Computer Networks: Principles and Architecture

Peter Druschel[1]

Department of Computer Science
Rice University

Fall 1995

[1] Based in part on material by Larry Peterson and Mic Bowman

Copyright Notice

Overview

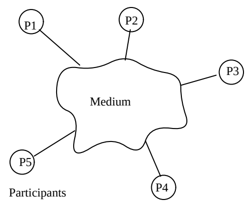
This course is:

- A study of concepts and principles
- A study of system design, network architecture and software
- Influenced by ARPA/Internet perspective
- A study of OS communications management

This class is not:

- A survey of standards and existing protocols
- A study of hardware and low level communication
- Guided by the ISO model of networking
- Queuing theory disguised as computer networks

Networking versus Communication



Networking is the problem of delivering a message from a source to a destination. How does the medium deliver the message? Host to host part of the problem.

- Encoding
- Synchronization
- Routing
- Congestion control

Communication is the problem of giving semantics to the messages that are received. What does it mean to receive a message?

- Interpreting the data in the message
- Inferring information about the state of the other process when the message is delivered
- Interprocess communication

Components:

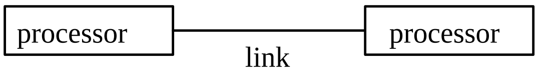
- participants
- medium
- messages

In general, the network may

- corrupt messages
- lose messages
- deliver duplicate messages
- deliver messages out of order

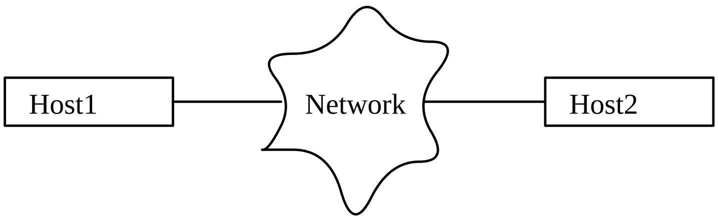
Hierarchical View of Connectivity

Processor—Link—Processor (frames)



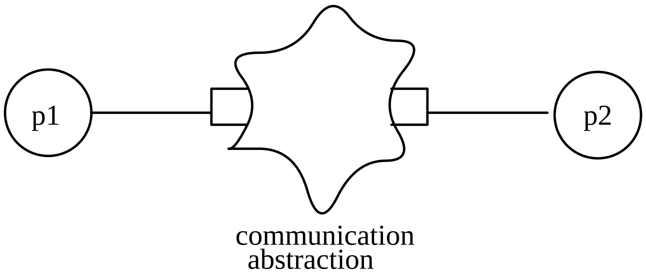
- A *processor* is dedicated to bit exchanges
- Physical data communication: bits as electrical/optical signals
- Data link: dedicated physical medium
- Frame is unit of information exchange

Host—Network—Host (packets)



- A *host* performs general-purpose computations
- Packet-switched network
  - internal nodes are dedicated switches
  - packet switching nodes (PSN) or routers
- Contention network
  - also called broadcast, multiple access networks
  - one wire which everyone connects to
  - e.g. Ethernet and FDDI LANs
- Meta-network or *internetwork*
  - collection of networks (subnets)
  - packet-switch nodes between subnets are called *gateways*

Process—Communication-abstraction—Process (messages)



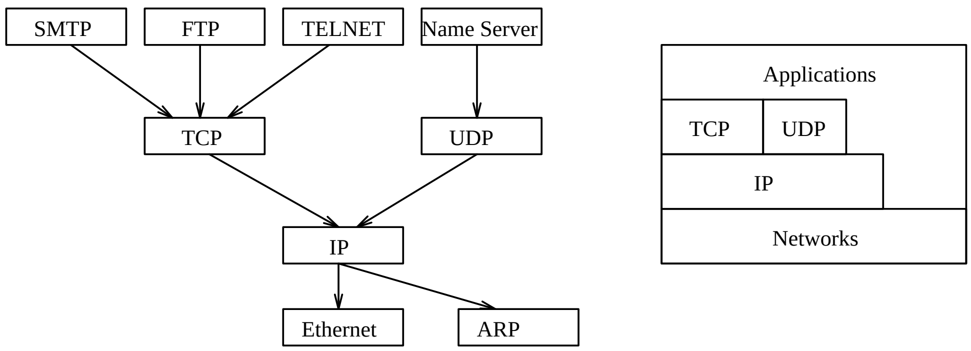
- Processes exchange messages through a communication abstraction.
- An abstraction is a “veneer” that hides the details of the networking that is performed and gives meaning to the data in the message.
  - minimally this is an unreliable datagram service
  - maximally it is an error free, sequenced communication channel such as a pipe
- there is a “logical” connection among processes

ISO OSI Reference Model

Application	e-mail, file transfer, WWW, etc.
Presentation	compression, encryption, data conversion
Session	synchronization
Transport	reliability (end-to-end), congestion control
Network	routing, addressing
Data Link	framing, error detection
Physical	physical connection, electrical signals

- This is a *prescriptive* model, i.e. it is given as a rule of action. The model was given and then protocols developed to fit the model.
- It is *linear* in that there is one path through the protocol suite.
- It is often used by phone companies.

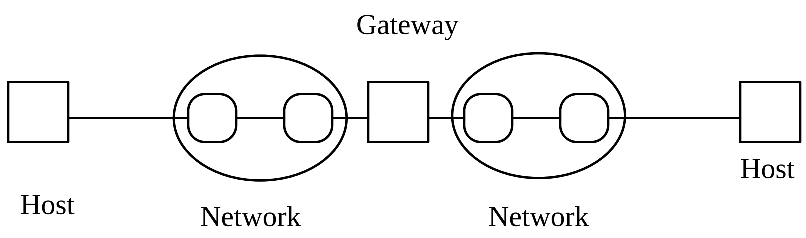
Internet Model



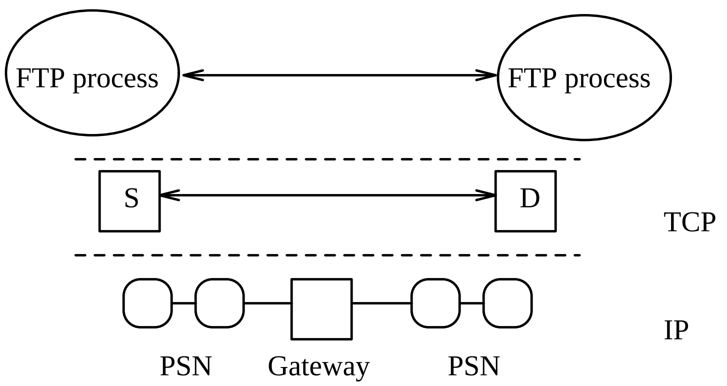
- This is a *descriptive* model, i.e. it describes a suite of protocols that already exist.
- It is *non-linear* because there are many dependencies among protocols and intermediate protocols can be bypassed.
- It is the model/protocol suite of choice in the computer industry.

File transfer example

- Protocol stack:
  - FTP: put/get file
  - TCP: open/close circuit, send/receive data
  - IP: send/receive datagram
  - Network (e.g. Ethernet): send/receive packet
- Interface points



- Peer levels



Fundamental Issues

A vertical cut defines issues common to all layers.

- Data encoding
  - Physical: bits on a signal
  - Intermediate: message consisting of header and data
  - IPC: primitive and compound data types on bytes
- Synchronization
  - Physical: transmitter and receiver (bit timing)
  - Intermediate: message flow (flow control when sender is faster than receiver)
  - IPC: process synchronization
- Resource management (Bandwidth and Buffer Space)
  - Bandwidth in contention network
  - Buffers and Links in packet-switch network
  - Buffers in end hosts

Fundamental Issues

- Reliability
  - Data Link: error detection/correction
  - Intermediate: link and PSN failure
  - IPC: delivery guarantees
- Naming/Addressing/Routing (Identifying Resources)
  - Data link: Node to node
  - Networks: Host to host
  - IPC: Process to process

Definitions

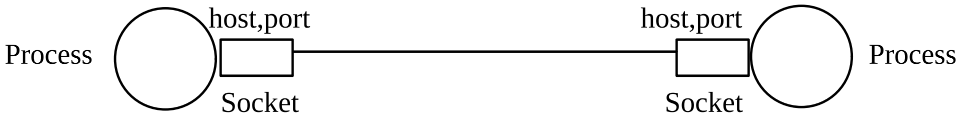
- Protocol/Service Interface
  - The *service interface* defines the syntax and semantics of the service that a communications (protocol) module provides to its clients.
  - A *protocol* defines the interaction between peer modules that jointly provide a communications service.
- Model/Standard
  - A *model* is a conceptual organization.
  - A *standard* is an enforced prescriptive model.
- Header/Body/Encapsulation
  - The message *header* contains control information such as sequence numbers, data length, etc.
  - The message *body* contains the data of the message.
  - Message *encapsulation* hides the "meaning" of the information in a message in transferring it to a lower level protocol.
- ACK/NAK
  - An *ACK* or *acknowledgment* indicates a successful condition.
  - A *NAK* or *negative acknowledgment* indicates an unsuccessful condition.

- Multiplexing/Demultiplexing
  - *Multiplexing* combines several circuits on a single lower level circuit.
  - *Demultiplexing* separates several circuits from a single lower level circuit.
- Fragmentation/Reassembly
  - The *fragmentation* of a message involves breaking it into several smaller pieces.
  - Fragments may be *reassembled* to form the original message.
- Datagram/Virtual Circuit
  - A *datagram* resembles a letter sent through the mail; information is passed in self-contained pieces that may travel different routes at different speeds.
  - A *virtual circuit* resembles a telephone call; information is passed along a particular route that is established before communication actually starts.
- Packet Switching/Circuit Switching
  - *Packet switching* divides communication into pieces (packets) that are multiplexed onto communications links.
  - *Circuit switching* forms a dedicated circuit between two points, such as a telephone connection.

- PSN/Gateway
  - A *PSN*, or Packet Switching Node, exchanges packets within a single network.
  - A *gateway* exchanges packets between two different networks.
- Layers/Levels

  Communication systems are divided into *layers* or *levels* so that layer n at the destination receives the same object sent by layer n at the source.
- Within network/End to end
  - *Within network* communication occurs between entities within the medium (a network) such as PSN to PSN.
  - *End to end* communication occurs between application programs.
- Unicast/Broadcast/Multicast
  - A *unicast* sends a single message to a single entity in the network.
  - A *broadcast* sends a single message to every entity in the network.
  - A *multicast* sends a single message to a subset of the entities in the network.

Operating System Support: Unix

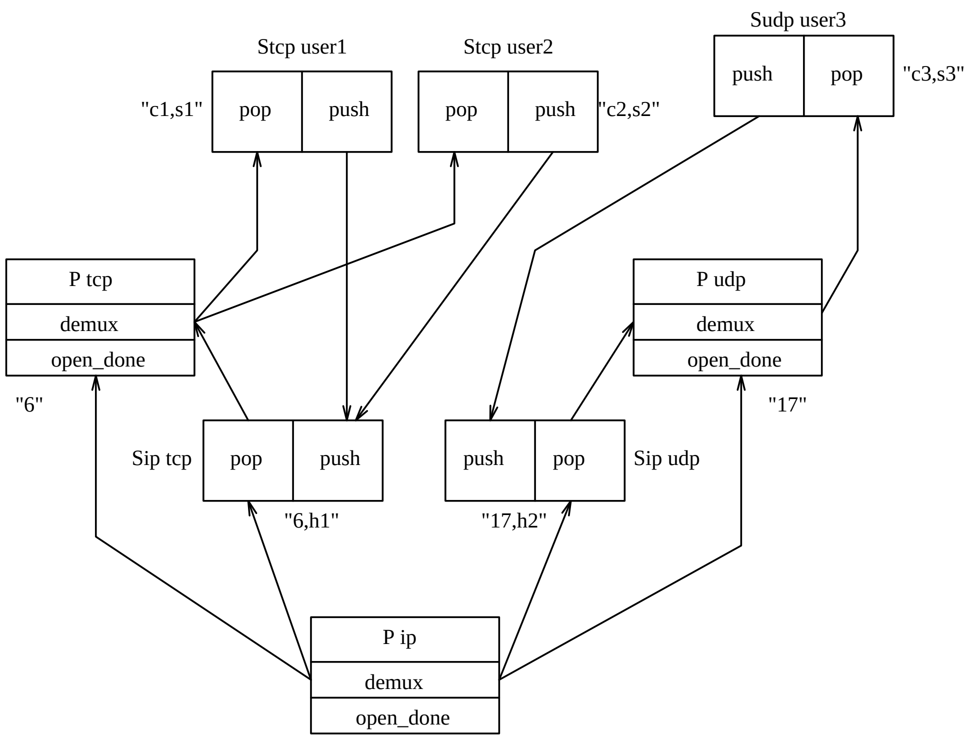


- A *socket* is an operating system abstraction in which a port is embedded.
- A *port* is a communication endpoint.
- Operations
  - fd = socket(): creates the socket
  - bind(fd,port): binds socket to local port (address)
  - connect(fd,port): establish a connection to a remote port
  - send(), recv(), write(), read(): operations for sending and receiving data
  - close(fd): close the connection

Operations System Support: x-kernel

- Object-based infrastructure
  - Protocol object:
    - open() operation creates a new session (active open)
    - openenable() and opendone() allow passive session creation
  - Session object:
    - push operation sends a message to a lower level protocol (session)
    - pop operation sends a message to a higher level protocol (session)
  - Demux operation determines the proper high level protocol to receive a message
  - Control operations
- Support library
  - Message ADT for efficient manipulation of messages
  - Id mapper for naming
  - Event mapper for events such as timeouts

Example



Sudp user3
Stcp user1
Stcp user2
"c1,s1"
pop
push
pop
push
"c2,s2"
push
pop
"c3,s3"
P tcp
demux
open_done
P udp
demux
open_done
"6"
"17"
Sip tcp
pop
push
push
pop
Sip udp
"6,h1"
"17,h2"
P ip
demux
open_done

- Active Open
  - Call open operation of a protocol
  - Creates a new session
  - Returns a handle for the new session
- Passive Open
  - Protocol P1 calls openenable operation of protocol P2 with address T (T is usually a partially wildcarded address)
  - When P2 demux receives a message that matches address T it calls the opendone operation of P1 creating a new session.
  - P2 demux calls the pop operation of the new session.

- Receiving a message
  - The demux operation of the bottom protocol is called.
  - The demux and pop operations process part of the message, e.g. they may strip header information off the message.
  - It decides which session should receive the message and calls that session's pop operation.
- Sending a message
  - A message is sent by calling the push operation of a particular session.
  - The push operation processes the message, e.g. it may add necessary header information.
  - The push operation calls the push operation of the next protocol.

Physical Data Communication

- physical layer of OSI reference model
- transmission of bits over a physical medium

Data

- information to be transmitted

Signal

- physical phenomenon that is generated by the transmitter, detected by receiver

Data encoding

- transmit information by varying some property of the signal

Transmission Impairments

Bandwidth-limited Channels

- frequency of change limited to finite range ("bandwidth")
- e.g., telephone line 300Hz – 3400 Hz

Noise

- thermal noise
- intermodulation noise
- crosstalk
- impulse noise

Attenuation

- signal strength diminishes with distance on transmission medium

Maximum Data Rate

Shannon's Law (1948)

- information-theoretic limit on digital data rate over bandwidth-limited channel
- upper bound—hard to achieve

$C = B\ log_2(1 + S/N)$

C: channel capacity [bits/sec]

B: channel bandwidth [Hz]

S/N: signal power to noise power ratio

Data encoding

digital data on digital signal

- mapping from serial data bits to signal elements
- Nonreturn to zero (NRZ)
- Return to zero (RZ)
- many more...

analog data on analog signal

- Amplitude modulation (AM)
- Frequency modulation (FM)

analog data on digital signal (digital voice)

- convert analog data to digital data
- Pulse code modulation (PCM)
- Delta Modulation (DM)

digital data on analog signal (modem)

- Amplitude-shift keying (ASK)
- Frequency-shift keying (FSK)
- Phase-shift keying (PSK)

Hardwire Transmission Media

Removable Storage Media

- high bandwidth
- very high latency
- low cost

Twisted Pair

- telephone local loops, local area networks
- low cost
- up to 1 Mbits/sec for several kilometers
- up to 100 Mbits/sec over short distances (100m)

Coaxial Cable

- cable TV, telephone system trunks (broadband)
- LANs (baseband)
- hundreds of Mbits/sec for hundreds of kilometers (broadband)
- 1 Gbits/sec for 1000m

Hardwire Transmission Media

Optical Fiber

- modern telcom trunks (single-mode)
- LANs (multi-mode)
- low attenuation, electromagnetic isolation
- hundreds of Mbits/sec for 1000m (multi-mode)
- several Gbits/sec for many kilometers (single-mode)

Softwire Transmission Media

Terrestrial Microwave

- 2-40 GHz carrier
- line-of-sight
- bandwidth comparable to coax, but lower attenuation

Satellite Microwave

- geosynchronous orbit (22,356 miles above sea level)
- high latency (250-300 milliseconds)

Radio

- 30 Mhz to 1 GHz carrier
- low bandwidth
- cellular radio

Digital Data Communication

Hierarchy of synchronization tasks

- bit synchronization (physical layer)
- character/word synchronization (physical/data link layer)
- frame synchronization (data link layer)

Asynchronous transmission (e.g., RS-232 serial lines)

- old and simple
- provides bit and character synchronization
- optional simple error detection (parity bit)

Synchronous transmission

- more efficient than asynchronous for large data blocks
- bit synchronization
  - separate clock
  - clock embedded in signal through appropriate encoding
- character-oriented (BSC/BISYNC)
- bit-oriented (SDLC, HDLC)

Data Link Level Communication

- data link layer of OSI reference model
- transmission of frames over a direct network link
  - point-to-point link
  - broadcast channel
- *essential property of direct link:* bits are delivered in the order they were sent.

Issues:

- framing
- error detection and correction (optional)
- flow control (optional)
- media access control (broadcast channels)

Framing

delineating sequences of bits into *frames*.

Four approaches:

- clock-based
- sentinels
- character count
- physical layer invalid codes

Clock-based framing (example: SONET)

- fixed length frames (transmission time or #bits)
- highly reliable bit synchronization critical

Sentinel-based framing

- use special character codes (sentinels) to delineate frames
- problem: what if special codes appear in data?
- solution: escape code: data link escape (DLE)
- if escape code appears in data, send as DLE-DLE
- 2 approaches:
  - precede each sentinel with DLE (example: IMP-IMP)
  - precede sentinel values in data with DLE (BISYNC)
- length of frames on the wire depends on data values!

Framing

Character count-based framing

- include frame length at fixed position in frame header
- problem: what if length field is corrupted?
- still need start-of-frame sentinel for robustness!
- (example: DECNET DDCMP)

Physical layer invalid codes

- requires redundancy in physical layer bit encoding
- use invalid codes as delimiters
- (example IEEE 802.4)

Error Control

handle data corruption during transmission

not a required function at data link layer (can be handled in higher layers)

- backward error correction (predominant method)
- forward error correction

Backward error correction

- error-detecting codes: add redundancy to data that allows detection of errors
- retransmission: ask sender for retransmission of erroneous frames

Error detecting codes

- parity
- two-dimensional parity
- cyclic redundancy check (CRC)

Error Detecting Codes

Parity bit

- add one bit to each character of a frame
- detects any odd number of bit errors in a character

$R_j = b_{1j} \oplus b_{2j} \oplus ....b_{nj}$

R_j: parity bit of jth character

b_{ij}: ith bit in jth character

$n = \#$ of bits in character

Two-dimensional parity

- add an additional parity check character to each frame
- reduces rate of undetected errors by 2-4 orders of magnitude

$C_i = b_{i1} \oplus b_{i2} \oplus ....b_{im}$

C_i: ith bit of parity check character

$m = \#$ of characters in frame

Error Detecting Codes

Cyclic redundancy check (CRC)

- sender appends a n-bit frame check sequence (FCS) to a k-bit data frame, such that the $k + n$ bit sequence is exactly divisible by a constant P
- receiver divides frame by P
- if there is a remainder, then the frame is in error

T = $(k + n)$-bit frame, $n < k$

M = k-bit data frame

F = n-bit FCS

P = $(n + 1)$-bit constant divisor (generator polynomial)

$T = 2^n M + F$

$\frac{2^n M}{P} = Q + \frac{R}{P}$

$\frac{T}{P}$ has no remainder, when $F = R$

Received frame $T_r = T + E$, where

T = transmitted frame

E = error pattern with 1s in error positions

T_r = received frame

Errors are not detected iff E is exactly divisible by P

The following are not divisible by P, hence are detectable:

- any odd number of errors, if P has a factor (X + 1)
- all double-bit errors, if P has a factor with at least 3 terms
- any error burst with length less than n
- most longer error bursts

Common values of P:

- $X^{16} + X^{15} + X^2 + 1$ (CRC-16)
- $X^{16} + X^{12} + X^5 + 1$ (CRC-CCITT)
- $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$ (CRC-32)

Implementation

- shift register and XOR gates (hardware)
- 8-bit table lookup (software)

Retransmission

- receiver responds with a control frame to every received frame
  - positive acknowledgment (ACK): tell sender frame was OK
  - negative acknowledgment (NAK): ask sender to retransmit
- problem: lost frames, acknowledgments
- retransmit timeout: sender retransmits, if no ACK/NAK was received for a period larger than the (known or estimated) round-trip delay
- potential problem: duplicate frames, ACKs
- solution: add frame/ACK identifiers (e.g., sequence numbers)

Example Protocols

- IBM BISYNC/BSC
- DECNET DDCMP

Reliable Transmission

Algorithms based on backward error correction and retransmission:

- stop-and-wait
- sliding window
- concurrent logical channels

Stop-and-Wait

- sender waits for acknowledgment before sending next frame
- receiver sends
  - ACK when it receives a correct frame
  - NAK when it receives a corrupted frame
- sender retransmits if
  - no response from receiver (timeout)
  - NAK received
- need 1-bit sequence number, carried in both data and ACK/NAK frames!

$U = \frac{D}{H+D+A+2CI}$

U = channel utilization (efficiency)

D = # bits in frame body

H = # bits in frame header

A = # bits in ACK

C = channel capacity [bits/sec]

I = propagation delay

Stop-and-Wait

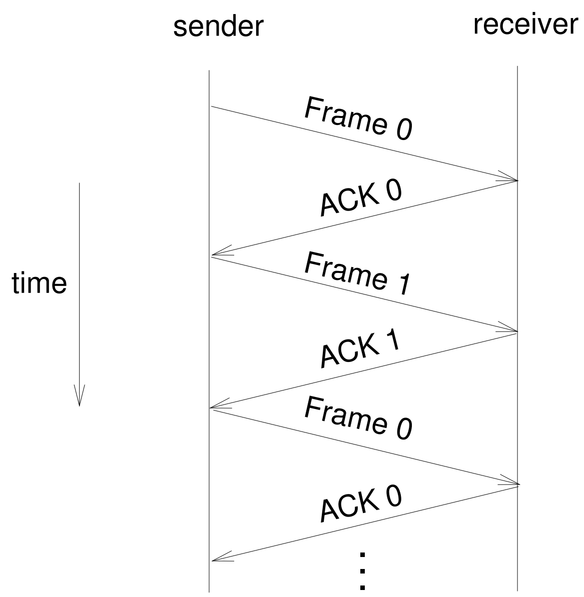
Operation:

- Frame successfully transmitted
  - receiver replies with ACK
  - sender gets ACK and discards its copy of the frame
- Frame is lost
  - sender times out and retransmits frame
- Frame arrives but contains an error
  - receiver replies with a NAK
  - sender retransmit upon receiving NAK
- NAK is lost
  - sender times out and retransmits frame
- ACK is lost
  - sender times out and retransmits frame
  - receiver discards duplicate frame (need sequence number!)

Stop-and-Wait

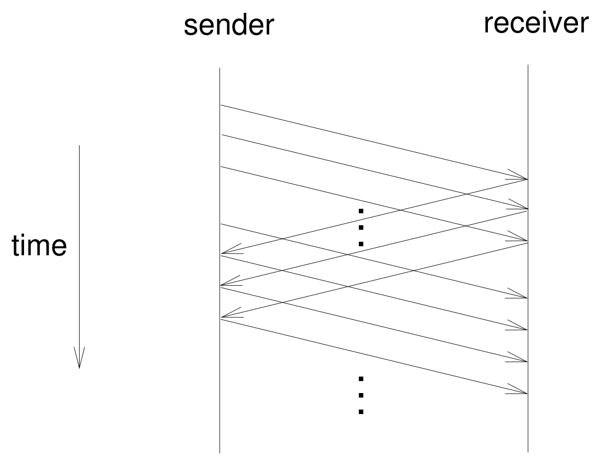
- frame or ACK is delayed
  - sender times out prematurely and retransmits frame0
  - sender receives delayed ACK0 and transmits frame1
  - frame1 is lost
  - receiver gets ACK0, discards (need sequence number in ACK/NAK!)

Stop-and-Wait



Poor utilization on links with high bandwidth-delay product!

Idea: Pipelining

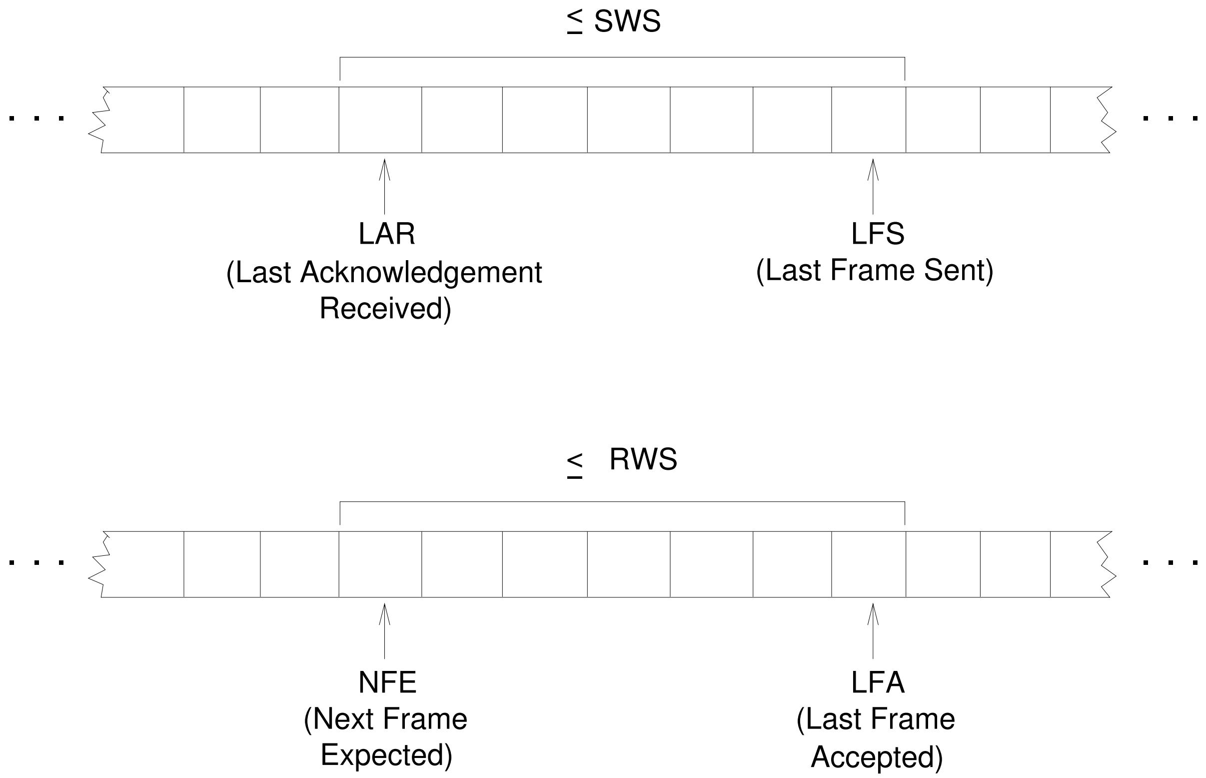


Sliding Window

Each frame (data and control) carries a sequence number, SEQNUM.

- Sender maintains a range of numbers, corresponding to the sequence numbers of frames sent but not yet acknowledged.
  – SWS sending window size (upper bound)
  – LAR last acknowledgment received
  – LFS last frame sent
  – LFS - LAR $<$ SWS
- Receiver maintains a range, corresponding to the frames it is willing to accept.
  – RWS receive window size (upper bound)
  – NFE next frame expected
  – LFA last frame acceptable
  – LFA - NFE $<$ RWS

Sliding Window



Sliding Window

Sender operation:

- can transmit up to SWS unacknowledged frames
- set a timer for each frame; cancel when frame is acknowledged; retransmit if timer expires
- when an ACK arrives with $SEQNUM > LAR$, let $LAR = SEQNUM$
- must buffer up to SWS frames.

Receiver operation:

- when a frame arrives with $NFE <= SEQNUM <= LFA$, accept frame, else discard it.
- if all frames with sequence number < SEQNUM have been received, send an ACK with SEQNUM, and let $NFE = SEQNUM + 1; LFA = SEQNUM + RWS$.
- must buffer up to RWS frames.

Window sizes:

- for best utilization (full pipeline), $SWS > 2CI[frames] + 1 frame$
- for best efficiency in the presence of lost and disordered frames, $RWS = SWS$
- if $SWS = RWS = 1$, stop-and-wait protocol.

Sliding Window

Sequence numbers

- sequence number in header has finite size.
- how many bits required?
- $SWS <= \frac{NUMSEQNUM}{2}$
- need at least $\log_2(2SWS)$ bits

Dealing with corrupted frames

- Solution 1: Go back N
  - receiver discards all subsequent frames
  - (implies RWS = 1)
  - no buffering needed at receiver
  - a lot of bandwidth can be wasted if error rate is high
- Solution 2: Selective repeat
  - receiver sends a NAK
  - receiver stores subsequent frames
  - sender retransmits the bad frame only
  - (implies receive window = SWS)
  - need to buffer up to RWS frames at receiver

Performance of Sliding Window

Case 1: $SWS >= 1 + \frac{2CI}{H+D}$ (large window, no errors)

$U = \frac{D}{H+D}$

Case 2: $SWS < \frac{2CI}{H+D}$ (small window, no errors)

$U = \frac{D}{H+D} * \frac{SWS}{1+\frac{2CI}{H+D}}$

Sliding Window

Piggy-backing

- exploit bi-directional traffic.
- try to add ACK information to data frames traveling in opposite direction
- if no traffic in opposite direction, send an explicit ACK

Sliding Window protocol service:

- reliable delivery
- ordered delivery

Concurrent Logical Channels

provides reliability, but no ordered delivery!

- several logical channels are multiplexed onto the communication link
- stop-and-wait is run on each channel.
- each frame carries channel#, 1-bit sequence#, piggy-backed ACKs for each channel
- header space required equivalent to sliding window protocol, where SWS corresponds to # of logical channels.
- no ordered delivery service

Flow Control

Throttle sender to prevent overloading of a slow receiver.

- Idea: delay ACKs until receiver is ready to receive next frame
- Problem:
  - delaying ACKs may cause retransmission
  - acknowledging a frame implicitly advances "send window"
- Solution:
  - decouple acknowledgment and control of send window
  - ACK carries sequence number plus new "advertised" SWS
  - can be used with all three of the above algorithms

Effective flow control requires explicit control of the sender window

Broadcast Networks

"multiple access networks" in OSI terminology

- simplest, cheapest way of providing multi-party communication
- popular in LANs (also satellite, packet radio networks)
- nodes communicate by sending frames over a shared link
  - bus
  - ring
- limitation: aggregate traffic cannot exceed shared link bandwidth
- key issue: mediating access to shared medium

Media Access Control

(MAC sublayer of data link layer in OSI)

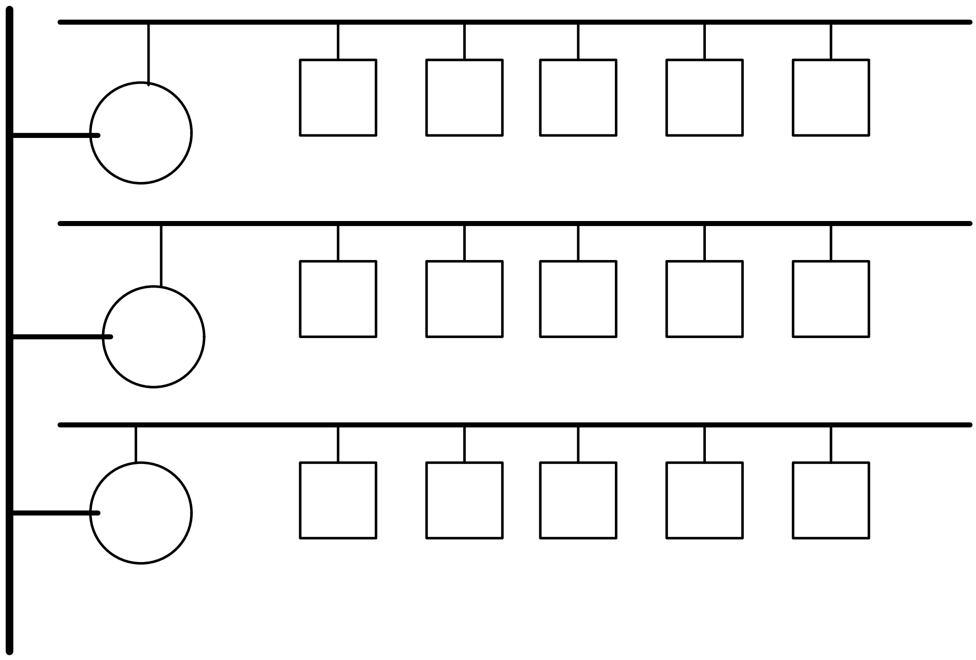
- static channel allocation
  - frequency-division multiplexing (FDM)
  - time-division multiplexing (TDM)
  - problem: severe under-utilization when traffic is bursty
- dynamic channel allocation
  - allocate channel only when data needs to be transmitted
  - contention
  - round-robin
  - reservation

Contention Networks

- each participant generates frames independently
- single channel that all participants use
- collisions occur when several participants use the channel simultaneously
- access may occur during a time continuum or at discrete time intervals (slots)
- there may or may not be carrier sensing

- ALOHA
- slotted ALOHA
- carrier sense multiple access (CSMA)
- CSMA/CD (collision detected)

CSMA/CD (Ethernet)



CSMA/CD: carrier sense, multiple access, collision detect

Ethernet

- shared cable (bus)
  - 50-ohm coax (10Base5, "thicknet"), 10 Mbits/s, < 500m
  - 75-ohm coax (10Base2, "thinnet"), 10 Mbits/s, < 200m
  - twisted pair (10BaseT), 10 Mbits/s, < 100m
  - twisted pair (100BaseT), 100 Mbits/s, < 100m
  - multi-mode fiber (100BaseF), 100 Mbits/s, < 2000m
- No more than 2 repeaters between any pair of hosts for a total of 1500m (IEEE 802.3 allows 4 repeaters)
  - a repeater relays electrical signals from one cable to another
  - they copy everything, including errors
  - software view is that of a single cable
- Maximum of 1024 hosts

Ethernet

Synchronous transmission, Manchester encoding

Frame format

- 64 bit preamble (alternating 1 and 0 for syncing purposes and a start of frame delimiter)
- 48 bit destination address, 48 bit source address
  - addresses are assigned by the NI manufacturer
  - broadcast is all 1's
  - multicast begins with a 1
- 16 bit frame type (length in 802.3)
- 368-12000 bits of data (46-1500 bytes)
- 32 bit CRC (cyclic redundancy code)
- 8 bit postamble (end-of-frame delimiter)

Preamble	Dest Addr	Src Addr	Type	Body	CRC	Postamble

Ethernet

Transmitter:

- Line Idle: transmit
  - maximum frame size provides an upper bound on the amount of time a sender controls the line
  - required minimum inter-frame gap (51.2 μsecs) prevents starvation of other senders.
- Line Busy: wait for line to become idle, start transmitting immediately
- Collision: detected immediately, transmit at least 512 bits (minimum frame size)
  - the time to detect a collision is round trip delay time, in this case 51.2 μsecs
  - exponential backoff strategy ($r2^c 51.2 \mu secs$ where r is a uniformly distributed random number with range [0..1], and c is the number of collisions)
  - note that starvation may occur (although very unlikely), participants are not guaranteed access to the network

Ethernet

Receiver:

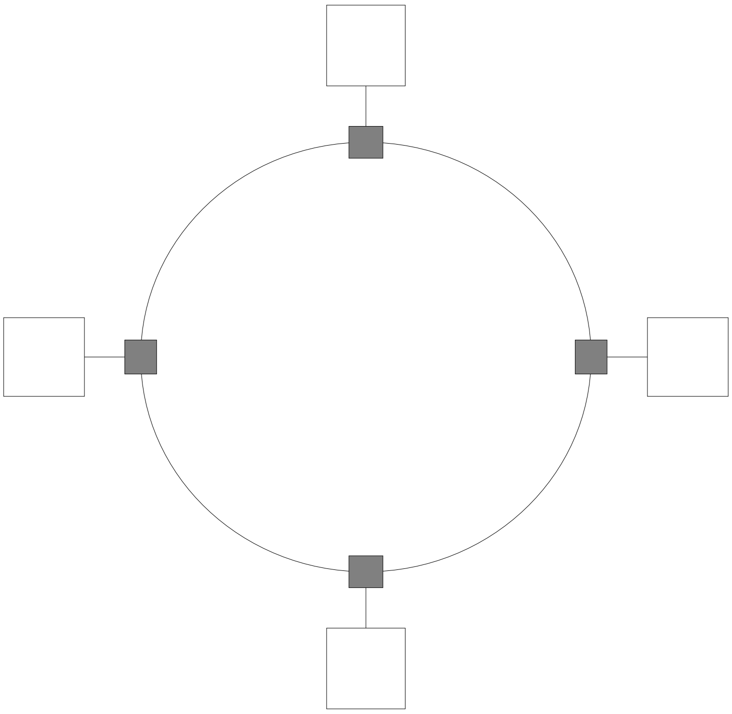
- accept all frames with the local address
- accept all frames with the broadcast address
- accept all frames with a programmed multicast address
- accept all frames if operating in promiscuous mode

Observe in practice

- 10-200 hosts (not 1024)
- cable length far shorter than 1500m; round trip delay closer to 5 μsecs
- higher level flow control exists (hosts don't just transmit data constantly)

Fiber Distributed Data Interface (FDDI)

Token Ring network (round-robin media access control)



- all frames flow around the ring in one direction (snake-like)
- stations forward frames after 1 bit delay
- multiple frames can be on a large ring simultaneously
- media access controlled by a *token* (special frame)

FDDI

Network adaptor operation:

- Copy/Listen mode: when not holding token
  - copy input to output
  - save copy of message in memory if it is addressed to me
- Transmit mode: when holding token
  - when token goes by, station "removes" it
  - sends its frame, followed by the token
  - station "removes" frame when it comes back around
- round-robin access guaranteed when everyone has something to send; therefore, at high loads the system is still fair

FDDI

- multi-mode fiber (also coax and twisted pair)
- up to 100km, 1000 stations
- 100 Mbit/s

Token maintenance (see book for details)

- controlling token rotation time to achieve bounded delay
- generating a token
- monitoring for valid token

FDDI Frame

Start of Frame	Control	Dest Addr	Src Addr	Body	CRC	End of Frame	Status

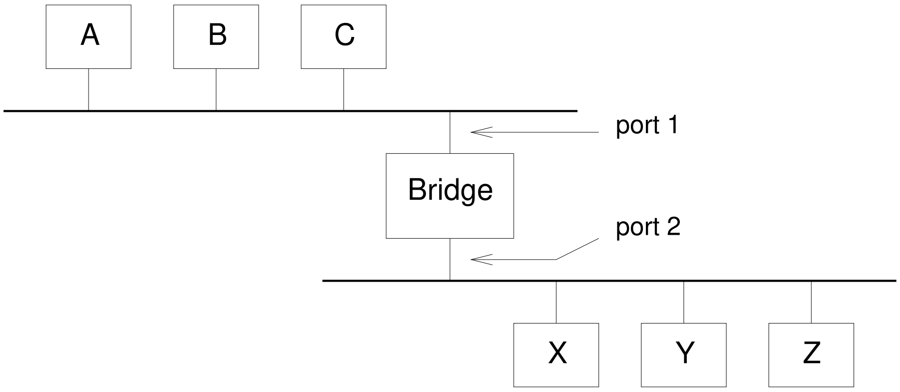
- 4 out of 5 encoding (not self-clocking, need highly stable clocks .005%)
- maximal frame size 4500 bytes (4478 bytes data)
- status field (3 bytes) used by receivers to report status info back to sender
  - cheap ACKs
  - flow control

Extended broadcast LANs

broadcast networks limited in size (#hosts, distance)

Idea: Bridge

- node connected to multiple broadcast nets with compatible addressing (e.g., Ethernet, FDDI)
- promiscuous mode (receive every frame)
- store-and-forward frames on other nets (using MAC)
- bridge operates at data link level (understands frames)



Problem: bridge forwards *all* frames

Extended LANs

Learning Bridges

- Learning: bridge looks at source address of incoming frames, "learns" to which port a host is connected; adds info to forwarding table.
- frames from hosts not listed in forwarding table are forwarded to all ports
- table entries expire to handle topology changes
- Problem: cycles
  - extra bridges for redundancy (fault tolerance)
  - extensions across administrative boundaries
- Solution: forward frames along a spanning tree

Extended LANs

Spanning tree algorithm

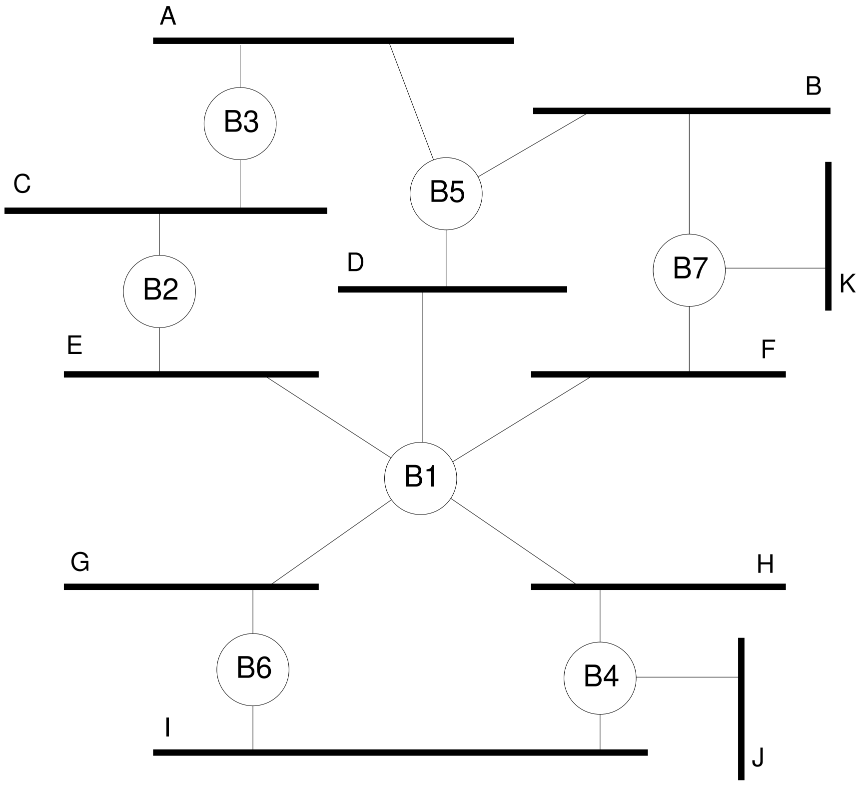
- Each bridge is assigned an ID; the bridge with the smallest ID is designated the *root bridge*
- Each LAN has a *designated bridge* chosen by minimum root path cost (closest to the root) with ties broken by smallest ID
- Each bridge chooses a port that is on its shortest path to the root
- Bridges forward and accept frames on ports for which they are "designated bridges", and on the port corresponding to the shortest path to the root.

Spanning Tree Algorithm

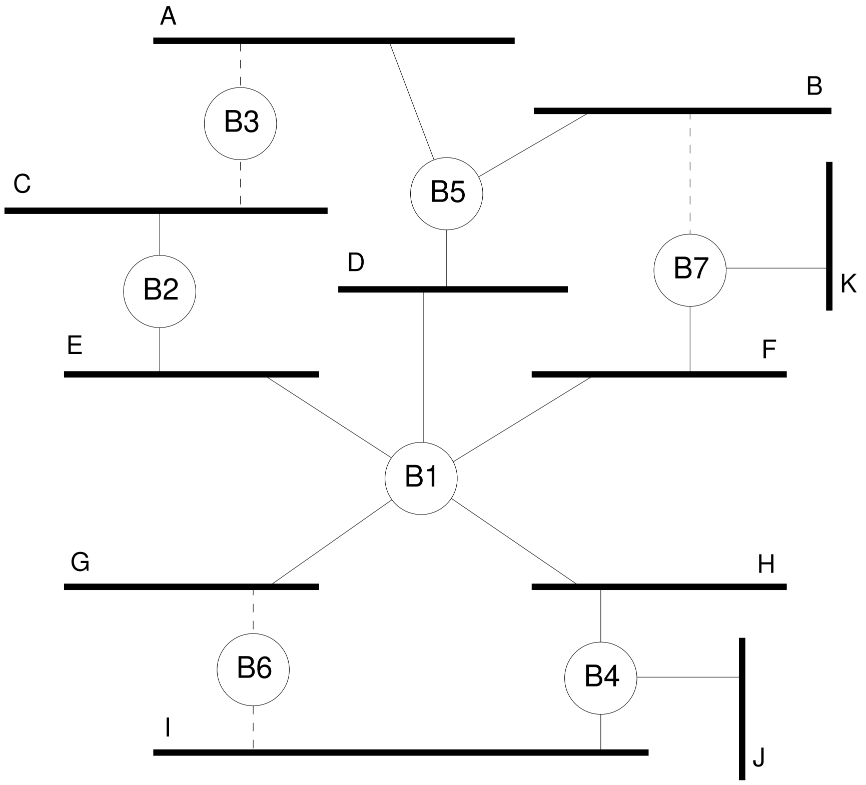
Election process

- Initially, all bridges assume that they are the root bridge
- Each bridge multicasts a BPDU (bridge protocol data unit) to all other bridges on the LAN containing
  - the id of this bridge
  - the id of the bridge that this bridge considers the root
  - the root path cost for this bridge
- every time a bridge receives a BPDU on a port, it updates its per-port information and potentially retransmits the BPDU. Per-port info is updated, if received BPDU has "better" info, i.e.,
  - it identifies a root with smaller id
  - it reports a shorter distance to the root
  - the sending bridge has a smaller id
- in steady state, only the root bridge periodically sends new BPDUs which are forwarded through the tree, each bridge indicating its root path cost
- if a bridge hasn't received BPDUs from the root for a while, it declares itself root and starts a new election process (dynamic adaptation to topology changes)

Spanning Tree Algorithm



Spanning Tree Algorithm



Extended LANs

Limitations

- Ethernet and FDDI LANs (have same addressing space)
- limited scalability (tens of LANs) due to broadcasts

Note: bridges drop frames; may reorder frames

Switched Networks

Switch: Device with multiple input/output ports

Use point-to-point links to connect

- hosts to switches
- switches to switches
- highly scalable topology!

Two classes of switched networks

- circuit-switched (connection-oriented)
  - links are allocated to one connection at a time.
  - at connection-setup, corresponding inputs and outputs in each switch are connected for the duration of the connection.
  - no contention, since all resources are allocated to a connection a priori.
  - used primarily in early phone networks.
- packet-switched
  - data is packetized.
  - links and switch ports are shared dynamically.
  - potential contention for output ports (links).
  - predominant approach in data networks.

Packet Switched Networks

Three issues:

- switching (forwarding)
- routing
- congestion control

Switching

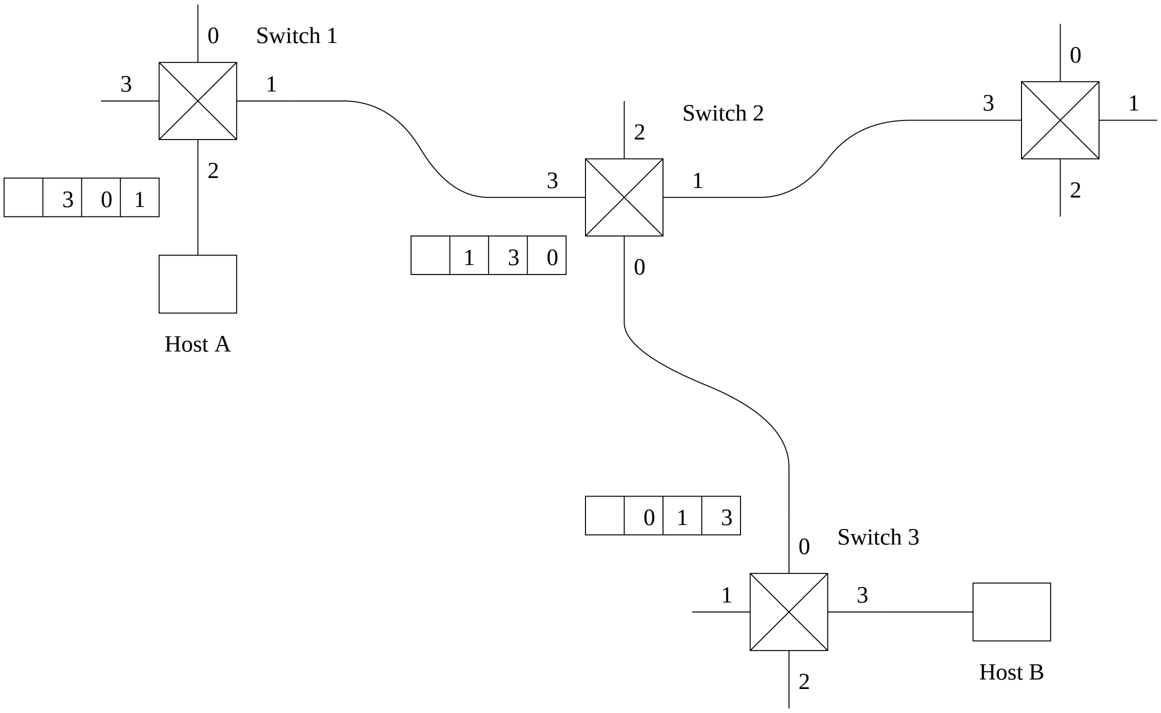
How does a switch decide on which output port an incoming packet should be forwarded?

Three approaches:

- source routing
- virtual circuit switching
- datagram switching

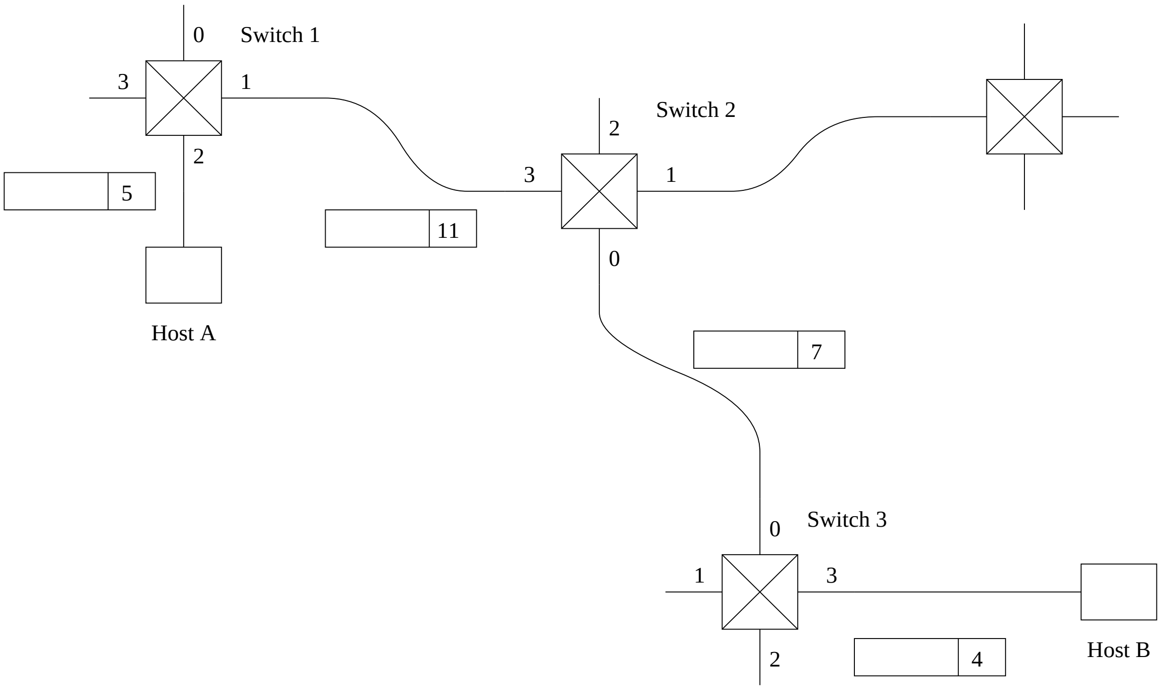
Source Routing

Sending host includes "route" in packet header (sequence of output port numbers, one for each switch.)



- sending hosts needs to know exact route
- large networks/changing topology pose problems
- mainly used in small, special-purpose networks (e.g., multiprocessor networks)

Virtual Circuit Switching (Connection-oriented)



Virtual circuit table for switch 1:

Input Port	Incoming Identifier	Output Port	Outgoing Identifier
2	1	2	4
2	4	0	3
2	5	1	11
2	6	0	4

Virtual Circuit Switching

Connection setup

- Host A sends "connection request" message, presenting
  - address of Host B
  - a VC identifier currently not used on the link host-A—switch-1
- foreach switch:
  - choose an output port according to routing table
  - pick an outgoing VC identifier currently unused on the link current-switch—next-switch
  - add a mapping $< inbound - port, inbound - VCid >$ $, < outbound - port, outbound - VCid >$ to virtual circuit table
  - forward connection request message using outbound-VCid
- Host B responds with a "connection accepted" message, causing a VC setup along the reverse path.

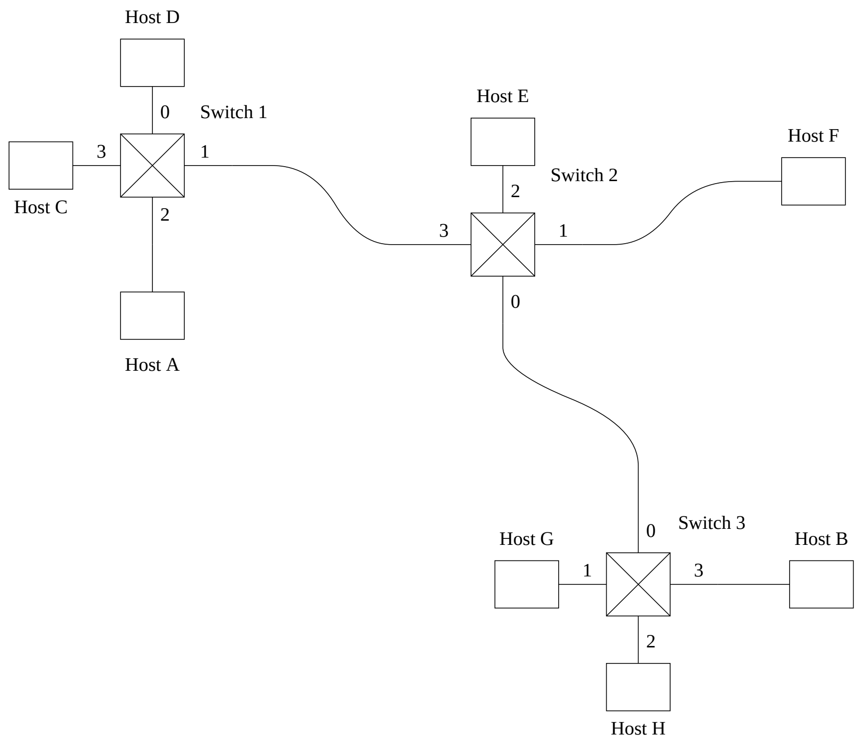
Virtual Circuit Switching

Packet transmission

- foreach switch:
  - lookup VC table, using $< inbound - port, inbound - VCid >$
  - replace VCid in packet header with $outbound - VCid$
  - forward packet on $outbound - port$

- minimum of one round-trip latency for connection setup
- data packet headers contain VCid only (not destination address)
- routing happens during connection setup
  - all data packets on a connection travel same route
  - link or switch failure breaks connection
  - need garbage collection for broken connections
- resource allocation possible during connection establishment

Datagram Switching (Connectionless)



Forwarding table for switch 2:

Destination	Switch Port
A	3
B	0
C	3
D	3
E	2
F	1
G	0
H	0

Datagram Switching

- no connection setup delay
- each packet is routed individually
  - can easily route around failed switches and links
  - can easily route around congested areas
  - per-packet routing overhead
- data packet headers contain full destination address
- resource pre-allocation not possible

Routing

- Goal: systematic progression of each packet through the network towards destination
- Factors that influence design
  - static: topology
  - dynamic: traffic
- Elements of design
  - performance criterion (hop count, delay, throughput)
  - decision time (when is route chosen)
  - decision place (who chooses the route)
  - information source (where is information managed)
  - strategy (protocol)
- Desirable properties
  - correctness
  - simplicity
  - robustness
  - stability
  - fairness, optimality

Model

- The network is given by a graph $N = (P, L)$ and a cost function $c : L \longrightarrow R$ for the links
- A **route** from p_s to p_d is a path through N originating at p_s and terminating at p_d; minimal cost path is "optimal"
- A switching function $S : A_d \times N_p \longrightarrow L_p$ maps destination address/network view to a link
- Two extremes of S
  - address contains all information about the route (address space)
  - address contains no information about the route (routing table)

Taxonomy of techniques

Non-Adaptive routing considers only "static" factors such as topology and bandwidth.

- a route is selected for each source/destination pair
- the route is only changed when the topology changes

Adaptive routing chooses paths based on both static and dynamic factors such as load.

- *centralized adaptive routing:* information from all nodes, centralized control; a centralized route server collects information from every processor and determines the routes chosen for every pair of processors
- *isolated adaptive routing:* local information, distributed control; each processor makes decisions based on information about the load on its links; e.g. queue length or round-trip delay
- *distributed adaptive routing:* local and global info, distributed control

Centralized Adaptive Routing

centralized route server periodically collects local information from each switch:

- load, bandwidth, latency of adjacent links
- traffic stats

server computes best routes from all hosts to all other hosts:

- calculate edge weights based on collected info
- use shortest path algorithm (e.g., Dijkstra's algorithm)

route distribution:

- server sends routes to all switches
- or, switches ask server for routes

Features:

- "optimal" routes based on complete information
- route recalculation expensive
  - need big server CPU
  - heavy routing traffic near server
  - slow to adapt
- single point of failure

Isolated Adaptive Routing

Each switch makes routing decisions based on local information only

- random walk routing: forward packet through a single, randomly chosen link
- hot potato routing: forward packet on link with shortest queue
  - good load distribution
  - unbounded delivery time
  - can be combined with static routing
- flooding: forward packet on every link (except incoming)
  - keep hop count in packet header and discard those that have traveled too long ($> maxpathlength$)
  - or, keep track of all packets that have been forwarded and discard any duplicates
  - optimal path (minimal delivery time)
  - high resource consumption (poor throughput)
  - highly robust
  - broadcast
  - selective flooding: send packet only on links that go in "roughly" right direction

Isolated Adaptive Routing

- backward learning
  - switches look at "weighted hop count" and source address in packet headers
  - keep track of best route for each destination
  - need route timeouts to accommodate topology changes
- delta routing (hybrid centralized/isolated)
  - routing center computes k best paths between each pair of ***switches***, such that the path costs are within δ and the initial links differ.
  - do random or hot potato within the k best paths.
  - shown to perform better than either pure centralized or pure isolated routing, with appropriate choice of k and δ.

Distributed Adaptive Routing

Switches periodically exchange info, recompute best known routes

- distance-vector algorithms
  - how far do you think it is to participant p?
- link-state algorithms
  - what is the load on your adjacent links?

Distance-vector Routing

originally used in ARPANET

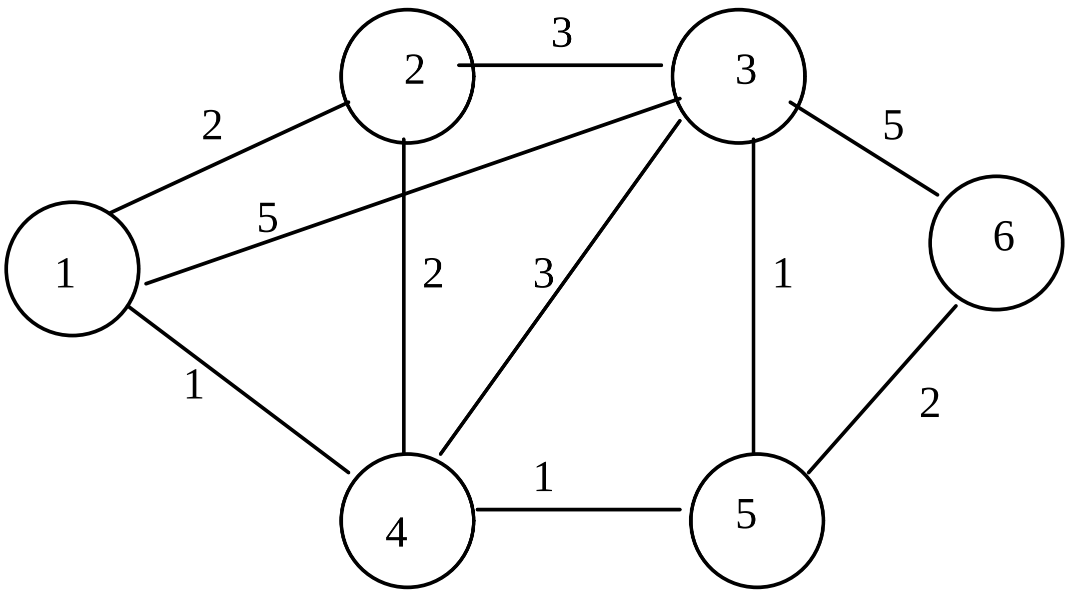
- initially, each node knows only its own identity, and the "distance" to its neighbors
- periodically exchange the whole routing table with directly connected neighbors
- save most recent distance vector from each neighbor
- recalculate routing table from distance vectors whenever we have "better" information.

Bellman-Ford algorithm

- Define:
  - s is the source node
  - d_{ij} is the link cost from i to j; $d_{ii} = 0$, $d_{ij} = \infty$ if nodes are not directly connected
  - h is the maximum links in a path at the current stage of the algorithm
  - $D_n^{(h)}$ is the cost of the least-cost path from s to n with no more than h links
- Initialize: $D_n^{(h)} = \infty$, for all $n \neq s$
- For each successive $h \geq 0$:

$$D_n^{(h+1)} = min_j[D_j^{(h)} + d_{jn}]$$

Example



Node 1 routing table:

h	Destination									
	2		3		4		5		6	
	Cost	Path	Cost	Path	Cost	Path	Cost	Path	Cost	Path
0										
1	2	1-2	5	1-3	1	1-4				
2	2	1-2	4	1-4-3	1	1-4	2	1-4-5	10	1-3-6
3	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6

ARPANET algorithm

ARPANET used a backward search algorithm based on Bellman-Ford.

- Each node i maintains two vectors
  - a delay vector $D_i = [d_{i1}, d_{i2}, \ldots, d_{iN}]$ where d_{ij} is the current estimate of minimum delay from i to j
  - a successor node vector $S_i = [s_{i1}, s_{i2}, \ldots, s_{iN}]$ where s_{ij} is the next node in the current minimum delay route
- Every 128ms, each node exchanges its delay vector with all of its neighbors
- The local vectors are updated as follows:
  - $d_{ij} = min_{k \in A}[l_{ik} + d_{kj}]$ where l_{ik} is the queue length associated with link ik and A is the set of directly connected neighbors
  - $s_{ij} = k$ that minimizes d_{ij}

Experience

- too much traffic to maintain tables (exchange a table with an entry for every node in the network)
- queue length is an artificial measurement of the time to reach a node (packets move towards shortest queue instead of destination)
  - didn't consider line speeds
  - instantaneous measurement
- too sensitive to minor changes (queue length changes rapidly and updates occur too often)
- too insensitive to major changes (slow convergence)

Link-state Routing

Each switch periodically broadcasts info about its adjacent links to all other switches.

Switches recompute routing tables based on the accumulated state of all links.

Reliable Flooding:

foreach switch:

- construct a link state packet (LSP):
  - originating switch id
  - list of adjacent switches and link costs
  - per-switch sequence number
  - time-to-live (TTL)
- send LSP on all links
- keep most recent copy of LSP from each other switch
- new LSP are forwarded to all neighbors (except sender), after decrementing TTL
- age stored LSPs (decrease TTL at suitable rate)
- re-flood LSPs whose TTL has reached 0.

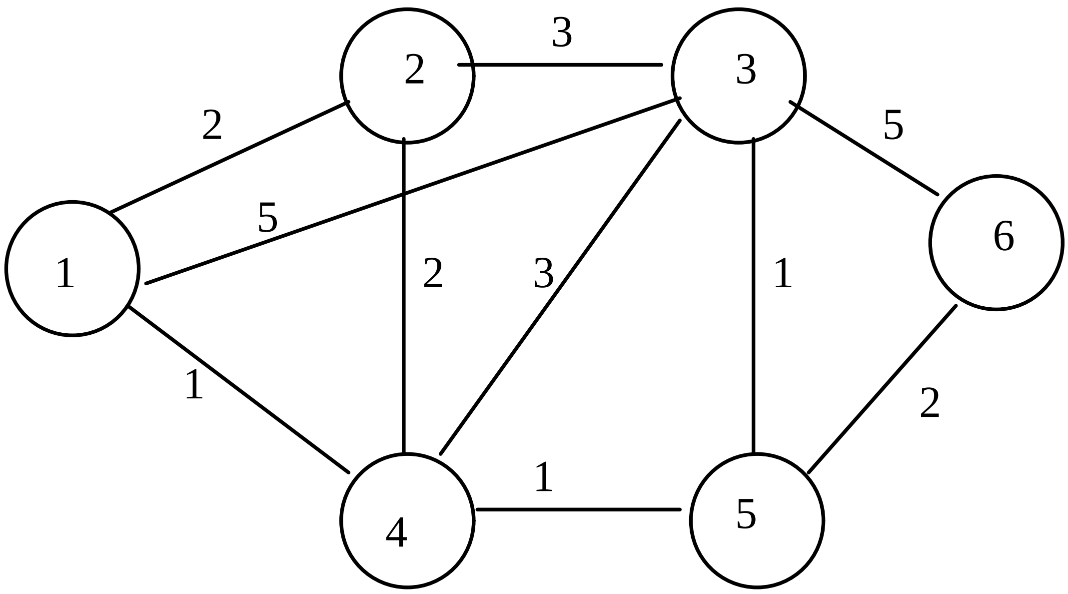
Route Calculation

use shortest path algorithm (Dijkstra)

Dijkstra algorithm

- Define:
  - N is the set of nodes in the network
  - s is the source node
  - M is the set of nodes incorporated so far
  - d_{ij} is the link cost from i to j
  - D_n is the cost of least cost path from s to n
- Initialize: $M = \{s\}$, $D_n = d_{sn}$ for $n \neq s$
- Find neighboring node with minimal path cost and add it to M: $w \notin M$ such that $D_w = min_{j \notin M} D_j$, add w to M
- Update minimal cost paths: $D_n = min[D_n, D_w + d_{wn}]$ for all $n \notin M$

Example



1 {1}	2 1-2	5 1-3	1 1-4		
2 {1,4}	2 1-2	4 1-4-3	1 1-4	2 1-4-5	
3 {1,2,4}	2 1-2	4 1-4-3	1 1-4	2 1-4-5	
4 {1,2,4,5}	2 1-2	3 1-4-5-3	1 1-4	2 1-4-5	4 1-4-5-6
5 {1,2,3,4,5}	2 1-2	3 1-4-5-3	1 1-4	2 1-4-5	4 1-4-5-6
6 {1,2,3,4,5,6}	2 1-2	3 1-4-5-3	1 1-4	2 1-4-5	4 1-4-5-6

New Internet Routing Mechanism

In 1980 the ARPANET converted routers to use a Link-State algorithm.

The new algorithm also measures delay directly.

- timestamp each incoming packet (at)
- record departure time (dt)
- if get ACK, delay = dt - at + transmission time + propagation delay

New Internet algorithm

The new algorithm is a forward search based on Dijkstra's algorithm.

- Each node maintains a vector containing the average delay to every directly connected node.
- Every 10s (if the delay change is sufficiently significant) broadcast average delay vector to every other node in the network
  - each receiving node retransmits on all links
  - the echo serves as an acknowledgment
  - old and duplicate update packets are dropped
- When an update arrives, recompute routing table using an incremental version of Dijkstra's algorithm

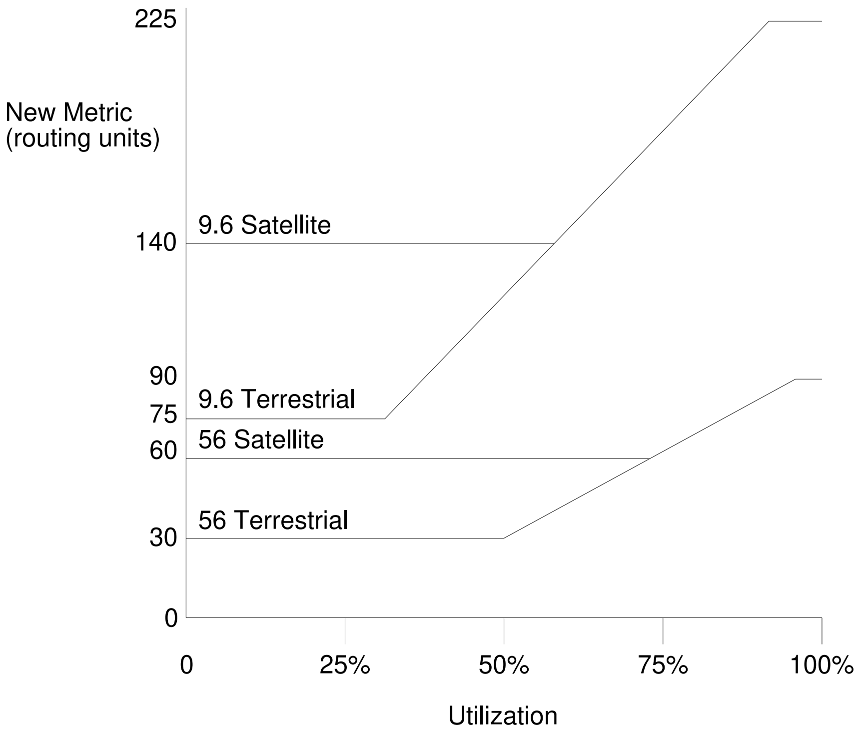
Experience

- reasonable amount of overhead (average update packet is 176 bits transmitted every 10s)
- responds rapidly to topological changes (down nodes and lines)
- stable under most conditions
- avoids congestion though oscillation possible (no loops since everyone performs the algorithm on the same database)
- per-node storage requirements substantial

Revised Internet Routing Metric

- Two problems to avoid
  - oscillation (don't spread congestion across network)
  - long, high capacity paths always have preference over short low capacity paths, even under heavy loads
- Goal under heavy loads: give the average route a good path rather than giving all routes the best path, i.e. long delays should be avoided, but not by all traffic
- Solution: normalize delay metric to account for hop count and link cost history with absolute limits
  - under light loads, be greedy and choose the best route for yourself, delay time is dominant
  - under heavy loads, the average route should be good, hop count is dominant
- Experience: shortest delay under light loads, improved performance under heavy loads

Revised Internet Routing Metric



- loaded link cost never exceeds 3 times its idle cost
- most expensive link only 7 times least expensive link
- high-speed satellite link more attractive than low-speed terrestrial link
- cost a function of utilization only at moderate to high loads.

High-Performance Parallel Interface (HIPPI)

Originally, a point-to-point connection technology (typically connecting supercomputers with peripheral devices)

- 800/1600 Mbits/sec
- up to 25 meters on 32/64 bit wide cables
- up to 10 km using serial fiber

Can be used as high-speed LAN in connection with a switch

- circuit-switched network!!
- very simple
- no multicast support

Asynchronous Transfer Mode (ATM)

Cell-switched network: virtual-circuit switched network, where packets (cells) are

- fixed-sized
- small

ATM technology designed to unify voice, video, and data networking.

Why fixed-sized cells?

- simple, thus potentially fast switching hardware
- low jitter

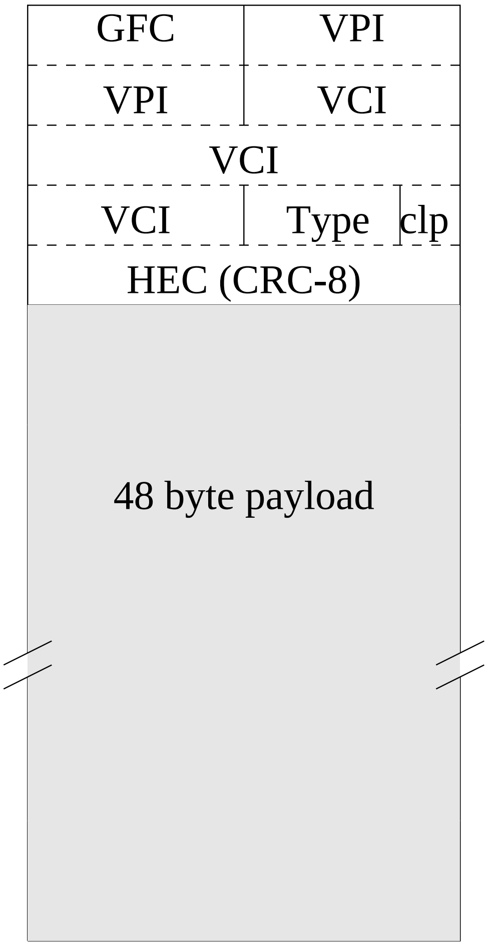
Why small cells?

- low delay
- BUT:
  - high overhead for segmentation/reassembly, switching
  - high space overhead (cell headers)

ATM

ATM cells are 53 bytes long

- 48 bytes "payload"
- 5 byte header
  - 4-bit "Generic Flow Control (GFC)" (host-switch only)
  - 8-bit "Virtual Path Identifier (VPI)"
  - 16-bit "Virtual Circuit Identifier (VCI)"
  - 3-bit control
  - 1-bit cell loss priority
  - 8-bit header CRC



Two-level virtual-circuit identifiers among switches

- no GFC field
- 12-bit "Virtual Path Identifier (VPI)"
- 16-bit "Virtual Circuit Identifier (VCI)"
- backbone switches use VPI only

Connection management ("signalling") not yet standardized

ATM

ATM features

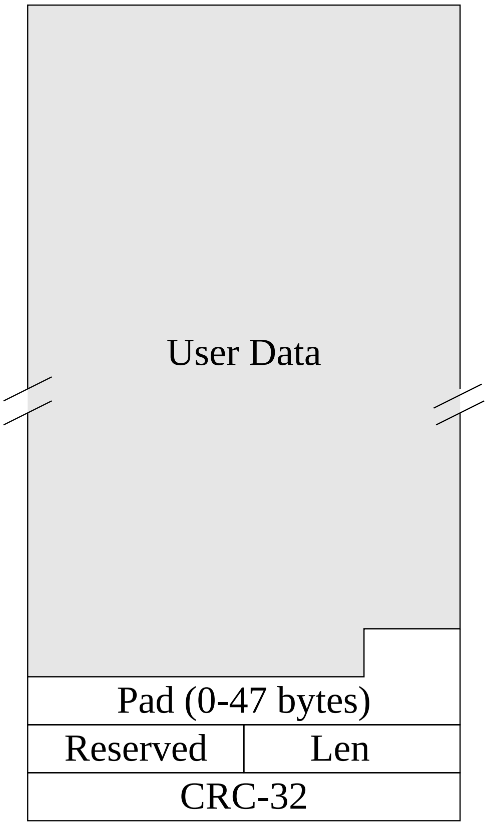
- high bandwidth: 155 Mbps, 622 Mbps, 1244 Mbps, ...
- low delay and jitter
- quality of service capabilities (reservation during virtual-circuit setup)
- no cell reordering
  - simplifies segmentation/reassembly at network edges
  - complicates switch design
- cell loss possible (congestion control not yet standardized)

ATM Adaptation Layers

"Adaptation Layer (AAL)" standards define how data is packaged into cells.

- AAL 1,2: continuous-rate data (voice, video)
- AAL 3/4: originally intended for packet data, high overhead
- AAL 5: new standard for packet data

AAL 5



- 1 bit in cell header used to identify last cell of AAL 5 frame
- low overhead
- CRC detects lost, corrupted cells
- up to 64 KByte frames

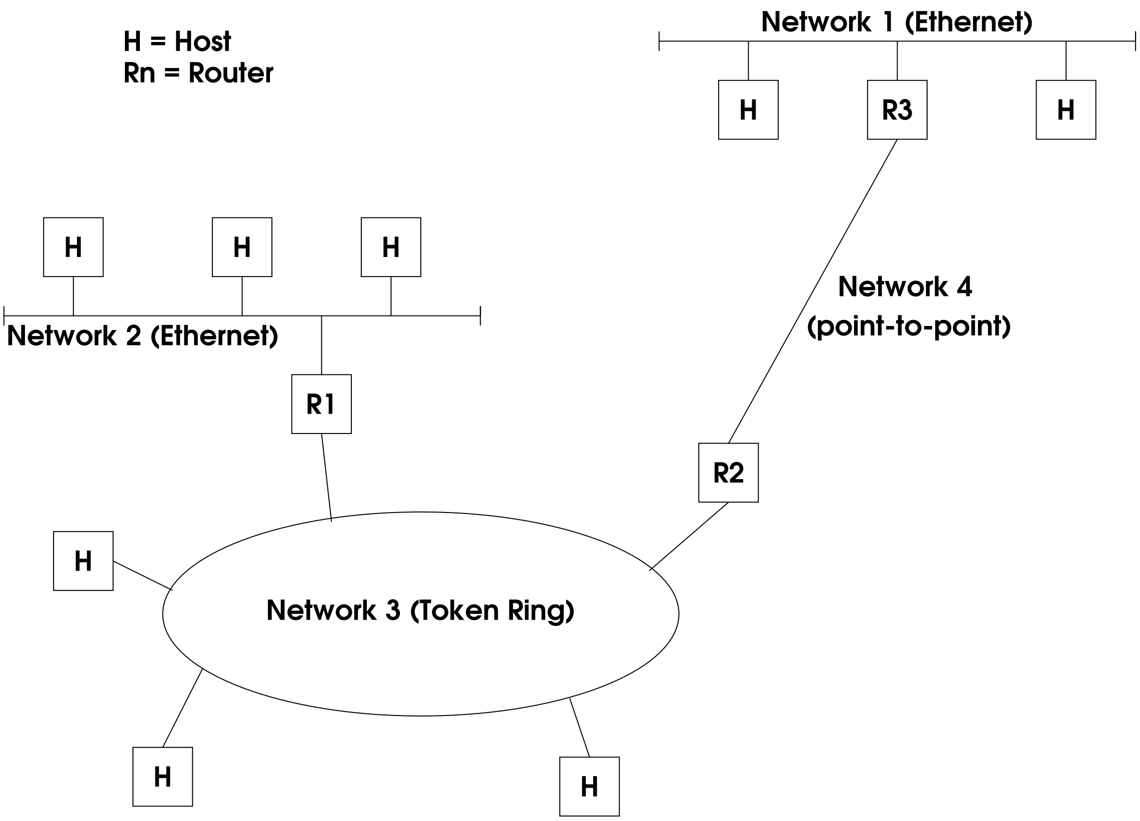
Internetworks

An *internet* is a software based interconnection of many separate and distinct physical networks functioning as a single, coordinated unit.

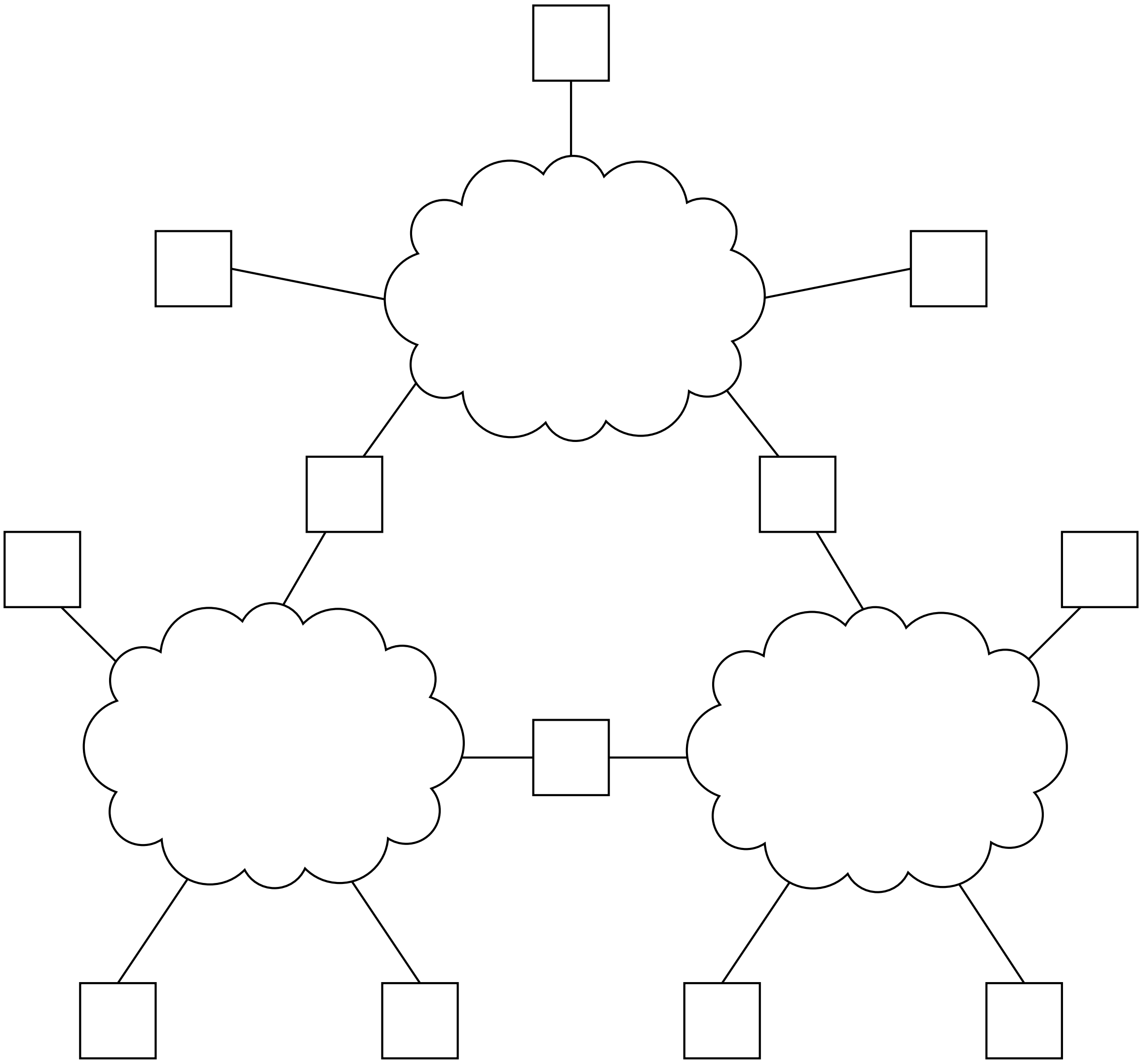
- no single network technology appropriate for all applications
- universal interconnection nevertheless desirable
- Goal of internetworking: to allow application processes running on different hosts connected to different networks to interoperate independent of vendor/OS/network technology

The *Internet*: the specific internetwork that grew out of ARPANET.

Internetworks



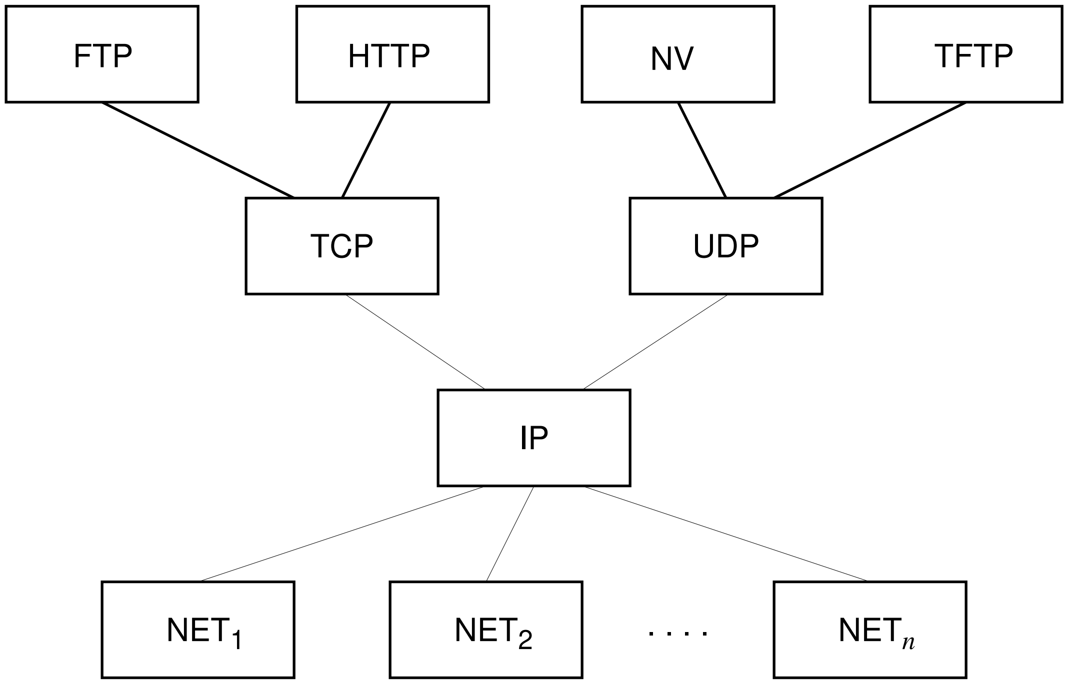
Internetworks



Internetworks

Interconnection approach

- application level
  - no uniform protocols required
  - applications must run on each gateway (internet router)
- network level (Internet approach)
  - uniform network level protocol (IP)
  - applications and upper layer protocols run on end-hosts only



Internet Characteristics

- Communication
  - Universal address space for hosts
  - Network-independent communications service model
  - Unreliable, best effort datagram delivery
- Size: 10^6 networks, 10^8 hosts, growing exponentially
- Administration
  - IAB (Internet Activities Board): oversee research and engineering task forces
  - FNC (Federal Networking Council): DARPA, NSF, NASA, DOE, HHS; funding and priorities
  - RFC (Request for comments): Internet tech reports; where standards are published: http://www.cis.ohio-state.edu/hypertext/information/rfc.html

Internet Protocol (IP)

Network-layer protocol that ties together heterogeneous physical networks

Issues:

- host addressing and address translation
- routing
- packet fragmentation/reassembly
- congestion control (not handled by IP)

IP Addressing

hierarchical addressing

- 32 bits usually written a.b.c.d
- network and host part
- three classes

		7	24
Class A	0	network	host

			14	16
Class B	1	0	network	host

				21	8
Class C	1	1	0	network	host

IP addresses identify host *interfaces*, not hosts!

Mapping IP Addresses

The IP address must be mapped to a lower level address to facilitate packet delivery on the local network.

- Direct mapping
  - ARPANET: 10.p.unused.i where the host is attached to port p at imp i
  - proNET: 192.12.69.x where x is the actual proNET address of the host
- Static table-based mapping
  - Table never changes because the mapping never changes.
  - X.25 uses an IP address to "phone number" table.
- Dynamic table-based mapping (ARP)
  - Maintain a dynamic table by querying hosts in the network for their address.
  - Necessary because the hardware address of a host may change (replacing Ethernet controller).

Address Resolution Protocol (ARP)

requires broadcast capability

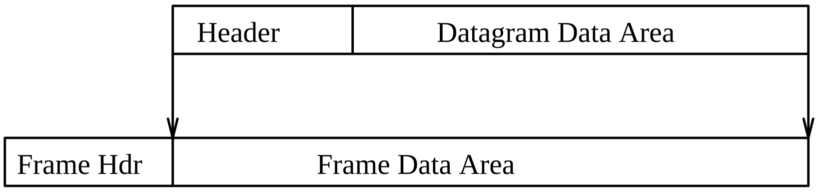
- Algorithm
  - requester broadcasts a packet including sender IP address, sender physical address, and target IP address
  - target host replies with a packet including sender IP address, sender physical address, target IP address (original requester), and target physical address (original requester); adds requester binding to its table.
  - each site caches bindings in a table; LRU when full; time to live associated with each binding, flush when out of date.
- Notes
  - powerful enough to support multiple physical networks (e.g. an extended LAN)
  - a similar facility (RARP) allows reverse translation of addresses that enables processors to learn their IP address (e.g. a diskless workstation that doesn't keep its address locally)

ARP Packet Format

16	16	8	8	16	48	32	48	32
HardwareType	ProtocolType	HLEN	PLEN	Operation	SourceHardwareAddr	SourceProtocolAddr	TargetHardwareAddr	TargetProtocolAddr

IP Datagrams

An IP datagram is encapsulated within a network frame.



IP packet header:

v4	hlen	Type of Svc	Length
Ident		Flags	Fragment Offset
Time to live	Protocol	Header Checksum	
Source Address			
Destination Address			
Options			Pad

IP Datagram Format

Size	Field	Description
4	version number	
4	header length	in 32 bit words
8	type of service	how datagram should be handled (precedence, low delay, high throughput, high reliability)
16	total length	in bytes including header and data
16	identification	uniquely identifies the original datagram
3	flags	fragmentation flags
	fragmentation	may the datagram be fragmented
	more fragments	are there more fragments
13	offset	position in the original datagram measured in units of 8 bytes (must fragment on an 8 bytes boundary)
8	time to live	time in seconds that a datagram may remain in the network
8	protocol	specifies the high level protocol to receive the frame
16	header checksum	
32	source addr	
32	destination addr	
	options	specify debugging or network control
8	option type	copy flag, option class, option number
8	option length	(optional)
	option data	(optional)
variable	padding	

Fragmentation and Reassembly

Problem

- Each physical network has a maximum transmission unit (MTU)
- Wasteful to restrict all IP datagrams to fit inside the smallest MTU
  - Minimal size MTU is 576 bytes (20 for header, 556 for data)

Solution

- Fragment a datagram as it is passed down to the physical network whenever the MTU is smaller than the datagram size (IP header plus IP body)
- Reassemble as it is received from the physical network at the destination

Fragmentation and Reassembly

<table>
<tr><td colspan="5">Start of header</td></tr>
<tr><td>ident = x</td><td></td><td></td><td>0</td><td>offset = 0</td></tr>
<tr><td colspan="5">Rest of header</td></tr>
<tr><td colspan="5">1400 data bytes</td></tr>
</table>

Unfragmented Packet

<table>
<tr><td colspan="5">Start of header</td></tr>
<tr><td>ident = x</td><td></td><td></td><td>1</td><td>offset = 0</td></tr>
<tr><td colspan="5">Rest of header</td></tr>
<tr><td colspan="5">556 data bytes</td></tr>
</table>

First Fragment

<table>
<tr><td colspan="5">Start of header</td></tr>
<tr><td>ident = x</td><td></td><td></td><td>1</td><td>offset = 556</td></tr>
<tr><td colspan="5">Rest of header</td></tr>
<tr><td colspan="5">556 data bytes</td></tr>
</table>

Second Fragment

<table>
<tr><td colspan="5">Start of header</td></tr>
<tr><td>ident = x</td><td></td><td></td><td>0</td><td>offset = 1112</td></tr>
<tr><td colspan="5">Rest of header</td></tr>
<tr><td colspan="5">288 data bytes</td></tr>
</table>

Last Fragment

Fragmentation and Reassembly

Each fragment is given its own IP header and is treated as an independent datagram.

- derived from the original header
- routed independently
- may arrive in different order

Reassembly

- Each original datagram is uniquely identified by the combination of ID, Source Address, Destination Address, and Protocol.
- The offset field describes the place to put the fragment in the original datagram.
- The more fragments field indicates that there are no more fragments to be processed.
- Problems
  - reassembly state (allocate on arrival of first fragment)
  - duplicate fragments (overwrite)
  - missing fragments (timeout and discard reassembly state along with received fragments)
- Algorithm: idea is to keep a hole descriptor for each known missing piece

IP Forwarding

Compare network portions of destination address and local address(es); if match, *direct delivery*, else *indirect delivery*.

direct delivery: source and destination are attached to same physical net (cloud).

- ask ARP for physical address of destination host
- encapsulate IP packet in a physical network frame
- send it

indirect delivery

- lookup routing table for the appropriate directly connected IP router, using destination address as key.
- use direct delivery to forward the packet to that router.

Datagram is forwarded from router to router until it can be delivered directly.

IP Forwarding

Routing table lookup:

- check for host-specific route
- else check for network-specific route
- else check for default route
- else declare routing error (drop packet)

Default routes

- helps keep routing tables small
- particularly effective for nets with only one router connected

Host-specific routes

- mainly used for testing and special case routing (security)

Scalability of Routing and Addressing

original scheme would have quickly caused exhaustion of address space (Class B addresses), and/or unmanageably large routing tables.

Observation: no need to maintain separate routes throughout the Internet for networks that are in close proximity.

Two approaches:

- share network addresses among multiple physical networks
  - transparent router
  - proxy ARP (ARP hack)
  - subnetting
- supernetting (classless routing, CIDR)

Scalability of Routing and Addressing

Transparent Router

- a router disguised as a host on a class A network
- works well only on class A networks

Proxy ARP

- allows sharing of an IP network number among two physical nets that use ARP
- a router connects the second, “hidden” network to the first net
- the router answer ARP requests for hosts on the hidden network (effectively lying about its IP address)
- transparent to other hosts
- does not generalize, security problems

<u>Subnetting</u>

- widely used
- multiple physical nets ("subnets") share a single IP network address (usually class B)
- some number of bits in the host portion of the IP address are interpreted as a *subnet* number.
- only local routers know about subnets
- local routers use a *subnet mask* to extract subnet number

Network Number	Host Number

Class B address

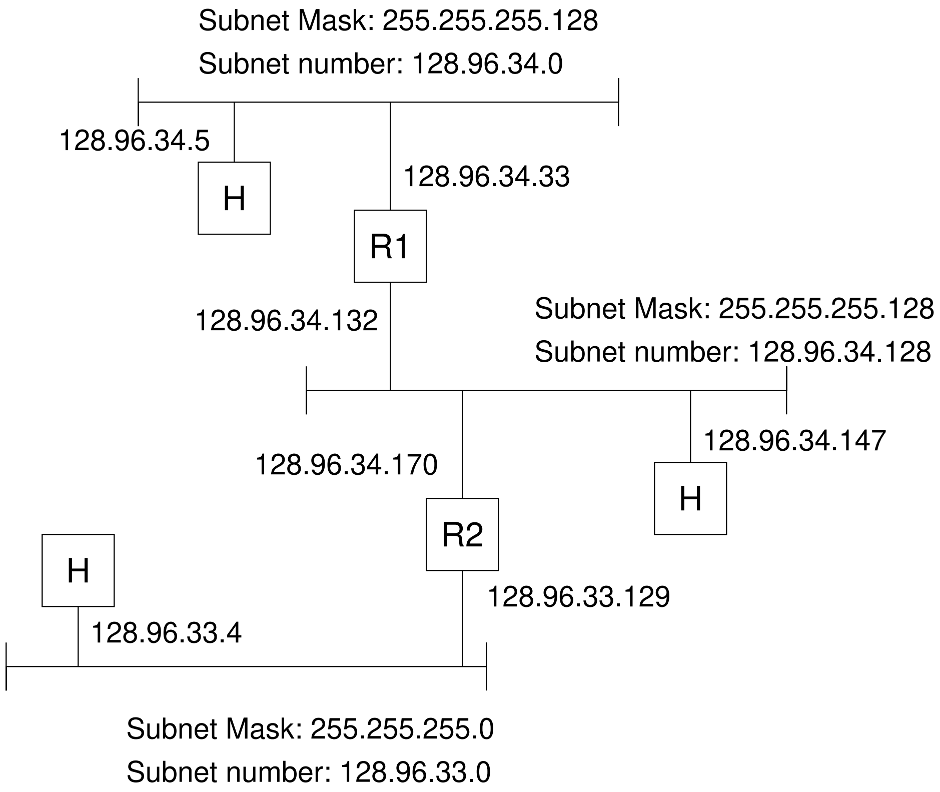
111111111111111111111111	00000000

Subnet Mask (255.255.255.0)

Network Number	Subnet ID	Host ID

Subnetted Address

Subnetting



Forwarding in the presence of subnets: as before, except

- routing table entries contain a subnet mask in additional to the network number.
- during lookup, bit-wise AND the destination address with the entry's subnet mask, then compare result with entry's network number.

Supernetting

Idea:

- assign blocks of contiguous network numbers to "close-by" networks
- for routing purposed, represent blocks by a single pair $< 1st\ network\ address, count >$
- in practice, restrict block size to powers of 2, use a bit mask (CIDR mask) to identify size of block.
- all routers need to understand CIDR addressing for this to be effective!

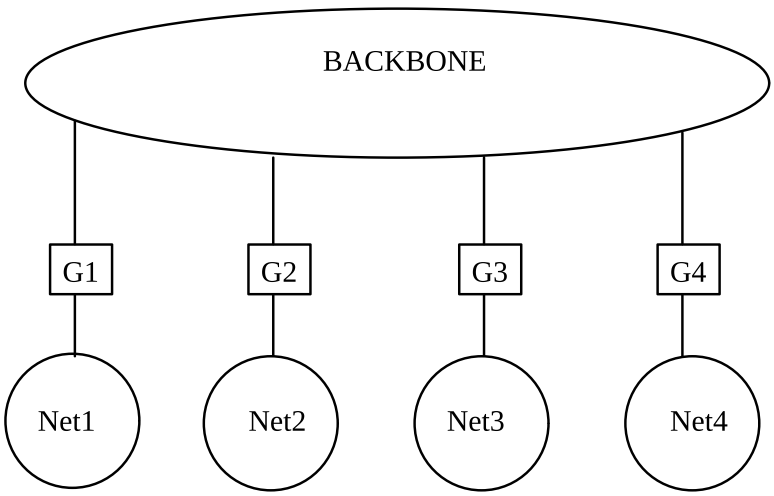
Internet Routing Architecture

What values to place in routing tables, and *how* to obtain those values?

Problem: dynamic routing algorithms do not scale to size of the Internet

Original Solution: Core system

- Single backbone (ARPANET) to which all networks connect via *core gateways* (routers).
- Core gateways exchange reachability information for all networks using GGP (gateway to gateway protocol, based on vector-distance). Metric is hop count.
- Outlying routers (noncore gateways) know about routes within their local network only. Default routes point toward the core.



Problems

The core system by itself did not scale well enough.

- Hidden network problem: no way for core gateways to automatically learn about configuration of the rest of the network; tables must be built by hand
- Core routers need to know about *all* networks
- Attempts to partition core system using default routes can lead to routing loops (consider a packet destined for a non-existent network).

Autonomous System (AS) approach

Allow several networks within a single administrative domain to be considered a single autonomous system; e.g., a University campus, a corporate network, a backbone.

Neighbors concept:

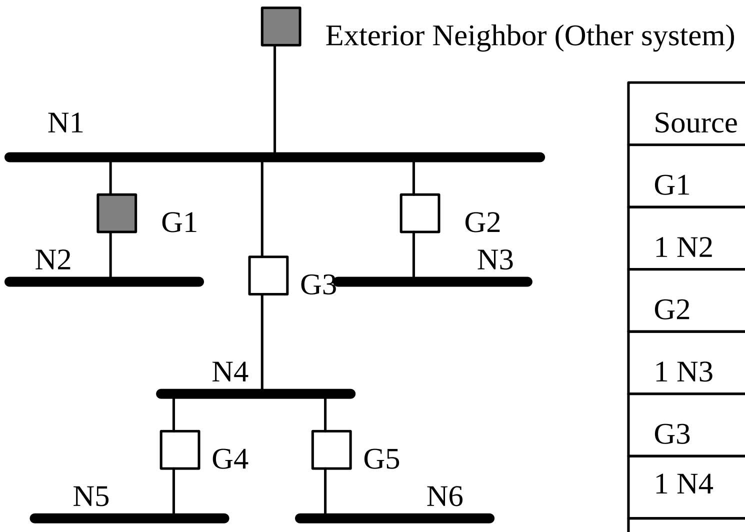
- Two routers are interior neighbors if they are in the same autonomous system; interior routers may choose their own protocol for sharing information (IGP)
- Two routers that share routing information are exterior neighbors if they are in different autonomous systems; a common protocol is defined for sharing information between exterior gateways (EGP)

ASs have unique 16-bit numbers, assigned by central authority

Exterior Gateway Protocol (EGP)

- Neighbor acquisition: one router requests that another be its peer; peers exchange reachability information
- Neighbor reachability: one router periodically tests to see if the other router is still reachable; exchange HELLO/ACK messages; since these can be lost, uses a k-out-of-n rule
- Routing updates: peers periodically exchange their routing tables (distance-vector).
  - EGP reports reachability only
  - reports list of routers, and list of destinations for each router

Example

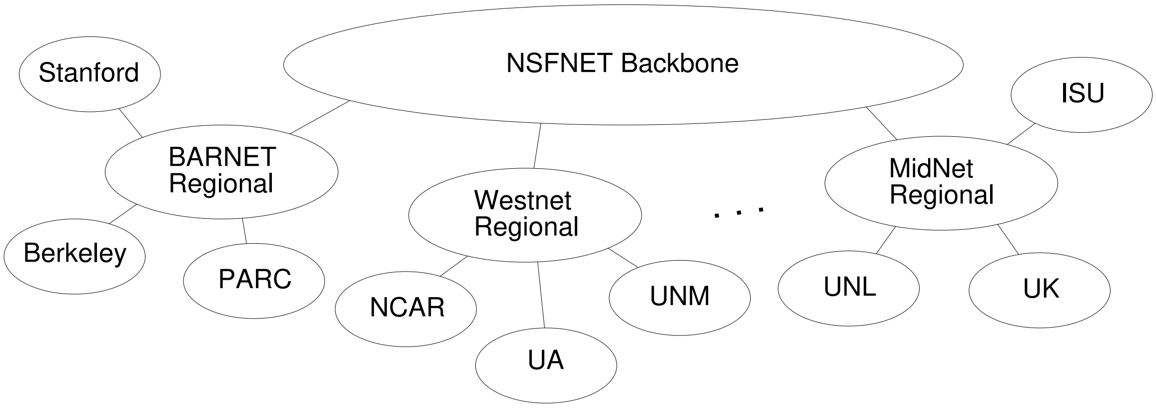


Source Net: N1
G1
1 N2
G2
1 N3
G3
1 N4
2 N5
2 N6

EGP

With EGP, the structure of the internet was constrained to a tree of autonomous systems rooted at a backbone network

- core routers, stub routers, interior routers



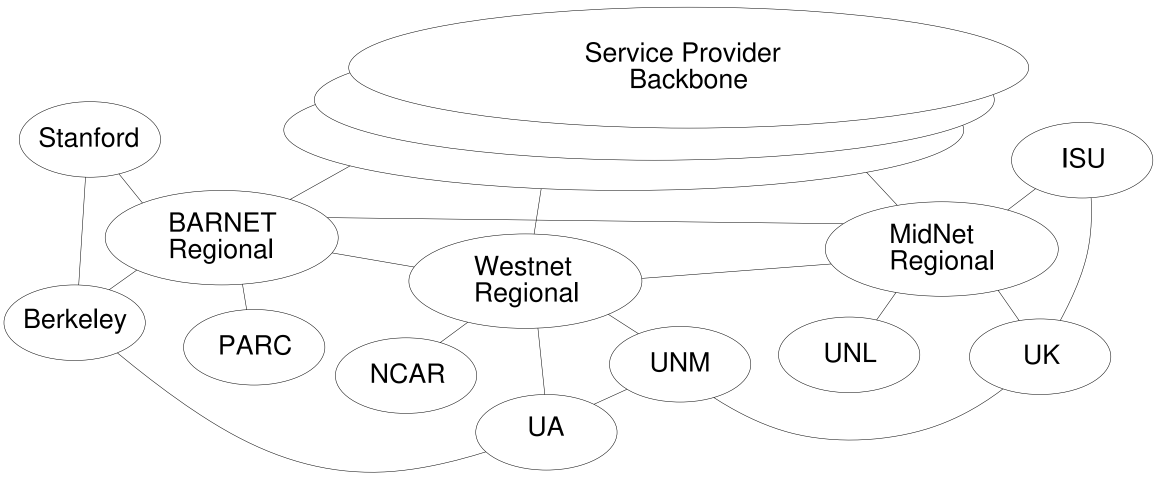
Border Gateway Protocol (BGP-4)

Currently used inter-AS routing protocol

Based on distance-vector algorithm

Three types of ASs:

- Stub AS: single connection to one other AS; local traffic only.
- Multihomed AS: connections to several ASs; refuses to carry transit traffic.
- Transit AS: connections to several ASs; carries both local and transit traffic.



BGP-4

Each AS has:

- one or more border routers
- one BGP speaker advertises
  - local networks
  - other reachable networks (transit AS only)
  - routes are enumerated lists of ASs (no count-to-infinity!)
  - route choice manually controlled ("policies")

Scalability:

- inter-domain (AS) routing O(# of ASs)
- intra-domain routing O(# of networks in AS)
- BGP speakers and border routers still need to know about every network
- BGP-4 understands CIDR network addresses!

Interior Gateway Protocols (IGPs)

Examples:

- GGP: gateway to gateway protocol, Bellman-Ford distance-vector algorithm.,
- Spread: core gateway protocol, link state routing algorithm.
- HELLO: from original NSFNET backbone gateways, uses delay as metric
- OSPF: recent standard; link-state algorithm.
- RIP: from XNS, distributed with Unix

Internet Control Message Protocol

An IP companion protocol that handles error and control messages.

- uses IP to forward its messages; used by IP for error reporting and control among routers and hosts.
- Time exceeded: router sends to source host if a datagrams TTL field reaches zero.
- Unreachable: router/host sends to source host to notify that datagram cannot be forwarded to the destination
  - network unreachable
  - host unreachable
  - protocol unreachable
  - port unreachable
  - fragmentation needed (and DF bit set)
  - source route failed
- Redirect: router sends to source host on directly connected network to inform it that there is a better route for a particular destination
- Source Quench: router sends to source when router has to drop one of the host's datagrams
- Echo: used to test host/router reachability; `/etc/ping`

IPv6

proposed standard for next generation IP

Major features:

- 128-bit addresses
- multicast support
- flow-based resource allocation
- authentication and security
- autoconfiguration
- end-to-end fragmentation
- protocol extensions

IPv6 Addresses

Classless addressing/routing (very similar to CIDR)

Notation: x:x:x:x:x:x:x:x (x = 16bit hex number)

- contiguous 0s can be compressed: 47CD::A456:0124
- IPv6 compatible IPv4 address: ::128.42.1.87

Address assignment

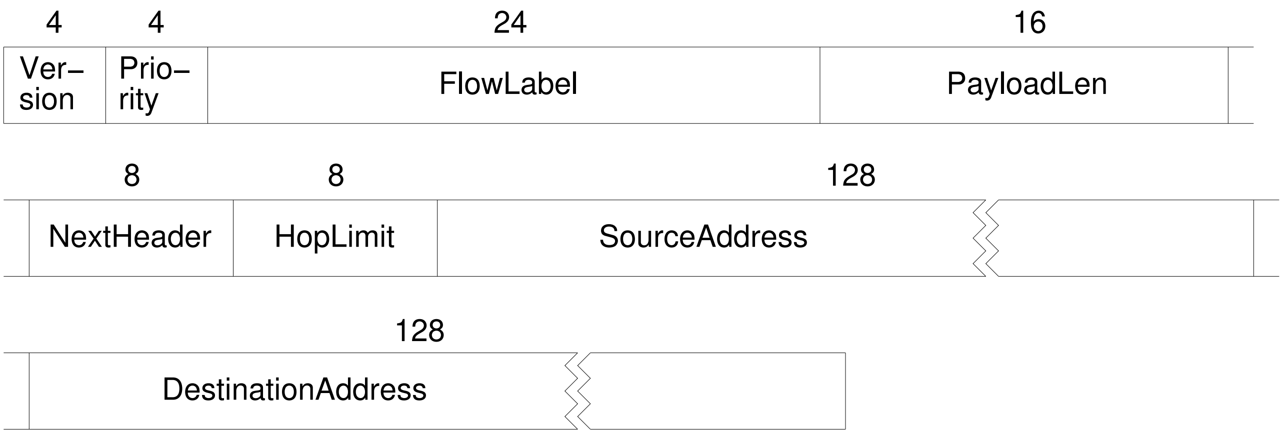
- provider-based
- geographic

3	m	n	o	p	125−m−n−o−p
010	Registry ID	Provider ID	Subscriber ID	Subnet ID	Interface ID

Prefix	Use
0000 0000	Reserved
0000 0001	Unassigned
0000 001	Reserved for NSAP Allocation
0000 010	Reserved for IPX Allocation
0000 011	Unassigned
0000 1	Unassigned
0001	Unassigned
001	Unassigned
010	Provider-Based Unicast Address
011	Unassigned
100	Reserved for Geographic-Based Unicast Addresses
101	Unassigned
110	Unassigned
1110	Unassigned
1111 0	Unassigned
1111 10	Unassigned
1111 110	Unassigned
1111 1110 0	Unassigned
1111 1110 10	Link Local Use Addresses
1111 1110 11	Site Local Use Addresses
1111 1111	Multicast Addresses

IPv6 header

- base header (40 bytes .,.)
- optional extension headers (fixed order, mostly fixed length)
  - fragmentation
  - source routing
  - authentication and security
  - other options



End-to-end fragmentation

8	8	13	3	1	32
NextHdr	Reserved	Offset	Res	M	Ident

- fragmentation normally restricted to source host
- requires path discovery
- reduces load on routers
- does not accommodate route changes

IP Multicast

2 Issues:

- multicast addressing and routing at the Internet level
  - class D IP addresses (IPv4)
  - multicast group management (IGMP)
  - multicast routing
- mapping IP multicast onto physical network multicast

Mapping IP multicast addresses to Ethernet/FDDI multicast addresses

- place least significant 23 bits of IP address into the 23 least significant bits of the ETH/FDDI address 01.00.5E.00.00.00

Internet Group Management Protocol (IGMP)

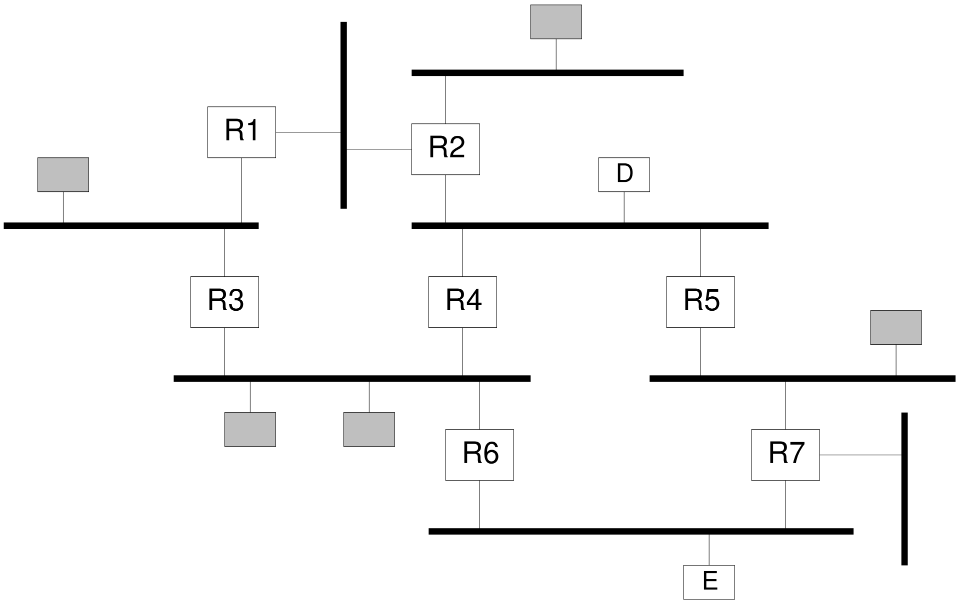
IGMP is another companion protocol of IP

- all hosts and routers that support IP multicast join the "all hosts" group.
- once a minute, a multicast router sends a poll message to the "all hosts" group.
- hosts start a timer with a random delay [0-10secs] for each group in which they have membership. Timer is cancelled if another host declares membership for the group. If timer goes off, send a message to the group, declaring membership in the group. Two events trigger start of a timer:
  - poll request from the multicast router
  - group membership request from an application

Multicast Routing

Link-state multicast routing

- add to link state the set of multicast groups that have membership on the other end of the link.
- multicast router compute multicast spanning tree on demand for each source/group pair, cache results.
- forward multicast packets accordingly



Multicast Routing

Distance-vector multicast routing (Reverse Path Multicast RPM)

- each physical network designates a "parent" router, relative to each source
  - router with shortest path is parent
  - ties are broken by lowest router address
- *leaf* network routers learn membership info via IGMP
- membership info propagated by augmenting distance vectors with set of groups with active membership
- routers forward a multicast packet
  - if the packet arrived on the shortest path link to the source, and
  - the router is the parent router relative the the source and the link it arrived on,
  - only on those links that have advertised membership in the group.

MBONE (Internet Multicast Backbone)

UNIX **Mrouted** program:

- multicast route propagation using *Distance Vector Multicast Routing Protocol (DVMRP)*
- multicast tunneling

Multicast tunneling

- Problem: most IP routers don't support multicast
- Solution: encapsulate multicast packets in regular IP packets for transmission across sections of the Internet that don't support multicast (tunneling)

Congestion Control

Congestion: performance degradation due to excessive load on the network.

- affects packet-switched networks (physical and internetworks)
- Cause: dynamic resource allocation ("statistical multiplexing")

Contributing factors:

- slow switch fabric (engineering tradeoff; can use faster switch)
- input traffic exceeds capacity of output lines
  - traffic on input lines funnels into a few output lines
  - capacity of input line exceeds that of output line

Note distinction:

- Flow control: preventing sender from overloading receiver (end-to-end)
- Congestion control: preventing sender(s) from overloading network (global)

Taxonomy of Congestion Control Schemes

- router-based: CC handled
  - inside the network
  - at network layer
- host-based: CC handled
  - at edges of network (end hosts)
  - at transport layer
- reservation-style CC
  - *a priori* reservation of resources along the path
  - implies router-based
- feedback-style CC
  - sender host adjusts rate according to feedback from network
  - explicit feedback: notification from router(s)
  - implicit feedback: dropped packets, delay, bandwidth
- window-based CC
  - sender adjusts amount (window) of in-transit data
  - controls buffer consumption in routers
- rate-based CC
  - sender adjusts rate of transmitted data
  - controls load on links in network

Congestion Control

Goals:

- effective resource allocation
  - maximize network throughput
  - minimize delay
  - prevent congestion collapse
- fairness
  - ensure each flow gets fair share of network resources

Congestion Control

Current Internet: host-based, feedback style, window-based

Window-based congestion control:

- sliding-window protocol used at transport layer
- for maximal xput, each flow should have a packet queued at each router (pipelining!)
- having more than one packet queued at each router can only give potential xput benefit at the cost of other flows
- two factors control size of send window:
  - buffer space at receiver (flow control)
  - buffer usage in routers (can't directly observe!)

Router Buffer Management

Problem: one output queue consumes all of router's buffers

m = max # buffers assigned to one queue

k = # buffers

q = # queues

- $m = k$: unrestricted case
- $m = k/q$: static partitioning
- $m = k/\sqrt{q}$: good choice for wide range of traffic
- in addition, dedicating a small # of buffers (e.g., 1) to each output line prevents starvation

Router Queuing Disciplines

Queuing discipline allocates

- bandwidth (which packet transmitted next)
- buffer space (which packet is discarded)

FIFO queuing

- simplest
- predominantly used in current Internet
- packets are sent in order they arrived (relative to an output queue)
- if a packet arrives while the appropriate output queue is full, that packet is dropped
- does not discriminate among sources
- aggressive sources can acquire a disproportionate share of resources

Router Queuing Disciplines

Fair queuing

- maintain a separate queue for each flow
- if running out of buffers, drop packet at the end of longest queue
- transmit by simulating bit-by-bit round-robin
  - transmit packet with earliest finish time, where
  - earliest finish time = max(arrival time, actual finish time of previous packet on same flow) + transmit time

Weighted Fair queuing

- add additional weight to each flow
- can assign arbitrary share of link capacity to flows
- requires a form of reservation

Hop-by-Hop Flow Control

Idea: flow control on each link of a path jointly controls congestion

- Problem: sender is not throttled until buffers in all routers along the path are full (fairness!)
- need flow-based queuing alg. (e.g., fair queuing) in routers to achieve fairness
- each link in network must support flow control (too restrictive for an internetwork)

Internet (TCP) Congestion Control

Idea: Transport protocol "learns" appropriate send window size to

- avoid congestion collapse
- achieve fairness (sort of)

Send window = min(Flow-control window, Congestion window);

- flow-control window advertised by receiver
- congestion control window adjusted based on implicit feedback from network:
  - dropped packets (current TCP)
  - delay, throughput (experimental TCP Vegas)

Maintaining the congestion window: *additive increase, multiplicative decrease*

- Assumption: dropped packets indicate congestion
- start with a congestion window of one packet (minimum)
- if no dropped packets during past congestion window, increase window by one packet
- if dropped packet, cut congestion window in half

Internet (TCP) Congestion Control

Problem: TCP periodically creates congestion to learn the appropriate send window size.

Idea: try to detect impending congestion before packets are dropped

- increase in delay due to growing queue length in routers
- flattening of throughput while approaching point of congestion (TCP Vegas)

DNA DECbit

explicit feedback-based CC

- routers set a congestion bit in header if average queue length > 1. (averaged over last busy+idle+current busy period)
- receiver copies congestion bit into ACK
- sender monitors congestion bits, sets congestion window accordingly
  - if $\geq 50\%$ of packets in last congestion window had bit set, multiply congestion window by .875
  - else, increase congestion window by one packet

Virtual Clock

Rate-based, reservation style

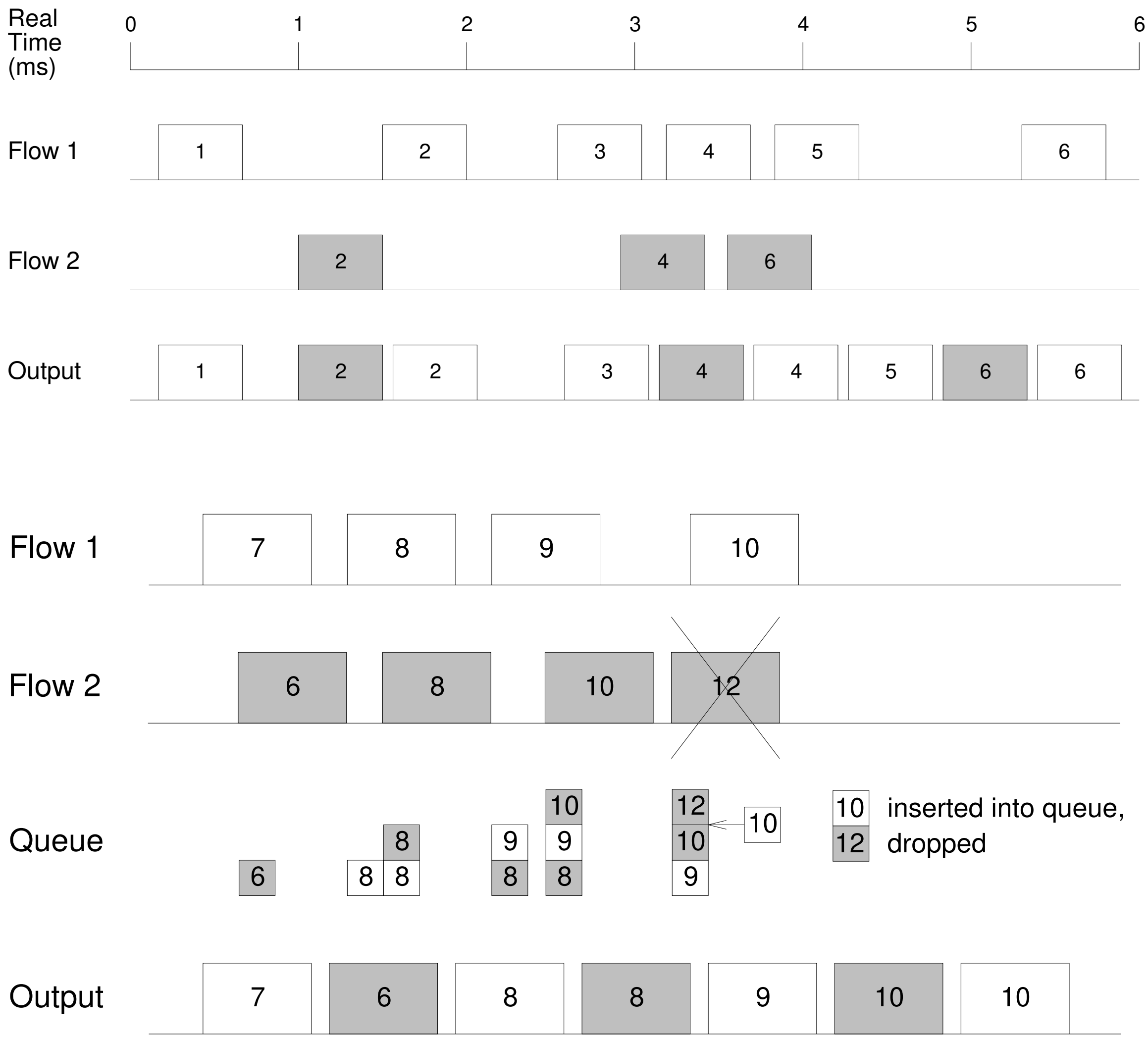
Addresses shortcomings of feedback-based CC w.r.t applications that need predictable performance (bandwidth, delay)

- applications negotiate resource needs during a flow setup phase. Parameters:
  - average rate (AR)
  - averaging interval (AI)
  - $(1/AR <= AI <= total\ duration\ of\ flow)$

Queuing

- maintain virtual clock VC_i for each flow i
- upon arrival of first packet of flow i, set $VC_i = realtime$
- for each packet on flow i:
  - stamp packet with value of VC_i
  - advance VC_i by "virtual transmission time" of packet (transmission time at requested rate AR)
- packets for each output line queued and serviced in order of increasing timestamps
- when queue is full, drop packet with largest timestamp

Virtual Clock



Virtual Clock

Monitoring Flows

Problem: don't want to allow flow to "accumulate credit", then use it all at once.

- $AIR_i = AI_i * AR_i$
- every AIR bytes of data, inspect flow:
  - if $VC_i > realtime$, send notification to sender
  - if $VC_i < realtime$, let $VC_i = realtime$ (prevents accumulation of credit)

Flow 0 is reserved for best-effort service.

Rate-based Congestion Control

In practice, burstiness of traffic sources at multiple time scales has to be controlled by separate mechanisms.

- bursts of short duration: absorbed by router buffers
- bursts of medium duration: "traffic shaping" at network edges (e.g., leaky buckets)
- bursts of longer duration: re-negotiate the flow

End-To-End (Transport) Protocols

Transport Layer in OSI terminology

Provide process-to-process communication.

Service Model:

- Addressing
  - direct (process id)
  - indirect (port)
- Connection-oriented/Connection-less
- Synchronous/Asynchronous
- Reliable/Unreliable (Quality of Service)
- Message-oriented vs. byte-stream
- Typed vs. untyped data ("inband" vs. "out-of-band" data)

End-To-End (Transport) Protocols

Issues:

- multiplexing (several process-to-process connections over a single network layer host-host connection)
- connection management
- fragmentation/reassembly
- reliability
- ordering
- flow control
- congestion control

UDP: User Datagram Protocol

Minimal transport protocol: Adds multiplexing only

- unreliable, unordered datagram service
- endpoint identification based on ports
- UDP endpoint address consists of IP address plus 16-bit port number
- optional 16-bit ones complement checksum (packets with bad CS are silently discarded)

16	16	16	16
Src Port	Dest Port	Check Sum	Length

TCP: transmission control protocol

- Reliable byte-stream ("virtual circuit") connection
  - connection establishment period
  - connection termination period
- Stream oriented
  - sending process writes bytes in arbitrary quantities
  - TCP packs bytes into segments to send via IP
  - receiving process reads bytes in arbitrary quantities
  - order is preserved
- Full duplex connection
  - information passes in both directions
  - this allows ACKs to be piggybacked on segments passed in the opposite direction
  - if one direction closes the connection it becomes half-duplex

TCP

- Buffered transfer
  - in general, number of bytes written is not equal to the number of bytes in a segment is not equal to the number of bytes read
  - bytes are buffered at both ends of a connection
  - TCP sends segment when
    - enough bytes written by application to fill a segment
    - buffered bytes have aged enough
    - application explicitly requests a push (flush)
- Unstructured byte stream
  - the data sent between processes has no meaning to TCP, it is just a stream of bytes
  - per-byte acknowledgments
    - each byte has a sequence number
    - ACKs are cumulative, indicate next byte expected

TCP segment header

16	16	32	32	4	6	6	16	16	16	variable
Src Port	Dest Port	SequenceNum	Acknowledgement	Off	res	Flags	Advertised Window	CheckSum	URP	options

- 16 bit source port, 16 bit destination port
  - TCP has separate port space
  - TCP connection specified with a ¡host,port¿ pair for both source and destination
- 32 bit sequence number
  - the sequence number of the first data octet in the segment
  - if SYN bit set then this is the "initial sequence number"
- 32 bit acknowledgment number
  - if ACK bit set then this gives the next sequence number that the sender of the message expects to receive
  - this number is sent with every message even if it gives redundant information
- 4 bit header length or data offset field
  - the number of 32 bit words in the TCP header
  - indicates where the data begins
- 6 bit reserved

TCP segment header

- 6 bit control field
  - URG bit: urgent bit, deliver immediately
  - ACK bit: acknowledgment field is valid
  - PSH bit: push request field (flush)
  - RST bit: reset the connection
  - SYN bit: synchronize the sequence numbers (establish connection)
  - FIN bit: no more data from sender (close connection)
- 16 bit window field
  - the number of bytes that the sender of the message is willing to accept beginning with the one indicated in the acknowledgment field
  - receiver "opens" window when receiving user process reads bytes and/or when OS gives the connection more buffers
- 16 bit checksum
  - it need not be computed
  - computed relative to the full segment plus a pseudo-header including the source and destination host addresses
- 16 bit urgent pointer
  - when URG bit is set this field gives the offset from the first byte in the segment where the urgent data is located
  - used, e.g., by telnet to send stop characters
- Options and padding

TCP Sliding Window

- go back n
- no NAKs
- "fast retransmit": multiple ACKs for same seqnum trigger retransmission (3 duplicate ACKs)
- advertised window (flow control)
- adaptive retransmission (discussed below)
- 32-bit sequence number
  - given 60 second segment lifetime, not sufficiently large for high-speed networks
  - TCP extension adds a 32-bit timestamp
- 16-bit advertised window size (64 KBytes)
  - insufficient for even moderate speed networks
  - TCP "window scaling option" allows window field to be scaled by a given factor

Adaptive Retransmission

Recall that TCP is intended for use in an internetwork.

Implications:

- since there are many different routes through the network, segments are not guaranteed to take the same route
- since an internet is composed of many different types of networks, two routes may vary widely in the time taken to transmit a segment from source to destination
- therefore, the distribution of transmission delays has a very high variance over time and over connections

TCP uses an adaptive retransmission scheme in order to accommodate widely varying network delays; the timeout associated with a connection is a function of measured round trip delay time

Adaptive Retransmission

Timeout is computed (as recommended in original TCP spec) by:

- $RTT_{new} = ((1 - \alpha) * RTT_{old}) + (\alpha * M)$, M is the measured round trip time for the last packet acknowledged, spec recommends $\alpha = 0.1$
- $Timeout = \beta * RTT$, spec recommends $\beta = 2$ (large β wastes time waiting, small β is too eager)

How should M be measured?

- ACKs for retransmitted segments are ambiguous
- assume ACK is for retransmitted segment:
  - underestimate M
  - cause more retransmissions, congestion
- assume ACK is for original message:
  - overestimate M
  - poor performance

Karn/Partridge Fix

- disregard RTT samples from retransmitted segments
- use exponential backoff to set retransmission timer for retransmitted segments
- keep using the resulting timer value until a valid RTT sample can be obtained

New RTT calculation

Problem: old RTT calculation did not adapt to high RTT variance

Solution: estimate both RTT and RTT variance (actually, mean deviation)

- $diff = M - RTT_{old}$
- $RTT_{new} = RTT_{old} + (\delta * diff)$
- $dev_{new} = dev_{old} + \delta(\mid diff \mid - dev_{old})$
- $Timer = RTT_{new} + 4 * dev_{new}$

Connection establishment and termination

Opening a connection requires a three way handshake protocol (active/passive):

- Protocol
  - A-to-P: SYN x
  - P-to-A: SYN y, ACK x+1
  - A-to-P: ACK y+1
- A and P both time out if they don't get an ACK; A doesn't need to know that its ACK got to P; A starts sending data with ACK set to y+1
- Example

TCP A	DIR	SEQ	ACK	CTL	**TCP B**
closed					closed
SYN sent	A-B	100		SYN	SYN received
established	B-A	300	101	SYN,ACK	SYN received
established	A-B	101	301	SYN,ACK	established

- Situations with half-open connections and simultaneous opens are more difficult

Connection establishment and termination

Closing a connection:

- Simple Protocol
  - A-to-B: FIN x
  - B-to-A: ACK x+1
- The protocol is more difficult because
  - nothing follows the last ACK
  - both have to close
- Example

TCP A	DIR	SEQ	ACK	CTL	**TCP B**
established					established
fin-wait 1	A-B	100	300	FIN,ACK	close wait
fin-wait 1	B-A	300	101	ACK	close wait
time-wait	B-A	300	101	FIN,ACK	last ack
time-wait	A-B	101	301	ACK	closed
closed					closed

TCP Congestion Control

- Assumption is that packets are rarely lost due to line errors; therefore, it is reasonable to assume that a timeout indicates network congestion
- But, can't tell if underutilizing the network
- Algorithm: add two variables to per-connection state:
  - *congestion window* (CW)
  - target congestion window (TCW)
  - when starting a connection, set
    CW = 1 segment; TCW = advertised window
  - for each ACK received,
    if CW $<$ TCW then CW = CW + 1 segment (exponential increase),
    else CW = CW + 1/CW (linear increase)
  - when there is a retransmission timeout, let TCW = CW/2; CW = 1 segment
  - send min(advertised window, CW)

TCP Congestion Control

- initially, window opens exponentially (slow start)
- once TCP is reached, continue with linear increase to avoid "overshooting"
- Problem: retransmission timeout are a late indication of dropped segments
- Idea: duplicate ACKs are an early indication of possible segment loss

Fast retransmit/ Fast recovery

- when three duplicate ACKs arrive
  - retransmit first unacknowledged segment (fast retransmit)
  - let TCW = CW = CW/2 (fast recovery)
- in practice, detects dropped segments early and avoids retransmission timeouts about half of the time

TCP Vegas Congestion Control

Idea: estimate amount of excess data in the network by measuring delay and throughput; avoid segment dropping altogether

- $BaseRTT$ = RTT when there is no congestion
- $ExpectedRate = CW/BaseRTT$
- $ActualRate$ = sending rate during last RTT
- $diff = ExpectedRate - ActualRate$ (≥ 0 by definition)
- two thresholds $\alpha < \beta$
- if $diff < \alpha$, increase CW linearly during next RTT
- if $diff > \beta$, decrease CW linearly during next RTT

Silly Window Syndrome

- Receiver computes offered window, sender computer usable window (offered - outstanding)
- Consider the following scenario:
  - R: offered window of 1000 bytes
  - S: sends 5 segments with 200 bytes each
  - S: ACK for first segment arrives, S computes usable as 200 bytes
  - Continue in equilibrium for awhile (S sends 200 byte segments as R pulls them off)
  - R: application reads 5 bytes, advertises 5 bytes, S sends a 5 byte segment
  - When ACK for 5 byte segment arrives, S sends another 5 byte segment; this continues forever
- Intuitively: once a small segment gets into the stream, it can't be removed.
- Solution is for sender to accumulate ACKs until usable window is big enough to make use of; receiver can also wait a small amount of time before sending an ACK (to see if more is being read by application)

Remote Procedure Call Protocols

RPC: Communication service suitable for client/server style (asymmetric) distributed computations

Components:

- argument/result marshalling: package argument/result data structures into messages
- demultiplexing: allow calls to many remote procedures using one endpoint (port)
- request/reply protocol: implement RPC semantics
- bulk data transfer: transfer arguments/results larger than network layer MTU

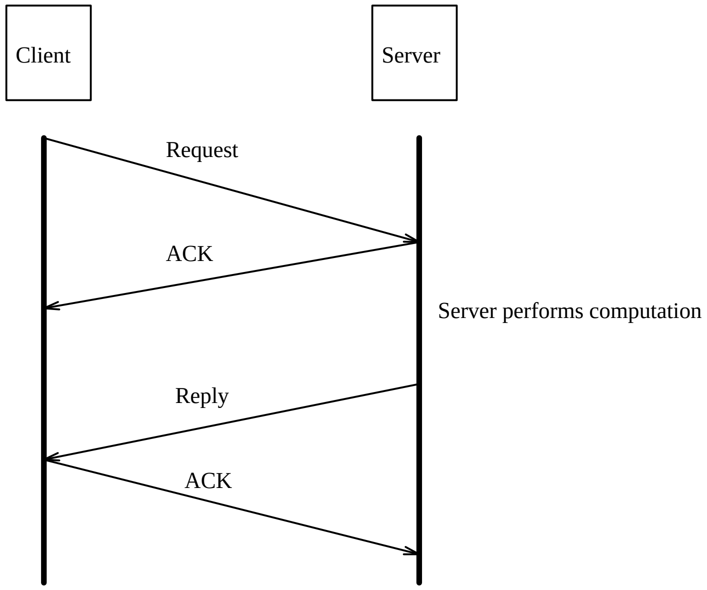
Request/Reply Protocol (CHAN)

Characteristics (asymmetric protocol):

- client: package arguments, send request, wait for response
- server: wait for request, extract arguments, execute server procedure, package results, send response, continue
- match up request messages with reply messages (synchronize)
  - allow one outstanding request per channel
  - in order to get concurrency we must use multiple channels
- preserve "at-most-once semantics" in case of failures
  - network drops packets
  - network duplicates packets
  - server host fails
  - server host is slow or has lots of work to do

Request/Reply Protocol (CHAN)

Basic Algorithm



- sender buffers request until it gets an ACK and receiver buffers reply until it gets an ACK
- each side maintains a retransmit timer
- the sequence number is used to identify a particular request/reply pair, duplicates are ignored
- Header format $< type, protnum, channel, MID, BID >$
  - type: REQ, REP, ACK
  - protnum: identifies high level protocol for demultiplexing
  - channel: unique id for this channel
  - MID: sequence number of the current request/reply pair
  - BID: boot id; incremented every time a host boots

Improvements

- Implicit ACKs: reply acknowledges request and next request acknowledges previous reply.
  - Hard to tell the difference between lost requests and slow servers
  - hard to tell the difference between lost replies and no more requests
  - PROBE messages: client periodically sends “are you alive” message if procedure takes a long time to execute; server responds with ACK

Bulk Data Transfer Protocol (BLAST)

Messages that do not fit in a single packet must be fragmented.

- Could use a standard protocol such as stop-and-wait or sliding window to transfer requests and replies.
- However, properties of RPC permit a more efficient strategy:
  - Fragment a message, send all of the packets back to back, reply with a single acknowledgment, retransmit missing pieces
  - It is like having a window size equal to the message size; we need to be able to buffer the entire message anyway, so we certainly have the buffer space.
  - retransmit individual fragments for performance, not reliability (reliability ensured by CHAN semantics)

BLAST

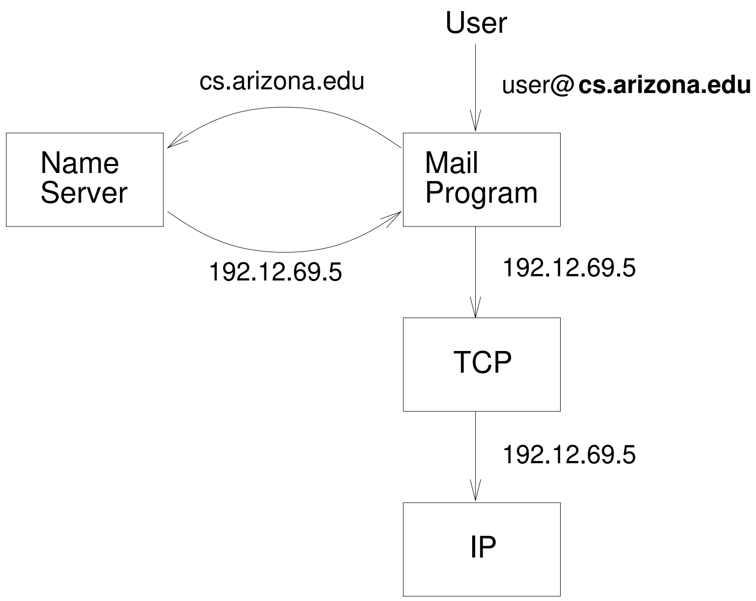
Header:

$< type, protnum, mid, fragmask, numFrags, length >$

- sender fragments message and sends each packet back-to-back; start DONE timer
- receiver gets fragments and puts message back together; start LASTFRAG timer when first fragment arrives
- receiver:
  - if all of the fragments arrive, then pass the message up to the high level protocol (the response will acknowledge the message)
  - if the last fragment arrives but some are missing, then send a SRR (selective retransmit request) for the missing pieces; start RETRY timer
  - if LASTFRAG timer expires, then send SRR for the missing pieces; start RETRY timer
  - if RETRY timer expires for the first time, then resend last SSR
  - if RETRY timer expires for the second time, then free buffers, cancel LASTFRAG, and give up on this message
- sender:
  - if the sender receives an SRR, then resend the requested fragments and reset DONE
  - if DONE timer expires, then free buffers and give up on this message
  - If a response arrive from the server, free buffers and cancel DONE.

Domain Name System (DNS)

Distributed database; provides a mapping from domain names (suitable for human use) to IP addresses



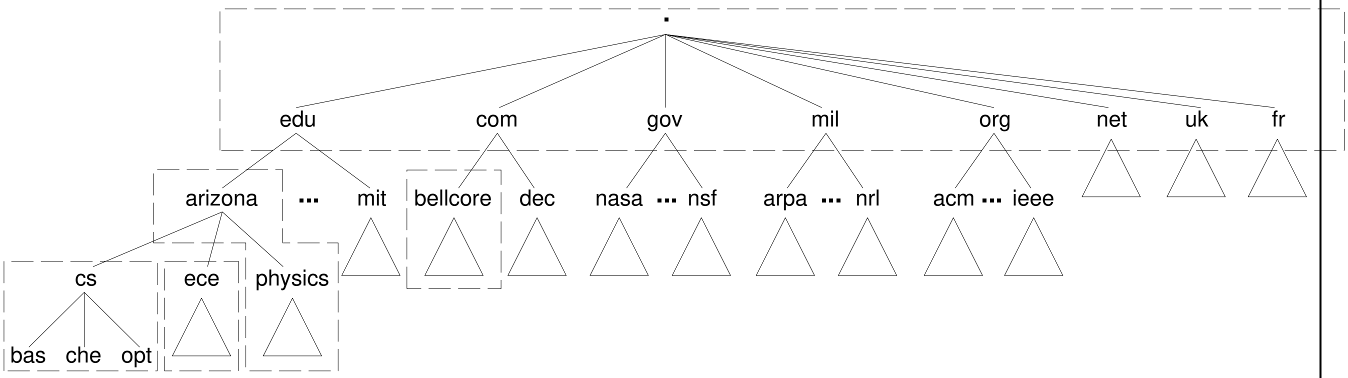
Key Issue: Scalability

- name space
  - hierarchical name space allows distributed management
- name servers
  - distributed database of bindings
  - name servers replicated for fault resilience
- resolution protocol

<u>Domain Hierarchy</u>

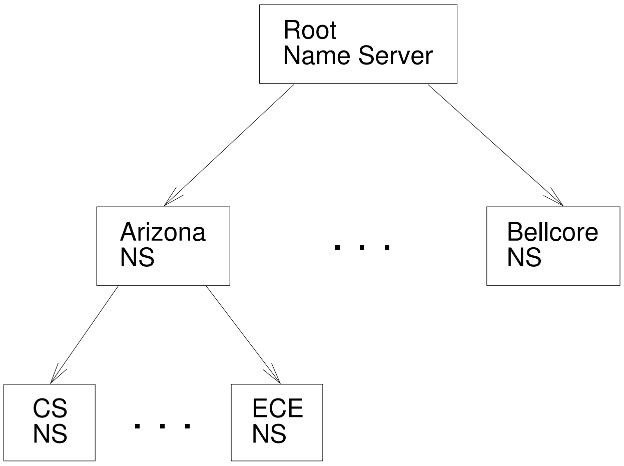
Hierarchical name space for Internet "objects" (mostly IP addresses)

Domain names interpreted from right to left:
`baskerville.cs.arizona.edu`



- domain hierarchy subdivided into *zones*
- zones correspond to administrative authorities responsible for that part of the name space
- each zone is implemented by a set of replicated name servers

Name Server Hierarchy



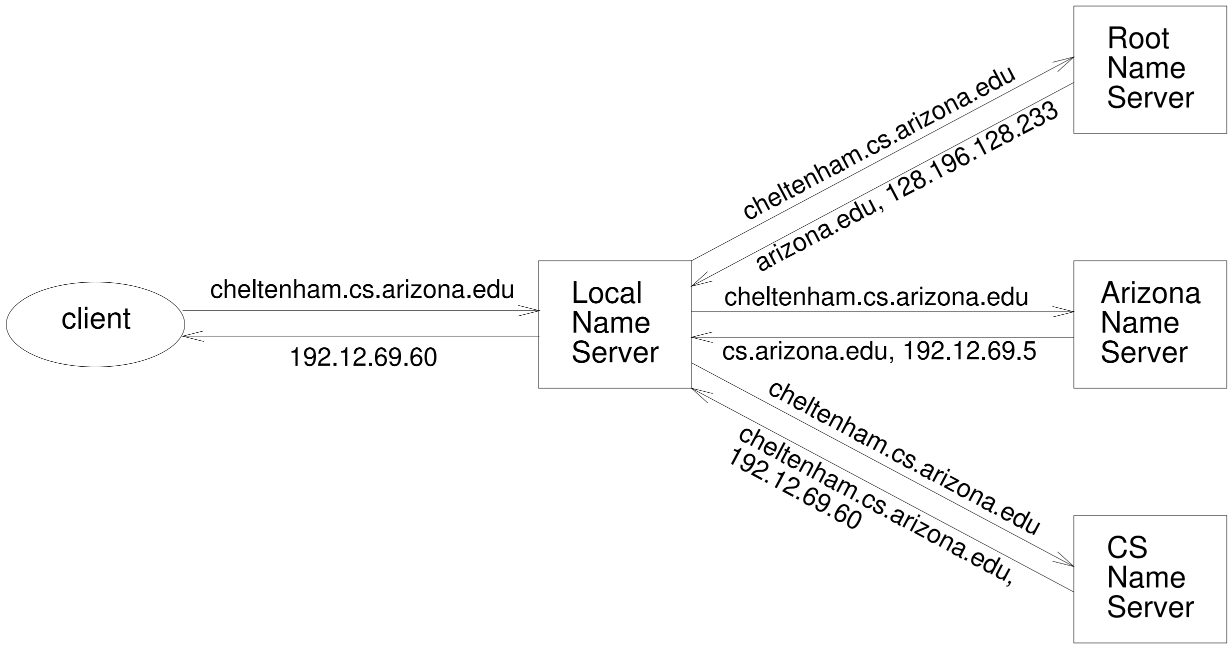
Each name server maintains a set of *resource records*:
$< name, value, type, class, ttl >$

- type: specifies interpretation of value field
  - A: IP address
  - NS: domain name of name server
  - CNAME: canonical name of host (aliases)
  - MX: domain name of mail server
- class: allows use of DNS for other purposes (each class has its own definition of types)
- ttl: specifies how long the binding may be cached

Name Servers

- root name servers maintain “pointers” to second level name servers:
  - a set of NS records with the domain name of each replicated server
  - a set of A records with the IP addresses of the replicated servers
- second level maintain “pointers” to third level servers
- A, CNAME, MX records for hosts in the same zone

Name Resolution Protocol



- each host is configured with address of local name server, and local domain name
- each local name servers is configured with address of root name server

Name Resolution Protocol

- resolution:
  - host adds local domain suffix to requested name, if necessary
  - host checks cache; if miss, query local name server
  - local NS checks cache; if miss, query root name server
  - root name server return NS entry with longest suffix match
  - local NS queries NS returned by root server
  - NS returns longest suffix match, and so on
- replicated name servers are contacted in sequence until a response is received

End-To-End Data Presentation

"Presentation Layer" in OSI reference model

Representation of application data in network messages, such that "meaning" of data is preserved. Issues:

- conversion between architecture-specific representations of elementary data types
  - character encoding
  - integral type conversion (byte swapping)
  - floating-point conversion
- marshalling ("flattening") of structured data
  - padding
  - pointers
- compression
- encryption

Conversion Strategy

Canonical intermediate form

- sender converts to standard representation
- receiver converts from standard representation
- only one pair of conversion functions necessary at any host
- inefficient when used between entities with identical native data representation

Receiver makes it right

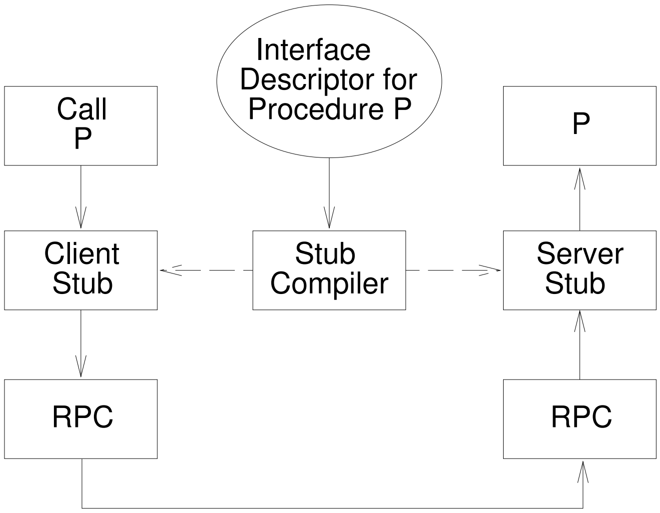
- sender transmits using its native representation
- receiver converts from sender's to its native representation
- no work required when used between entities with identical native data representation
- hosts must have a conversion functions from every representation to its own native representation

Structure Information

Information about the structure of data contained in a message. Two approaches:

- self-describing ("tagged")
  - type tag
  - length tag
  - architecture tag
- not self-describing ("untagged")
  - rely on application to provide structure info
  - avoids redundant tags
  - does not fit layering principle

Marshalling Functions



Sun XDR

- eXternal Data Representation scheme used by Sun RPC
- canonical intermediate form
- not self-describing
- supports C types (except function pointers)
- stubs are compiled from RPC interface description

ASN.1 (Abstract Syntax Notation)

OSI presentation layer protocol standard

- canonical intermediate form
- self-describing
  - type tags
  - length tags
- supports C types (except function pointers)
- data item represented by $< tag, length, value >$

NDR (Network Data Representation)

standard used in OSF's Distributed Computing Environment (DCE)

- receiver makes it right
- not self-describing
- supports C types (except function pointers)
- architecture tag at front of message

Integer Rep	Char Rep	FloatRep	Extension 1	Extension 2

Data Compression

Motivation:

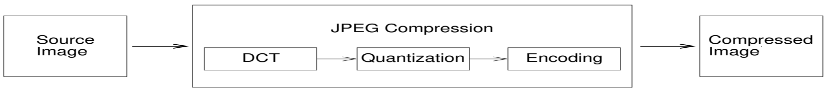
- save $ (if charged per amount of data transferred)
- increase effective bandwidth
  - possible if $B_c > \frac{r}{r-1} * B_n$
  - B_c = bandwidth of compression/decompression function
  - B_n = usable network bandwidth
  - r = compression ratio

  - loss-less compression
  - lossy compression

Loss-less Compression

- run-length encoding (RLE)
  - replace consecutive occurrences of same symbol with $< symbol, count >$
- differential pulse code modulation (DPCM)
  - choose an appropriate reference symbol
  - encode each symbol by its difference from the reference symbol
- dictionary-based (e.g., Lempel-Ziv)
  - choose a dictionary of sequences that frequently occur in the data
  - encode sequences by their index in dictionary
  - dictionary usually chosen for a particular data object, added to compressed object.
  - used in `compress`, `gzip`, GIF.

JPEG Image Compression



Discrete cosine transform (DCT)

- transforms 8x8 pixel matrix to 8x8 matrix of frequency coefficients

Quantization

- quantizes (rounds) coefficients according to a table
- $QuantizedValue(i, j) = IntRnd(DCT(i, j)/Quantum(i, j))$

Encoding

- encode quantized coefficients using RLE along a zig-zag traversal of the coefficient matrix
- DC coefficients of successive 8x8 matrices are encode using DCMP

MPEG Video Compression

uses compression similar to JPEG in each frame

in addition, take advantage of inter-frame correlation

- I frames (reference frames)
- P frames (difference from previous I frame)
- B frames (interpolation between previous or subsequent I or P frame)

Network Security

Threats:

- data privacy
- integrity (spoofing, tampering)
- unauthorized access to services
- denial of service

Tools:

- encryption
- authentication
- access control

Encryption

Basic idea: store and transmit information in an encoded form that bears no obvious relationship with the original message.

Secret-key cryptography

- M: plain-text messages
- E: encrypted message
- K: encryption key
- $E = f(M, K)$ (f is reasonably efficient to compute)
- $M = f^{-1}(E, K)$ (f^{-1} is reasonably easy to compute)
- $M = f'(E)$ (very hard to compute)
- $K = f''(M, E)$ (hard to compute)

- privacy: have to know the secret key to read the message
- authenticity: if it can be decrypted with the secret key, then the sender must have known the secret key

Problem: a secure channel is needed to distribute secret keys

Public-key Cryptography

- M: plain-text messages
- E: encrypted message
- K_1, K_2: encryption key pair
- $E = f(M, K_1)$
- $M = f(E, K_2)$
- $M = f(f(M, K_1), K_2)$
- $M = f(f(M, K_2), K_1)$

Idea: publish one of the keys of a pair, keep the other key secret

- anyone can encrypt a message for you using your public key, only you can decrypt it using the secret key.
- if you send a message encrypted with your secret key, anyone can verify the authenticity of the message by decrypting it with your public key.

Cryptographic Checksum (Message Digest)

- M: plain-text messages
- d: message digest (cryptographic checksum)
- $d = f(M)$ (very hard to invert f)

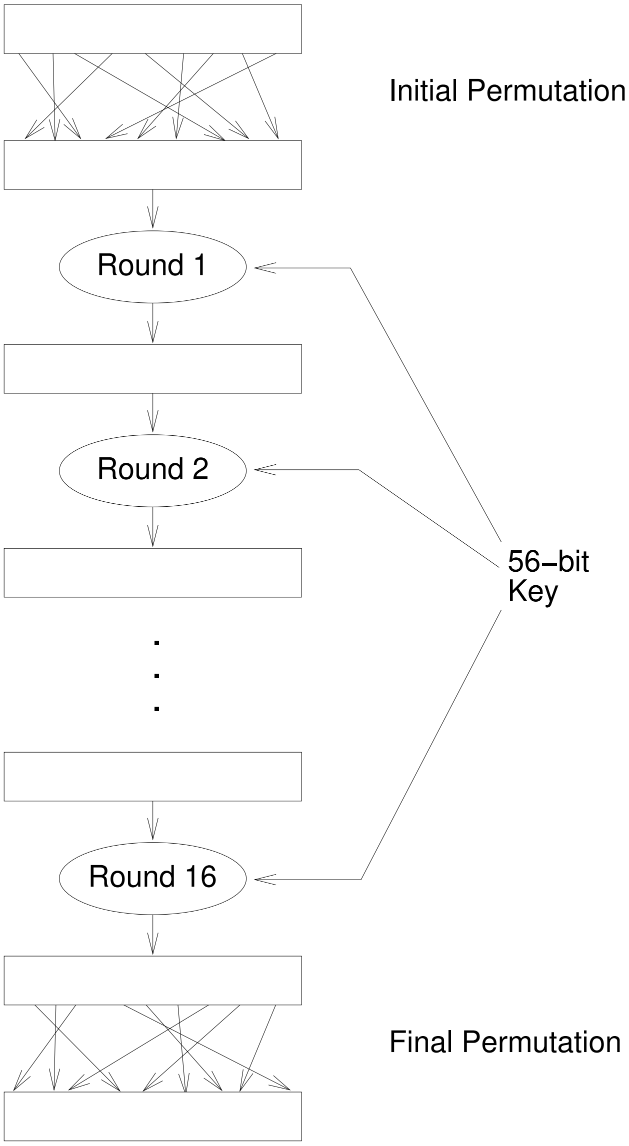
Can be combined with public-key encryption as an efficient method for ensuring message integrity:

- e.g., *keyed MD5*
- sender draws random number k
- sender transmits $m + MD5(m + k) + RSA(k, senderPrivateKey)$
- m: message
- $MD5(m + k)$: MD5 crypto checksum of message concatenated with random key
- $RSA(k, senderPrivateKey)$: RSA encrypted key, using sender's private key

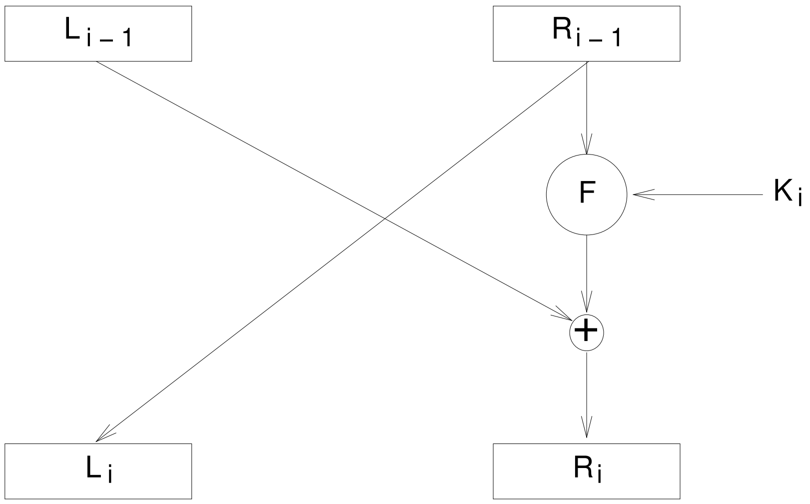
Data Encryption Standard (DES)

A secret key encryption algorithm

- works on 64 bit blocks
- key size 56 bits



Data Encryption Standard (DES)

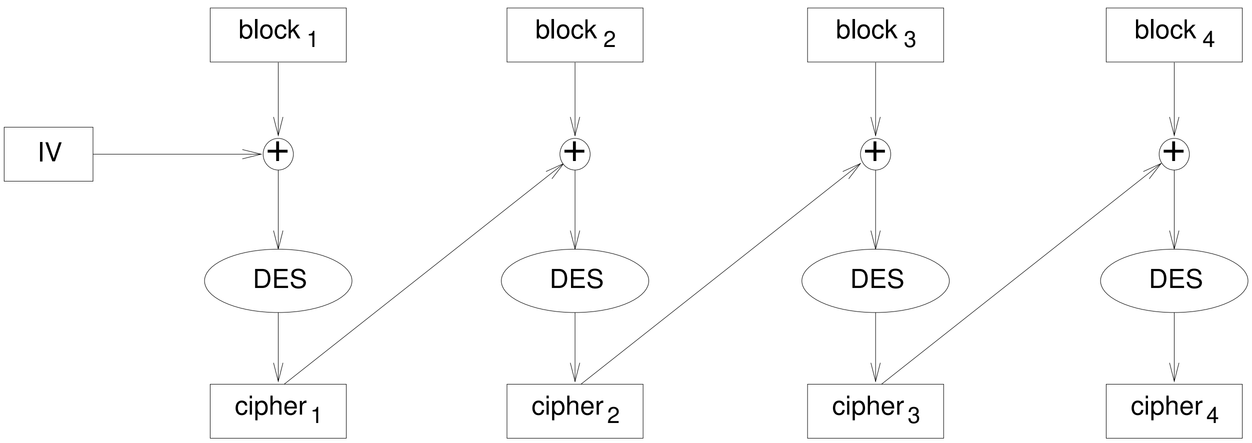


- K_i at round i derived from key through shifts and permutations according to a table
- combiner function F:
  - expand R_{i-1} to 48 bits (see below)
  - XOR with K_i
  - reduce to 32 bits by compressing contiguous 6-bit chunks into 4 bits using an *S-box*.
  - S-box is a chunk-specific translation table

Decryption function is identical, except that K_is are applied in reverse order.

Cyclic Block Chaining

allows encryption of messages larger than 64 bits



IV = random number that is encrypted and transmitted with the message

RSA Public Key Encryption

Key length is 512 bits

Choosing a pair of keys:

- pick two prime numbers p, q (about 256 bits long)
- $n = p * q$
- choose e, such that e and $(p-1)*(q-1)$ have no common factors greater than 1
- $d = \frac{mod((p-1)*(q-1))}{e}$
- key 1: $< e, n >$
- key 2: $< d, n >$
- discard p, q (cannot be compromised)

Encryption: $E = M^e(mod(n))$

Decryption: $M = E^d(mod(n))$

M must be no more than 512 bits long

longer messages encrypted by concatenation

Performance

DEC 3000/600 (175Mhz Alpha)

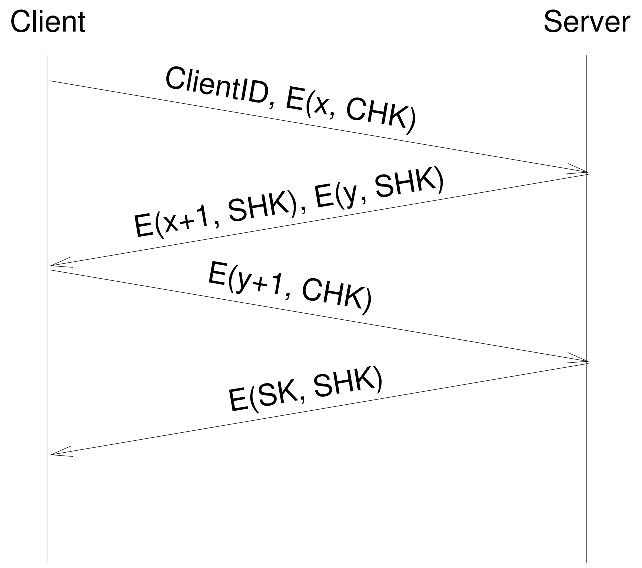
- DES: 36 Mbps
- RSA: 1 Kbps
- MD5: 80 Mbps

Custom VLSI:

- DES: 1 Gbps
- RSA: 64 Kbps

Authentication

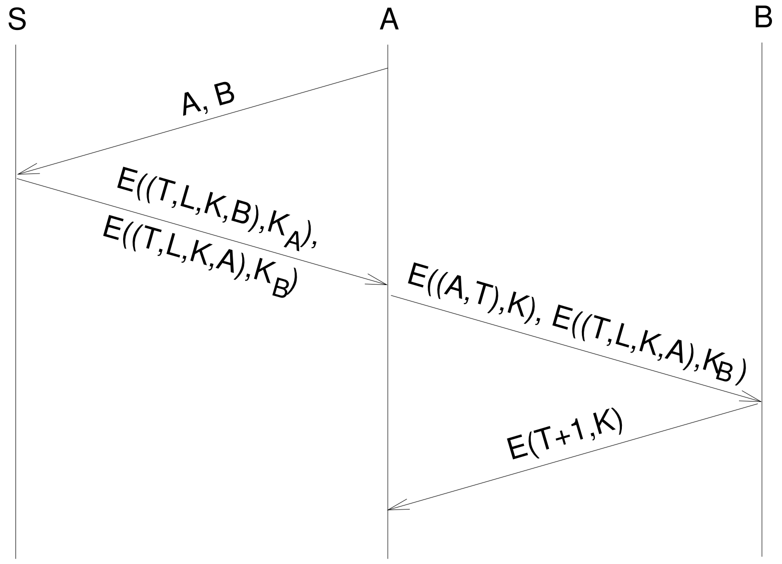
Three-way handshake



- $E(m, k)$ = encryption of message m with key k
- x, y: random numbers
- SHK, CHK: Client/Server handshake key (shared secret)
- SK: session key

Authentication

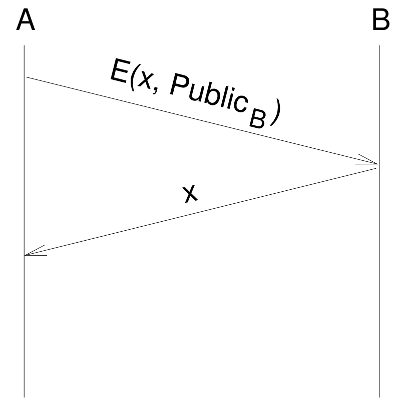
Using a trusted authentication server



- T: timestamp
- L: lifetime (of session key)
- K_A, K_B: A's/B's secret key (shared with server)
- K: session key

Authentication

Public Key Authentication



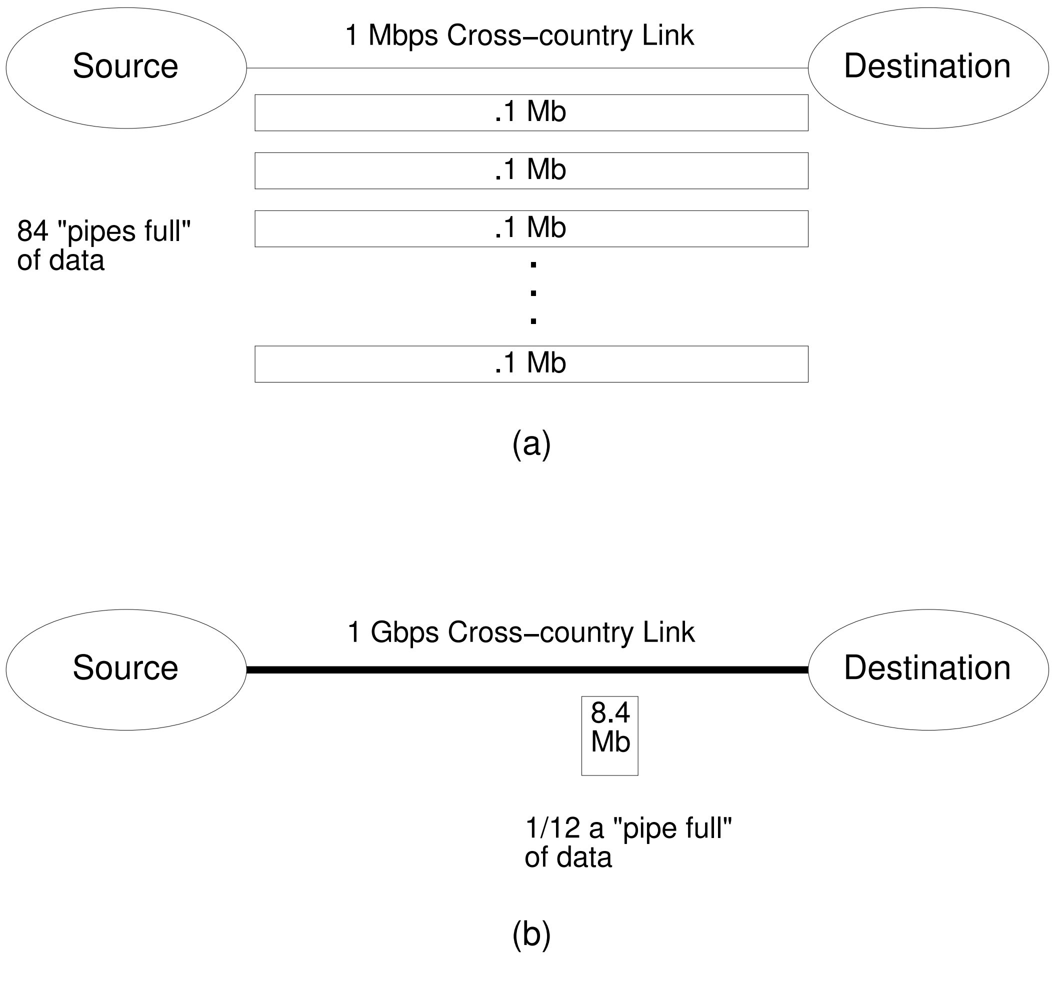
- x: random number
- $Public_B$: B's public key

High-Speed Networking

What changes?

- bandwidth greatly increased
- latency given by speed of light
- $\Rightarrow$ large bandwidth/delay product
- network bandwidth approaches that of main memory bandwidth
- new applications require service modes other than best-effort

Bandwidth-Delay Product

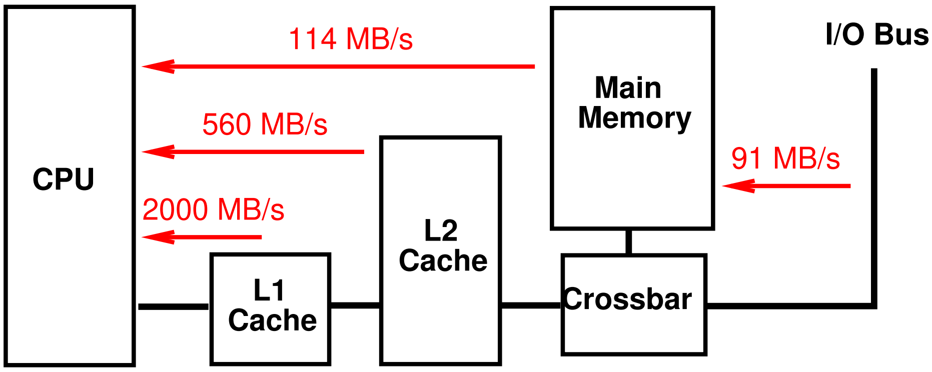


Bandwidth-Delay Product

Latency dominates delivery time in high-speed, wide-area networks

- retransmission is costly
- implications for congestion control
- implications for error control
- trading bandwidth for latency

Host Performance Issues



- network bandwidth same order of magnitude as memory bandwidth
- to keep up with network, can't read/write main memory more than once
- caches only help if accesses have good locality
  - avoid copying
  - scheduling for good locality
  - integrated layer processing

Avoiding Data Copying

- buffer management and message manipulations
- cross-domain data transfer
- interface design

Integrated Services Model