
{
// Find next file
    var URL
    URL = dirloc + "/page20b.htm"
// Some problems getting background to look nice...
    parent.frames[2].document.open()
    parent.frames[2].document.clear()
    parent.frames[2].document.writeln('<HTML><HEAD></HEAD><BODY
BGCOLOR="#FFFFFF">')
    parent.frames[2].document.writeln("<DL>")
//search stuff goes here
    URL = URL + "?" + form.keyword.value
    parent.frames[0].location.href = URL
}
// some hackery to force the bottom frame to be empty every
// time the page is reloaded
    var URL
    var loc = location.href.toString()
    var ep = loc.lastIndexOf("/")
    URL = dirloc + "/page20y.htm"
// force frame 0 to empty frame so re-execution is a no-op
    parent.frames[0].location.href = URL
// -->
</SCRIPT>
```

Summary

- ***Active* documents execute code in browser on user's computer**
- ***Java* is most widely used active document technology**
- **Java consists of:**
  - **Programming language**
  - **Run-time environment**
  - **Class library**
  - **Tags in browser for Java program invocation**

Chapter 32 - RPC and Middleware

Introduction

- ***Client-server* model most often used for networked applications**
  - **Fits well with program execution model**
  - **Modular**
- **May not be easy to program**
  - **Model not intuitive to coding experience**
  - **Lots of details to manage**

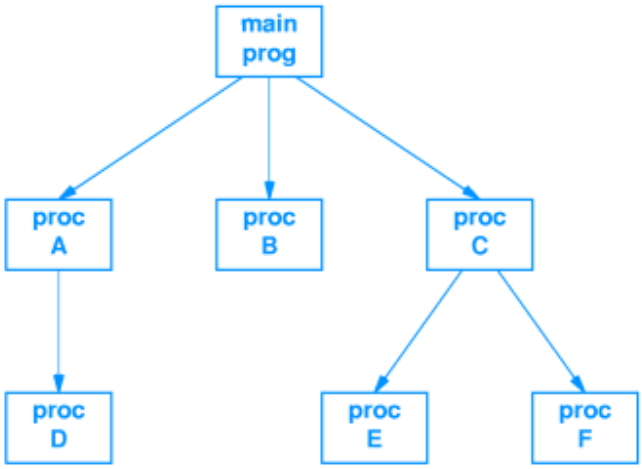
Tools for networked applications

- **Lots of details to manage**
  - **Connections between components of networked application**
  - **Synchronization**
  - **Robust data exchange**
  - **Data conversions**
  - **Error conditions**
- **Many details similar or identical in different networked applications**
- **Idea: use tools to handle routine parts of interface**

Programming with procedures

- **Modularizes code**
  - **Reusable components**
  - **Procedures operate on parameters**
- **Procedures may call other procedures**

Procedure call graph



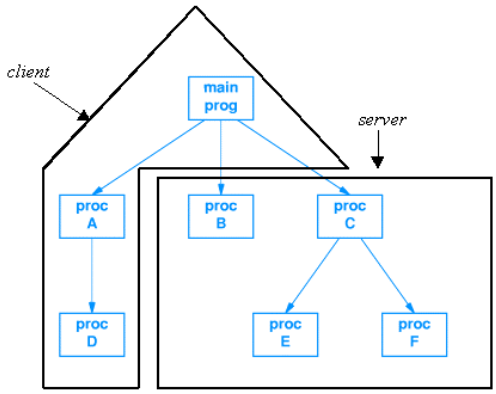
Remote Procedure Call

- ***Remote Procedure Call* (RPC) model provides structure for development of networked applications**
  - **Based on procedure call model in familiar languages**
  - **Related to client-server model**
- **Hides details of communication and synchronization beneath procedure call interface**

RPC paradigm

- **Network communication hidden by procedure calls**
  - **Procedure *may* be executed on different computer**
  - **RPC mechanism hides details**
- **Programmer decides which procedures are executed where**

RPC call graph

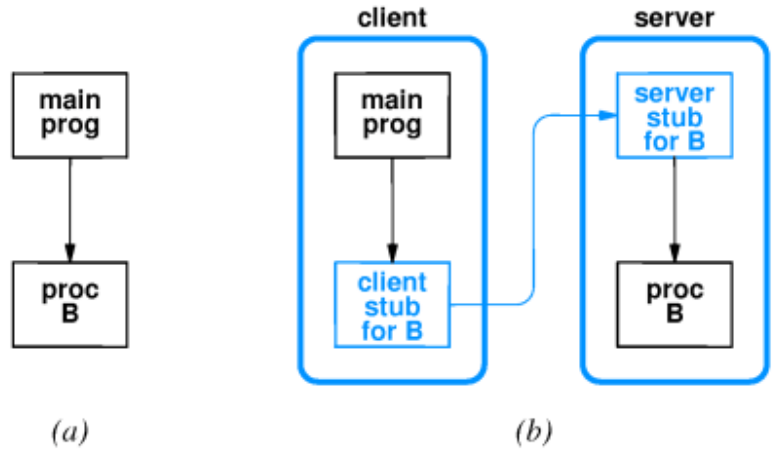


What does RPC mechanism have to do?

- **Caller:**
  - ***Marshal* arguments**
  - **Transmit procedure identifier and arguments to remote procedure**
  - **Wait for response**
- **Called procedure:**
  - **Unpack arguments**
  - **Execute procedure**
  - **Reply with any returned value**

Where is this work done?

- ***Client stub* with name of remote procedure on client**
- ***Server stub* "spoofs" caller on server**



External data representation

- **Machine-dependent data representations**
  - **Integer byte ordering**
  - **Floating point**
  - **Character strings (ASCII - EBCDIC; null-terminated - length encoded)**
- ***External data representation*: common format for data exchange**

Middleware and Object-oriented Middleware

- ***Middleware*: tools for generating RPC-based applications**
- ***Interface Definition Language* (IDL)**
  - **Defines specifications for procedure interfaces**
  - **Used by client and server programmers**
- **Object-oriented middleware: remote invocation of object *methods***

ONC RPC

- ***Open Network Computing Remote Procedure Call* (ONC RPC)**
  - **Designed by Sun Microsystems**
  - **Early example of middleware**
- **Used in many Sun applications; e.g. NFS**
- **Includes *eXternal Data Representation* (XDR) standard**

DCE RPC

- ***Open Software Foundation* (OSF) defined *Distributed Computing Environment* (DCE)**
- **Includes *DCE RPC* as middleware component**
- **Defines its own IDL**
- **Microsoft derived *Microsoft Remote Procedure Call* (MSRPC) from DCE RPC**

CORBA

- ***Common Object Request Broker Architecture* (CORBA)**
  - **Developed by *Object Management Group* (OMG)**
  - **Vendor-independent, interoperable spec for middleware**
- **Remote invocation instantiated by local object proxies**
  - **Proxies instantiated at runtime**
  - **Methods passed to "real" remote object**

Summary

- **Client-server programming may be difficult due to details and synchronization**
- **Procedure call model is "natural" paradigm for programmers**
- ***Remote Procedure Call* (RPC) uses procedure call paradigm for client-server communication**
  - **Client sends parameters for procedure call to server**
  - **Server executes procedure and replies with returned value**
- ***Middleware* is category of development tools for RPC**

Chapter 33 - Network Management: SNMP

Introduction

- **Network management is a** hard **problem**
- **Will discuss network management paradigm base on network communication and client-server model**
- **SNMP is TCP/IP standard**

Internet Management

- ***Network manager* or *network administrator* is responsible for monitoring and controlling network hardware and software**
  - **Designs and implements efficient and robust network infrastructure**
  - **Identifies and corrects problems as they arise**
  - **Must know both hardware and software**
- **Why is network management hard?**
  - **Most internets *heterogeneous***
  - **Most internets *large***

Types of problems

- **Catastrophic**
  - **Fiber broken by backhoe**
  - **LAN switch loses power**
  - **Invalid route in router**
  - ***Easiest* to diagnose**
- **Intermittent or partial**
  - **NIC sends frames too close together**
  - **Router has one invalid entry**
  - ***Hardest* to diagnose**

Problem with hidden failures

- **Some intermittent of partial failures may not be evident to user**
  - **Hardware may drop frames with data errors**
  - **Network protocols may recover from lost packet**
- **However, network performance decreases**

Network management software

- **Monitor operation and performance of network devices:**
  - **Hosts**
  - **Routers**
  - **Bridges, switches**
- **Control operations through rebooting, changing routing table entries**

Network management model

- **Network management does *not* have an internet or transport layer protocol**
- **Defines application layer protocol using TCP/IP transport layer protocol**
- **Based on client-server model; names changes**
  - ***Manager* == client; run by network manager**
  - ***Agent* == server; runs on managed device**
- **Manager composes requests for agent; agent composes response and returns to manager**

SNMP

- **TCP/IP standard is *Simple Network Management Protocol* (SNMP)**
- **Defines all communication between manager and agent**
  - **Message formats**
  - **Interpretation of messages**
  - **Data representation**

SNMP data representation

- **SNMP uses *Abstract Syntax Notation.1* (ASN.1)**
  - **Platform-independent data representation standard**
  - **Strongly-typed**
  - **Can accommodate arbitrary data types**
- **Example - integer representation**
  - **Length octet - number of octets containing data**
  - **Data octets - value in big-endian binary**

decimal integer	hexadecimal equivalent	length octet	octets of value (in hex)
27	1B	01	1B
792	318	02	03 18
24,567	5FF7	02	5F F7
190,345	2E789	03	02 E7 89

Fetch-store paradigm

- **Manager-agent interaction based on *fetch-store* paradigm**
  - ***Fetch* retrieves a value from the agent**
  - ***Store* changes a value on the agent**
  - **Any other information is extracted from the *fetch*ed data and displayed by the manager**
- ***Fetch* used to monitor internal data values and data structures**
- ***Store* used to modify and control data values and data structures; also used to control behavior by setting "reboot" object**

SNMP operations

- ***Get* (fetch) retrieves value of object**
- ***Set* (store) stores new values into object**
- ***Get-next* retrieves *next* object (for scanning)**

Identifying objects with SNMP

- **SNMP is not tied to any particular set of data structures**
- **Operates on a collection of related objects identified in a *Management Information Base* (MIB)**
- **Objects in a MIB are identified by ASN.1 naming scheme**
  - **Hierarchical naming structure**
  - **Authority for new names delegated as in DNS**
- **Example - count of incoming IP datagrams:**

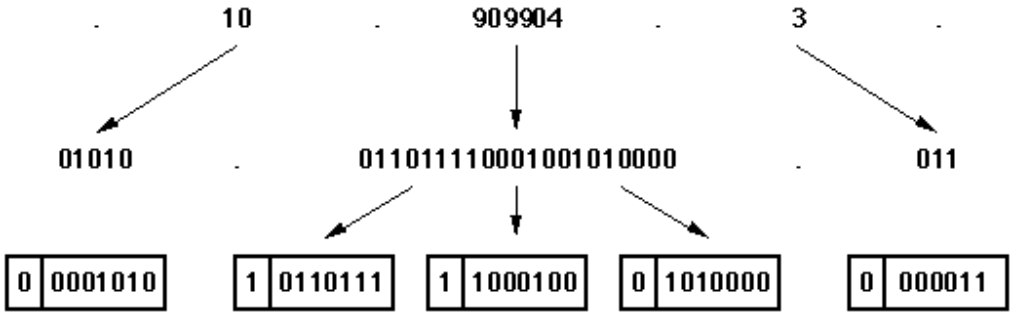
```
iso.org.dod.internet.mgmt.mib.ip.ipInReceives
```

- **For efficiency, each name has a numeric equivalent; e.g.:**

```
1.3.6.1.2.1.4.3
```

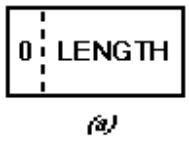
Storing ASN.1 numeric values

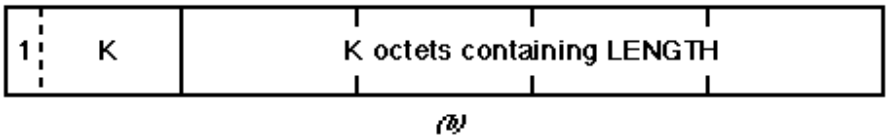
- **Value stored in sequence of octets**
- **Leftmost bit is 0 in *last* octet**
- **Example:**



Storing ASN.1 lengths

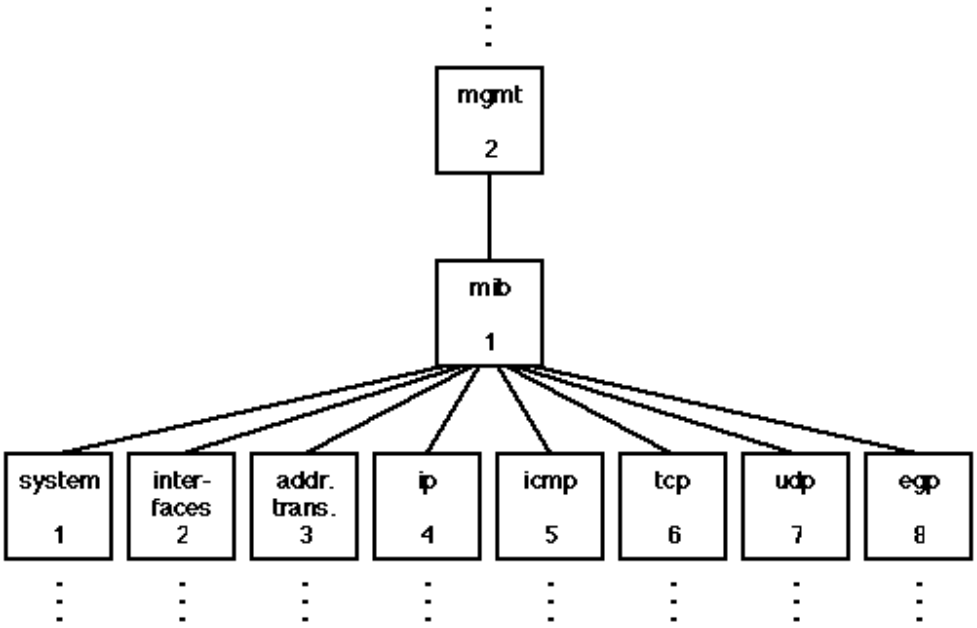
- **Leftmost bit 0 means length in same octet**
- **Leftmost bit 1 means length in *k* octets**





Types of MIBs

- **Very flexible structure**
- **MIBs defined for protocols, devices, network interfaces**
- ***MIB I* is original TCP/IP standard for protocol suite; *MIB II* extends that original version**



Arrays in MIBs

- **Some types of data - such as a routing table - is most naturally stored as an array**
- **ASN.1 supports variable length, associative arrays**
  - **Number of elements can increase and decrease over time**
  - **Each element can be a structured object**
- **Indexing is implicit**
  - **Manager must know object is an array**
  - **Manager must include indexing information as suffix**

Array example

- **Routing table is an array:**

`ip.ipRoutingTable`

- **List of routing table entries is indexed by IP address**
- **To identify one value:**

`ip.ipRoutingTable.ipRouteEntry.ipRouteNextHop.IPdestaddr`

  - **`ipRouteEntry` indicates indexing**
  - **`ipRouteNextHop` is a field in a routing table entry**
  - **`IPdestaddr` is 32-bit IP address**

Summary

- **TCP/IP includes *SNMP* as network management protocol**
- **SNMP is an *application protocol* that uses UDP for transport**
- **Based on *fetch-store* paradigm**
  - **Controls operation as side-effect of *store* operations**
  - ***Get-next* used top scan objects**
- ***Management Information Base* (MIB) defines structure of objects**
- ***Abstract Syntax Notation.1* (ASN.1) used for data representation and object identification**

Chapter 34 - Network security

Introduction

- **Routers forward packets - from *any source***
- **Bad guys can send in packets from outside**
- **How to avoid security breaches?**

Secure networks and security policies

- **Can't describe a network as *secure* in the abstract**
- **University may have different notion of security than military installation**
- **Must define a *security policy***
- **Many possibilities to consider:**
  - **Data stored on servers**
  - **Messages traversing LANs**
  - **Internal or external access**
  - **Read/write versus read-only access**

Components of security policy

- **Describes items to be protected and rules for protection**
- **Must cover computer systems, LANs, interconnection devices, ...**
- **Development must include assessment of cost of protected information versus cost of protection**

Aspects of security

- **Data accessibility - contents accessible**
- **Data integrity - contents remain unchanged**
- **Data confidentiality - contents not revealed**

Responsibility and control

- **Must be able to delegate and control responsibility**
- **Accountability - who is responsible for tracking access to data**
- **Authorization - who is responsible for who access data**

Integrity mechanisms

- **Checksum, CRC - protects integrity**
- **Message digest - more robust integrity check**

Encryption and privacy

- **Encryption - rewrite contents so that they cannot be read without key**
  - **Encrypting function - produces encrypted message**
  - **Decrypting function - extracts original message**
  - **Encryption key - parameter that controls encryption/decryption; sender and receiver share secret key**
- **Sender produces: *E = encrypt(K, M)***
- **Sender transmits *E* on network**
- **Receiver extracts: *M = decrypt(K, E)***

Digital signatures

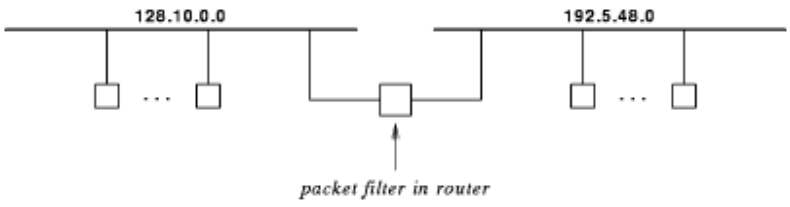
- **Goal - guarantee that message must have originated with certain entity**
- **Idea - *encrypt* with private key, *decrypt* with public key**
- **Only owner of private key could have generated original message**

Digital signatures and privacy

- **Can combine techniques - signed by A, private 10 B**
- **A forms: *X = encrypt(PUBB, encrypt(PRVA, M))***
- **B extracts: *M = decrypt(PUBA, decrypt(PRVB))***

Packet filtering

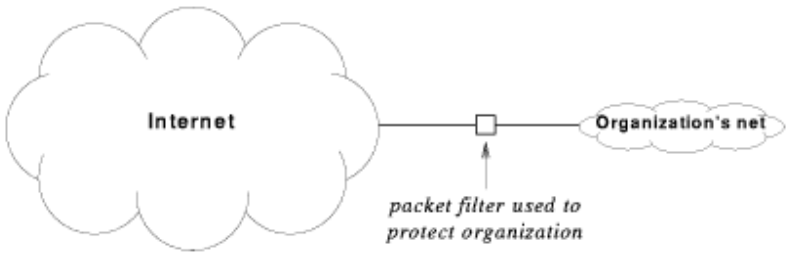
- **Can configure packet forwarding devices - esp. routers - to drop certain packets**
- **Consider example:**



- **Suppose 192.5.48.0 is test network and 128.10.0.0 has controlling workstations**
  - **Install filter to allow packets only from 192.5.48.0 to 128.10.0.0**
  - **Keeps potentially bad packets away from remainder of Internet**

Internet firewall

- **Packet filter at edge of intranet can disallow unauthorized packets**
- **Restricts external packets to just a few internal hosts**



- ***Proxies* forward packets through firewall after authorization**
- ***DMZ* net adds extra layer of access**
- ***Net 10* and *network address translation* (NAT) boxes also add security**

Summary

- **Security is a problem because Internet is not owned by one entity**
- **Organizations can use *firewalls* to prevent unauthorized access**
- ***Encryption* and *digital signatures* can provide confidentiality and secure identification**

Chapter 35 - Initialization

Introduction

- **How does protocol software begin operation**
- **From where is the software loaded**
- **How are configuration parameters determined**

Bootstrapping

- **Generally. initialization process is called *bootstrapping***
- **Key idea is to perform initialization in stages, each of which enables more functions**
- **Protocol software may be:**
  - **Run out of on-board PROM**
  - **Loaded with bootstrap program from disk**
  - **Loaded with operating system from disk**

Protocol parameters

- **Protocol software needs specific information for operation**
- **Software employs *parameters* for operation on a specific hardware and network**
- **Process of supplying parameters to protocol software is called *configuration***

Examples

- **IP address - depends on network, must be unique on network**
- **Default router address - where to send packets aimed at remote network**
- **Subnet mask - to specify if subnet addressing is used and what the subnet is**
- **DNS server address - for DNS queries**
- **Server addresses**

Modes of protocol configuration

- **Manual**
- **Local disk file**
- **Automated through network**

Why automate?

- **Centralized configuration for groups of computers**
- **Automatic address assignment**

Methods of automated protocol configuration

- **Decentralized - host discovers parameters independently**
- **Server-based - host contacts server**

The chicken-and-egg problem

- **How can host use network to get network address?**
- **Use broadcast-based *link-layer protocol***
  - **RARP**
  - **AppleTalk**

RARP

- ***Reverse Address Resolution Protocol*** **(RARP)**
- **Maps MAC address to IP address**
- **Host broadcasts RARP request with its MAC address**
- **Server replies:**
  - **Must be on same subnet**
  - **Looks up MAC address in table**
  - **Replies with IP address**

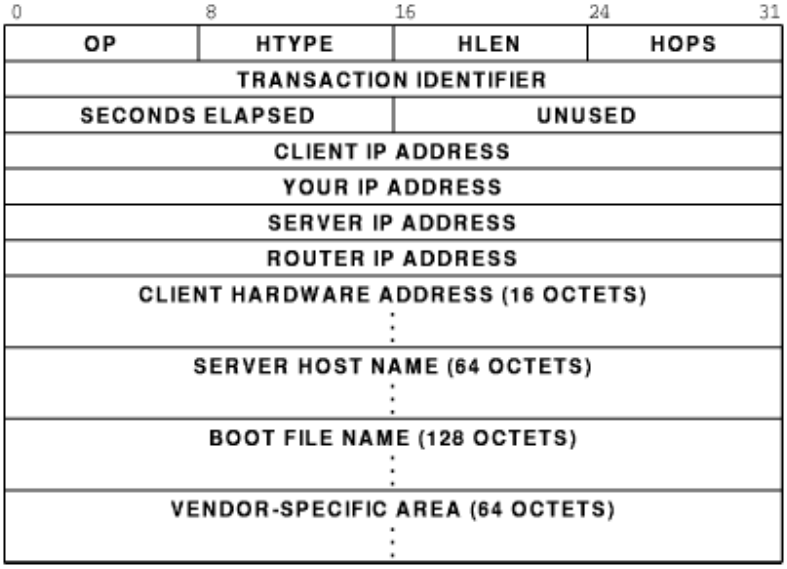
ICMP

- ***ICMP address mask request*** **- find subnet mask**
- ***ICMP gateway discovery*** **- find default router**
- **Sequence:**

1. **Broadcast RARP request**
2. **Extract IP address from RARP response**
3. **Broadcast ICMP address mask request**
4. **Extract subnet mask from ICMP address mask reply**
5. **Broadcast ICMP gateway discovery request**
6. **Extract default router from ICMP gateway discovery reply**

BOOTP

- ***Bootstrap Protocol*** **(BOOTP) provides multiple parameters**
- **BOOTP request broadcast on subnet**
- **Server returns reply with IP address, subnet mask, servers, etc.**
- **Format:**



BOOTP forwarding

- **Maintaining BOOTP server on each subnet can be expensive**
- ***BOOTP relay agent* listens for BOOTP requests**
  - **Forwards to BOOTP server**
  - **Returns responses from server back to BOOTP client**

Automatic address assignment

- **BOOTP requires that client be present in server database *before* BOOTP arrives**
- **Can addresses be allocated automatically?**
  - **AppleTalk**
  - **IPX**

DHCP

- ***Dynamic Host Configuration Protocol* (DHCP) can be used to allocate addresses to hosts without pre-configuration**
- ***Plug-and-play* networking**
  - **Host sends DHCP request message**
  - **Server finds unused address**

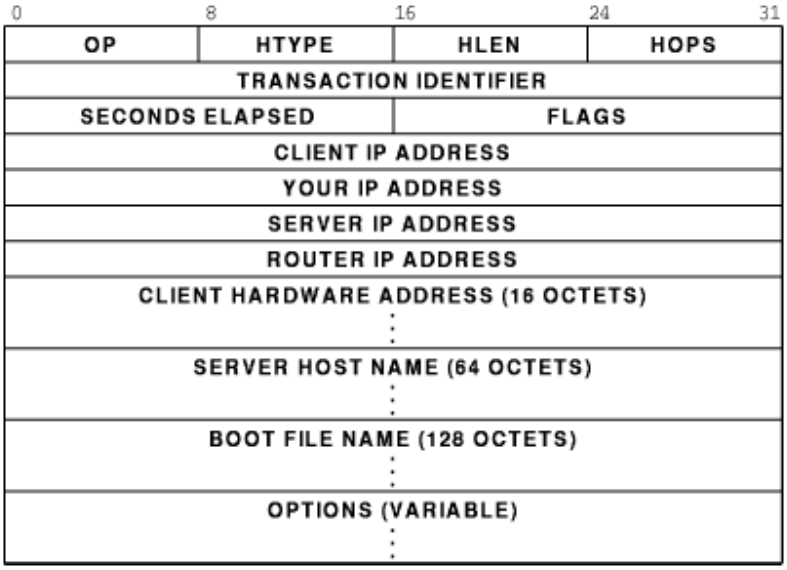
- **Adds to local list of addresses**
- **Returns address to host**

Address leases

- **Suppose host leaves subnet?**
- **Address no longer in use; server should reassign**
- **How does server know when to reassign?**
- **Address is assigned with a *lease***
  - **Client cannot use address after lease expires**
  - **Client can ask for *extension* prior to expiration**

DHCP message format

- **Extension of BOOTP; includes new options**



DHCP and DNS

- **New addresses should be entered in DNS**
- **DNS just recently added capability to automate entry updates**
- **"Soon", DHCP client or server will be able to add new entries to DNS**

Summary

- **Protocol software requires *configuration parameters***
- **Small, heterogeneous networks can use decentralized configuration**
- **IP uses server-based configuration**
  - **BOOTP**
  - **DHCP**