Chapter 1

Communications links: fiber, copper, radio and satellite

Bandwidth: transmission rate. Bits transmitted by second.

Packet switches: forward packets (routers and switches)

Internet: Network of networks

Protocols: A method of communications between devices on a network. Defines format, order of messages sent and received among network entities, and actions taken on message transmission, receipt.

Internet standard: RFC, IETF

Network edge: Clients and server.

Access networks, physical media: wired, wireless communication links.

Network core: interconnected routers.

Digital subscriber line (DSL): use existing telephone dedicated line to central office DSLAM. (No frequency conflict because phone is low Hz and internet high Hz)
Copper wire (Slow compared to hybrid fiber coax)

Cable network: use frequency division multiplexing on a shared cable allowing different channels transmitted in different frequency bands.
HFC: Hybrid fiber coax (Faster than Copper wire used by telephone)

Ethernet is faster than **wireless** because it has less noise.

Switch: Forward packets within a LAN network.

LAN within building, **WAN** within city or more.

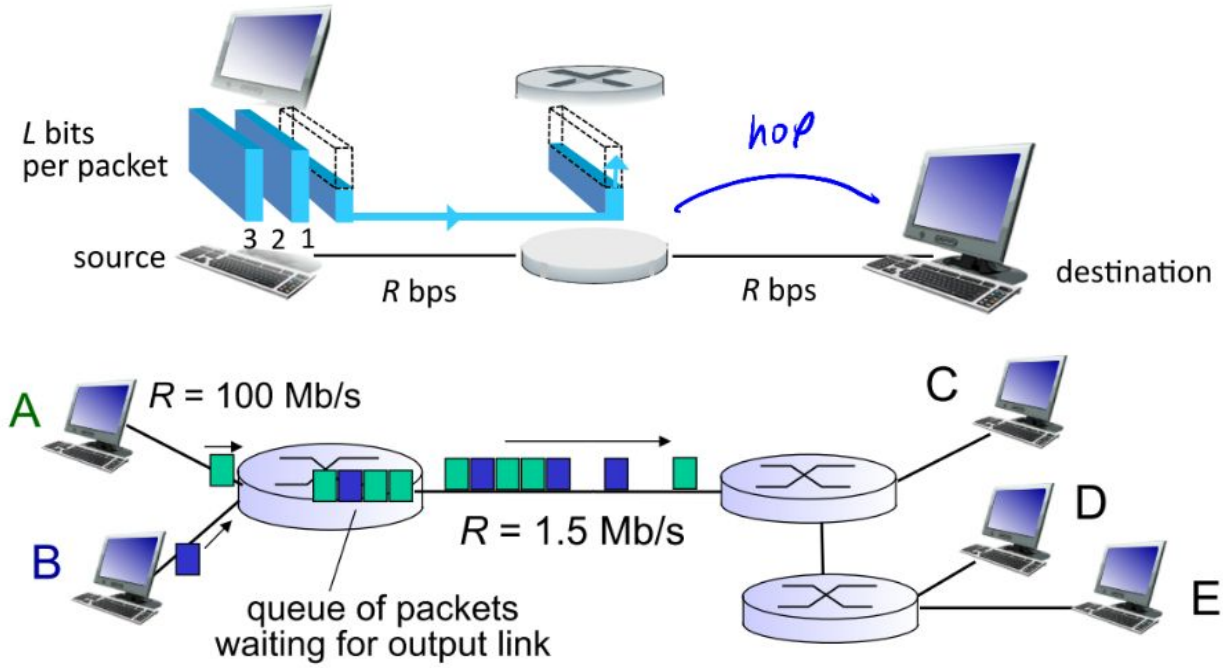
Packet transmission delay = **time needed to transmit L-bits packet into link** = L/R
L(bits)= length of one packet
R(bits/sec)= bandwidth

Guided media: copper, fiber etc..

Unguided media: Radio etc…

Packet switching and circuit switching

Packet switching: Forward packet from one router to another. Using the **store and forward** mechanism where the entire packet must arrive at router before it can be transmitted on next link.



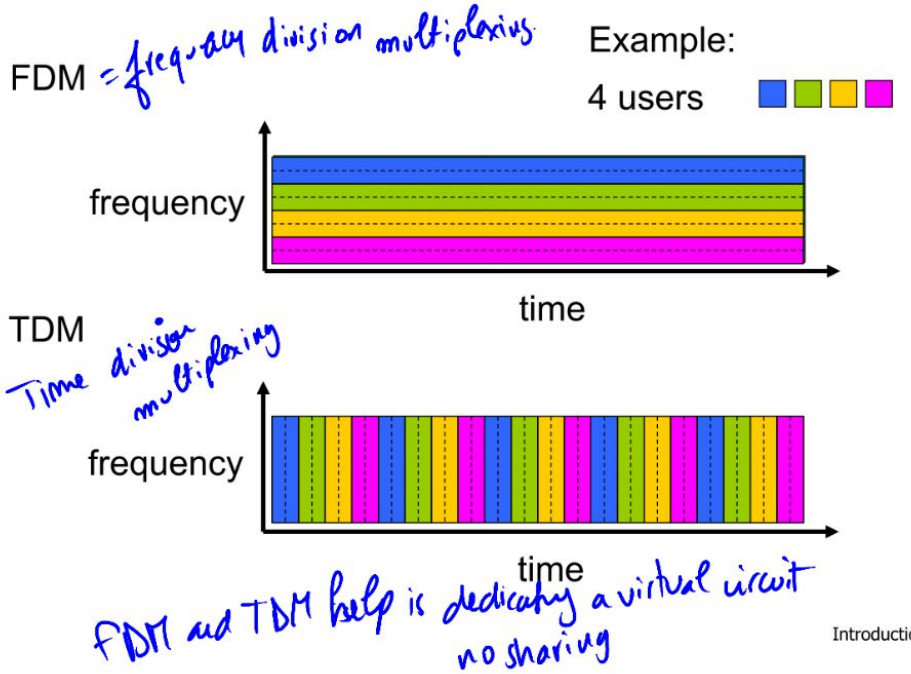
If **arrival rate** to link exceeds **transmission rate** of link for a period of time, packet will be queued and possibility of losing some of them if memory buffer fills up. Packet lost may be retransmitted by previous node,source or not at all.

Routing: determines source-destination route taken by packets. Routing algorithm is based on a dynamic table mapping system.

Forwarding: move packets from router's input to appropriate router output based on the routing algorithm generated table.

Circuit switching: reserved path from source and destination. The resources are dedicated (no sharing), and the circuit segment is idle but still reserved. This switching is used in telephone network. Guaranteed performance.

Circuit switching: FDM versus TDM



P.S C.S

Packet switching versus circuit switching

packet switching allows more users to use network!

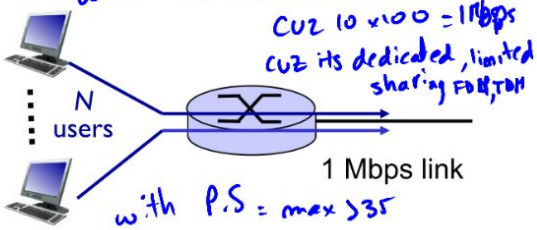
90% used

example:

- 1 Mb/s link
- each user:
  - 100 kb/s when "active"
  - active 10% of time

10 × 100 kbps = 1 Mbps

- *circuit-switching:*
  - 10 users (limited sharing previous slide)
- *packet switching:*
  - with 35 users, probability > 10 active at same time is less than .0004 *



Q: how did we get value 0.0004?

Q: what happens if > 35 users ?

$p(10+ \text{ active}) = p(11 \text{ active}) + p(12 \text{ active}) + \dots + p(35 \text{ active}) = 0.0004$

$p(11 \text{ active}) = \binom{35}{11} \times 0.1 \times 0.9^{35-11}$

$p(R \text{ active}) = \binom{35}{R} \times 0.1^{R} \times 0.9^{35-R}$

* Check out the online interactive exercises for more examples

if we have 0.0005, we can find N as unknown

Advantages of Packet switching:

- Great for short data
- Resource sharing
- No call setup

Disadvantages of packet switching:

- Excessive congestion can result in packet loss or delay
- Needs a protocol for reliable data transfer and congestion control

Tier-1 commercial ISP (AT&T, Sprint): National and international coverage.

Content provider network (Google): private network that connects its data centers to the internet, often bypassing tier-1 and regional ISP.

Delay

d-nodal = d-proc + d-queue + d-trans + d-prop

d-proc:

- **Check bit errors**
- **Run routing algorithm**
- **Takes less than 1ms**

d-queue:

- **TIme waiting in the queue**

d-trans:

- **how long it takes to get *all* the bits *into* the wire in the first place**
- **d-trans = L/R**

d-prop:

- **how long it takes *one* bit to *travel* from one end of the "wire" to the other**
- **d: length of physical link (m)**
- **s: propagation speed (m/s)**
- **d-prop = d/s**

Delay problem

Given
Propagation speed: 100 km/h
1st and 2nd link length: 100 km
1st and 2nd toll both = L/R = 12 s

Question
How long until ten car is lined up before 2nd toll booth ?

Answer
d-prop = 100 km / 100 km/s = 1h = 3600 s
d-trans = 12 s
d-prop + d-trans = 3600 +12 * 10 = 3720 s = 62 min

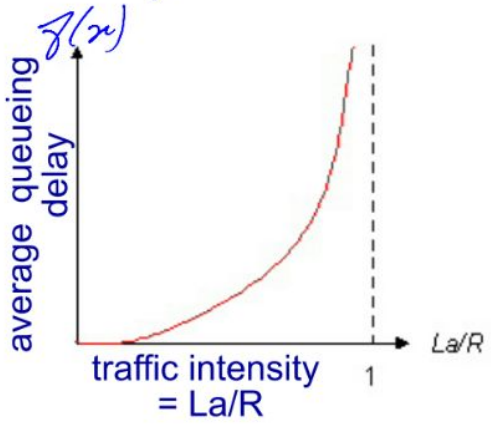
Question
Propagation speed: 1000 km/h
Toll booth takes = L/R = 60 s
Will cars arrive to 2nd booth before all cars serviced at first booth ?

Answer
d-prop = 100 km / 10000 km/h = 0.1 h = 6 min
d-trans = 60 s = 1 min
First car takes 1 min + 6 min = 7 min to arrive
Tenth car takes 1 min * 9 = 9 min to arrive to the first toll booth
Therefore, yes 1st car arrives at second booth, three cars still at 1st booth.

Queueing delay (revisited)

- *R:* link bandwidth (bps)
- *L:* packet length (bits)
- a: average packet arrival rate (packet / second)

∴ La : average arrivale bits to queue



- *La/R* ~ 0: avg. queueing delay small
- *La/R* -> 1: avg. queueing delay large
- *La/R* > 1: more "work" arriving than can be serviced, average delay infinite!

La/R ~ 0

La/R -> 1

* Check out the Java applet for an interactive animation on queuing and loss

to verify

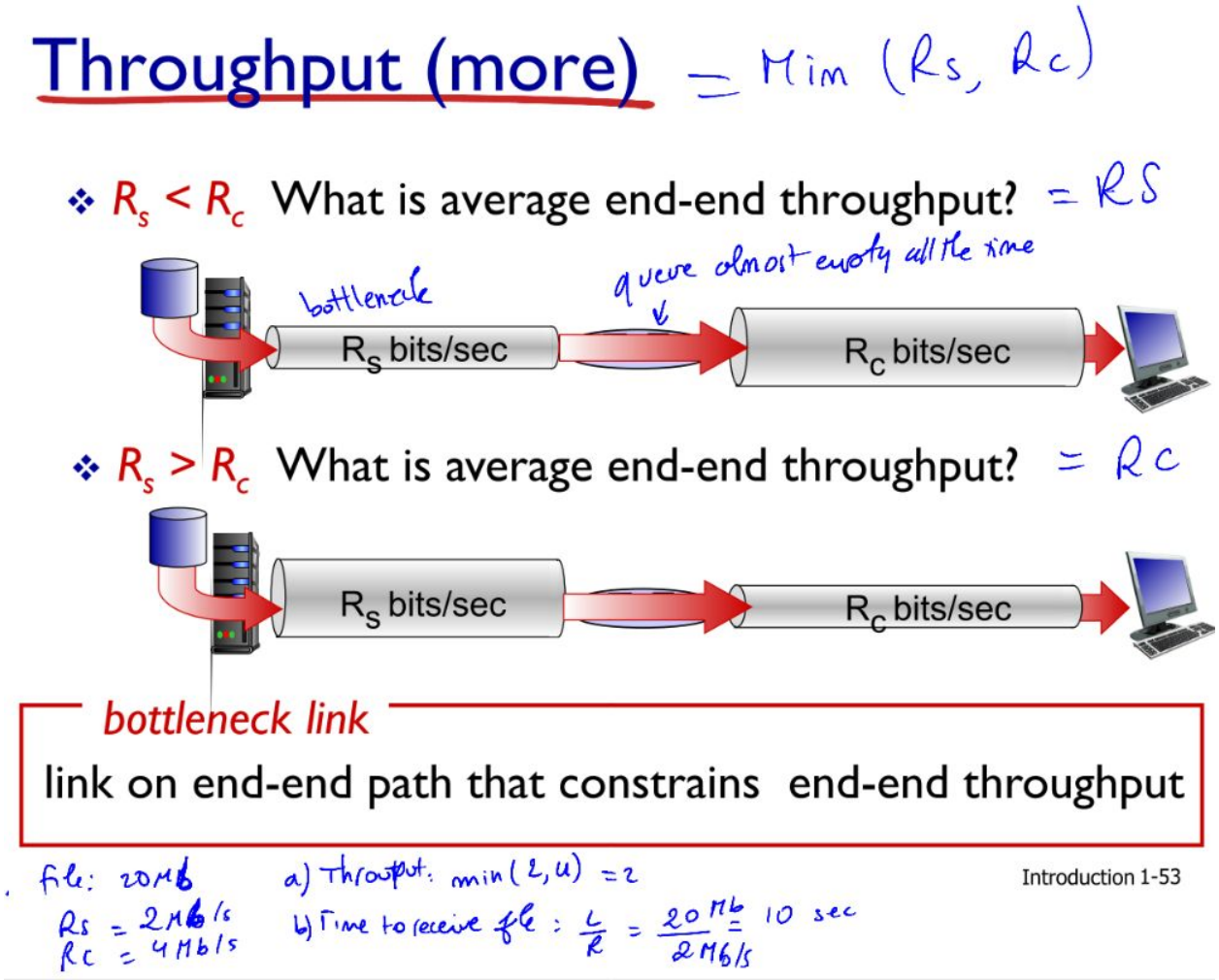
$f(La/R) = \frac{1}{N}\left(0 + \frac{L}{R} + \frac{2L}{R} + \cdots + (N-1)\frac{L}{R}\right)$

delay for first person — delay for second person

Throughput

Throughput: rate at which bits transferred between sender and receiver. (bits/unit of time).

- Example is bandwidth.
- Instantaneous: Rate at given point in time (download progress bar)
- Average: rate over longer period of time. (speedtest.net)



If we have more than 2 tubes, the average end-end throughput is the min of R1, R2, … Rn

Layering

Layering allow ease of maintenance, update etc…

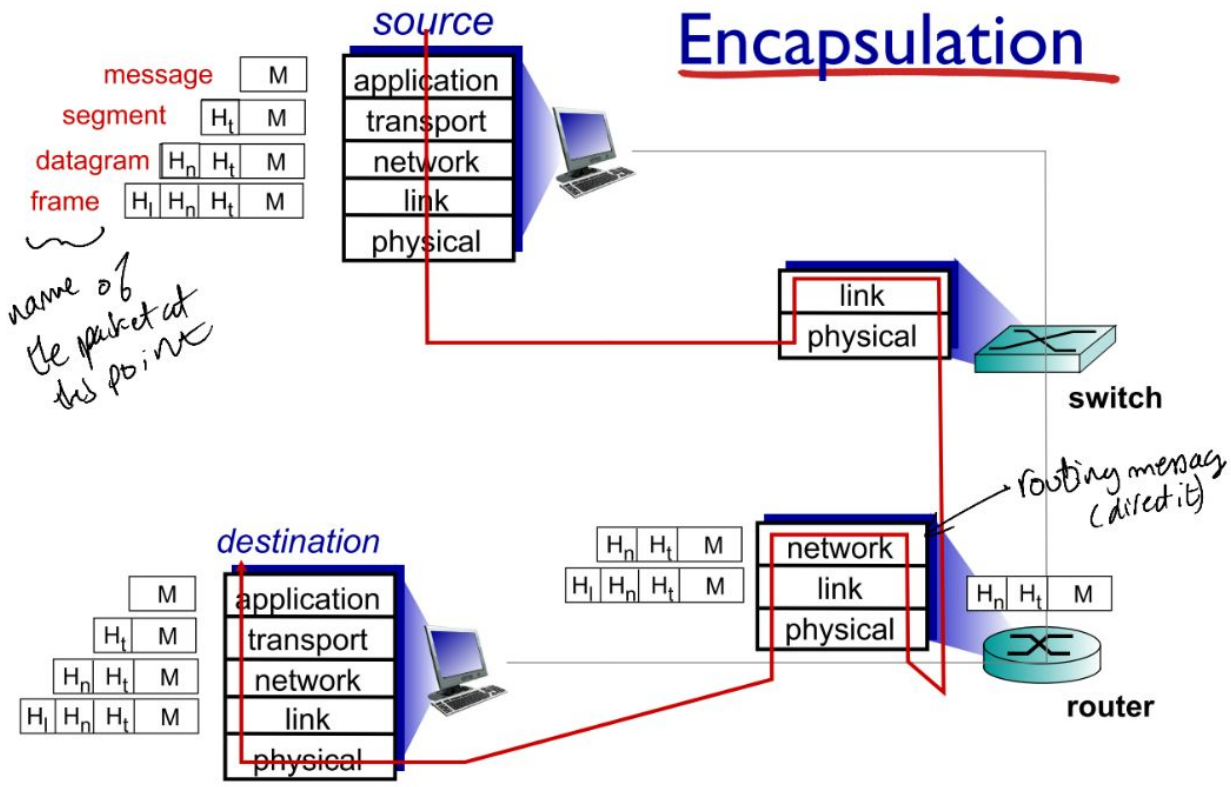
Internet protocol stack

- **Application**: network application FTP, SMTP, HTTP
- **Transport**: process data transfer TCP-UDP
- **Network**: Routing of datagrams from source to destination
- **Link**: Data transfer between neighboring network elements (wire, wifi)
- **Physical**: bits on wire

The old model (OSI) has two more layers above the transport layer:

- **Presentation**: Allow applications to interpret meaning of data (encryption, compression)
- **Session**: Synchronization recovering of data exchange.

Those two layers if needed by the developer, they will have to do them manually in the application layer.



Network security

Virus: self-replicating infection by receiving/executing objects

Worm: self-replicating infection by passively receiving objects that gets itself executed.

A worm is similar to a virus by design and is considered to be a sub-class of a virus. Worms spread from computer to computer, but unlike a virus, it has the capability to travel without any human action. A worm takes advantage of file or information transport features on your system, which is what allows it to travel unaided.

Spyware malware: record keystrokes, web history, upload info.

Denial of service (DoS): attackers make resources unavailable to legitimate traffic by overwhelming resource with bogus traffic.

Packet sniffing: network interface reads/records all packets passing by broadcast media (wifi, shared wire). Ex: Wireshark

IP spoofing: send packet with false source address

Chapter 2

Network program: run on different end systems over a network without caring about the network core. (web server communicate with browsers)

Application architecture:
- Client server
- Peer to peer (p2p)

Client server

Example: Web service

Client:
- Communicate with server
- May have Dynamic IP
- Do not communicate with other client directly

Server:
- Always on host
- Static IP
- Data center for scaling

Process: program executing in the host

Interprocess communication: Two or more process communicate on the same host.

Message communication: Two or more process communicate on different hosts.

Client process: Start the communication.

Server process: Listen for communication from client.

P2P

Example: Torrent

Peer to peer:
- Not always on server
- End system communicate and serve with each other
- Self scalability, the more end systems available the more services can be served
- May have dynamic IP

Sockets

Process send/receive messages to/from its socket
Sending and receiving socket relies on transport infrastructure on other side.

Identifier:
- IP + port number
- IP address is not sufficient for identifying the process because multiple process might be running on the same machine.
- Common port numbers: HTTP server: 80, Mail server: 25

App-Layer Protocols

A **protocol** is a set of rules that governs the communications between computers on a **network**. In order for two computers to talk to each other, they must be speaking the same language.

Defines:
- **Types of messages exchanged**: Request, response.
- **Message syntax**: Structure of the message, fields and delimiter.
- **Message Semantics**: meaning of information
- **Rules:** when and how processes send and respond to messages
- **Open Protocols**: e.g. HTTP, SMTP.
- **Proprietary protocols**:. Communication protocol owned by a single organization. e.g. Skype

Transport Services

- **Data Integrity:** does transport have to be 100% reliable or can it tolerate some loss.
- **Timing:** Does it need low delay to be effective.
- **Throughput:** Elastic apps make use of whatever throughput they receive.
- **Security:** How secure does the data need to be.

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	
interactive games	loss-tolerant	few kbps up	yes, few secs
text messaging	no loss	elastic	yes, 100's msec yes and no

TCP

- Reliable Data Transport
- Flow Control: Sender won't overwhelm receiver
- Congestion Control: Throttle sender when network overloaded
- Connection Oriented: Connection setup required between client and server processes.
- Doesn't Provide: Timing, Minimum Throughput guarantee, Security

UDP

- Unreliable Data Transfer
- Less Latency than TCP
- Cheaper than TCP
- Doesn't Provide: Reliability. Flow or Congestion control. Timing, Throughput guarantee, Security or Connection setup.

SSL is at the app layer. It provides TCP with an encrypted connection, data integrity, endpoint authentication.

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Web and HTTP

HTTP - Web application layer Protocol:

- Client(Browser) - Server Structure
- Uses TCP
- Stateless

Round Trip Time (RTT): time for a small packet to travel from client to server and back.

HTTP Connections:

Non-Persistent Connection

- One object sent over TCP connection then connection is closed. Multiple objects require multiple connections
- 2*RTT + File Transmission time For each object
- OS overhead for each TCP connection

Persistent Connection

- Multiple objects can be sent over one TCP connection.
- As little as RTT for all referenced objects.
- Open TCP connection

Types of HTTP Messages:

1. **Request - ASCII**
2. **Response - (ASCII?)**
    - ➔ **Status Codes:**
        - **200 OK**
        - **301 Moved Permanently**
        - **304 Not Modified**
        - **400 Bad Request**
        - **404 Not Found**
        - **505 HTTP Version Not Supported**

//structure of request and response messages

HTTP methods:

1. **Post Method:** Content is sent to server via entity body.
2. **Get Method:** Content is uploaded into URL field of request Line.
3. **Head:** Asks server to leave requested object out of response
4. **Put:** Uploads file in entity body to path specified in URL field
5. **Delete:** Deletes file specified in the URL field.

Cookies

A tool designed to maintain state on websites. When initial HTTP requests arrives at site, site will create: A unique ID and an entry in the backend database tied to that ID.
Cookies can be used for authorization, shopping carts, recommendations, user session state.

Components:

1. Cookie header line of HTTP response message; Creates a unique ID and entry.
2. Cookie header line in next HTTP request Message; Tells browser to set cookie ID.
3. Cookie file kept on host, managed by user's browser.
4. Back-end database at Website.

Web caches / Proxy Servers

- Satisfy client requests without involving origin server.
- Browser sends all HTTP request to cache
  If object in cache -> cache will return object
  else cache requests object from origin server, then returns object to client.
- Cache acts as both client and server. Client to origin server, Server to original client.
- Installed by ISP

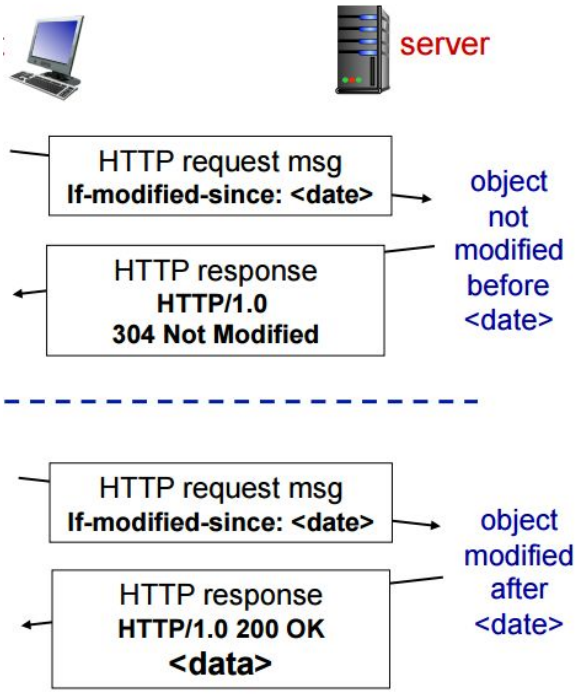
Advantages of Proxy Servers

1. Reduce response time for client request
2. Reduce traffic on an institution link
3. Saves bandwidth
4. Enables poor providers to deliver content effectively.

Conditional Get

Goal - Don't send objects to cache if cache has up-to-date cached version. This results in no object transmission delay, lower link utilization.

Client requests an object.
In the request message it will specify the date of cached copy with the If-modified-since: <date> statement
The server will respond with no object if the cached copy is up to date with a 304 Not Modified message else
if the cached copy is out of date it will respond with a 200 message and the object(s).



File Transfer Protocol

Client - Server Model. RFC 959, Port 21. Uses TCP. Maintains State - such as authentication and directory.
Out of Band control - control connection.

1. FTP client contacts FTP server at port 21, using TCP.
2. Client authorized over control connection
3. Client browses remote directory, sends commands.
4. Server receives command, the server will open a 2nd TCP connection using port 20 to transfer the file to client
5. After transferring one file, server closes data connection.

6. Repeat 4-5 for each file.

Commands	Return codes
● Sent as ASCII text over control channel	● Status code and phrase
● USER username	● 331 Username OK, password required
● PASS password	● 125 data connection already open; transfer starting
● LIST return list of file in current directory	● 425 Can't open data connection
● RETR filename - retrieves (gets) file	● 452 Error writing file
● STOR filename - stores (puts) file onto remote host	

Electronic Mail

Uses In band control - where data and control is sent over same connection

Components

- ➔ **User Agents** - Some program that allows you to view, create, and manipulate mail messages. Example: Outlook, Mail, Thunderbird
- ➔ **Mail Servers** -
  - ◆ Mailbox: contains incoming messages for user.
  - ◆ Message Queue outgoing messages.
  - ◆ SMTP used between mail servers to send and receive email.
- ➔ **SMTP -** Uses TCP, port 25
  - ◆ Phases of transfer: Handshaking, Transfer of messages, closure.
  - ◆ Commands in ASCII text
  - ◆ Response: status code and phrase
  - ◆ Messages must be in 7-bit ASCII
  - ◆ Persistent Connection

Sample Codes:

- 220 - <domain> Service Ready
- 221 - <domain> Service Closing Transmission channel
- 250 - Requested mail action okay, completed
- 354 - start mail input: end with <CRLF>.<CRLF>

Emai Message format: Header(contains To, From, Subject, etc.), Blank line, Message.

Mail Access Protocol - Retrieval from server

- **POP:** Post Office Protocol, RFC 1939. Authorization, download. Download and delete, once file is downloaded it is deleted from server. Messages kept on client(?). Stateless
- **IMAP:** Internet Mail Access Protocol, RFC 1730: more features such as manipulation of stored messages on server. All messages kept on server. Allows use of folders. Maintains state.
- **HTTP:** Gmail, Hotmail, Yahoo, etc.

DNS: Domain Name System

Uses UDP, Port 53.

Services

- Distributed Database implemented in hierarchy of many name servers
- Application-Layer Protocol: name-to--address translation.
- Host aliasing
- Mail Server Aliasing
- Load Distribution

Classes of DNS Servers that belong to the hierarchy

1. Root Name Servers:
2. Top-Level Domain (TLD) Servers: Responsible for com, org, net, edu, aero, jobs, museums, and all top level country domains such as ca, uk, fr
3. Authoritative DNS Servers: Organization's own DNS server(s), providing authoritative hostname to IP mapping. Maintained by either ISP or organization.

Local DNS server:

- ➔ Does not strictly belong to hierarchy
- ➔ Maintained by ISP
- ➔ When host makes a DNS query, query is sent to its local DNS server
    - ◆ Maintains a cache of recent translations
    - ◆ Acts as proxy.

Resolution Methods:

- ➔ **Iterated Query:** Initially contacted server contacts a DNS server, if mapping is not known it will direct you to a server that might know the answer until resolution is known. Local DNS will then return resolution to client
- ➔ **Recursive queries:** Puts burden of name resolution on initially contacted server. It must query the other servers and get the resolution to the host.

DNS caching: Once mapping is learned it will be cached for some time, they may be out of date.
TLD servers typically cached in local name servers thus Root servers not typically contacted.

P2P applications

Torrent: group of peers exchanging chunks of a file.
Tracker: Tracks peers participating in torrent.
Churn: Peers may come and go

P2P File Distribution

- ➔ Files are packetized into 256KB
- ➔ Peers in torrent send/receive file packets
- ➔ Peer joining torrent: Accumulates packets over time, registers with trackers to get list of peers and connects to a subset of peers.
- ➔ While downloading, peer uploads packets to other peers
- ➔ Peer may change peer which it exchanges with
- ➔ Once peer has entire file, it may leave and remain in torrent to continue sharing file.

Socket Programming

UDP: Unreliable DataGram

- ➔ No connection between client and server
  - ◆ No handshaking
  - ◆ sender explicitly attaches IP destination address and port # to each packet
  - ◆ receiver extracts sender IP address and port # from received packet
- ➔ Data may be lost or received out of order
- ➔ UDP provides unreliable transfer of Datagrams between client and server

TCP: reliable, byte stream-oriented

- ➔ Client must contact server
  - ◆ Server process must be running
  - ◆ Server must have created socket that accepts client's contact
- ➔ Clients contact server by
  - ◆ Creating TCP socket, specifying IP address, Port, and number of server process
  - ◆ When Client creates Socket: client TCP establishes connection to server TCP
- ➔ When contacted by client, server TCP creates new socket for server process to communicate to that client
  - ◆ Allows server to talk with multiple clients
  - ◆ Port numbers used to distinguish clients
- ➔ TCP provides reliable, in order byte-stream transfer (Pipe) between client and server

Chapter 3 - Transport Layer

Transport Layer Services and Protocol:

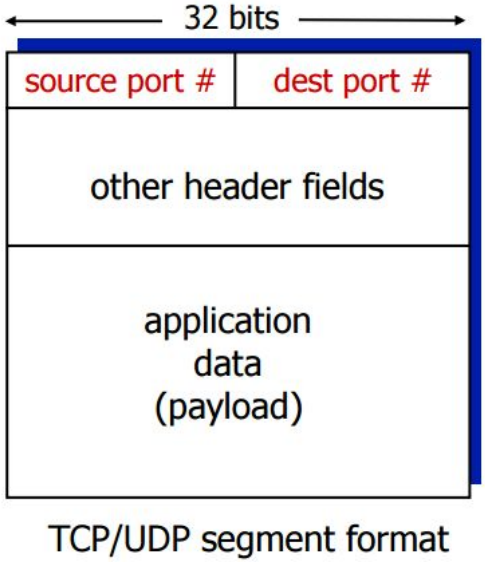
- ➔ Provides logical communication between app processes on different hosts.
- ➔ Transport protocol run in end systems
  - ◆ Sender: Segments app messages, passes to network layer.
  - ◆ Receiver: Reassembles segments into messages, passes to application layer.
- ➔ Multiple protocols available for transport. TCP and UDP.
- ➔ Relies on, enhances network layer services.

Network layer: Logical communication between hosts.

Multiplexing/Demultiplexing

Multiplexing at sender: Handle data from multiple sockets, add transport(4 tuples) header.
Demultiplexing at receiver: Use header info to deliver received segments to correct socket.

Datagram Format:
//32 bit = 16 bit host port + 16 bit receiver port.



TCP/UDP segment format

Each datagram has source and destination IP address,
source and destination port numbers,
and carries one segment.

IP and Port numbers are used to direct segment to appropriate socket.

Connectionless Demultiplexing
IP datagrams with same destination port number,
but different source IP and/or source numbers will be directed to same socket at destination.

Connection-oriented Demultiplexing

- ➔ TCP socket identified by 4 tuple: Source IP and Port, Destination IP and Port.
- ➔ Receiver uses all four values to direct segments to appropriate socket.

- ➔ Server may support many simultaneous TCP sockets. Each socket is unique by the 4 tuples.

UDP Socket vs TCP Socket

UDP has one socket per application for all different clients src IP
TCP has one socket per combination of application and client src IP

Connectionless Transport: UDP

Why use UDP:

- ➔ No connection means no connection delay
- ➔ Simple
- ➔ Small header size
- ➔ No congestion control, UDP delivers as fast as it can.

Reliable transfer over UDP:

- ➔ Add reliability at application layer.
- ➔ Application-specific error recovery.

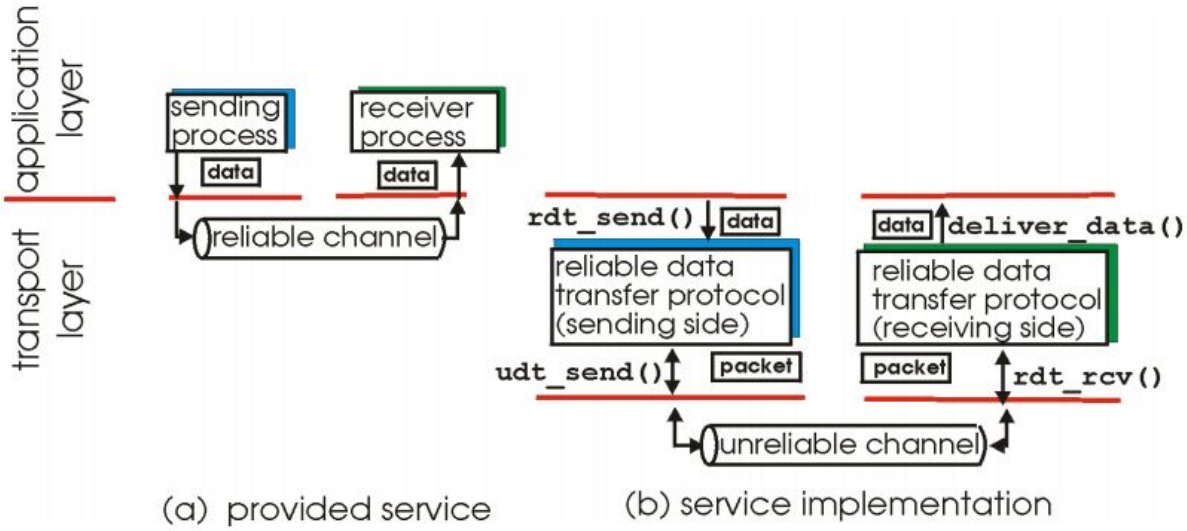
Segment header is made up of source/destination port + length + checksum + data.

Checksum:
Used to detect error(e.g. flipped bits) in transmitted segment.

- ➔ Sender
  - ◆ Treat segment contents, including header fields, as 16 bit ints
  - ◆ Checksum: addition of segment content. (One's complement)
  - ◆ Checksum is placed into UDP checksum field.
- ➔ Receiver
  - ◆ Compute checksum of received segment
  - ◆ Check if computed checksum is equal to checksum in field.
  - ◆ If YES - No error detected. Possible to have still have errors.
  - ◆ If NO - Error detected.

Principles of Reliable Data Transfer

Important in Application, Transport, and Link layers.
Characteristics of unreliable channel will determine complexity of reliable data transfer protocol.



rdt_send(): Called from above(app layer). Passed data to deliver to receiver upper layer.
udt_send(): Called by RDT, to transfer packet over unreliable channel to receiver.
rdt_rcv(); Called when packet arrives on receiver side of channel.
deliver_data(): Called by RDT to deliver data to upper layer.

Consider only unidirectional data transfer, however control will be sent both directions.
Use Finite State Machine (FSM) to specify sender, receiver.

Quick explanation of different rdt:

- **rdt1.0**: Sender sends data and receiver receives data (no error check, no loss check)

- **rdt2.0**: Sender sends data with error checksum, receiver receives data: if corrupted, send NAK, otherwise ACK

- **rdt2.1**: Same as rdt2.0 but the receiver sends error checksum with the ACK/NACK. If sender receives a corrupted ACK or NACK, it will resend the last packet but the receiver will ignore it if it was originally ACK.

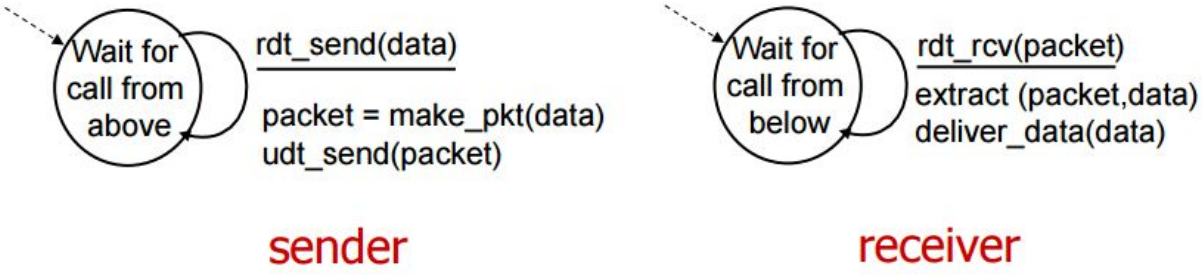
- **rdt2.2**: Same as rdt2.1, but instead of NACK, we introduce a sequence number (0 | 1), and the NACK is replaced by an ACK of the last packet.

- **rdt3.0**: Same as rdt2.2, but the Sender adds a timer for each packet, and it will resend the packet if the timer timed out. (Solve lost packet problem)

RDT1.0: reliable transfer over a reliable channel.

➔ Underlying channel perfectly reliable.
  - ◆ No bit errors or loss of packets

➔ Separate FSMs for Sender/Receiver
  - ◆ Sender sends data into underlying channel
  - ◆ Receiver reads from underlying channel.



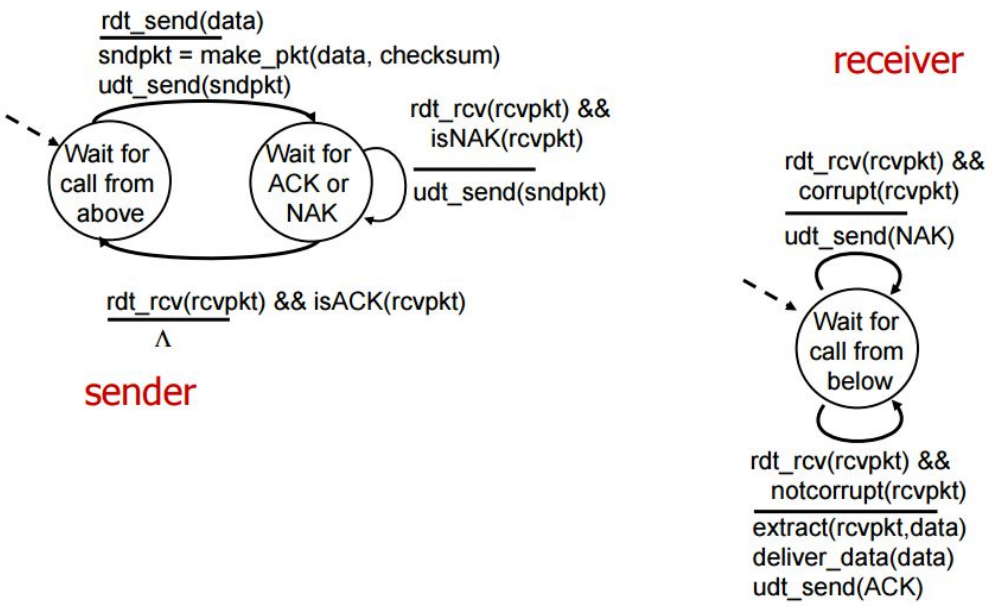
RDT2.0: Channel with bit errors

Contains improvements from RDT1.0 in the form of error detection and feedback control messages from receiver to sender.

➔ Underlying channel may flip bits in packet
  - ◆ Checksum to detect bit errors.

➔ To recover from errors
  - ◆ Acknowledgements(ACKs): Receiver explicitly tells sender that packet is received OK
  - ◆ Negative Acknowledgements(NAKs): Receiver explicitly tells sender that packet contains errors

- If receiver returns NAK then sender retransmits packet.



Stop and Wait: Sender sends one packet then waits for receiver response

RDT2.0 fatal flaw:

- ➔ If ACK/NAK corrupted
  - Sender doesn't know what happened at receiver
  - Can't just retransmit because it can cause duplicate packets being sent over
- ➔ To handle duplicates
  - Sender retransmits current packet if ACK/NAK corrupted.
  - Sender adds sequence number to packets.
  - Receiver discards duplicate packets.

RDT 2.1

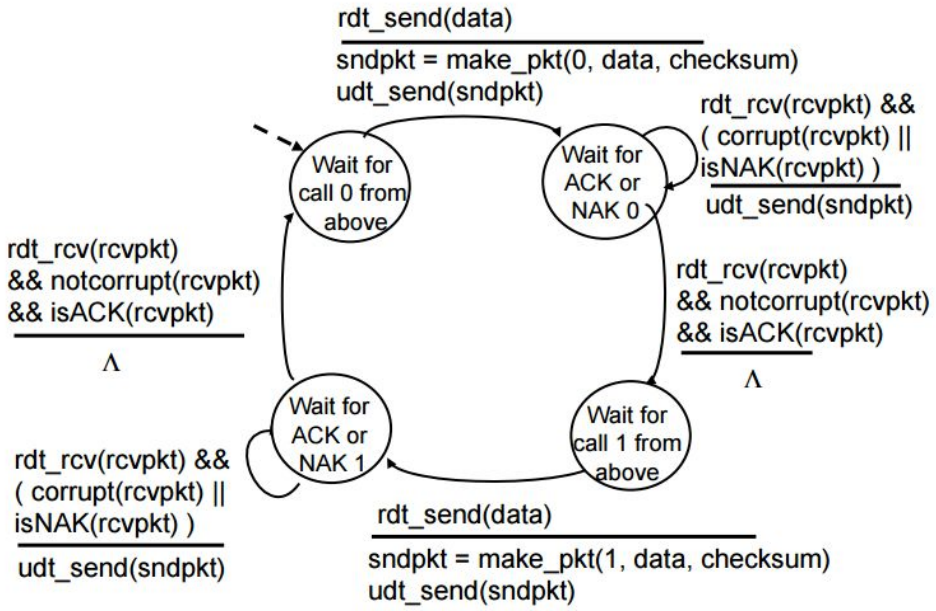
Sender

- ➔ Sequence # added to packet
- ➔ Two sequence numbers (0,1) will suffice
- ➔ Must check if received ACK/NAK corrupted
- ➔ Twice as many states: state must remember whether expected packet should have sequence number 0 or 1.

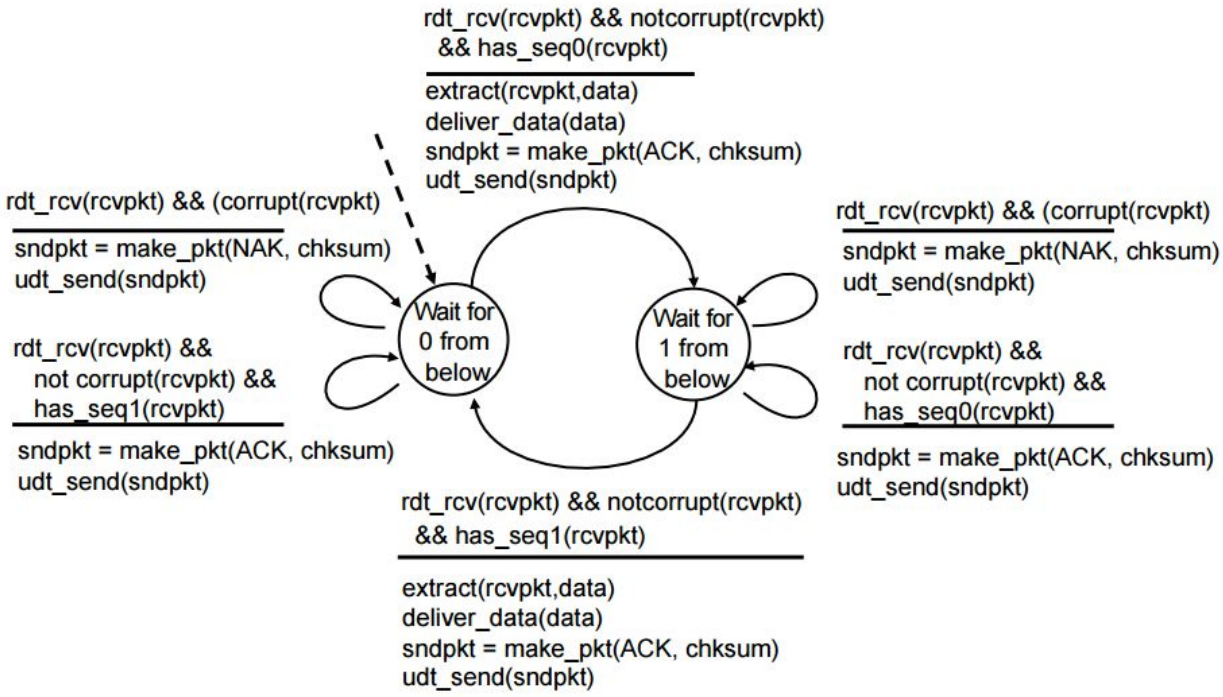
Receiver

- ➔ Must check if received packet is duplicate
- ➔ State indicates whether 0 or 1 is expected packet sequence number
- ➔ Receiver can not know if its last ACK/NAK received OK at sender

Sender

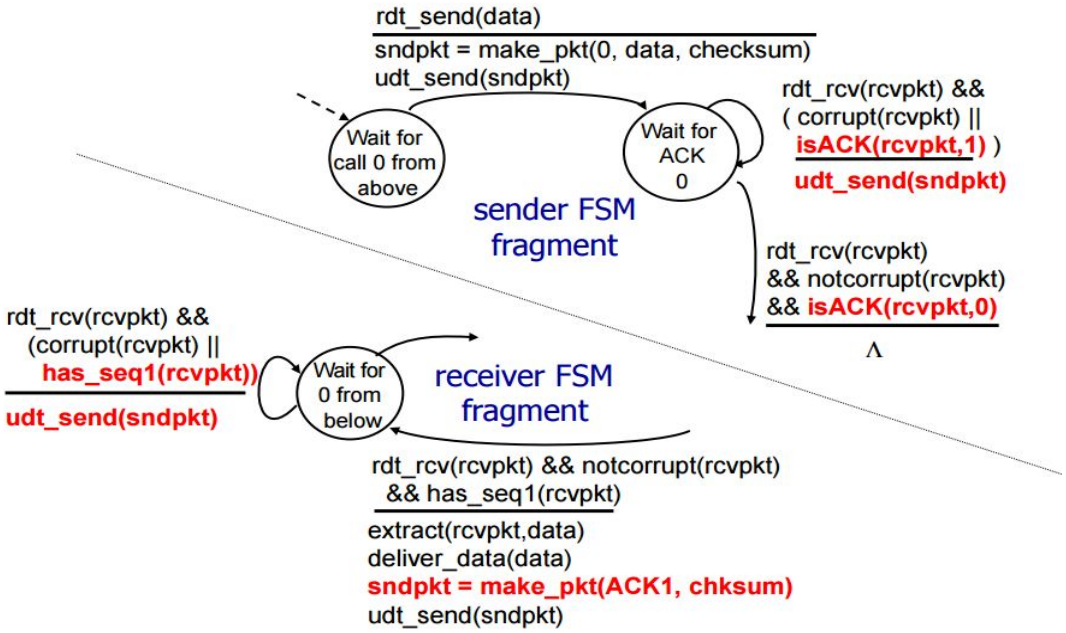


Receiver


rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq0(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)
sndpkt = make_pkt(NAK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) &&
not corrupt(rcvpkt) &&
has_seq1(rcvpkt)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)
Wait for
0 from
below
Wait for
1 from
below
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)
sndpkt = make_pkt(NAK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) &&
not corrupt(rcvpkt) &&
has_seq0(rcvpkt)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq1(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)

RDT 2.2 - A NAK free protocol

- ➔ Same functionality as RDT2.1, using only ACKs.
- ➔ Instead of NAK, receiver sends ACK for last pkt received OK
  - ◆ Receiver must explicitly include sequence # of packet being ACKed
- ➔ Duplicate ACK at sender results in same action as NAK; retransmission.

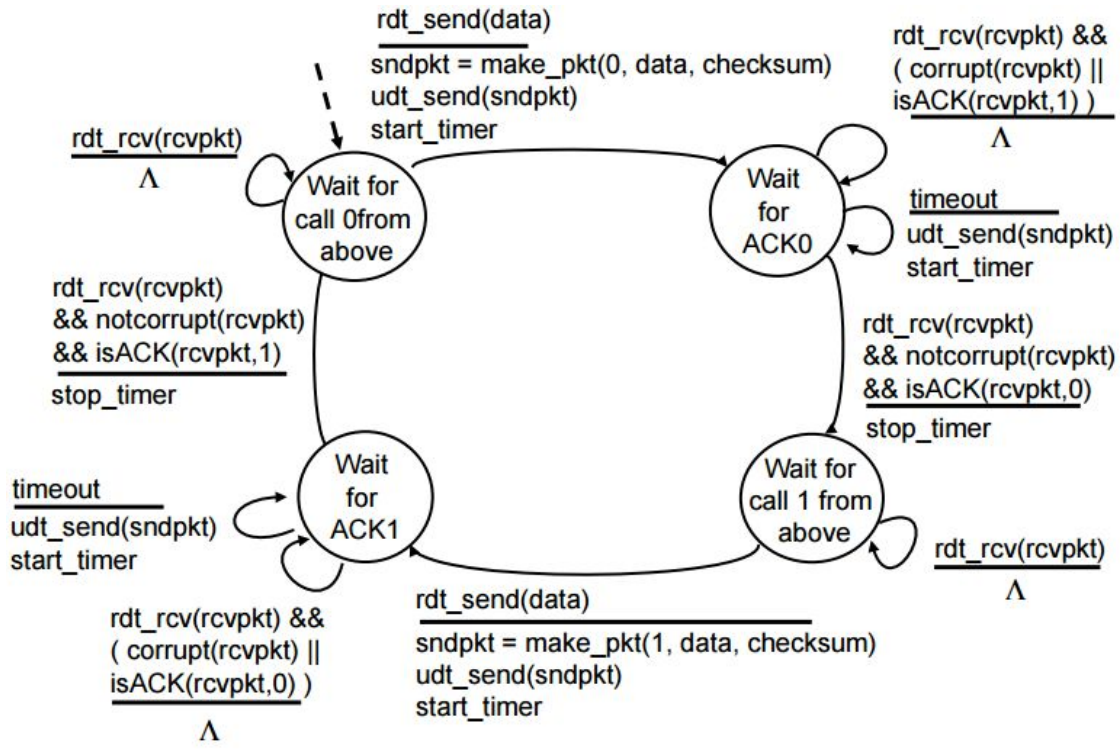


RDT 3.0 - Channels with error and loss.

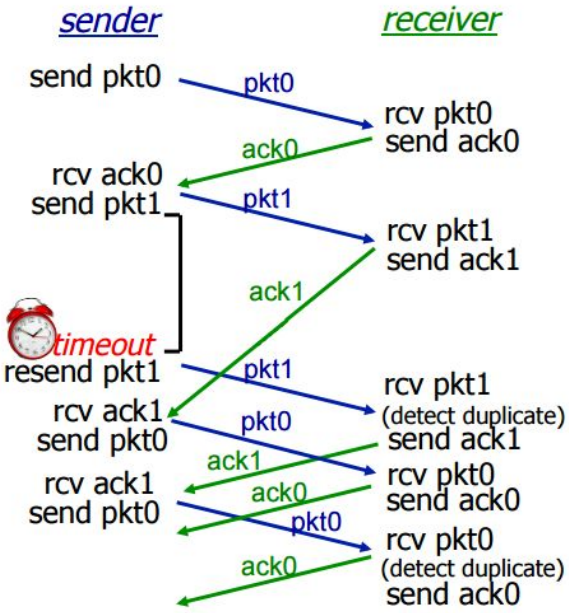
- ➔ Underlying channel can also lose packets - Data/ACKs
  - ◆ Checksum, sequence #, ACKs, retransmission are useful but not a complete solution.
- ➔ Solution: Sender waits “reasonable” amount of time for ACK
  - ◆ Retransmit if no ACK received in this time

- if packet (or ACK) just delayed not lost - retransmission will be duplicated, sequence number deals with this problem. Receiver must specify seq # of packet being ACKed.
- Requires countdown timer.

Sender



RDT 3.0 in action.



(d) premature timeout/ delayed ACK

Performance of RDT 3.0

- ➔ RDT 3.0 is correct but performance is less than desireable.
- ➔ Network Protocol limits use of physical resources.
- ➔ Example
  - ◆ Assume Transmission time = 8 microsec, RTT = 30 msec
  - ◆ Utilization = T / (RTT + T) = 0.008 / 30.008 = 0.0027
  - ◆ 1kb pkt every 30 msec: 33kB/sec thruput over 1Gbps link

Pipeline Protocols

Pipelining - sender allows multiple yet to be acknowledged packets to be sent to receiver.

- ➔ The range of sequence numbers must be increased.
- ➔ Buffering at sender and/or receiver.
- ➔ Two generic forms of pipelined protocols
  - ◆ Go-Back-N
  - ◆ Selective Repeat

Go-Back-N

- ➔ Sender can have up to N unACKed

Selective Repeat

- ➔ Sender can have up to N unACKed

packets in pipeline	packets in pipeline
➔ Receiver only sends cumulative ACK - doesn't ACK packet if there is a gap	➔ Receiver sends individual ack for each packet
➔ Sender has timer for oldest unACKed packet - when timer expires, retransmit all unacked packets	➔ Sender maintains timer for each unacked packet - when timer expires, retransmit only that unacked packet
➔ Sender only has a buffer	➔ Sender and receiver have buffer

Go-Back-N

ACK-only:

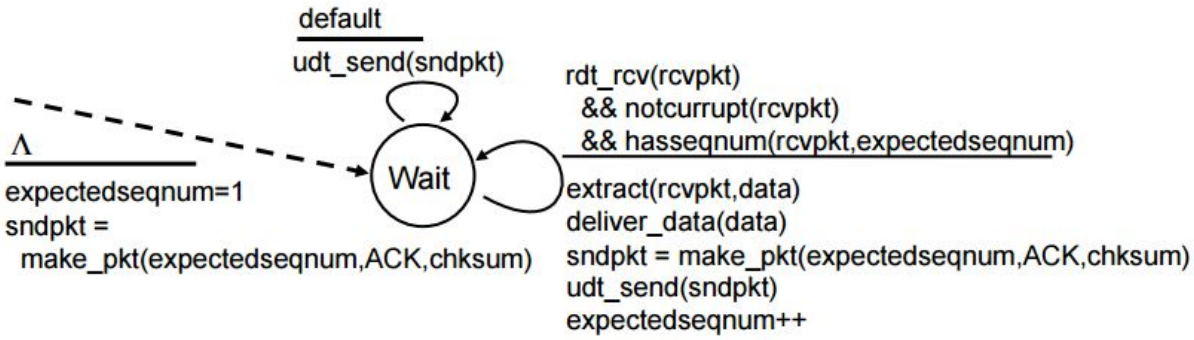
Always send ACK for correctly-received packet with highest in-order sequence #.

- This may generate duplicate ACKs.
- Only needs to remember expectedSeqNum.

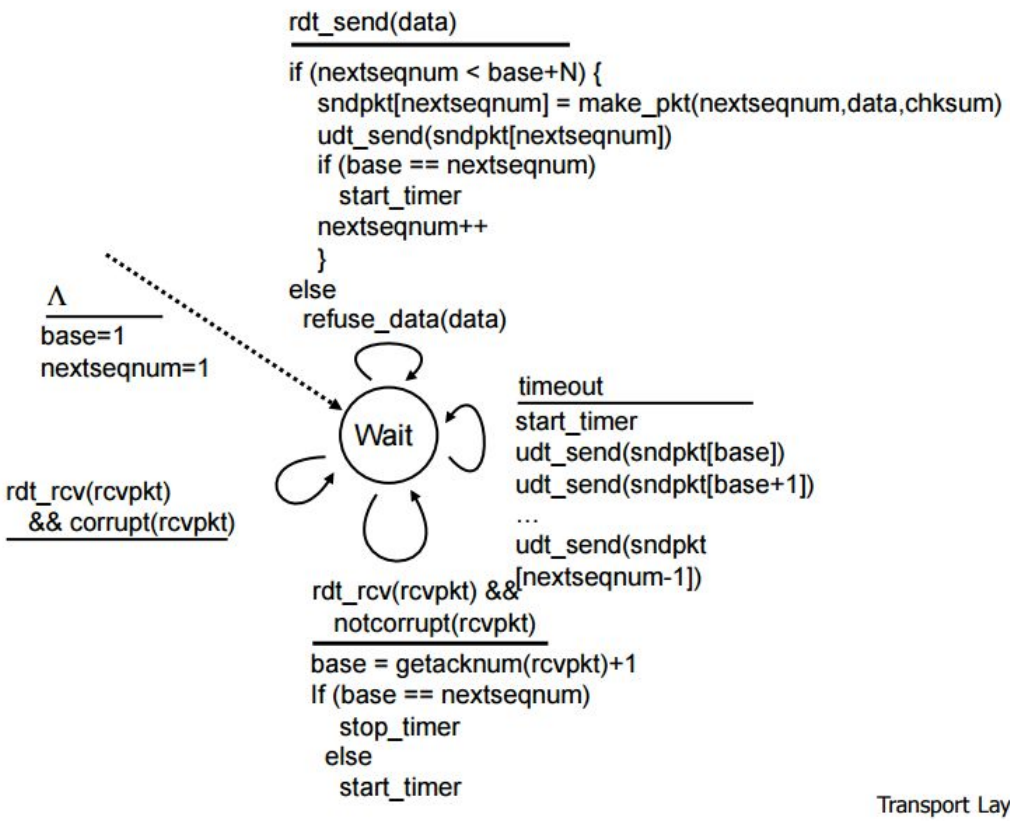
However for out of order packets:

- Discard (Don't Buffer): No receiver buffering.
- Re-ACK packet with highest in-order seq #.

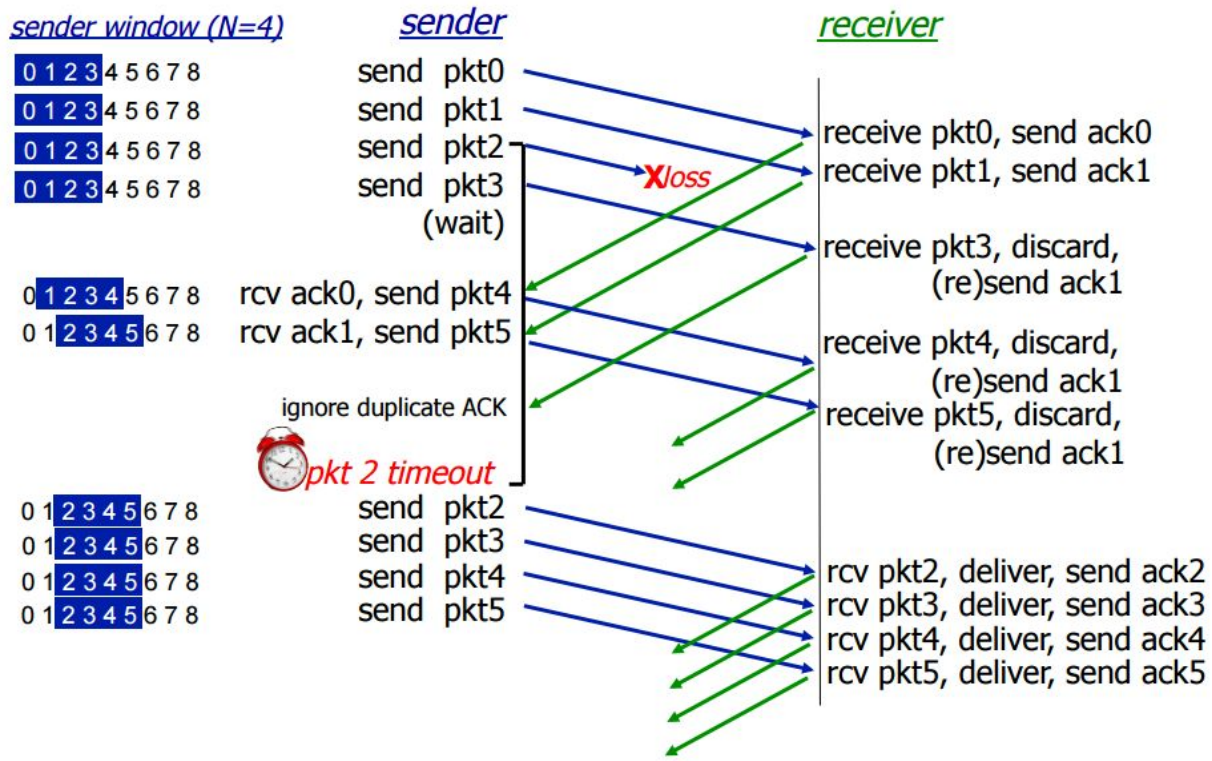
Receiver FSM



Sender FSM


rdt_send(data)
if (nextseqnum < base+N) {
sndpkt[nextseqnum] = make_pkt(nextseqnum,data,chksum)
udt_send(sndpkt[nextseqnum])
if (base == nextseqnum)
start_timer
nextseqnum++
}
else
refuse_data(data)
Λ
base=1
nextseqnum=1
Wait
timeout
start_timer
udt_send(sndpkt[base])
udt_send(sndpkt[base+1])
…
udt_send(sndpkt
[nextseqnum-1])
rdt_rcv(rcvpkt)
&& corrupt(rcvpkt)
rdt_rcv(rcvpkt) &&
notcorrupt(rcvpkt)
base = getacknum(rcvpkt)+1
If (base == nextseqnum)
stop_timer
else
start_timer

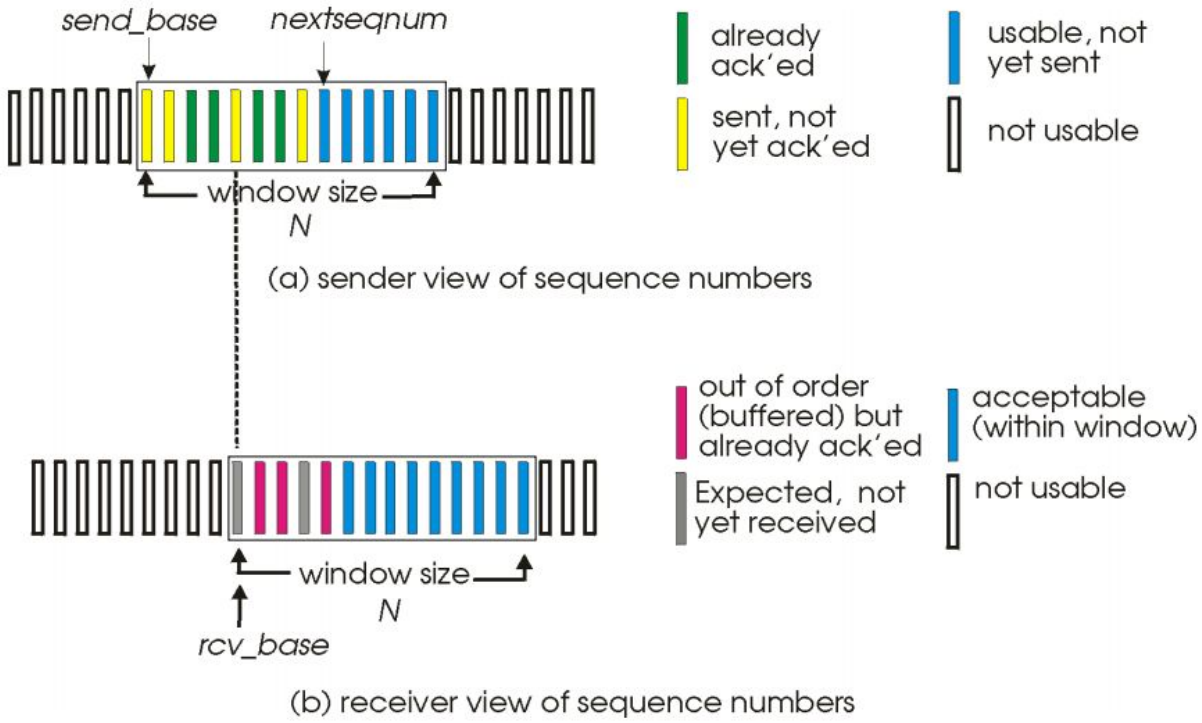
GBN IN ACTION



Selective Repeat

- Receiver ACKs individual Packets
- Sender only resends packets for which ACK was not received.
- Sender window has N consecutive sequence Number ands and has a limit on the sequence numbers sent with unACKed packets.
- Individual timer for each packet

Selective repeat: sender, receiver windows

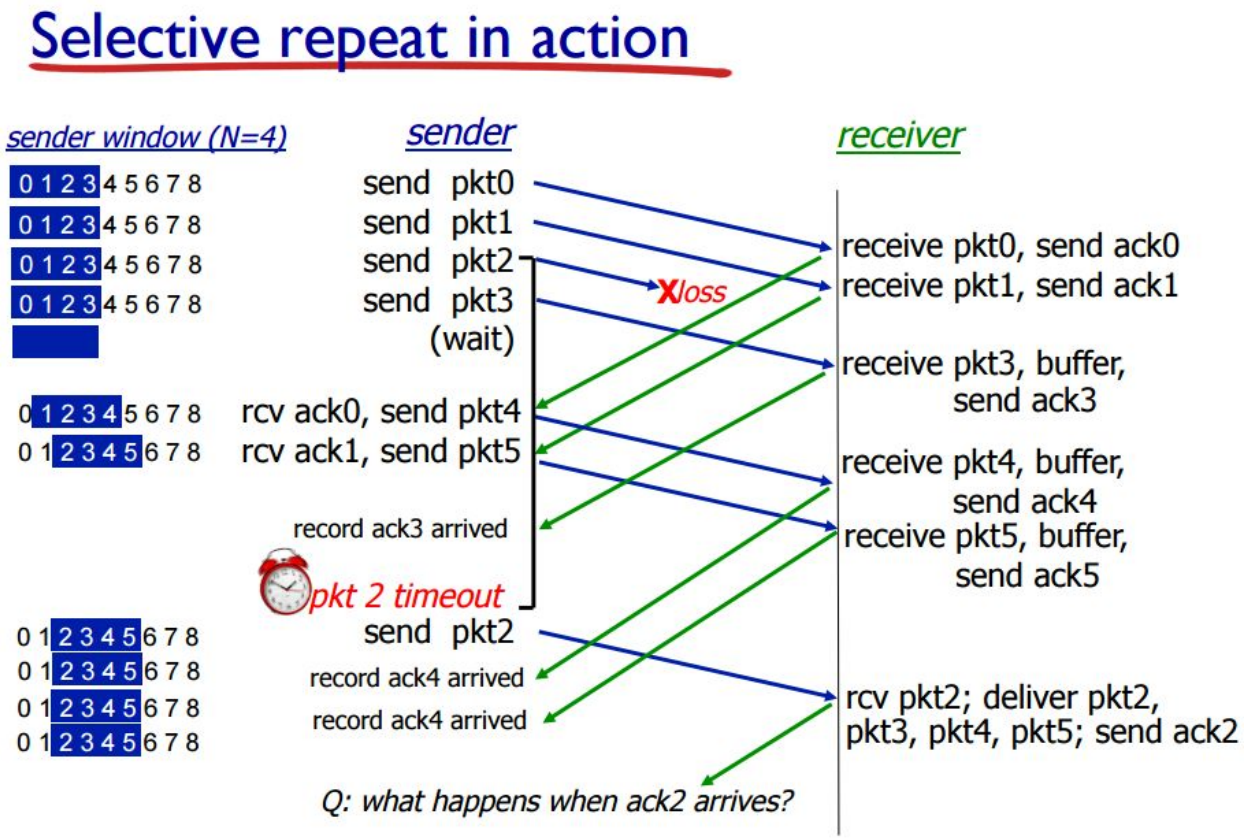


Sender

- If next available sequence number is in the window, send that packet
- If timeout occurs on packet n, resend the packet and restart the timer.
- Mark each packet as received if the that packet's ACK was received. Move window if necessary.

Receiver

- Send ACK(n) for packet n as long as it is in the window. If it isn't ignore it.
- If packet is out of order, buffer that packet.
- If packet is in order, the move the window to the next not-yet-received packet.



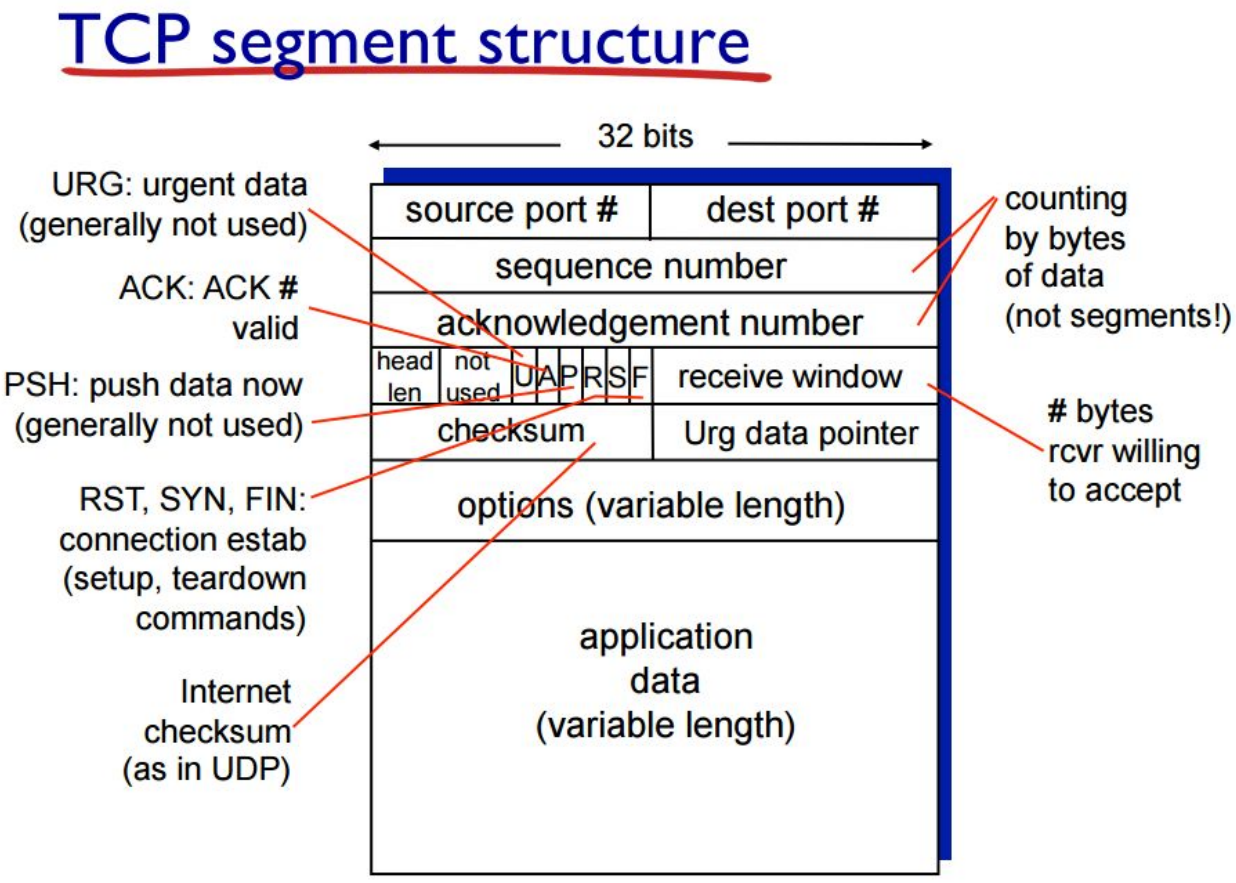
Selective Repeat dilemma

To solve the overlap of packets problems make sequence numbers ranging from 0 to 2*WINDOWSIZE (+1 maybe?)

Connection-Oriented Transport: TCP

TCP Overview

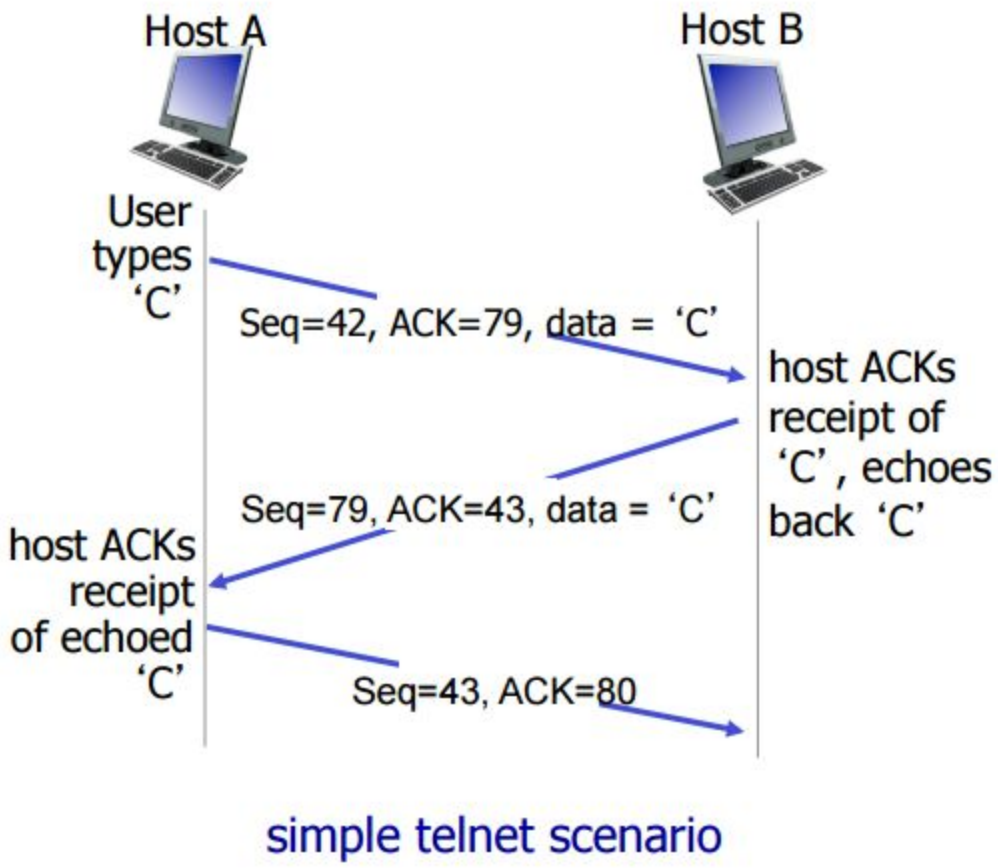
- One sender, One receiver. Point to point
- Reliable, in-order byte stream. No message boundaries
- Pipelined: TCP congestion and flow control set window size
- Full duplex data: bi-directional data flow in same connection.MSS: max segment size
- Connection Oriented: Handshake initializes sender and receiver state before data exchange.
- Flow controlled: sender will not overwhelm receiver.



TCP sequence numbers, ACKs

Sequence numbers: byte stream number of first byte in segment's data
Acknowledgements: Sequence number of next byte expected from other side. Cumulative ACKs.

How the receiver handles out of order segments is up to the implementor.



simple telnet scenario

TCP RTT and timeout

If RTT is too short: premature timeout, unnecessary retransmissions. If RTT is too long: slow reaction to segment loss.

To estimate RTT

- SampleRTT measures time from segment transmission until ACK receipt. Ignore retransmissions
- Average several SampleRTTs to get a better EstimatedRTT.

EstimatedRTT = (1-a)EstimatedRTT + a(SampleRTT)
Where a is usually 0.125

Timeout interval: estimatedRTT plus safety margin.
A large variation in EstimatedRTT leads to a larger safety margin.

Estimate SampleRTT deviation from EstimatedRTT:
DevRTT = (1-b)*DevRTT + b*|SampleRTT - EstimatedRTT|
Where b is usually 0.25

Which leads to
Timeout Interval = EstimatedRTT + 4*DevRTT. DevRTT is the safety margin.

TCP reliable data transfer

- TCP creates RDT service on top of IP's unreliable service.
    - Pipelined segments
    - **Cumulative acks**
    - Single retransmission timer which is triggered by timeout events and duplicate acks

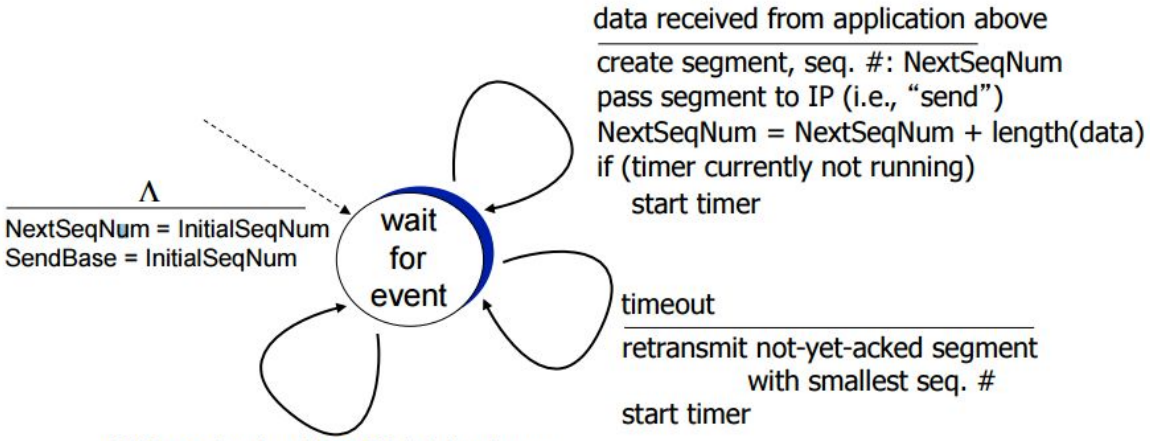
TCP Sender events:

Data Received From Application
- Create segment with sequence number.
- Sequence number is byte-stream number of first data in segment
- Start timer if it is not already running. Expiration interval: TimeOutInterval.

Timeout: retransmit segment that caused timeout and restart timer.

ACK received
- If ack acknowledges previously unacked segments
- Update what is known to be ACKed
- Start timer if there are still unacked segments.

TCP sender (simplified)



Λ

NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

data received from application above

create segment, seq. #: NextSeqNum
pass segment to IP (i.e., "send")
NextSeqNum = NextSeqNum + length(data)
if (timer currently not running)
    start timer

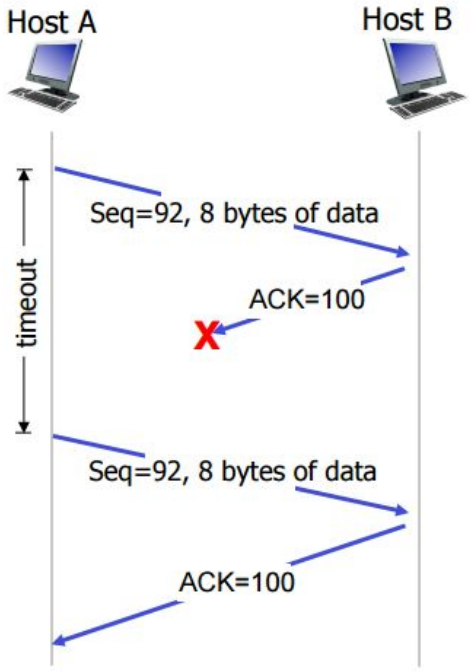
timeout

retransmit not-yet-acked segment
        with smallest seq. #
start timer

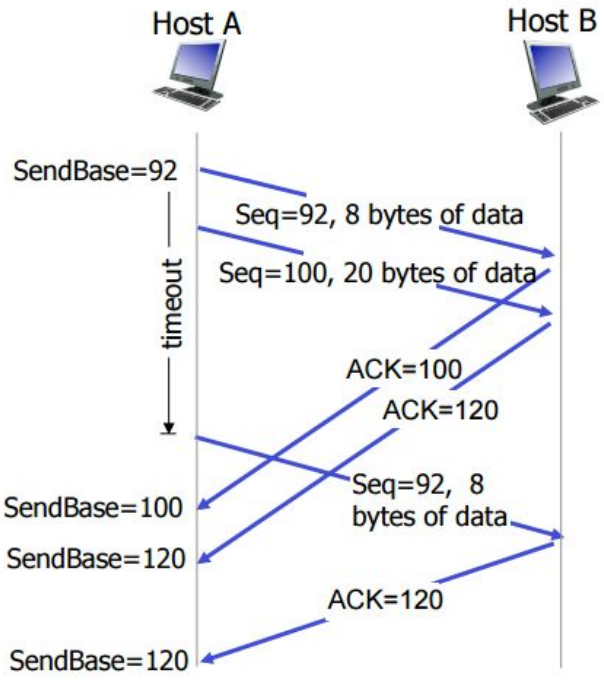
ACK received, with ACK field value y

```
if (y > SendBase) {
   SendBase = y
   /* SendBase–1: last cumulatively ACKed byte */
   if (there are currently not-yet-acked segments)
       start timer
     else stop timer
    }
```

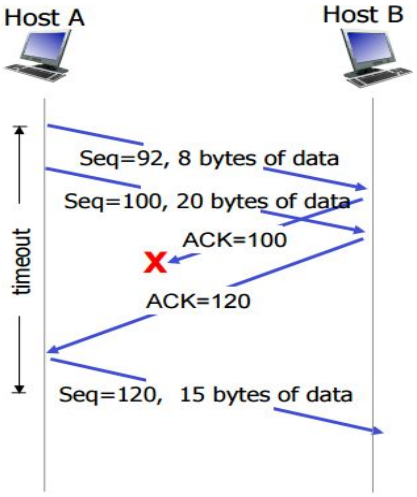
TCP: retransmission scenarios



lost ACK scenario



premature timeout



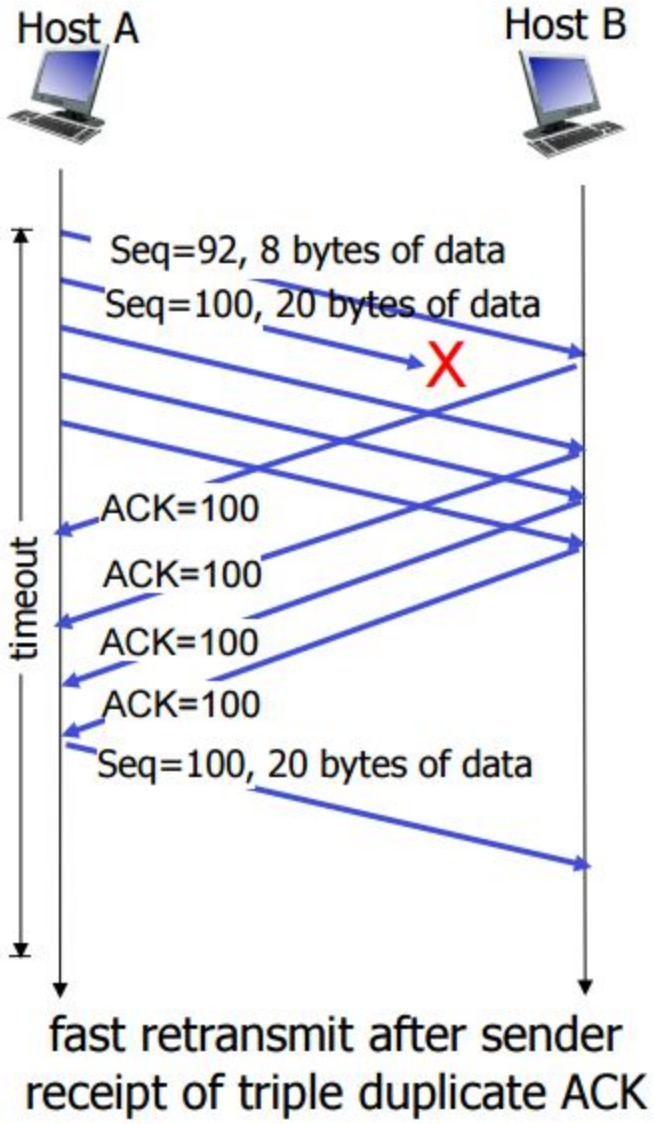
cumulative ACK

Event at receiver	TCP receiver action
- Arrival of in-order segment with expected sequence number. All data up to expected sequence number already ACKed	- delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
- Arrival of in-order segment with expected seq #. One other segment has ACK pending	- Immediately send single cumulative ACK, ACKing both in-order segments
- Arrival of out-of-order segment higher-than-expect seq. # . Gap detected	- Immediately send duplicate ACK, indicating seq. # of next expected byte
- Arrival of segment that partially or completely fills gap	- Immediate send ACK, provided that segment starts at lower end of gap

TCP fast retransmit

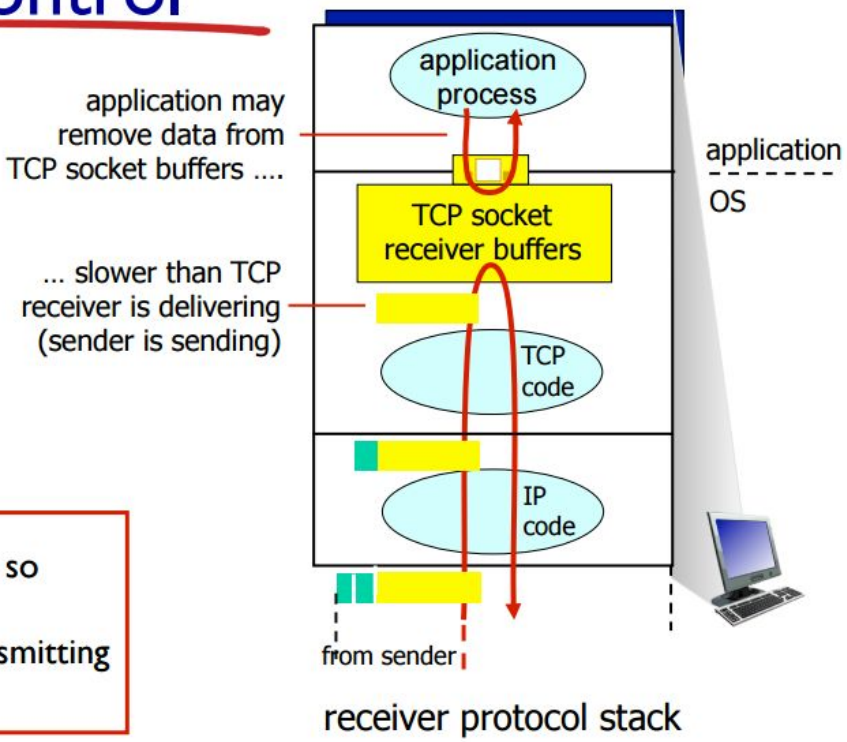
- Timeout period often relatively long. Long delay before resending lost packets
- Detect lost segments via duplicate ACKS. Sender often sends many segments back to back. If segments is lost, there will be many duplicate ACKs.

If sender receives 3 ACKs for the same data (triple duplicate ACKs) resend unacked segment with smallest sequence number. Like that unacked segment lost, so don't wait for timeout.



TCP flow control

TCP flow control

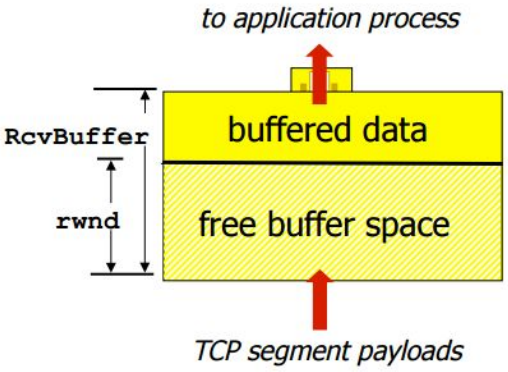


flow control

receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

- Receiver advertises free buffer space by including rwnd value in TCP header of receiver-to-sender segments
    - RcvBuffer size set via socket options (typical default is 4096 bytes)

- Sender limits amount of unacked (in flight) data to receiver's rwnd value
- Guarantees receive buffer will not overflow.

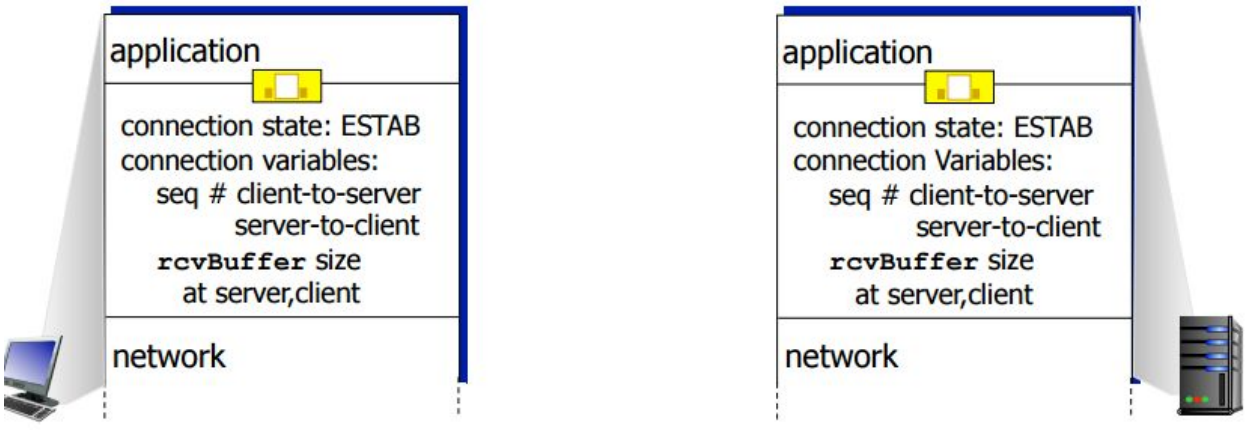


receiver-side buffering

.

TCP connection management

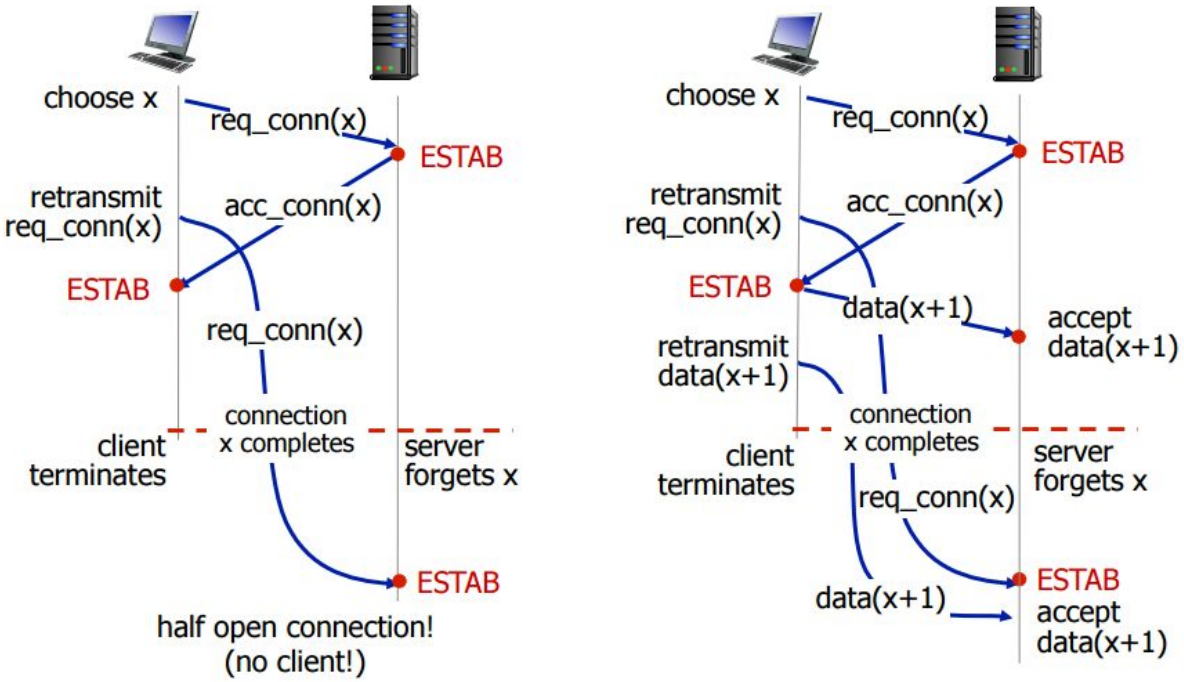
Initially the sender/receiver must handshake before data exchange. They both have to agree to establish a connection and on connection parameters.




```
Socket clientSocket =
  newSocket("hostname","port
  number");
```

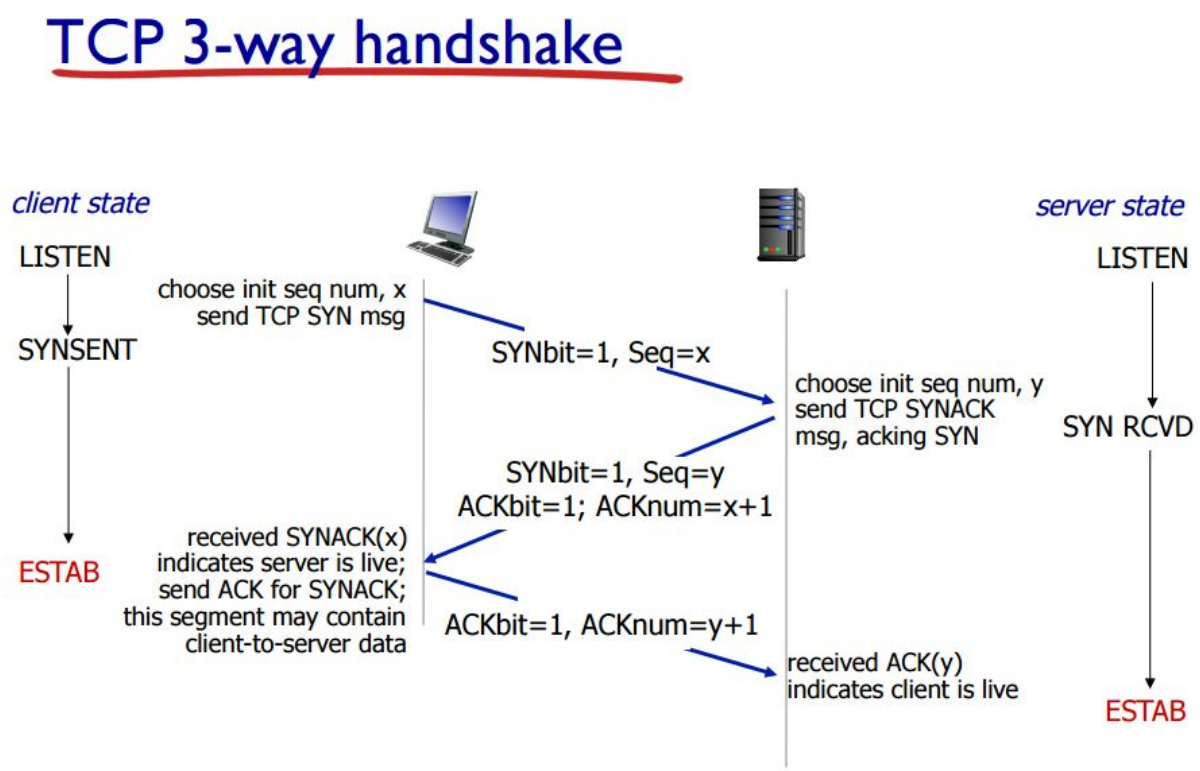
```
Socket connectionSocket =
  welcomeSocket.accept();
```

2-way handshake failure scenarios:

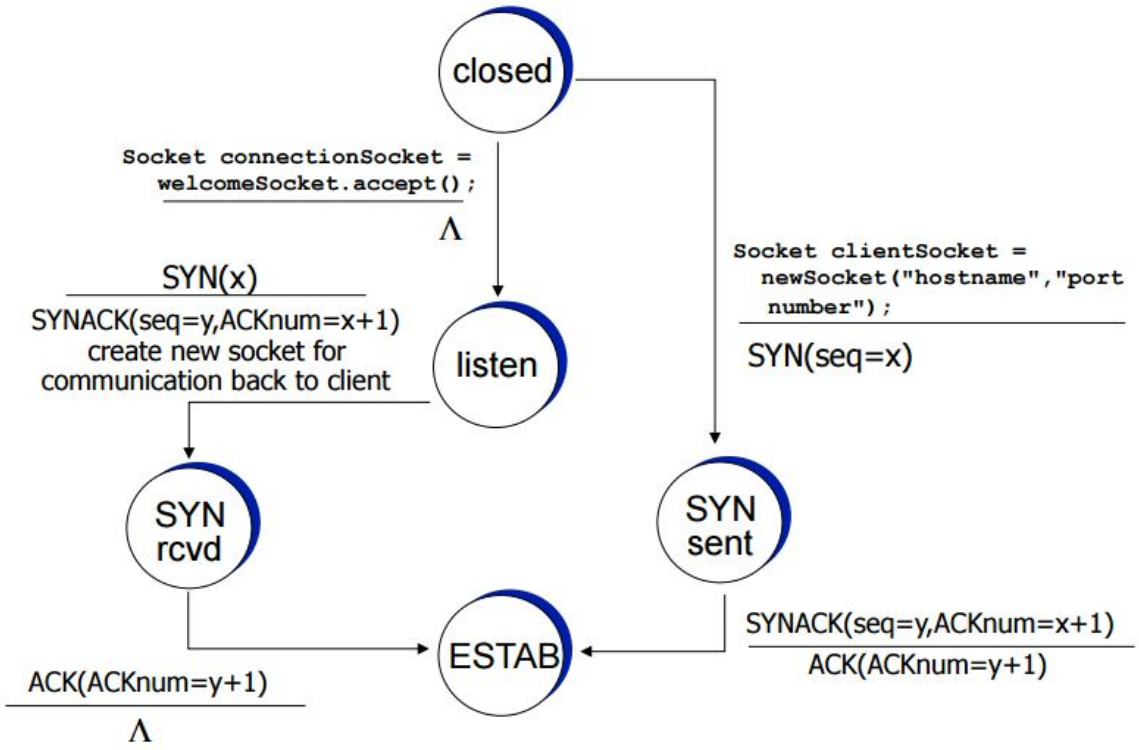


TCP connections can fail due to

- Variable delays
- Retransmitted messages due to message loss
- Message reordering
- Cant see the other side

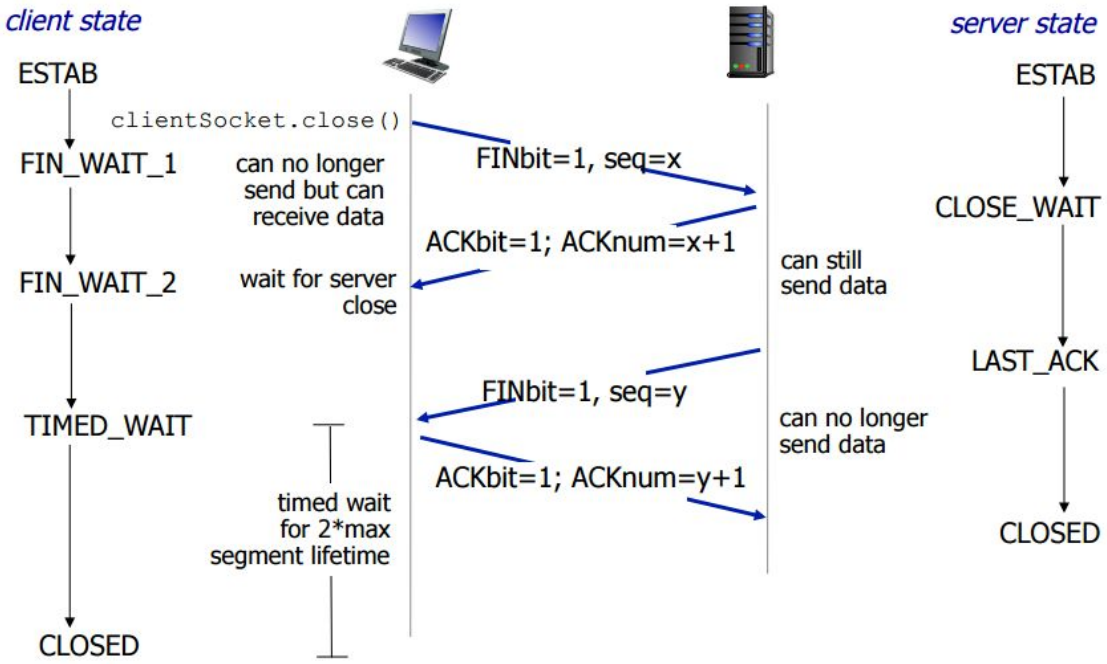


TCP 3-way handshake: FSM



Closing a TCP connection

- Client, server each close their side of connecion
    - Send TCP segment with FIN bit = 1.
- Respond to received FIN with ACK
    - On receiving FIN, ACK can be combined with own FIN
- Simultaneous FIN exchanges can be handled.



Principles of Congestion Control

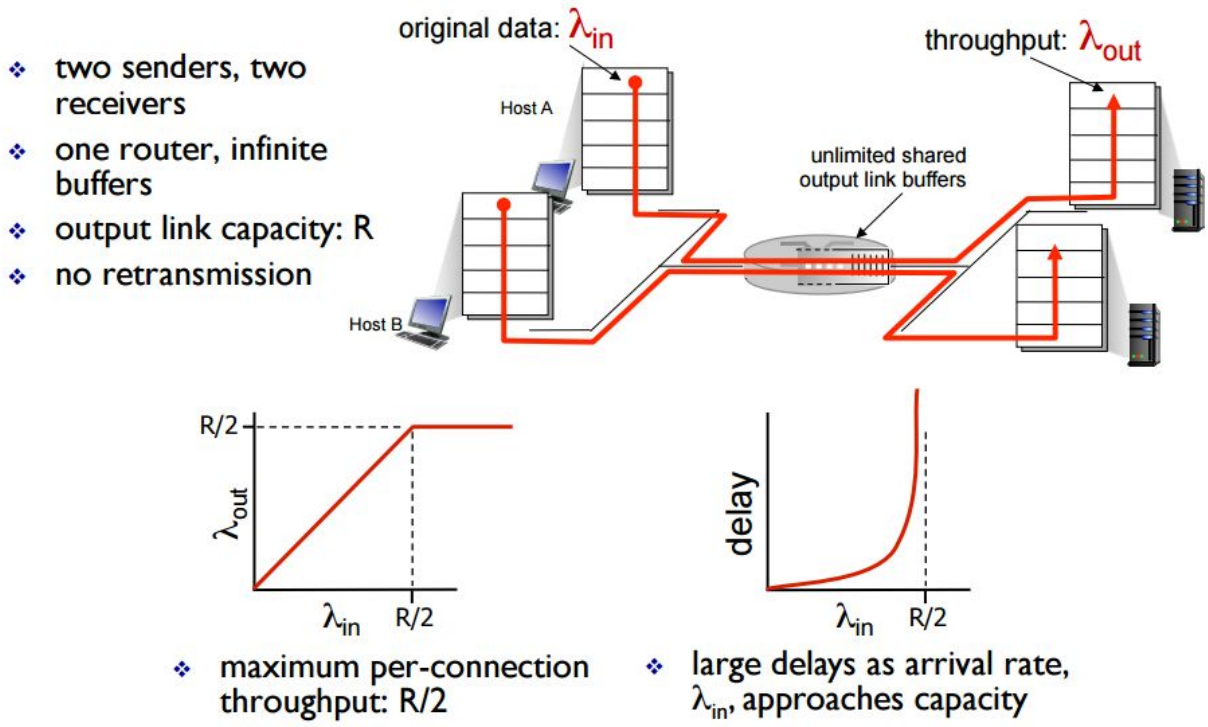
Congestion: Too many sources sending too much data too fast for network to handle.

Congestion can manifestate in 2 ways:

- Lost packets (buffer overflow at routers)
- Long delays (Queueing in router buffers)

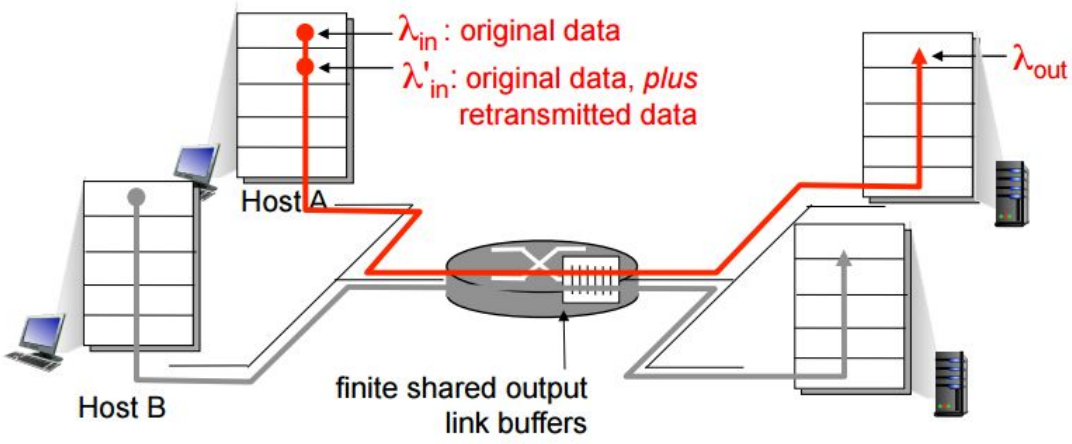
Congestion and flow control are different!

Scenario 1:



Scenario 2:

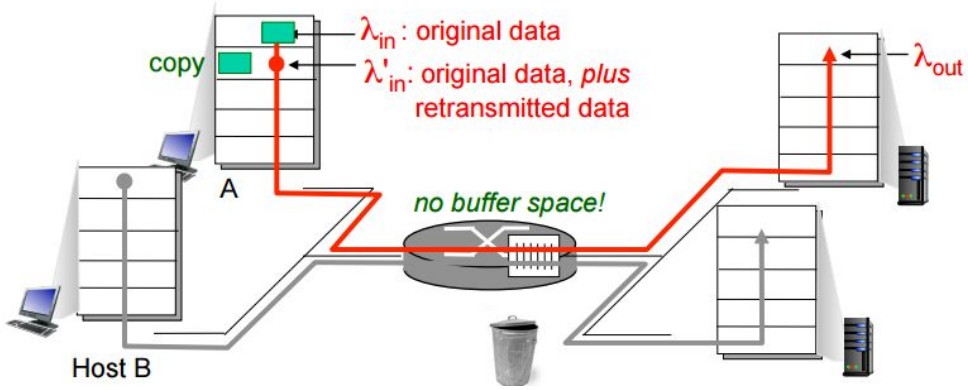
- one router, *finite* buffers
- sender retransmission of timed-out packet
  - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
  - transport-layer input includes *retransmissions* : $\lambda'_{in} \geq \lambda_{in}$



Idealization: known loss

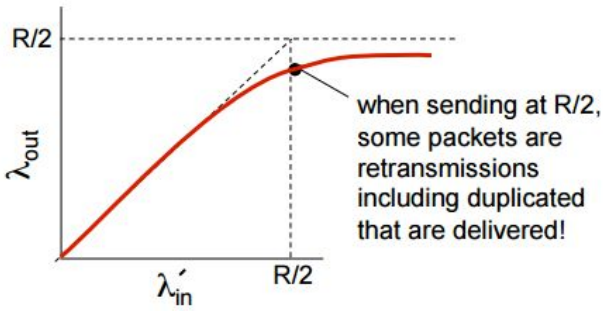
packets can be lost, dropped at router due to full buffers

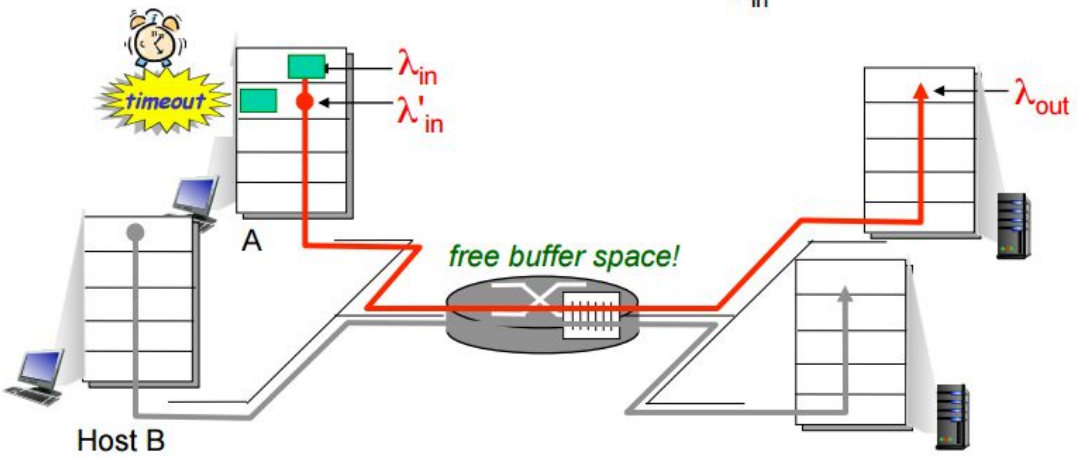
- sender only resends if packet *known* to be lost



Realistic: duplicates

- packets can be lost, dropped at router due to full buffers
- sender times out prematurely, sending *two* copies, both of which are delivered





Costs of congestion:

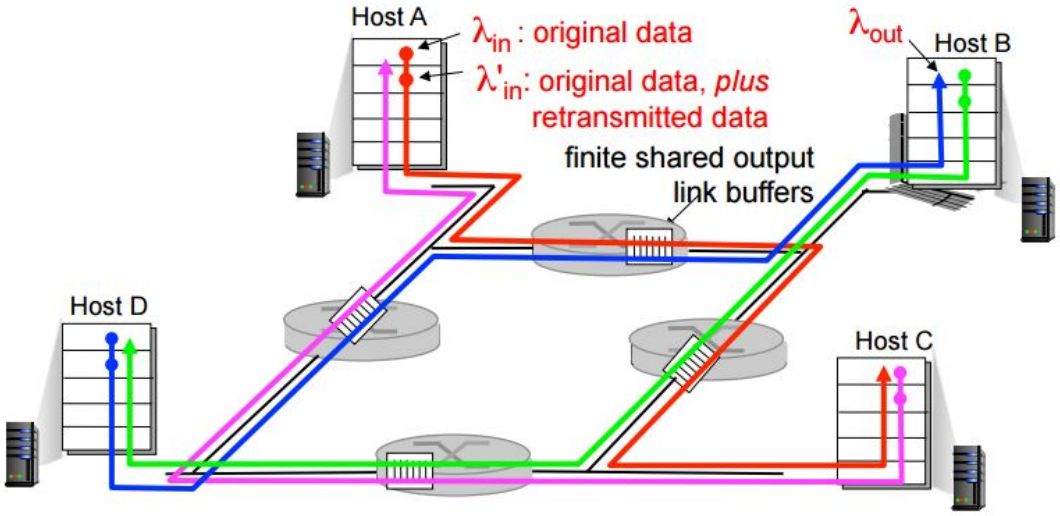
- More work (retransmission) for given goodput
- Unneeded retransmissions: link carries multiple copies of packet. This decreases goodput.
- When packet dropped, any " upstream transmission capacity used for that packet was wasted!

Scenario 3:

- four senders
- multihop paths
- timeout/retransmit

Q: what happens as λ_{in} and λ_{in}' increase ?

A: as red λ_{in}' increases, all arriving blue pkts at upper queue are dropped, blue throughput → 0



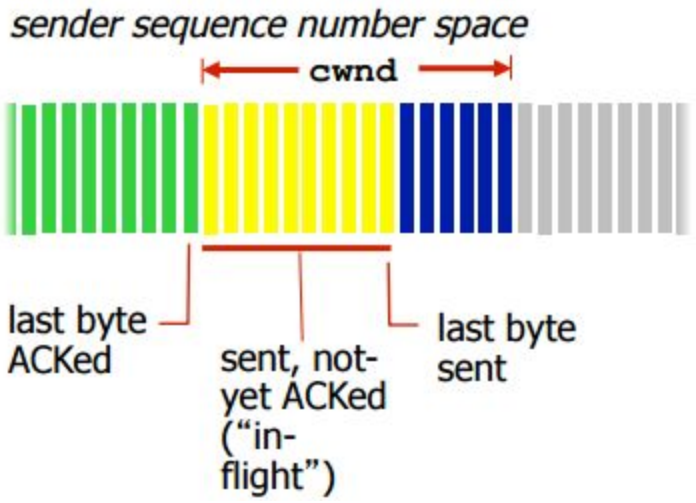
TCP congestion control

Additive Increase, Multiplicative Decrease.

Approach: Sender increases transmission rate(window size), probing for usable bandwidth, until loss occurs.

Cwnd = congestion window size.

Additive Increase: Increase cwnd by 1 MSS every RTT until loss detected.
Multiplicative decrease: cut cwnd in half after loss



Sender limits transmission: (LastByteSent)-(LastByteAcked) ≤ cwnd
Cwnd is dynamic, it is a function of perceived network congestion.

TCP sending rate: send cwnd bytes, wait RTT for ACKS then send more bytes

$$Rate = \frac{cwnd}{RTT} \; bytes/sec$$

Cwnd = window size * segments (??)

TCP Slow Start
Initial rate is slow but ramps up exponentially fast

When congestion begins, increase rate exponentially until first loss event.

- Initally set cwnd = 1 MSS
- Double cwnd every RTT
- Done by incrementing cwnd for every ACK received

Reacting and Detecting loss

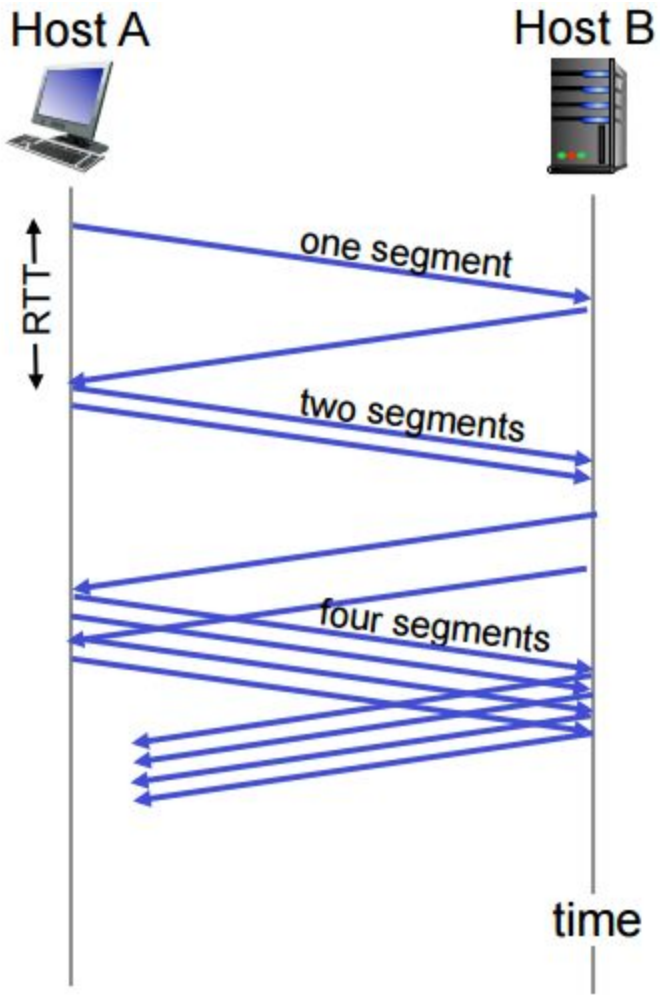
Loss is indicated by timeout:

- Cwnd set to 1 MSS
- Window then grows exponentially(slow start) to a threshold, then grows linearly

Loss indicated by 3 duplicate ACKs : TCP RENO

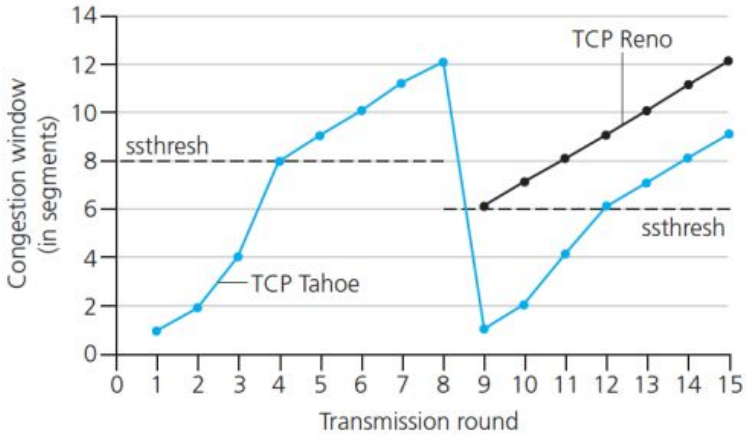
- Duplicate ACKs indicate network capable of delivering some segments
- Cwnd is cut in half window then grows linearly

TCP Tahoe always sets cwnd to 1 (timeout or 3 duplicate ACKs)



Q: when should the exponential increase switch to linear?

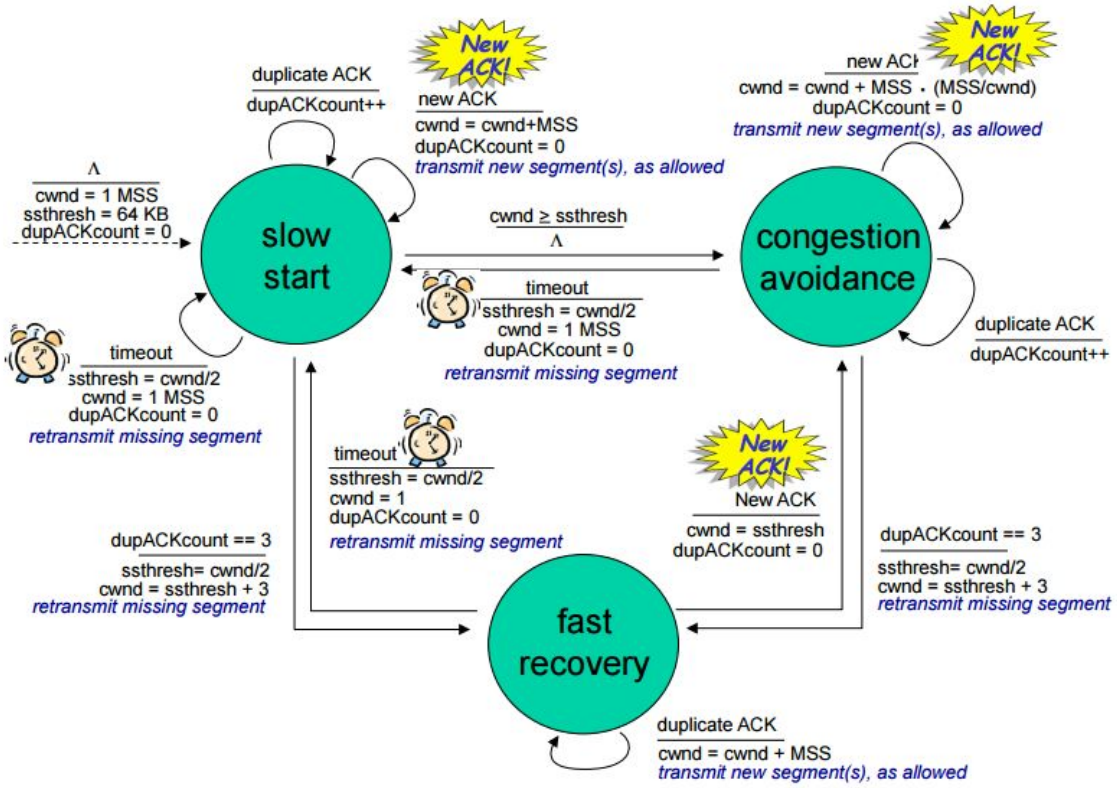
A: when **cwnd** gets to 1/2 of its value before timeout.



Implementation:

- variable **ssthresh**
- on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event

Summary: TCP Congestion Control



TCP throughput

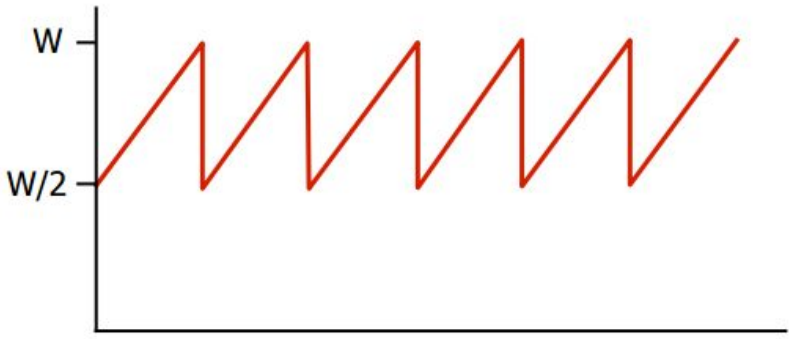
Average TCP throughput as function of window size and RTT?

- Ignore slow start, assume always data to send

W: Window Size(measure in bytes) where loss occurs

- Average window size (# in flight bytes) is ¾ W
- Average throughput is 3/4W per RTT

$$\text{avg TCP thruput} = \frac{3}{4}\frac{W}{RTT} \text{ bytes/sec}$$



TCP over long fat pipes

Example:

1500 byte segments, 100ms RTT, want 10 Gbps throughput.
Requires W = 83,333 in-flight segments
Throughput in terms of segment loss probability, L [Mathis 1997]:

$$TCP\ Throughput = \frac{1.22 * MSS}{RTT\sqrt{L}}$$

➜to achieve 10 Gbps throughput, need a loss rate of L = 2·10-10 – a very small loss rate!

new versions of TCP for high-speed

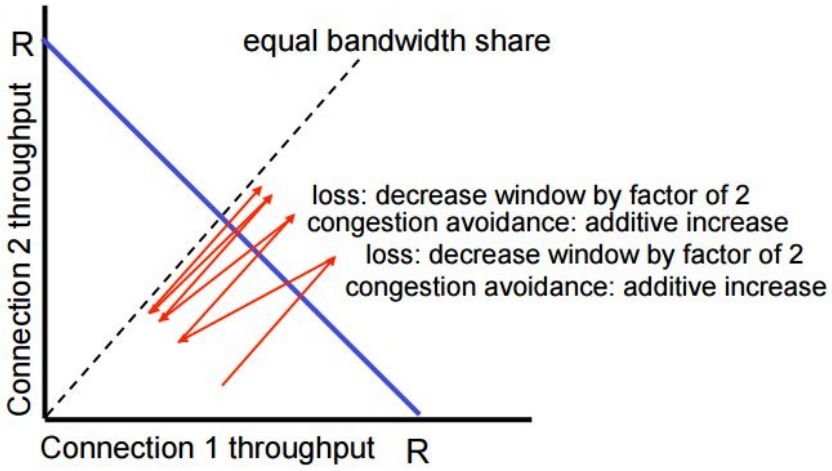
TCP Fairness

Fairness goal: if N TCP sessions share same bottleneck link of bandwidth R, Each should have an average rate of R/N

TCP is fair because

two competing sessions:

- additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally



Fairness and UDP

- multimedia apps often do not use TCP
  - Do not want rate throttled by congestion control
- Instead use UDP:
  - Send audio/video at constant rate, tolerate packet loss

Fairness Parallel TCP connections

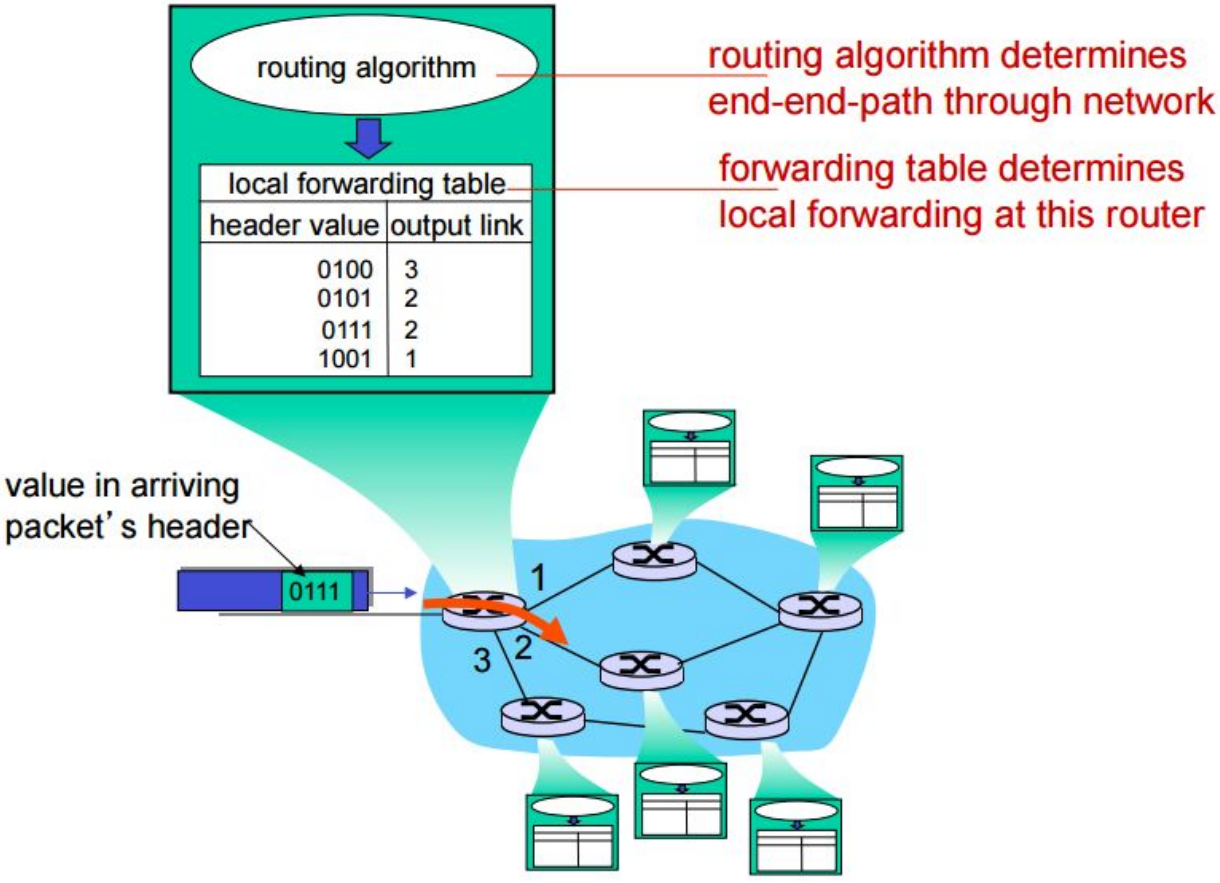
- Application can open multiple parallel connections between two hosts
- Web browsers do this
- e.g., link of rate R with 9 existing connections:
- new app asks for 1 TCP, gets rate R/10
- new app asks for 11 TCPs, gets R/21

Chapter 4 - Network Layer

- Transport segment from sending to receiving host
- On sending side encapsulates segments into datagrams
- On receiving side, delivers segments to transport layer
- Network layer protocols in every host, router
- Router examines header fields in all IP datagrams passing through it

2 key functions of the network layer **//EXAM QUESTION.**

- **Forwarding:** move packets from router's input to appropriate router output
- **Routing**: determine route taken by packet from source to dest. Via routing algos.



Connection Setup: Before datagrams flow, two end hosts and intervening routers establish virtual connection. The routers will get involved

3rd important function in some network architectures: ATM, frame relay, X.25

Network service model

What service model for “channel” transporting datagrams from sender to receiver?

Example Services for an individual datagram:

- Guaranteed delivery.
- Guaranteed delivery with less than 40 msec delay

Example Services for a flow of datagram:

- In-order datagram delivery
- Guaranteed minimum bandwidth to flow
- Restrictions on changes in inter-packet spacing

Network Architecture	Service Model	Guarantees ?				Congestion feedback
		Bandwidth	Loss	Order	Timing	
Internet	best effort	none	no	no	no	no (inferred via loss)
ATM	CBR	constant rate	yes	yes	yes	no congestion
ATM	VBR	guaranteed rate	yes	yes	yes	no congestion
ATM	ABR	guaranteed minimum	no	yes	no	yes
ATM	UBR	none	no	yes	no	no

Virtual Circuit and Datagram Networks

Datagram Network provides network-layer **connectionless** service.
Virtual-Circuit network provides network-layer **connection** service.

Analogous to TCP/UDP connection-oriented / connectionless transport-layer services, but:

- Service is host to host
- No choice: network provides one or the other
- Implementation in network core

Virtual Circuits

Source to dest path behaves much like telephone circuit

- Performance-wise
- Network actions along source to dest path

- Call setup, teardown for each call before data can flow
- Each packet carries VC identifier (not destination host address)
- Every router on source-dest path maintains " state " for each passing connection
- Link, router resources (bandwidth, buffers) may be allocated to VC (dedicated resources = predictable service)

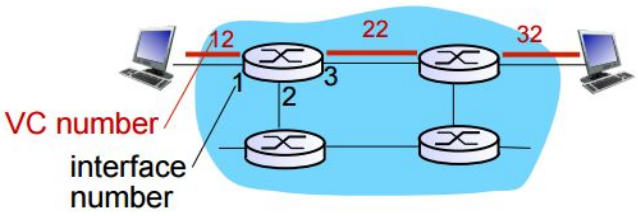
Implementation

A VC consists of:

1. Path from source to destination
2. VC numbers, one number for each link along path
3. Entries in forwarding tables in routers along path.

Packet belonging to VC carries VC number (rather than dest address). VC number can be changed on each link. Those new VC numbers come from forwarding tables.

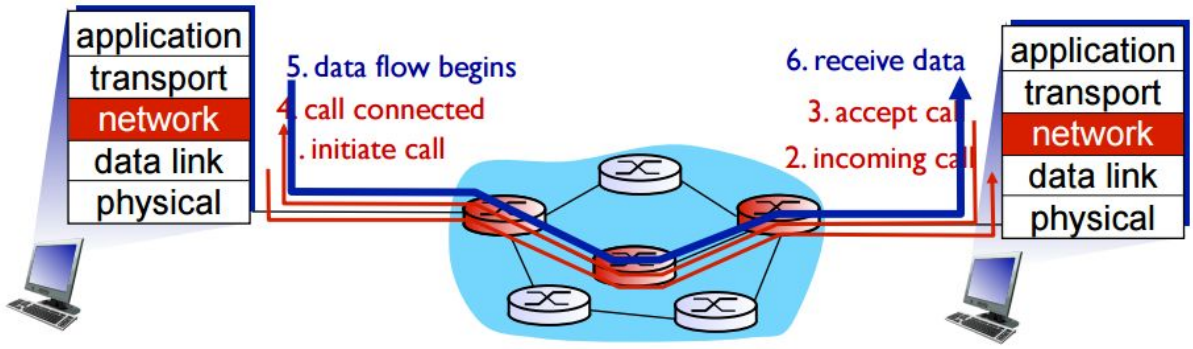
VC forwarding table



forwarding table in northwest router:

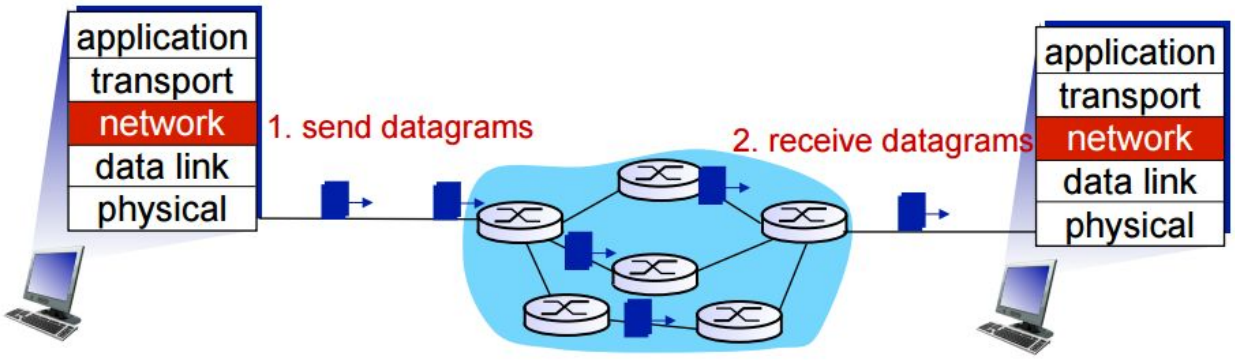
Incoming interface	Incoming VC #	Outgoing interface	Outgoing VC #
1	12	3	22
2	63	1	18
3	7	2	17
1	97	3	87
…	…	…	…

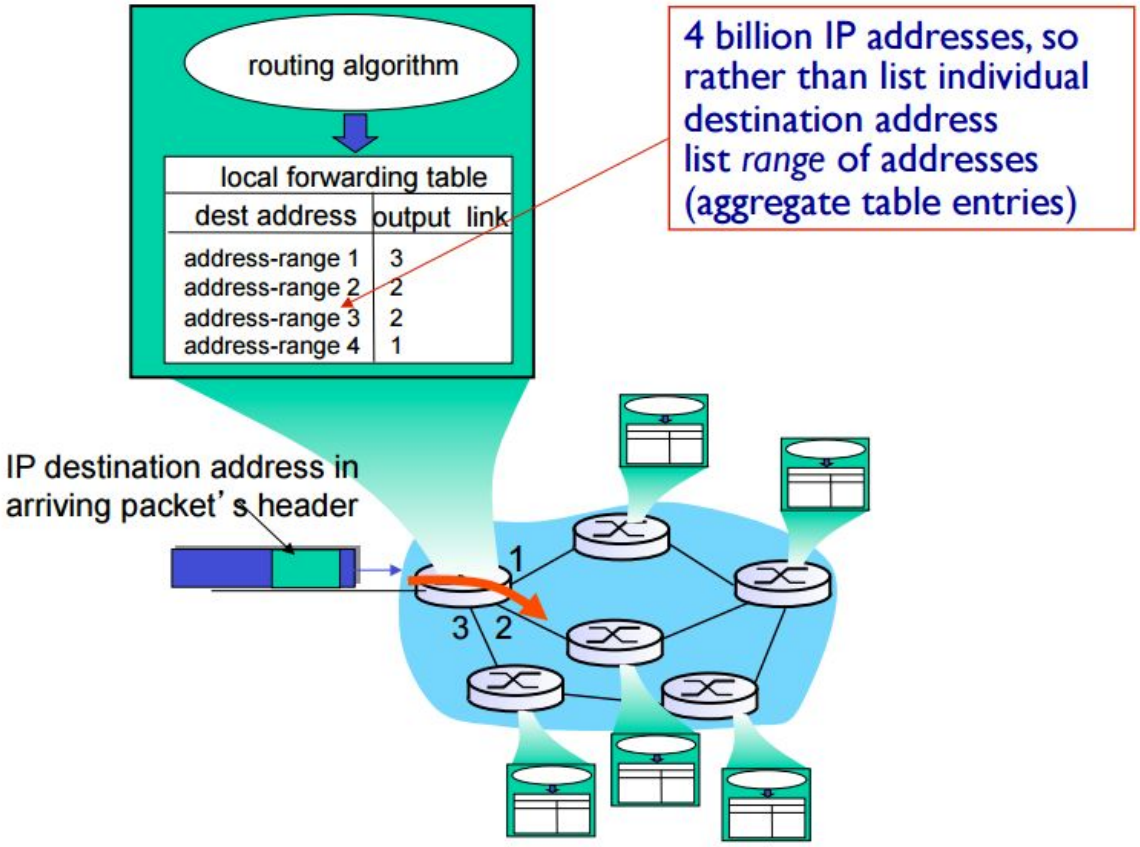
VC routers maintain connection state information!



Datagram networks

- No call setup at network layer.
- Routers: no state about end to end connections.
    - No network-level concept of “connection”
- Packets forwarded using destination host address.





Longest Prefix Matching: when looking for forwarding table entry for given destination address, use longest address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010*** *********	0
11001000 00010111 00011000 *********	1
11001000 00010111 00011*** *********	2
otherwise	3

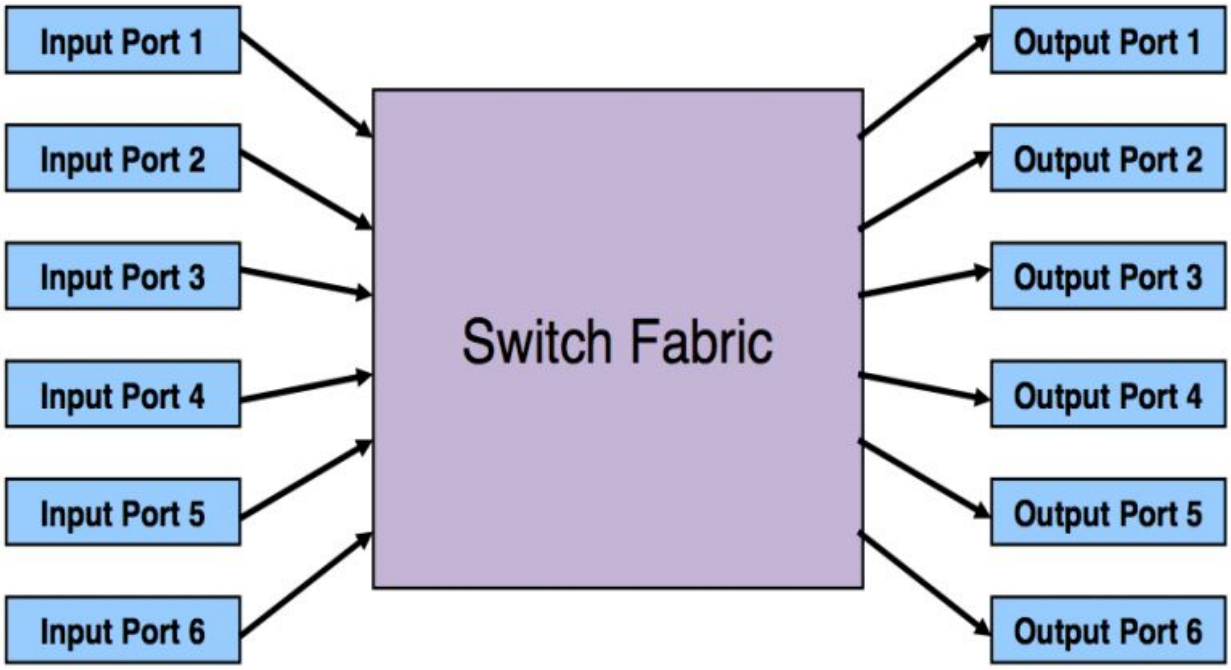
Internet(Datagram)

- Data exchange among computers
  - "Elastic" service.
- Many link types
  - Different characteristics
  - Uniform service difficult
- Smart end systems
  - Can adapt, perform control, error recovery
  - **Simple inside network, complexity at edge**

ATM(VC)

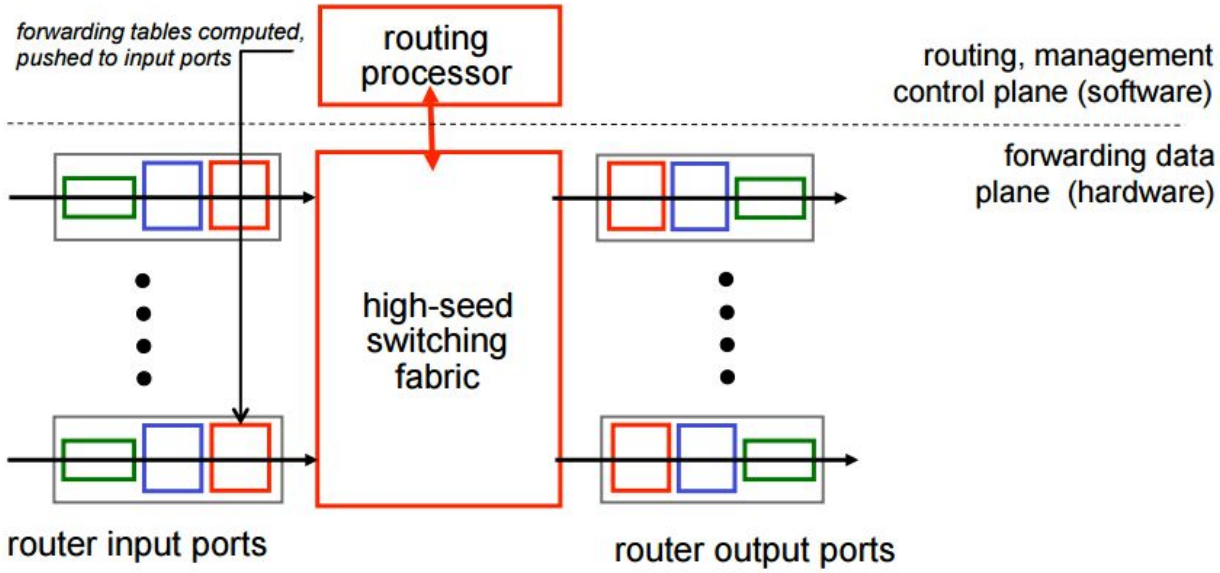
- Evolved from telephony
- Human Conversation:
  - Strict timing, reliability requirements
  - Need for guaranteed service
- Dumb end systems
  - Telephones
  - **Complexity inside network**

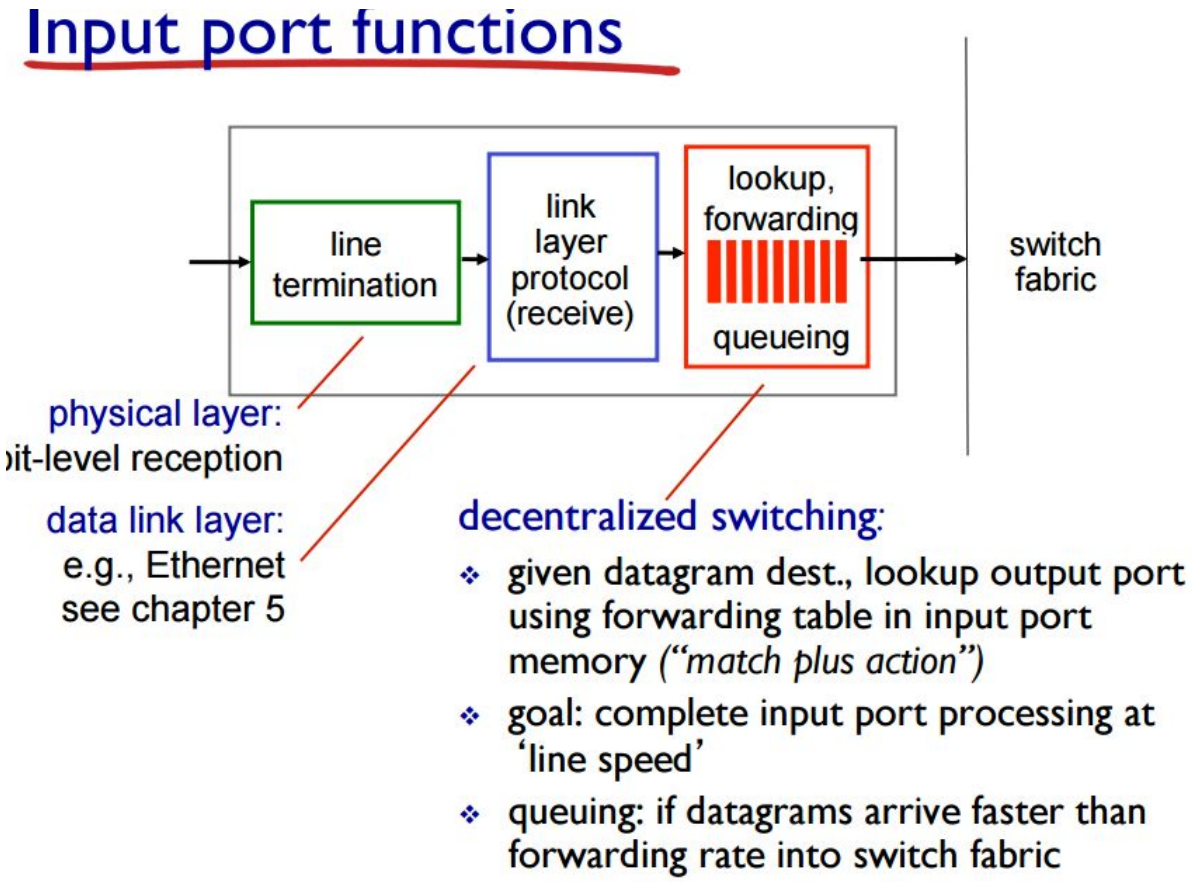
Routers



Two key routing functions:

- Run routing algorithms/protocol (RIP, OSPF, BGP)
- Forwarding datagrams from incoming to outgoing link

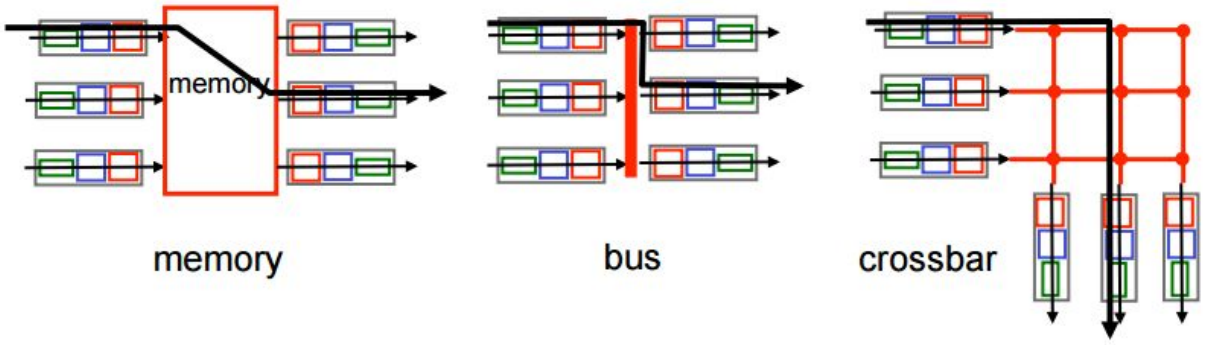




Switching fabrics

- Transfer packet from input buffer to appropriate output buffer
- Switching rate: rate at which packets can be transferred from inputs to outputs
    - Often measured as a multiple of input/output line rate
    - N inputs: switching rate N times line rate desirable

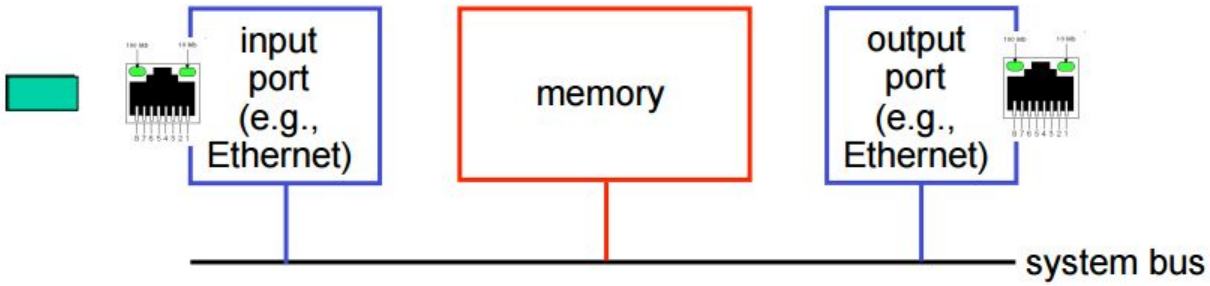
Types of switching fabrics:



Switching via memory

First generation routers:

- Traditional computers with switching under direct control of CPU
- Packet copied to system's memory
- Speed limited by memory bandwidth (2 bus crossing per datagram)



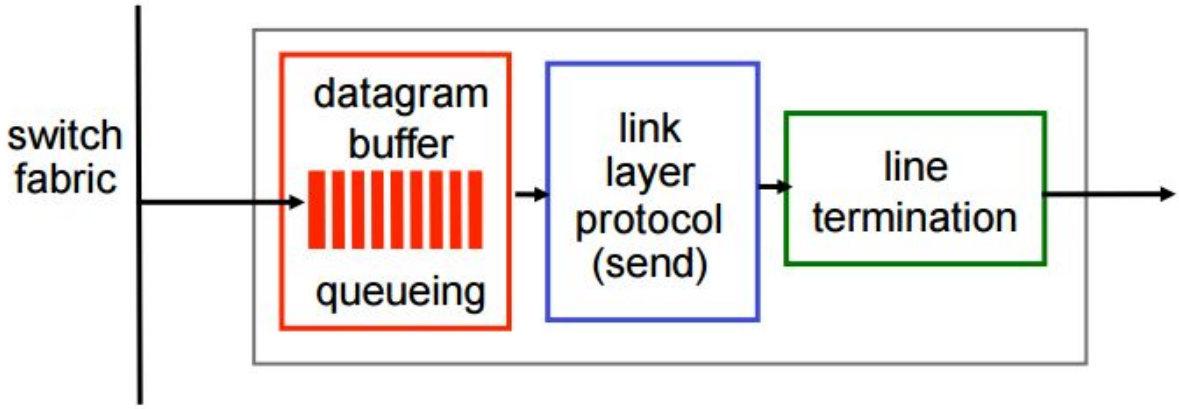
Switching via a bus

- Datagram from input port memory to output port memory via a shared bus
- Bus connection: switching speed limited by bus bandwidth
- 32 GBps bus, Cisco 5600: sufficient speed for access and enterprise routers

Switching via interconnection network

- Overcome bus bandwidth limitations
- Banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor
- Advanced design fragmenting datagram into fixed length cells, switch cells through the fabric

Output Ports



Buffering required when datagrams arrive from fabric faster than the transmission rate
Scheduling discipline chooses among queued datagrams for transmission.

Output Port Queueing

buffering when arrival rate via switch exceeds output line speed queueing (delay) and loss due to output port buffer overflow!

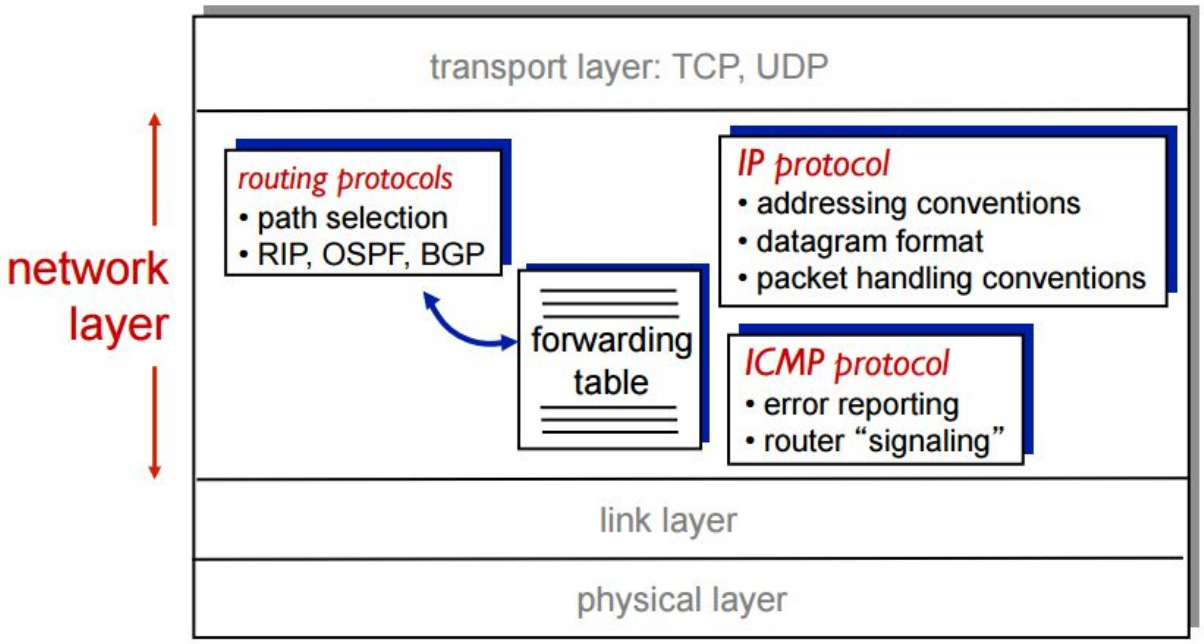
To calculate buffering
$\frac{RTT * C}{\sqrt{N}}$ where N is number of flows and C is the link capacity

Input port Queueing

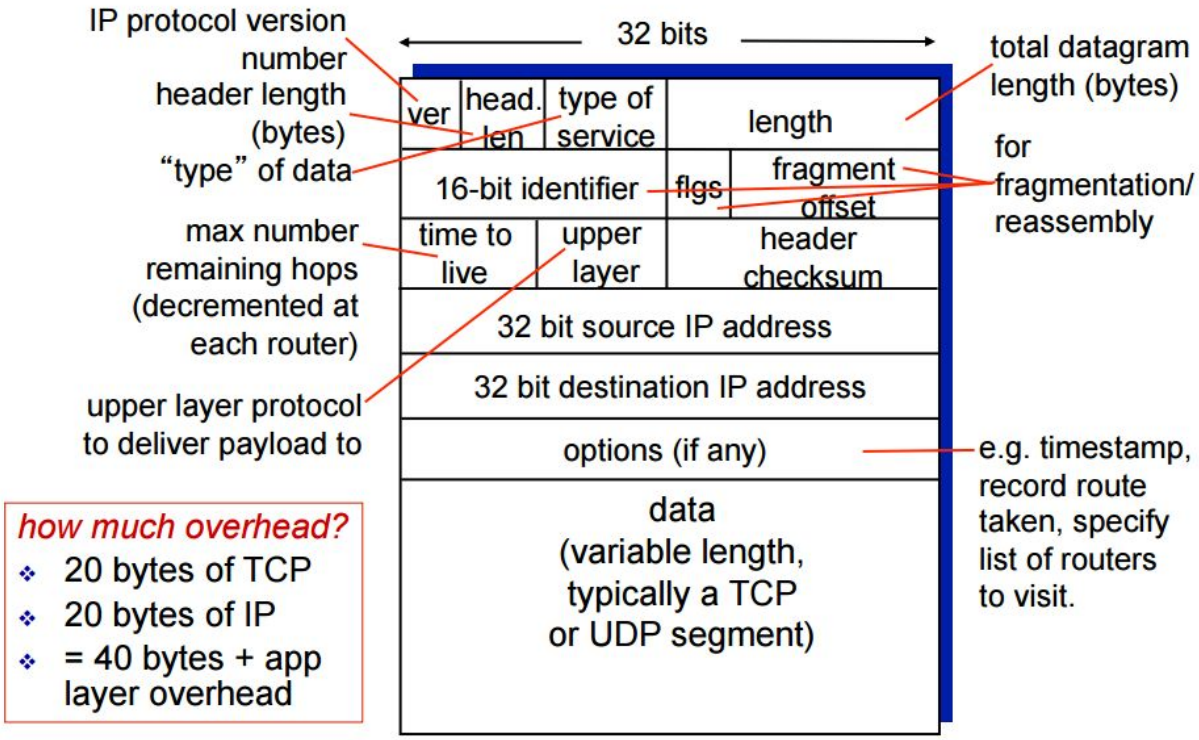
Fabric slower than input ports combined -> queueing may occur at input queues. Queueing delay and loss due to input buffer overflow

Head-of-the-line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward.

IP: Internet Protocol

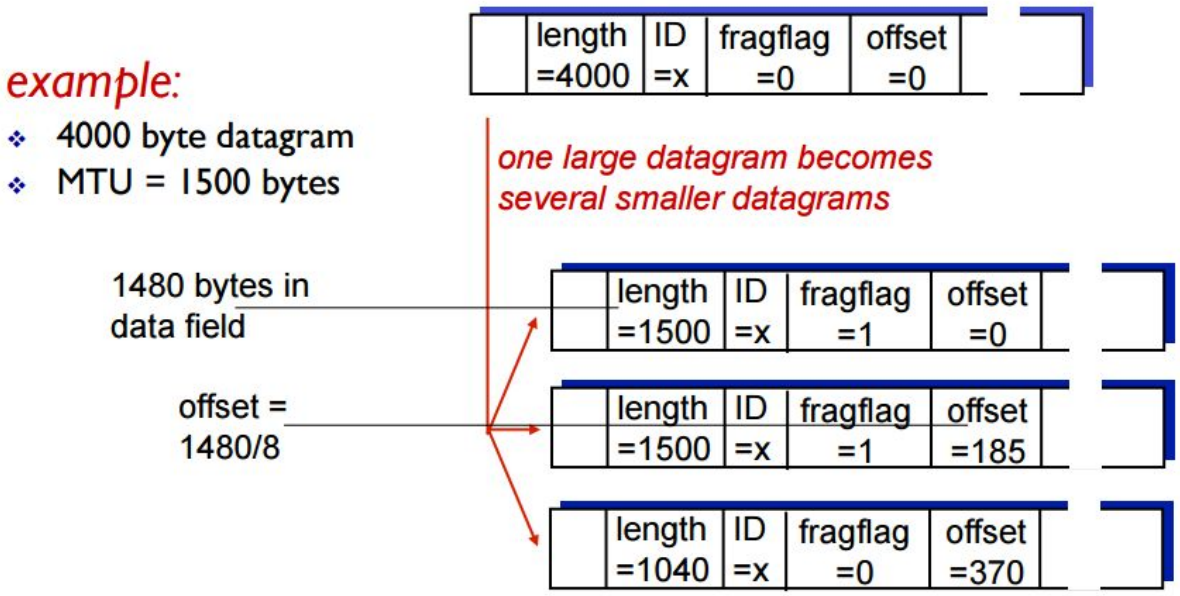


IP datagram format



IP fragmentation, reassembly

- Networks links have MTU (maximum transfer size) - largest possible link-level frame
    - Different link types, different MTUs
- Large IP datagram divided (fragmented) within net
    - One datagram becomes several datagrams
    - Reassambled only at final destination
    - IP header bits used to identify, order related fragments



IP addressing: introduction

Ip address: 32-bit identifier for host, router interface
Interface: connection between host/router and physical link

- Router typically have multiple interfaces
- Host typically has one or two interfaces(wired/wireless)

IP addresses associated with each interface.

Subnets
IP address:

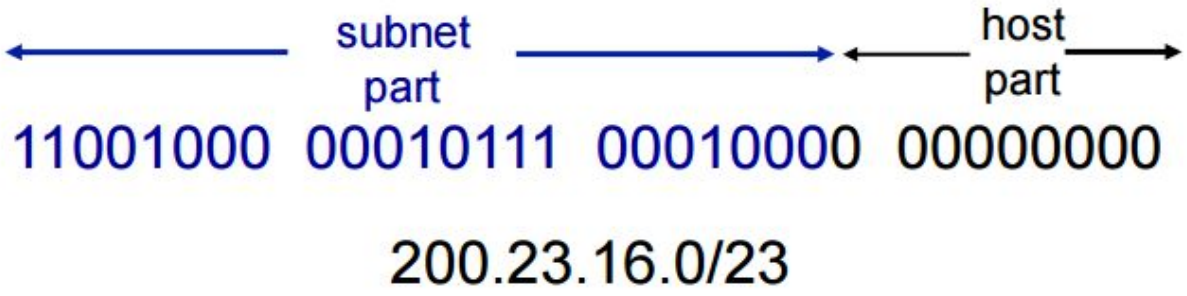
- Subnet part - high order bits
- Host part - low order bits

A subnet is a device interfaces with same subnet part of IP address, can physically reach other without intervening router.

To determine the subnets, detach each interface from its host or router, creating islands of isolated networks

CIDR: Classless InterDomain Routing

- Subnet portion of address of arbitrary length
- Address format: a.b.c.d/x, where x is # bits in subnet portion of address



How to obtain an IP address

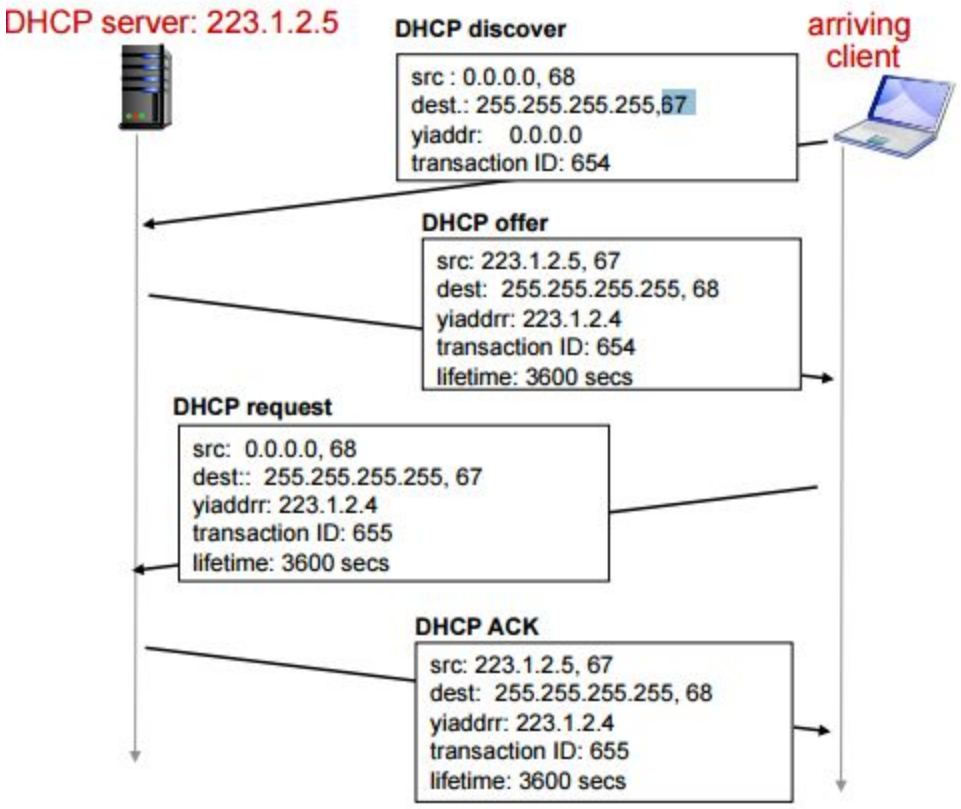
1. Hard Coded
2. **DHCP**: Dynamic Host Configuration Protocol: dynamically get address from a server "plug and play"

Dynamic Host Configuration Protocol:
Goal: allow host to dynamically obtain its IP address from network server when it joins network.

- Can renew its lease on address in use
- Allows reuse of addresses (only hold address while connected or on)
- Support for mobile users who want to join the network.

Overview

- Host broadcasts "DHCP discover" msg [optional]
- DHCP server responds with "DHCP offer" msg [optional]
- Host requests IP address: "DHCP request" msg
- DHCP server sends address: "DHCP ack" msg

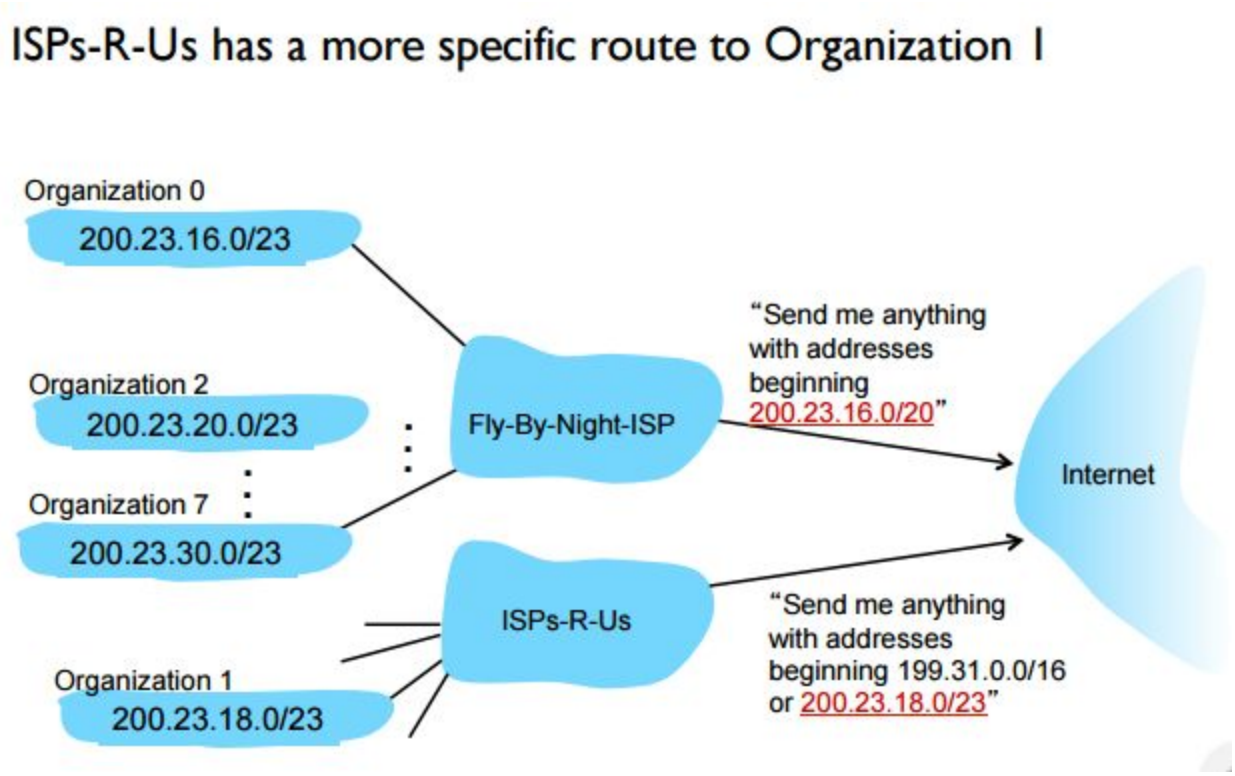


DHCP can return more than just allocated IP address on subnet:

- Address of first-hop router for client
- Name and IP address of DNS sever
- Network mask (indicating network versus host portion of address)

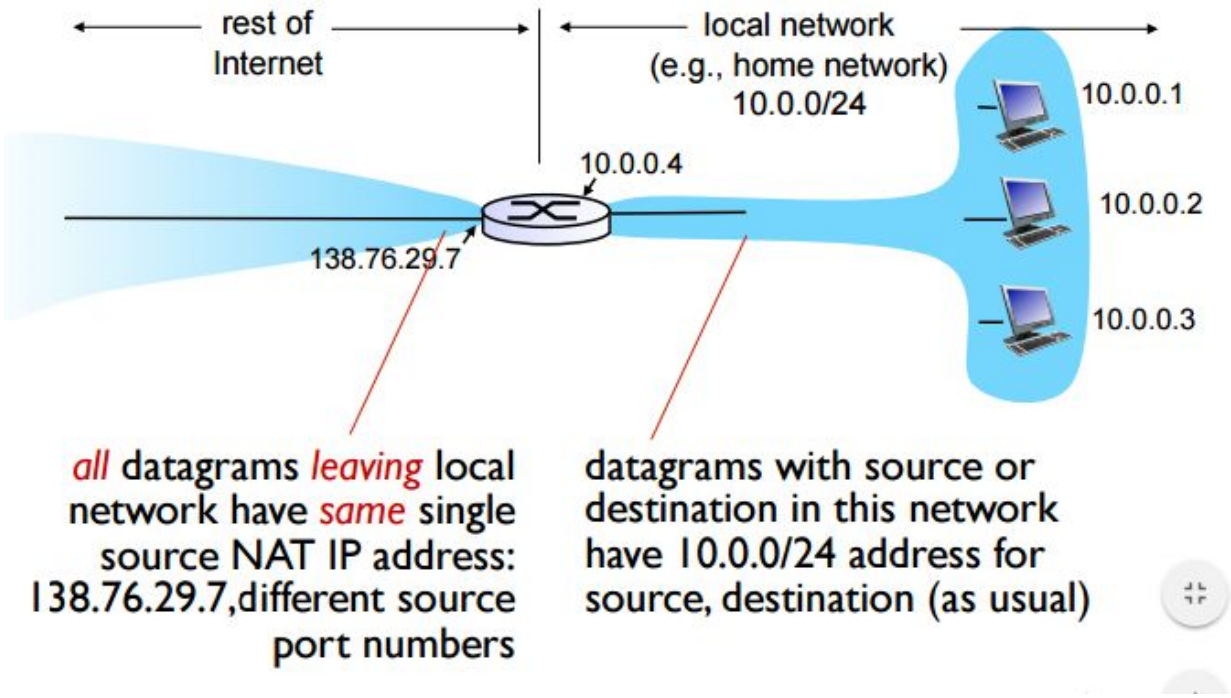
How does network get subnet part of IP addr? Gets allocated portion of its provider ISP's address space

ISP's block	11001000 00010111 00010000	00000000	200.23.16.0/20
Organization 0	11001000 00010111 00010000	00000000	200.23.16.0/23
Organization 1	11001000 00010111 00010010	00000000	200.23.18.0/23
Organization 2	11001000 00010111 00010100	00000000	200.23.20.0/23
...	....	....	....
Organization 7	11001000 00010111 00011110	00000000	200.23.30.0/23



How does an ISP get block of addresses?ICANN: Internet Corporation for Assigned Names and Numbers http://www.icann.org/ allocates addresses, manages DNS, assigns domain names, resolves disputes

NAT: Network Address Translation

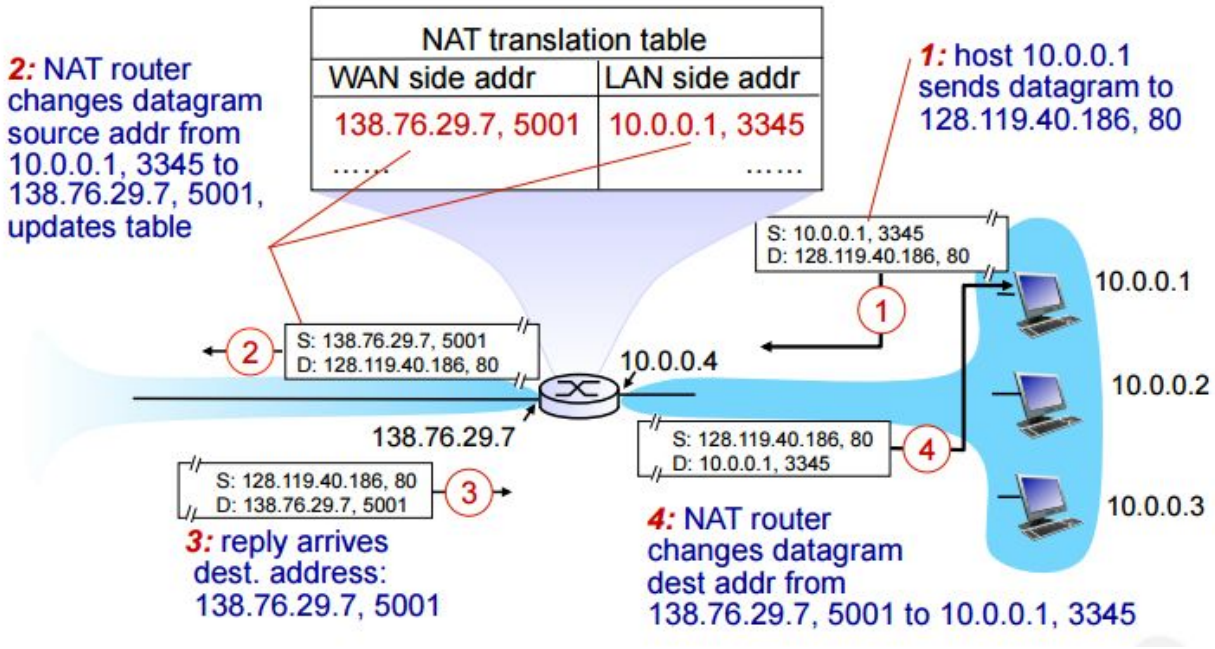


motivation: local network uses just one IP address as far as outside world is concerned:

- Range of addresses not needed from ISP: just one IP address for all devices
- Can change addresses of devices in local network without notifying outside world
- Can change ISP without changing addresses of devices in local network
- Devices inside local net not explicitly addressable, visible by outside world (a security plus)

A NAT router must

1. **Outgoing datagrams: replace** (Source IP, port #) of every outgoing datagram to (NAT IP address, new port #). Remote client/servers will respond using (NAT IP address, new port #) as destination address.
2. **Remember (in NAT translation table)** every (source IP address, port #) to (NAT IP address, new port #) translation pair.
3. **Incoming datagrams: replace** (NAT IP address, new port #) in destination fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table.

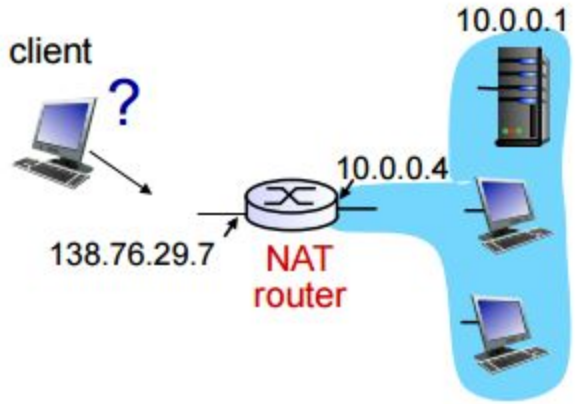


16- bit port number field:
60,000 simultaneous connections with a single LAN-side address

Nat is controversial:

- Routers should only process up to layer 3
- Violates end-to-end argument. NAT possibility must be taken into account by app designer such as p2p applications
- Address shortage can be solved with IPv6.

NAT Traversal Problem



Client wants to connecto to server with address 10.0.0.1

- Server address 10.0.0.1 local to LAN(client can't use it as destination addr)
- Only one externally visible NATed address: 138.76.29.7

Solution 1

Statically configure NAT to forward incoming connection requests at given port to server
E.g (138.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000

Solution 2

Universal Plug and Play (UPnP) internet Gateway Device (IGD) protocol. Allows NATed host to:

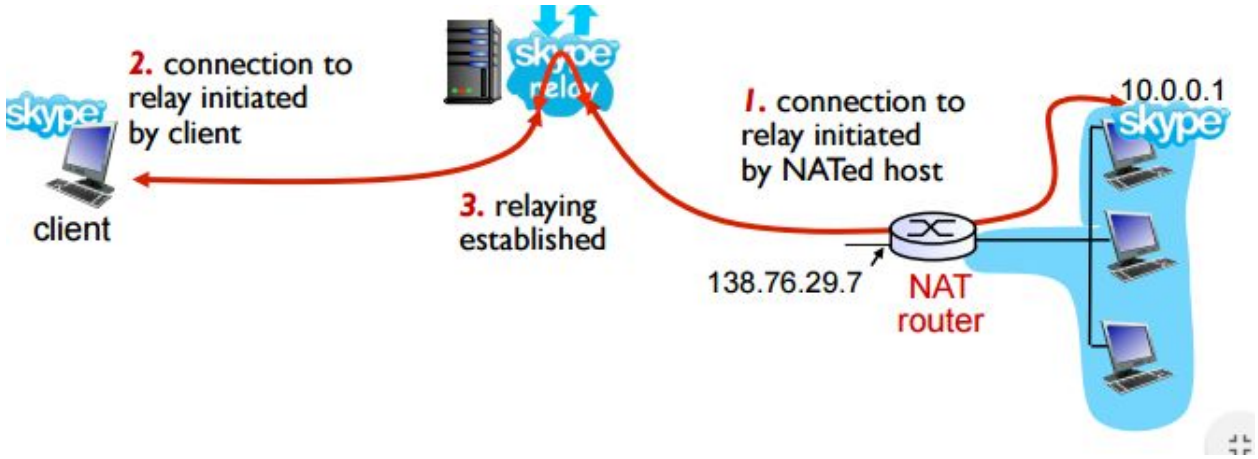
- Learn public IP address (138.76.29.7)
- Add/remove port mappings (with lease times)

i.e. automate static NAT port map configurations

Solution 3

Relaying

- NATed client establishes connection to relay
- External clent connects to relay
- Relay bridges packets between to connections



ICMP: internet control message protocol

- used by hosts & routers to communicate networklevel information
    - Error reporting: unreachable host, network, port, protocol
    - Echo request/reply (used by ping)
- network - layer “above” IP
    - ICMP messages carried in IP datagram
- ICMP message: type, code plus first 8 bytes of IP datagram causing error

Traceroute and ICMP

Source sends series of UDP segments to dest

- First set has TTL = 1
- Second set has TTL = 2, etc.
- Unlikely port number.

When nth set of datagrams arrives to nth router:

- Router discards datagrams
- And sends source ICMP messages (type 11, code 0)
- ICMP messages includes name of router & IP address
- When ICMP message arrives, source records RTTs

Stopping criteria:

- UDP segment eventually arrive at destination host
- Destination returns ICMP “port unreachable” message (type 3, code 3)
- Source stops.

IPv6:

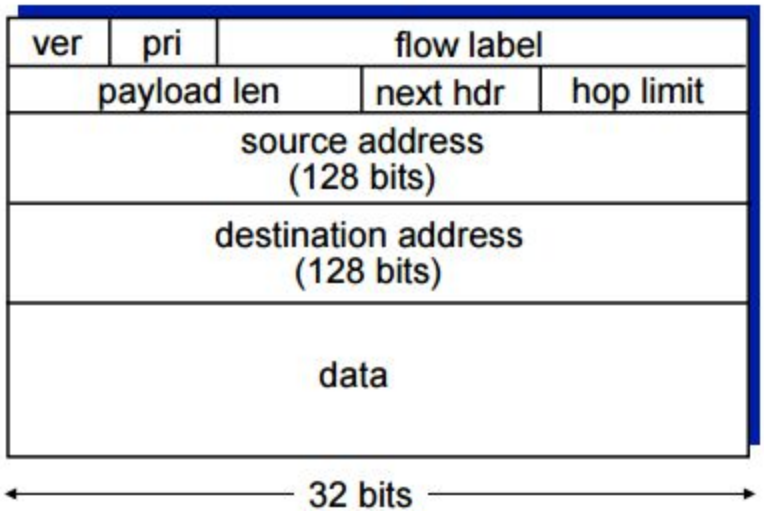
Motivation:

- 32-bit address space will soon be completely allocated
- Header format helps speed processing/forwarding
- Header changes to facilitate QoS

IPv6 datagram format

- Fixed length 40 byte header
- No fragmentation allowed.

priority: identify priority among datagrams in flow
flow Label: identify datagrams in same "flow."
(concept of"flow" not well defined).
next header: identify upper layer protocol for data

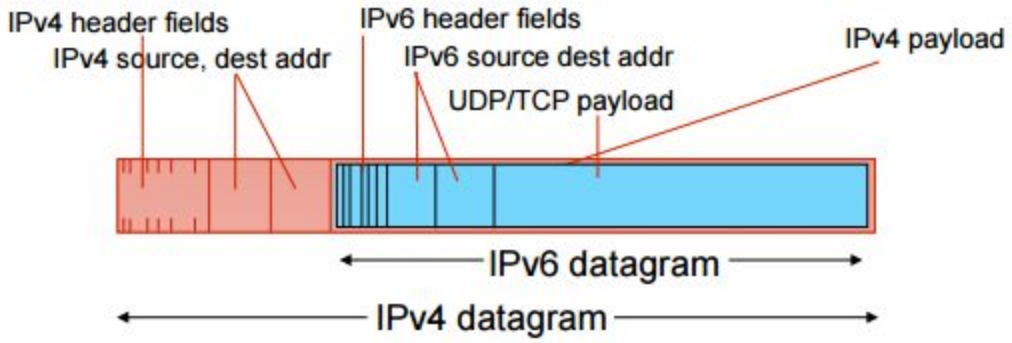


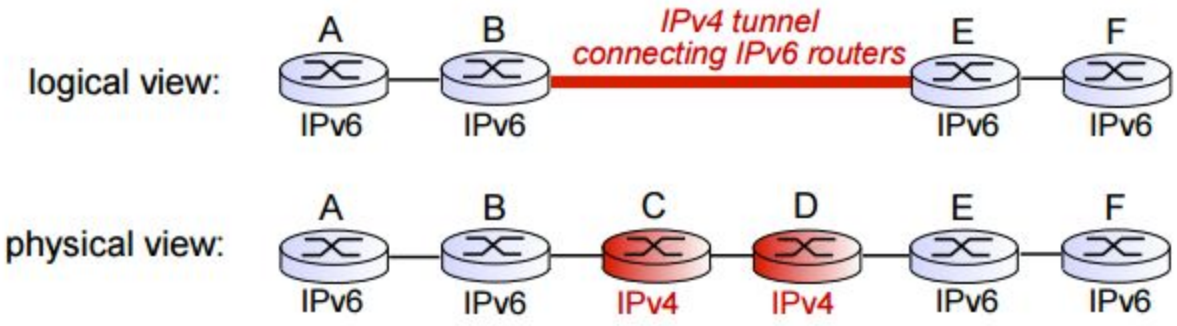
Changes from IPv4

checksum : removed entirely to reduce processing time at each hop.
Options: allowed, but outside of header, indicated by "Next Header" field
ICMPv6: new version of ICMP

- Additional message types, e.g. packet too big
- Multicast group management functions

How will network operate with mixed IPv4 and IPv6 routers?
Tunneling: IPv6 datagram carried as payload in IPv4 datagram among IPv4 routers


IPv4 header fields
IPv4 source, dest addr
IPv6 header fields
IPv6 source dest addr
UDP/TCP payload
IPv4 payload
IPv6 datagram
IPv4 datagram


logical view:
A
B
IPv4 tunnel
connecting IPv6 routers
E
F
IPv6
IPv6
IPv6
IPv6
physical view:
A
B
C
D
E
F
IPv6
IPv6
IPv4
IPv4
IPv6
IPv6

Routing Algorithms

Global	Decentralized
- All routers have complete topology, link cost info - **Link state** algorithms	- Router knows physically-connected neighbors, link costs to neighbour. - Iterative process of computation, exchange of info with neighbors. - **Distance Vector** algorithms.

Static	Dynamic
- Routes change slowly over time	- Routes change more quickly - Periodic update, in response to link cost changes.

Link state Algorithm

Dijkstra's Algorithm

- Net topology, link costs known to all nodes.
    - Accomplist via link state broadcast
    - All nodes have same info
- Computes least cost paths from one node (source) to all other nodes
    - Gives forward table for that node
- Iterative: after k iterations, knows least cost to k destinations.

Notation:

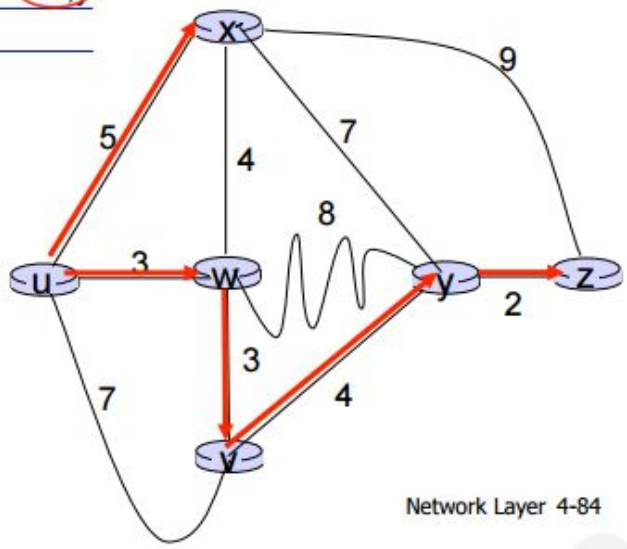
- **c(x,y)**: link cost from x to y. ∞ if not directly connected.
- **D(V)**: current value of cost of path from source to dest. v.
- **p(v):** predecessor node along path from source to v.
- **N':** set of nodes whose least cost path definitively known.

Dijkstra's algorithm: example

Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

notes:

- construct shortest path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)



Algorithm complexity: n nodes

- each iteration: need to check all nodes, w, not in N
- $n(n+1)/2$ comparisons: $O(n^2)$
- more efficient implementations possible: O(nlogn)

Distance Vector Algorithm

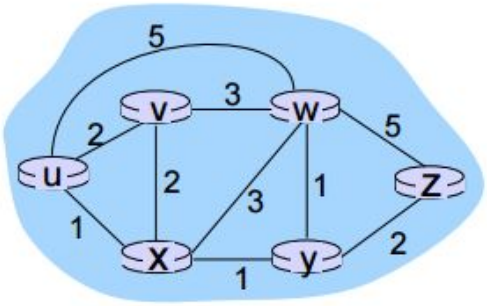
Bellman-Ford equation (dynamic programming)

$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

$d_v(y)$: cost from neighbor v to destination y

$c(x,v)$: cost to neighbor v

$\min_v$: *min* taken over all neighbors v of x



clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$d_u(z) = \min \{ c(u,v) + d_v(z), \; c(u,x) + d_x(z), \; c(u,w) + d_w(z) \}$$
$$= \min \{2 + 5, \; 1 + 3, \; 5 + 3\} = 4$$

node achieving minimum is next hop in shortest path, used in forwarding table

Node x:

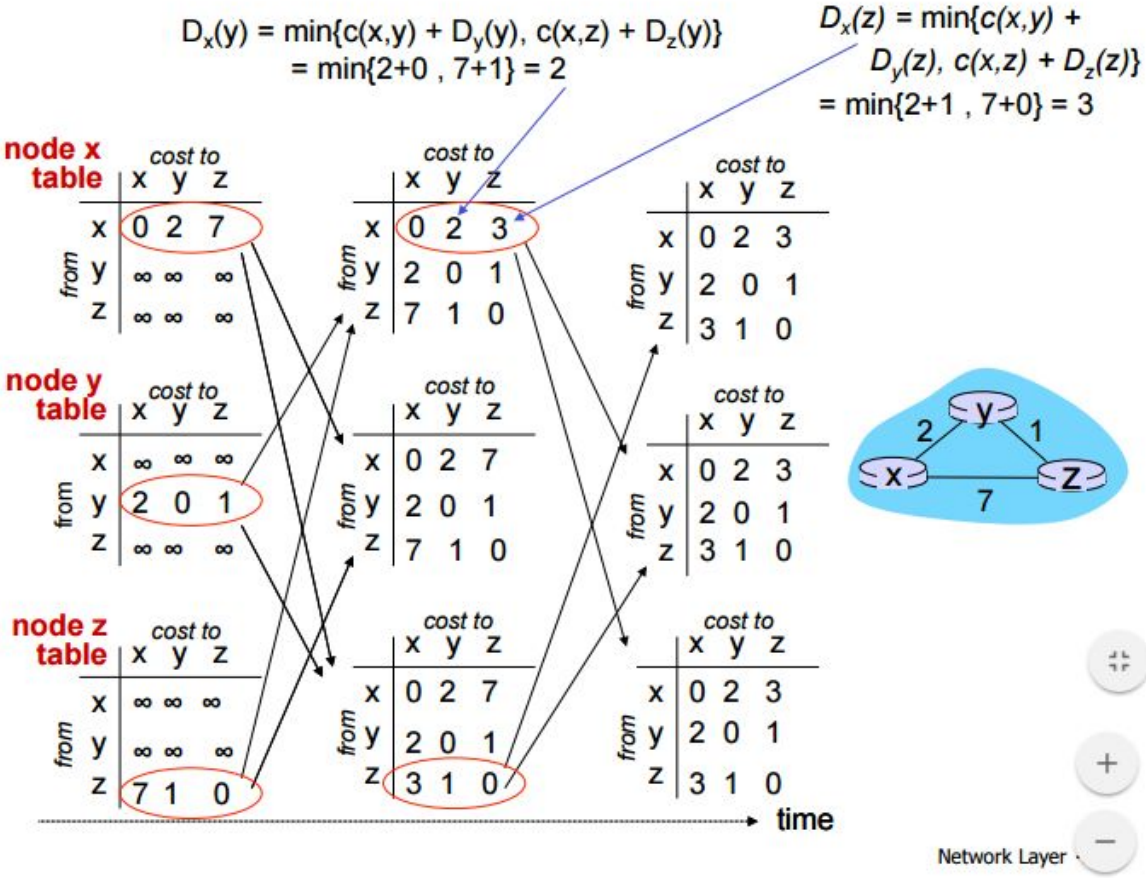
- Knowns cost to each neighbor v: c(x,v)
- Maintains its neighbor's distance vectors.
- For each neighbor v, x maintains $D_v = [D_v(y): y \varepsilon N]$

Key ideas

- From time to time, each node sends its own distance vector estimate to neighbors.
- When x receives new DV estimate from neighbor, it updates its own DV using B-F equation
- Under minor, natural conditions, the estimate $D_x(y)$ converge to the actual least cost $d_x(y)$

Distance vector algorithm is

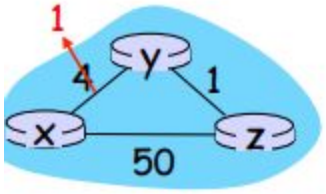
- Iterative, asynchronous: each local iteration caused by:
  - Local link cost change
  - DV update message from neighbor
- Distributed
  - Each node notifies neighbors only when its DV changes. Then neighbors notify their neighbors if necessary.



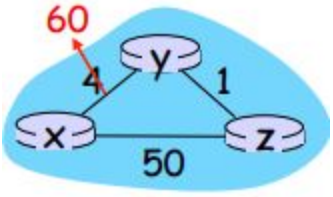
Link cost changes:

- Node detects local link cost change.
- Updates routing info, recalculates distance vector
- If DV changes, notify neighbors.

"Good News travel fast" this would only take 3 iterations to stablize



- Node detects local link cost change.
- bad news travels slow - " count to infinity" problem!
- 44 iterations before algorithm stabilizes.



poisoned reverse:

- If Z routes through Y to get to X
  - Z tells Y its (Z's) distance to X is infinite (so Y won' t route to X via Z)

Link State (Dijkstra's)	**Distance Vector(B-F)**
- Message complexity: with n nodes, E links, O(nE)	- Message complexity: exchange messages between neighbors only. Convergence time varies.
- Speed of Convergence:$O(n^2)$ algorithm requires O(nE) messages. May have oscillations	- Convergence time varies. May be routing loops. Count-to-infinity problem.

- Robustness: what happens if router malfunctions.
  - Node can advertise incorrect link cost
  - Each node computes only its own table.
	- Robustness: what happens if router malfunctions.
	  - DV node can advertise incorrect path cost
	  - Each node's table used by others. Error will propagate through the network.

Hierarchical Routing

- Our routing study thus far is an idealization
- All routers identical
- Network flat

Scale with 600 million destinations:

- Can't store all destination in routing tables
- Routing table exchange swamp links

Administrative autonomy

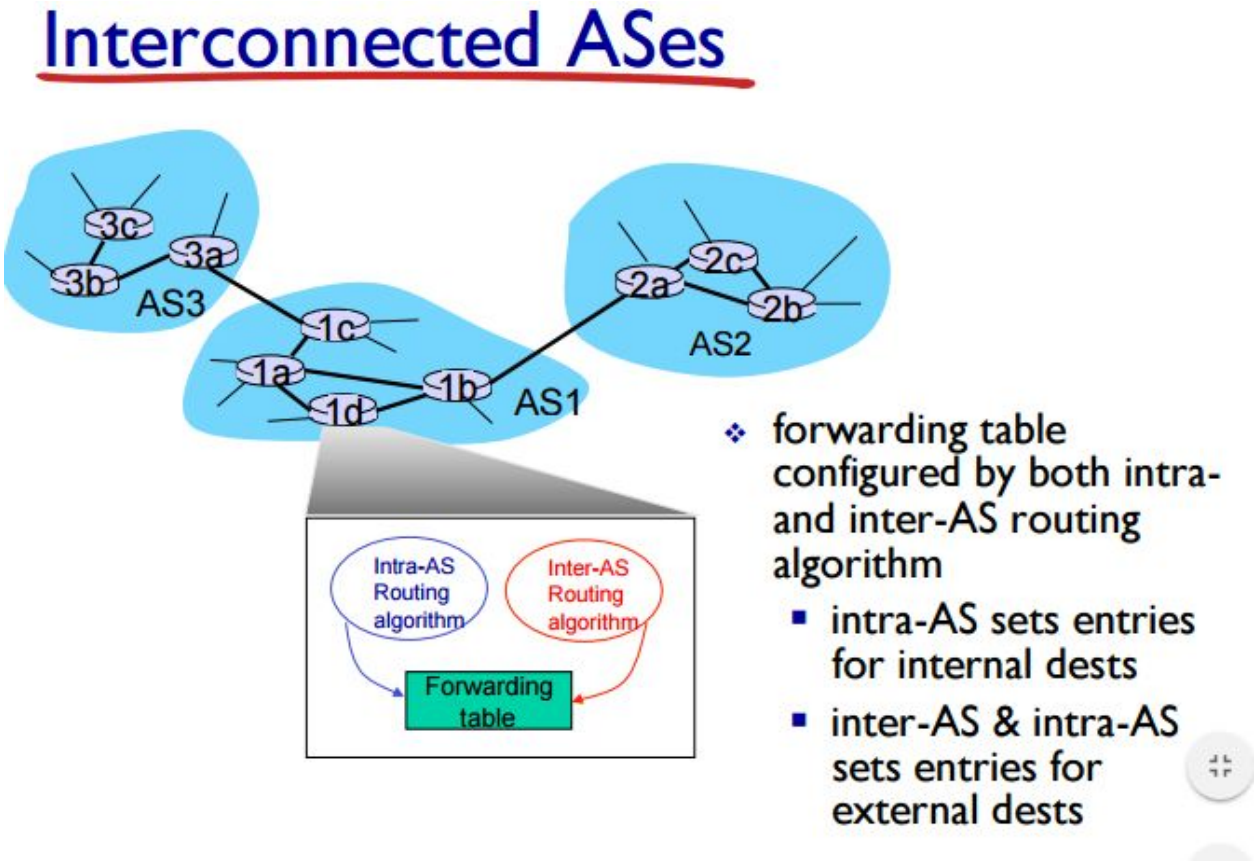
- Internet = network of networks
- Each network admin may want to control routing in its own network.

Aggregate routers into regions, "Autonomous systems" (AS)
Routers in same AS run same routing algorithm

- Intra-AS routing protocol
- Routers in different AS can run different intra-AS routing protocol

Gateway router:

- As "edge" of its own AS
- Has link to router in another AS.

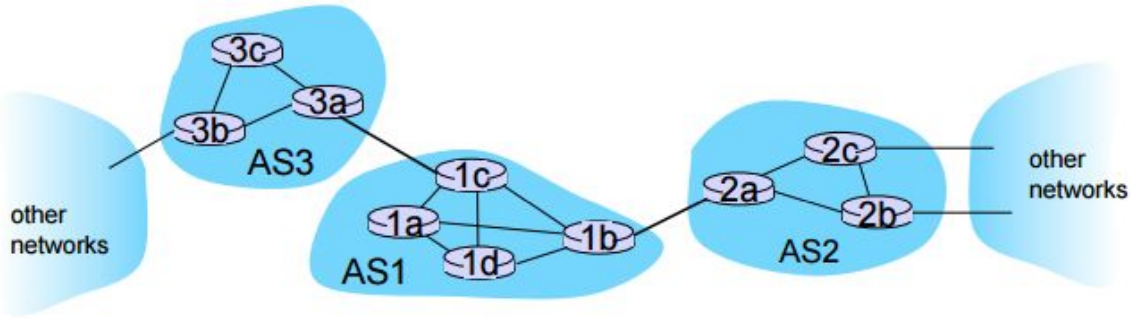


Inter-AS taks

- Suppose router in AS I receives datagram destined outside of AS I
- Router should forward packet to gateway router but which one?

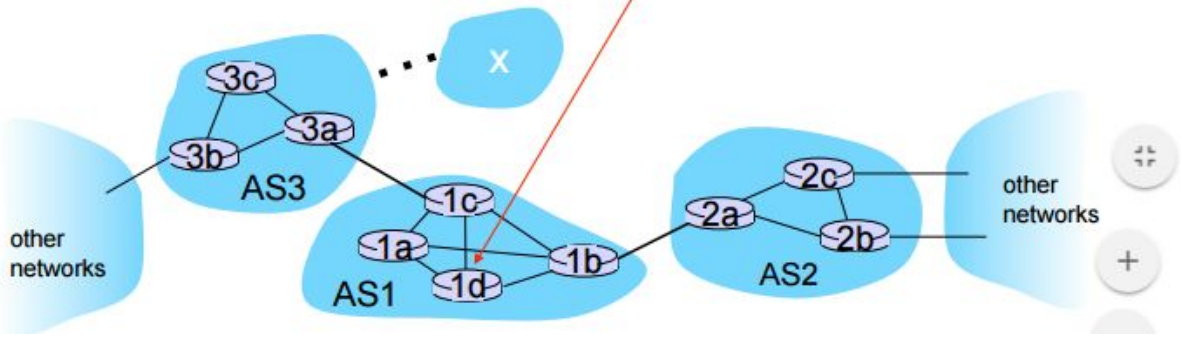
AS I must:

1. Learn which dests are reachable through AS 2, which through AS 3.
2. Propagate this reachability info to all routers in AS I. Job of inter-AS routing.



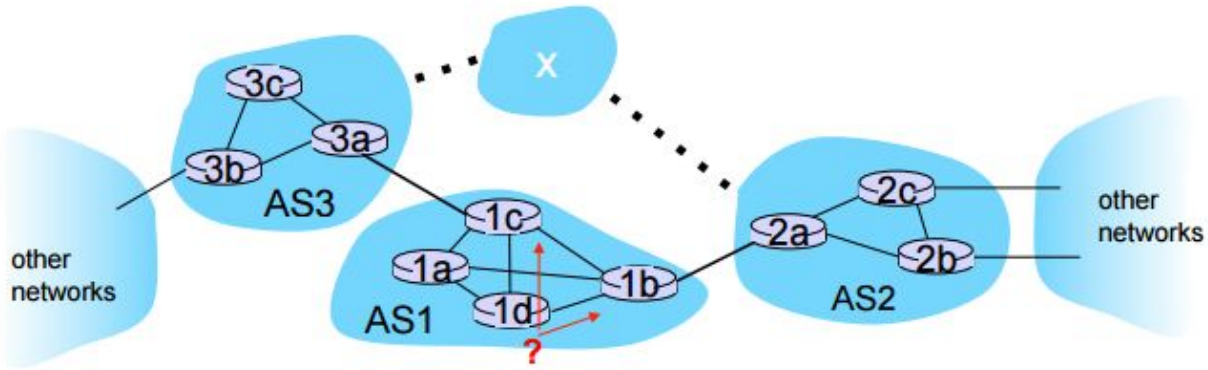
Example: setting forwarding table in router 1d

- suppose AS1 learns (via inter-AS protocol) that subnet *x* reachable via AS3 (gateway 1c), but not via AS2
  - inter-AS protocol propagates reachability info to all internal routers
- router 1d determines from intra-AS routing info that its interface *I* is on the least cost path to 1c
  - installs forwarding table entry *(x,I)*

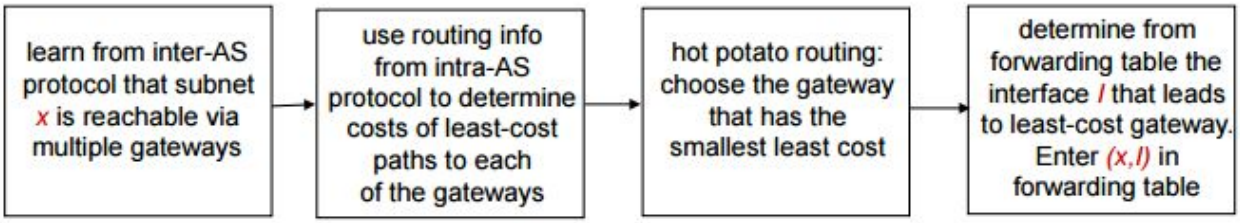


Example: choosing among multiple ASes

- now suppose AS1 learns from inter-AS protocol that subnet *x* is reachable from AS3 *and* from AS2.
- to configure forwarding table, router 1d must determine which gateway it should forward packets towards for dest x
  - this is also job of inter-AS routing protocol!



hot potato routing: send packet towards closest of two routers.



Intra-AS routing also known as interior gateway protocols (IGP)

Most common intra-AS routing protocols:

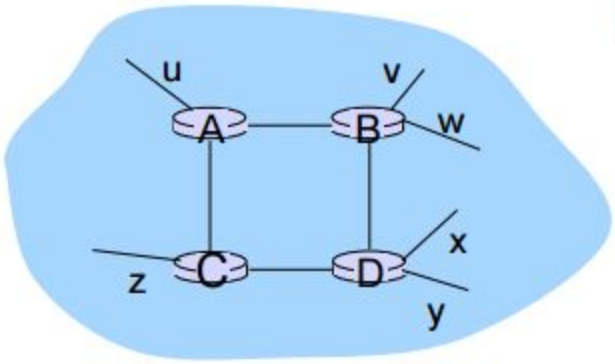
- RIP: Routing Information Protocol
- OSPF: Open Shortest Path First
- IGRP: Interior Gateway Routing Protocol (cisco propreitary)

Routing In the Internet

RIP: Routing Information Protocol

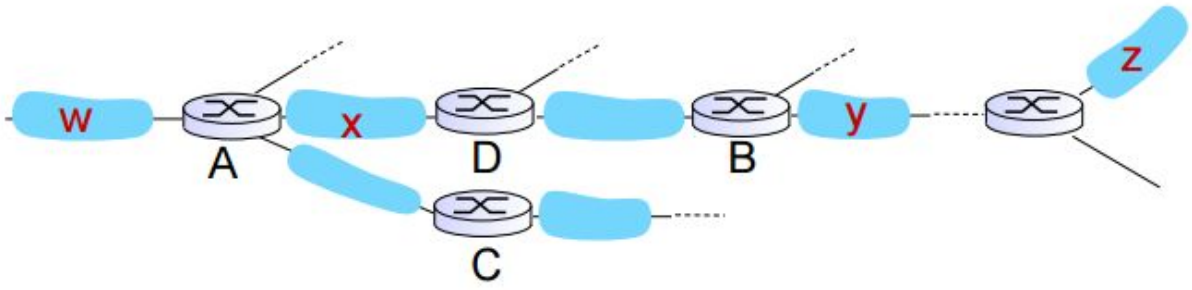
Distance Vector Algorithm

- distance metric: # hops (max = 15 hops), each link has cost 1
- DVs exchanged with neighbors every 30 sec in response message (aka advertisement)
- Each advertisement: list of up to 25 destination subnets (in IP addressing sense)



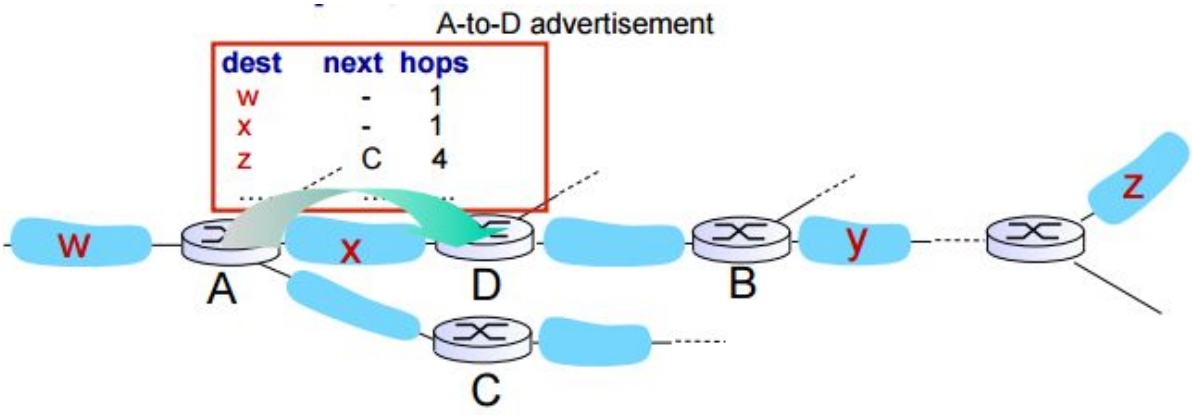
from router A to destination *subnets:*

subnet	hops
u	1
v	2
w	2
x	3
y	3
z	2



routing table in router D

destination subnet	next router	# hops to dest
w	A	2
y	B	2
z	B	7
x	--	1
....	....	....



routing table in router D

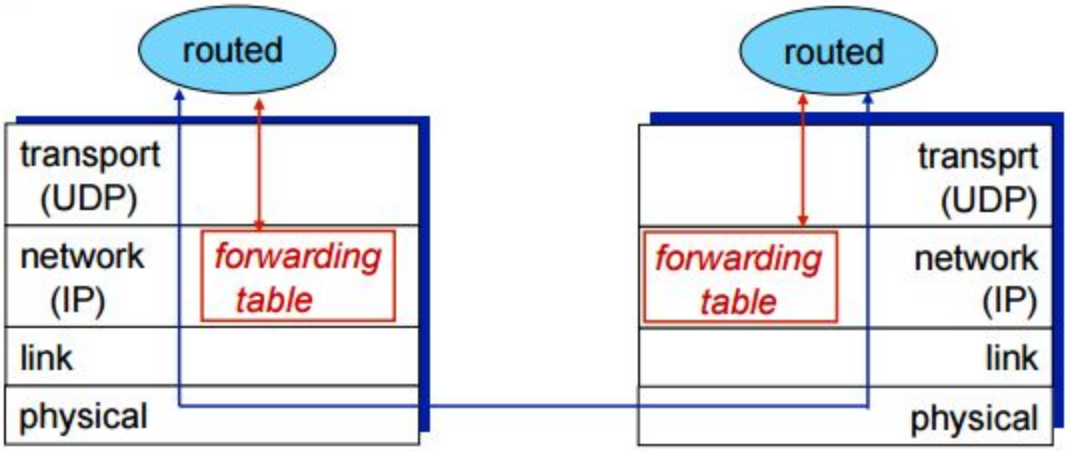
destination subnet	next router	# hops to dest
w	A	2
y	B	2
z	~~B~~ A	~~7~~ 5
x	--	1
....	....	....

RIP: link failure, recovery

If no advertisements heard after 180 seconds then neighbor/link decleared dead

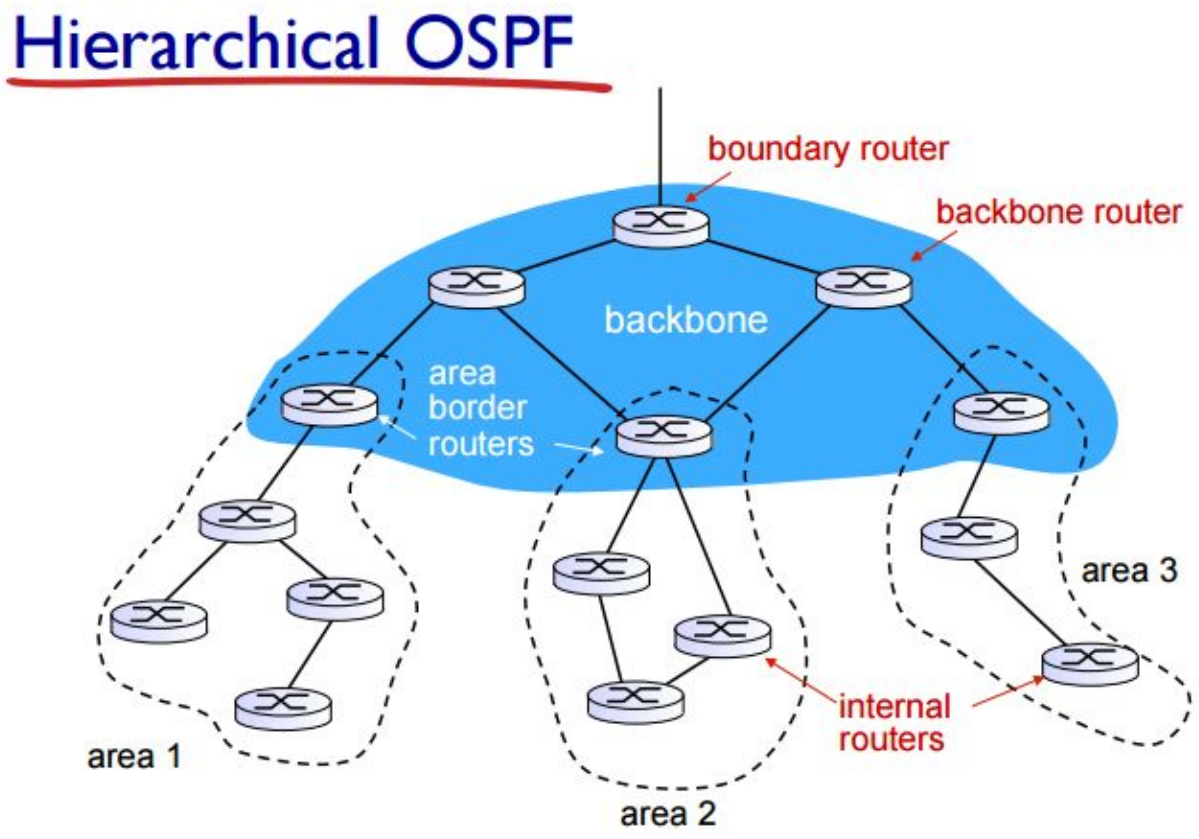
- routes via neighbor invalidated
- new advertisements sent to neighbors
- neighbors in turn send out new advertisements (if tables changed)
- link failure info quickly (?) propagates to entire net
- poison reverse used to prevent ping-pong loops (infinite distance = 16 hops)

- RIP routing tables managed by *application-level* process called route-d (daemon)
- advertisements sent in UDP packets, periodically repeated



OSPF: Open Source Path First

- “open” is publicly available
- Uses Link State algorithm
    - LS packet dissemination
    - Topology map at each node
    - Route compute using Dijkstra’s Algorithm
- OSPF advertisement carries one entry per neighbor
- Advertisements flooded to entire AS
    - Carried in OSPF messages directly over IP(rather than TCP/UDP)
- IS-IS routing protocol: nearly identical to OSPF
- Security: all OSPF messages authenticated(to prevent malicious intrustion)
- Multiple same-cost paths allowed (only one path in RIP)
- For each link, multiple cost metrics for different TOS
- Integrated uni- and multicast support
    - Multiple OSPF (MOSPF) uses same topology data base as OSPF
- Hierarchical OSPF in large domains



- **Two-level hierarchy:** local area, backbone.
    - Link-state advertisements only in area
    - Each nodes has detailed area topology; only know direction (shortest path) to nets in other areas.
- **Area border routers:** “summarize” distances to nets in own area, advertise to other Area Border routers.
- **Backbone routers**: run OSPF routing limited to backbone.
- **Boundary routers:** connect to other AS’ s.

Why different Intra-, Inter-AS routing?

Policy

- Inter-AS: admin wants control over how its traffic routed. Who routes through its net.
- Intra-AS: single admin, so no policy decisions needed

Scale

- Hierarchical routing saves table size, reduced update traffic

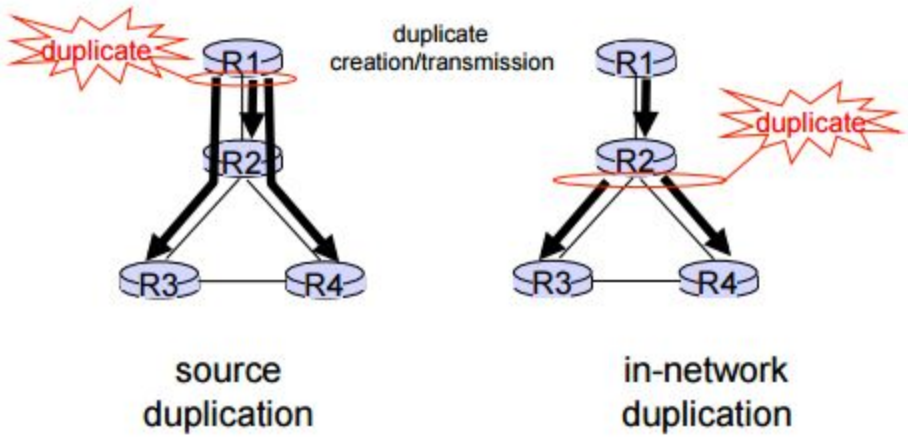
Performance

- Intra-AS: can focus on performance
- Inter-AS: policy may dominate over performance

Broadcast and Multicast Routing

Broadcast routing

- Deliver packets from source to all other nodes
- Source duplication is inefficient.

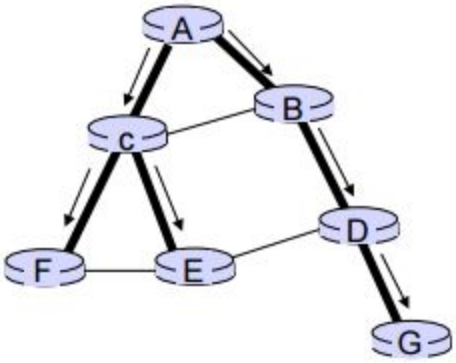


In-Network Duplication

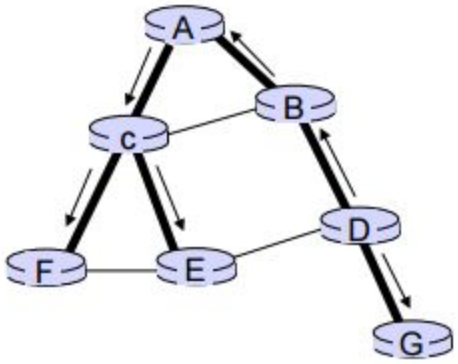
- **Flooding:** when node receives broadcast packet, sends copy to all neighbors
  - Problems: cycles & broadcast storm
- **Controlled Flooding:** node only broadcasts packet if it hasn't broadcasts same packet before
  - Node keeps track of packet ids already broadcasted
  - Or reverse path forwarding (RPF): only forward packet if it arrived on shortest path between node and source
- **Spanning Tree:** no redundant packets received by any nodes

Spanning Tree

- First construct a spanning tree.
- Nodes then forward/make copies only along spanning tree



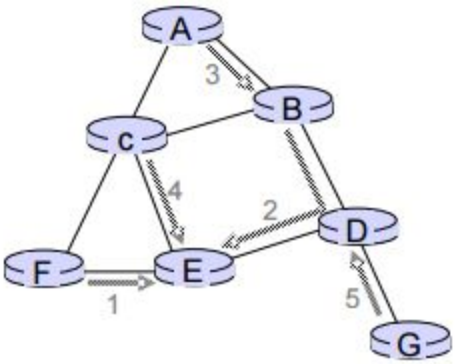
(a) broadcast initiated at A



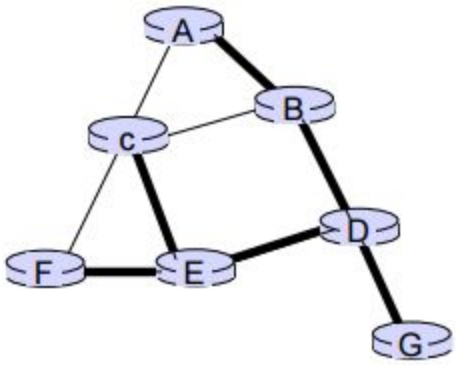
(b) broadcast initiated at D

Creation:

- Center node
- Each node sends unicast join message to center node
    - Messages forwarded until it arrives at a node already belonging to spanning tree.



(a) stepwise construction of spanning tree (center: E)

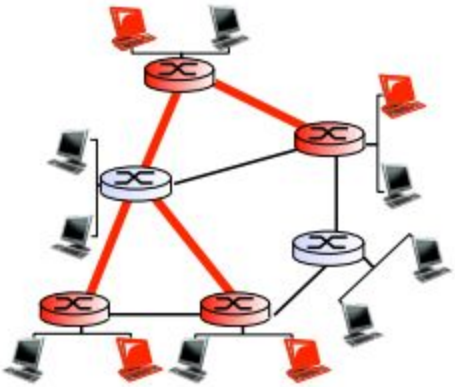


(b) constructed spanning tree

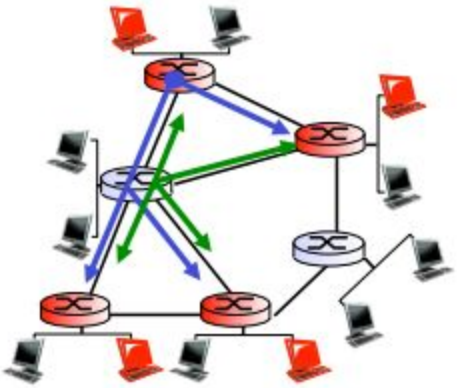
Multicast routing: problem statement

Goal: find a tree(s) connecting routers having local mcast group members

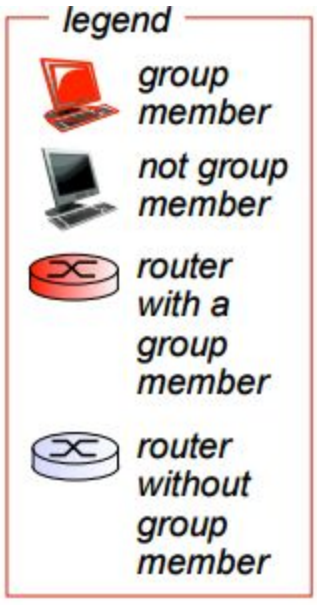
- Tree: not all paths between routers used
- Shared-Tree: same tree used by all group members
- Source based: different tree from each sender to receiver.



shared tree



source-based trees



Chapter 5 - Link Layer

Data Link layer is responsible for transferring the datagram from one node to the physically adjacent node over a link

Frame:
Packets in this layer are called frames
Adds a header and trailer
MAC addresses used in frame headers to id source, destination
    48 bits, hexadecimal format

Services:
Flow control: pacing between adjacent sending/receiving nodes
Error detection: errors from noise, receiver detects errors, signals retransmission or drop frame
Error correction: receiver ids and corrects bit error(s) without retransmit
    This is done at each node to make fewer errors propagated
Half-duplex and Full-duplex: half duplex allows both ends of link to transmit, not at same time

Link Layer in both hardware and software in an “Adaptor”
Adaptor Sending Side: encapsulates datagram in frame, adds error checking bits, rdt, flow control
Receiving Side: check errors, rdt, flow control, extract datagram, passes to upper layer

Error Detection
EDC= error detection and correction bits
D = data protected by error checking, may include header fields

Error detection not 100% reliable, may miss errors rarely, larger EDC field = better detection and correction

Parity Checking
Single bit parity:
Detect only 1 bit error

2d bit parity:
Detect and correct single bit errors

Internet Checksum
Goal: detect errors in packet, only at transport layer
Sender:
Treats as sequence 16-bit ints
1's compliment addition of segment contents
Puts into udp checksum field
Receiver:
Compute checksum, check if equals to field value
    No- error detected
    Yes- none detected may still be there

Cyclic Redundancy Check
More powerful error detection
View data bits, D, as binary num
Choose r+1 bit pattern (generator), G
Goal: choose r CRC bits, R such that

- <D,R> divisible by G (modulo 2) exactly
- Receiver knows G, does division, if non-zero remainder, error detected
- Can detect all burst errors < r+1 bits
- Used in practice

Multiple Access Links and Protocols
Point-to-point

- PPP for dial up
- Point-to-point link between Ethernet switch, host

Broadcast(shared wire or medium)

- old -fashioned Ethernet
- 802.11 wireless LAN

Single shared broadcast channel
Two+ simultaneous transmissions by nodes: interference

- Collision if node receives two+ signals at the same time

Multiple access protocol

- Distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- Communication about channel sharing must use channel itself

MAC protocols: taxonomy
Channel partitioning: divide channel into smaller pieces (time, frequency, code) and allocate piece to node for exclusive use
Random access: channel not divided, allow collisions, recover from collisions
Taking turns: nodes take turns, but nodes with more to send can take longer

Channel Partitioning: TDMA
Time division multiple access
Accesses in rounds, each station gets fixed length slot in each round, unused slots are idle

Channel Partitioning: FDMA
Frequency division multiple access
Channel spectrum divided into frequency bands, each station gets fixed frequency, unused transmission time in frequency bands go idle

Random access protocols
When node sending:

- Transmit at full channel data rate R
- No a priori coordination among nodes

Two+ transmitting nodes = collision
Random access MAC protocols specifies how to detect and recover from collisions

Slotted ALOHA
Assumptions:

- Frames same size
- Time divided in equal size
- Can only transmit at beginning of a slot
- Synchronized nodes
- If 2+ transmit, all nodes detect collision

Operation:

- When node obtains fresh frame, transmit in next slot
    - No collision: node can send new frame next
    - Collision: node retransmits frame in each subsequent slot with prob p until success

Pros:

- Active node can transmit at full R
- Highly decentralized: only slots in nodes need to be in sync

Cons:

- Collisions, wasted slots, idle slots
- Nodes may be able to detect collision in less than time to transmit packet
- Clock synch

Max efficiency = 1/e = .37

Pure ALOHA

When frame first arrives, transmit immediately
Higher collision probability: collides with other frames sent in [t0-1, t0+1]
Max efficiency = 1/(2e) = .18

CSMA, CSMA/CD, CSMA/CA

Carrier Sense Multiple Access
Listen before transmit
If channel idle: transmit frame
If busy: defer transmission
Propagation delay means collisions may still occur
CSMA/CD = CSMA collision detection

- Collisions detected in short time, colliding transmissions aborted

Collision detection:

- Easy in wired LANs: measure signal strengths, compare transmitted, received signals
- Difficult in wireless

Ethernet CSMA/CD Algorithm

1. NIC(Network Interface Card) receives datagram from network layer, creates frame.
2. If NIC senses channel idle, starts frame transmission. If NIC sense channel busy, waits until channel idle, then transmits
3. If NIC transmits entire frame without detecting another transmission, NIC is done with frame.
4. If NIC detects another transmission while transmitting aborts and sends jam signal.
5. After aborting, NIC enters binary(exponential) backoff::
    a. After mth collision, NIC chooses K at random from {0, 1, 2, … 2^m-1} NIC waits K*512 bit times, returns to step 2.
    b. Longer backoff interval with more collisions.

CSMA/CD Efficiency

$\mathbf{T_{prop}}$ = max prop delay between 2 nodes in LAN

$\mathbf{T_{trans}}$ = time to transmit max-size frame

$$efficiency = \frac{1}{1 + 5\frac{t_{prop}}{t_{trans}}}$$

Efficiency goes to 1

- As t_{prop} goes to 0
- As t_{trans} goes to infinity

Better performance than ALOHA: and simple, cheap, decentralized.

"Taking turns" MAC protocols

- Channel partitioning MAC protocols:
  - Share channel efficiently and fairly at high load.
  - Inefficient at low load: delay in channel access, 1/N bandwidth allocated even if only 1 active node.

- Random access MAC protocols
  - Efficient at low load: single node can fully utilize channel
  - High load: collision overhead

  Taking turns protocols looks for the best of both worlds.

Polling:

- Master node "invites" slave nodes to transmit in turns.
- Typically used with "dumb" slave devices.
- Concerns:
  - Polling overhead
  - Latency
  - Single point of failure (master)

Token Passing

- Control token passed from one node to next sequentially
- Token message
- Concerns:
  - Token overhead.
  - Latency
  - Single point of failure (token)

Summary of MAC protocols

- Channel partitioning. By time, frequency or code
    - Time/Frequency division.
- Random access (Dynamic)
    - ALOHA, S-ALOHA, CSMA, CSMA/CD
    - Carrier sensing: easy in some technologies(wire), hard in others(wireless(
    - CSMA/CD used in Ethernet
    - CSMA/CA used in 802.11
- Taking turns
    - Polling from central site, token passing
    - Bluetooth, FDDI (Fiber Distributed Data Interface), Token Ring.

LANs

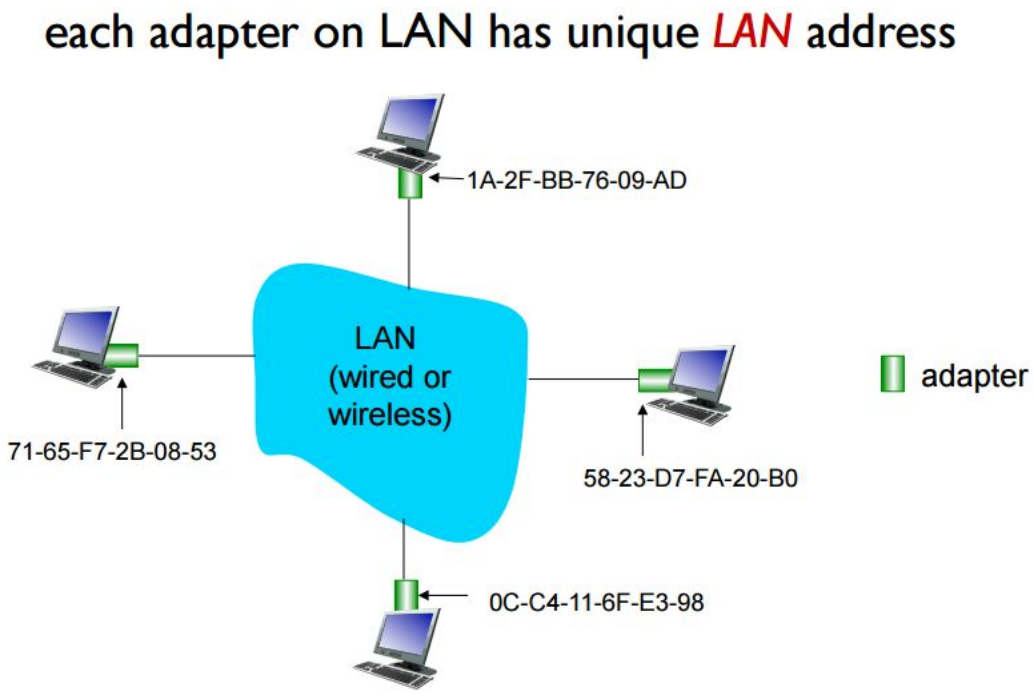
Addressing, ARP

32-bit IP address:

- Network-layer address for interface
- Used for layer 3 forwarding

MAC: Media Access Control Address(LAN, Physical, Ethernet address)

- Function: used locally to get frame from one interface to another physically-connected interface (same network, in IP-addressing sense)
- 48 bit MAC address(for most LANs) burned in NIC ROM,
- E.g. 1A-2F-BB-76-09-AD. Hexadecimal (base 16) notation each number represents 4 bits.

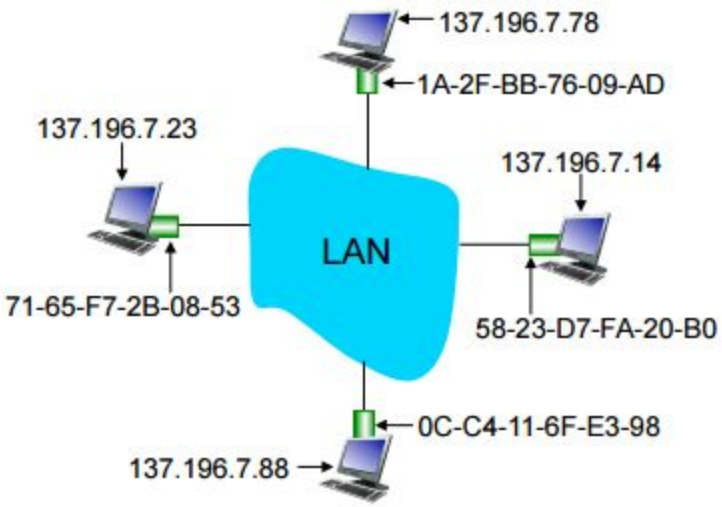


- MAC address allocation administered by IEEE
- Manufacturer buys portion of MAC address to assure uniqueness
- Analogy: MAC address = SSN, IP = Postal Code
- MAC flat address leads to portability
    - Can move LAN card from one LAN to another
- IP hierarchical address not portable
    - Address depends on IP subnet which node is attached

ARP: Address Resolution Protocol

ARP table: each IP node(host, router) on LAN has table

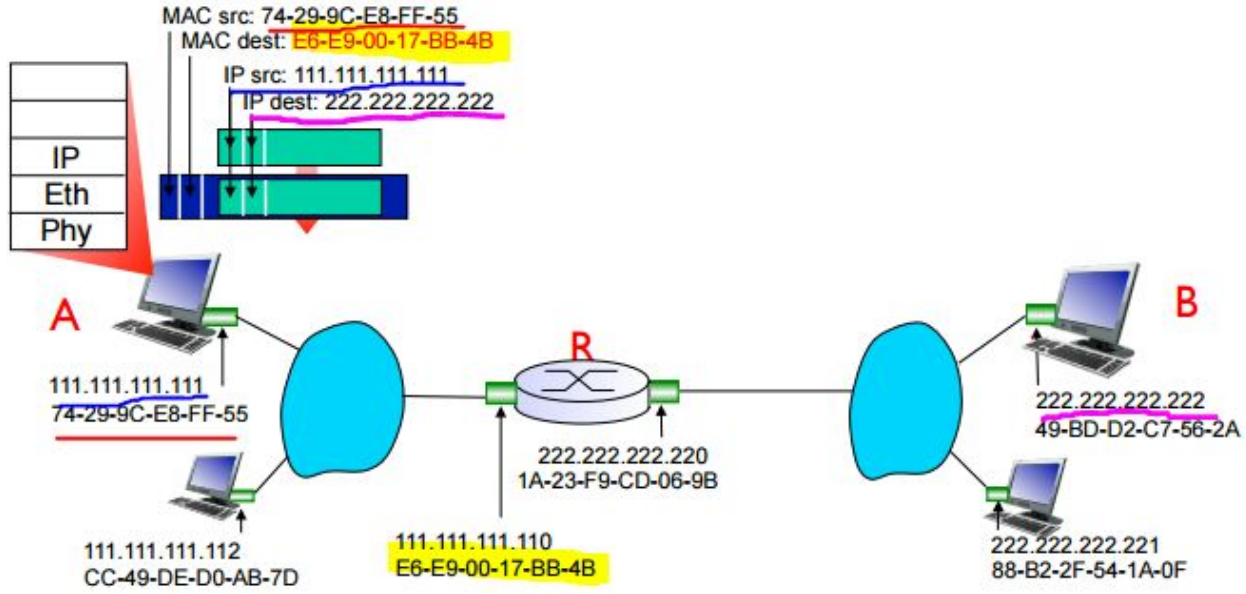
- IP/MAC address mappings for some LAN nodes: <IP address; MAC Address; TTL>
- TTL: time after which address mapping will be forgotten(typically 20 min)



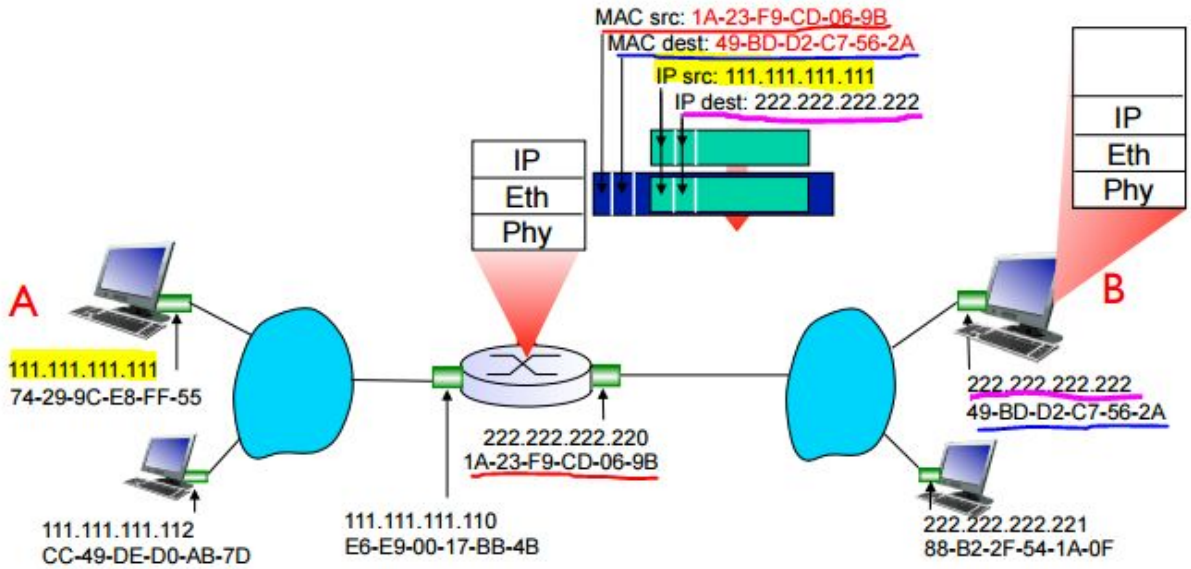
ARP: Same LAN

- A wants to send datagram to B. B's MAC address is not in A's ARP table.
- A broadcasts ARP query packet, containing B's IP address
    - Dest MAC address = FF-FF-FF-FF
    - All nodes on LAN receive ARP query
- B receives ARP packet, replies to A with its MAC address
    - Frame sent to A's MAC address(unicast)
- A Cache (saves) IP-TO-MAC address pair in its ARP table until information expires
    - Soft state: information that times out (goes away) unless refreshed.
- ARP is Plug and Play
    - Nodes create their ARP tables without intervention from net admin.

Addressing: routing to another LAN

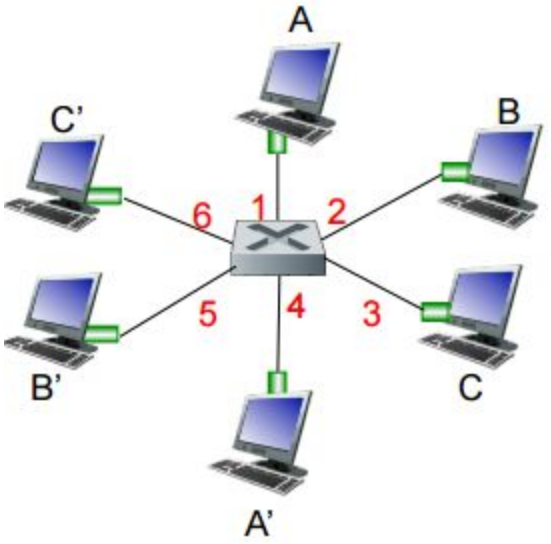


Frame sent from A to R. Frame received at R, datagram removed, passed up to IP
R forwards datagram with IP source A, destination B.
R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram



Switch: Multiple Simultaneous Transmissions

- Hosts have dedicated, direct connection to switch.
- Switches buffer packets
- Ethernet protocol used on each incoming link, but no collisions; full duplex
    - Each link is its own collision domain.
- **Switching:** A-to-A' and B-to-B' can transmit simultaneously, without collisions.



switch with six interfaces (1,2,3,4,5,6)

Q: how does switch know A' reachable via interface 4, B' reachable via interface 5?

A: each switch has a **switch table**, each entry:

- (MAC address of host, interface to reach host, time stamp)
- Looks like a routing table!

Switch: Self-Learning

- Switch **learns** which hosts can be reached through which interfaces
    - When frame received, switch learns location of sender; incoming LAN segment
    - Records sender/location pair in switch table

- Frame Destination, A', location unknown: **flood**
- Destination A location known: selectively send on just one link

MAC addr	interface	TTL
A	*1*	*60*
A'	*4*	*60*

switch table (initially empty)

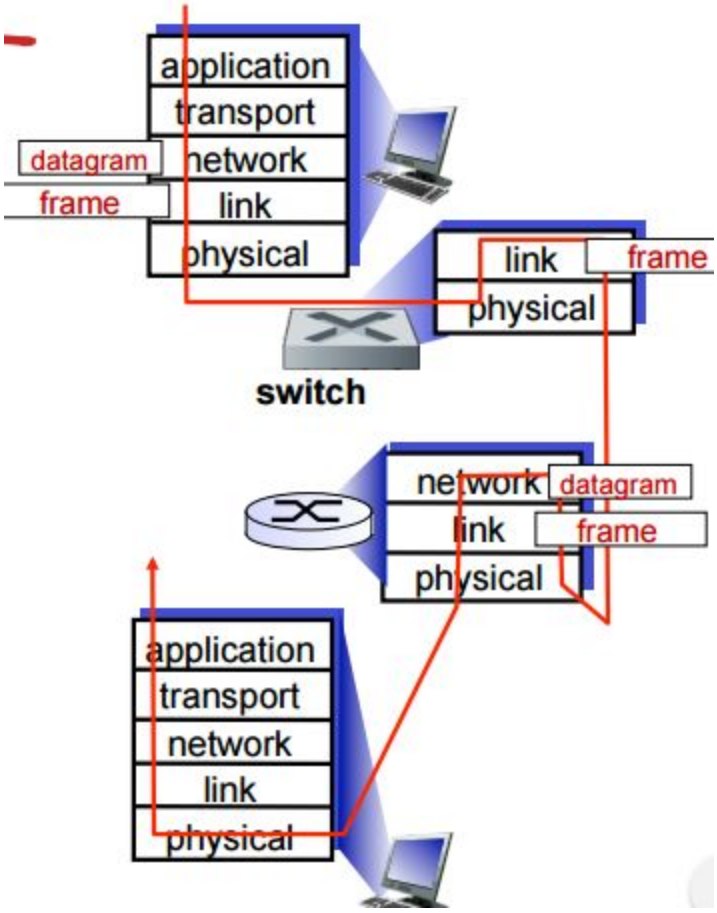
Switches vs. Routers

Both are store and forward

- Routers: network-layer devices (examine network-layer headers)
- Switches: Link-Layer devices (examine link-layer headers)

Both have forwarding tables:

- Routers: compute tables using routing algorithms, IP addresses
- Switches: Learn forwarding table using flooding, learning, MAC addresses.



List Of Acronyms:

DSL - Digital Subscriber Line.
FDM - Frequency division multiplexing
TDM - Time Division Multiplexing.
TCP - Transmission Control Protocol.
UDP - User Datagram Protocol
FTP - File Transfer Protocol
SMTP - Simple Mail Transfer Protocol.
RFC - Request for comments.
SSL - Secure Sockets Layer.
HTTP - HyperText Transfer Protocol
DNS - Domain Name System
POP - Post Office Protocol
IMAP - Internet Mail Access Protocol
SNMP - Simple Network Management Protocol
RDT - Reliable Data Transport
IETF - Internet Engineering Task Force

T.A. programming solution for TCP and UDP client/server
http://1drv.ms/1Qgq0Ry

Aiman Hanna slide's username and pw: iso & osi
http://aimanhanna.com/concordia/index.htm

Question

- Should we know the cable spects example: Slide 21 in Chapter 1 ?
- With the cable internet, people share the wire, is that circuit switching ?
- Slide 48 in Chapter 1, re-explain ?
- Slide 53 in Chapter 1, why the average is the min ? What if we add a third tube ? In the example of file, why the time to receive the file is the d-trans ?
-

How many RTT if the connection is persistent ?

- 1 RTT for initial connection handshake
- 1 RTT for the HTML file
- 1 RTT for all the ref'ed objects

As opposed to a non persistent connection

- 1 RTT for initial connection handshake
- 1 RTT for the request (page or object)
- Repeat for all objects

non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

persistent HTTP:

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

Diff between Connection-oriented and connectionless demux ?

TCP Create socket for each unique combination of Src IP, Src Port, Dest IP and Dest Port
UDP Create socket for each unique Dest Port:

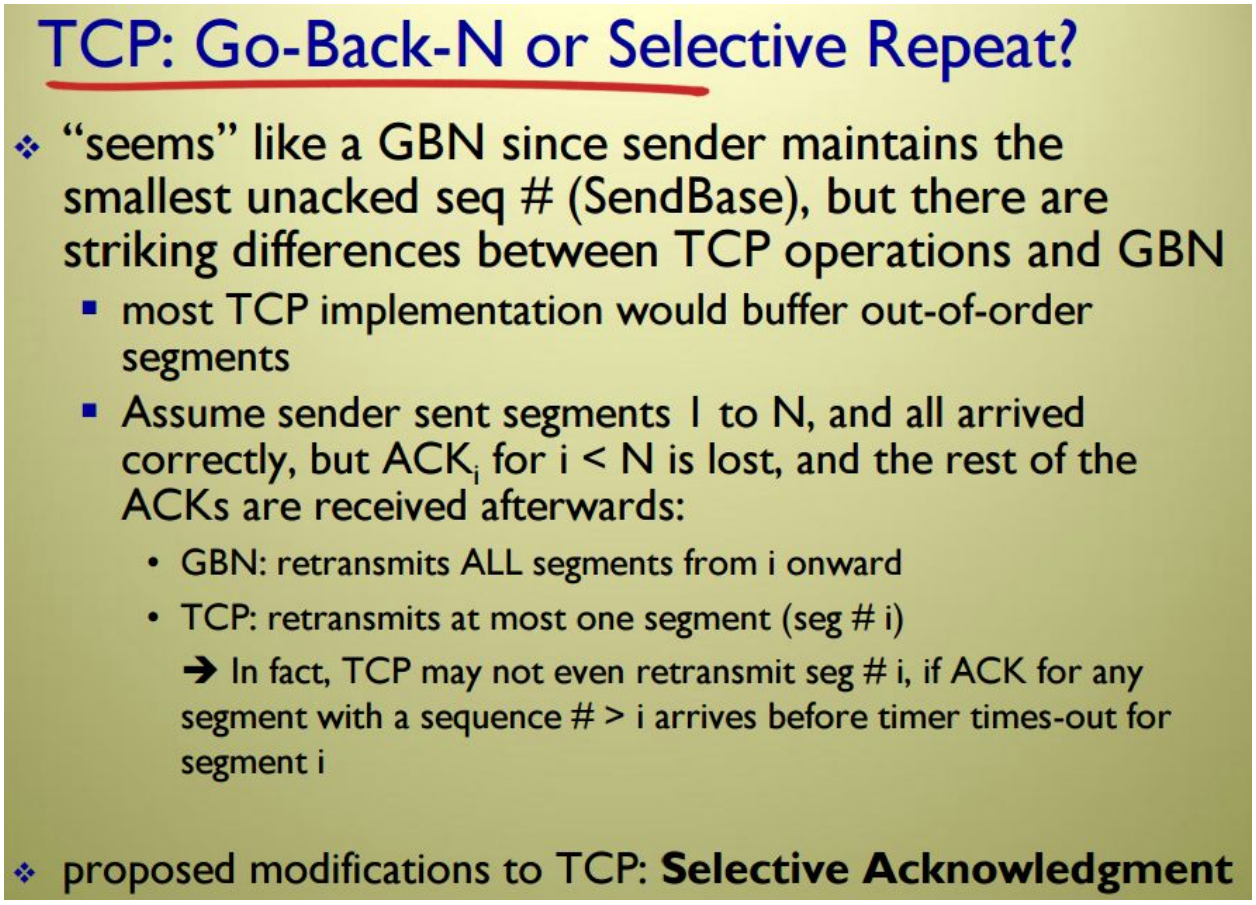
- We don't need a handshake method, so Src IP is not a reason to create new socket
- Any error must be handled by the application in the sender side (Make it reliable).

Questions for professor: -- Office hours 1:15-2:30

1. Do we need to know the formulas for estimatedRTT and DevRTT or any formula based on previous knowledge?

Exam Questions:
Chapter 3 -
Question 1:

Similarities between TCP/Selective Repeat/GO-BACK-N



	GBN	SR	TCP
Sender Window	**Yes**	**Yes**	**Yes**
Receiver Window	**No**	**Yes**	**Yes**
Receiver expected packet	**Yes**	**No**	**No**
Cumulative ACK	**Yes**	**No**	**Yes**
ACK Left+length	**No**	**No**	**Yes**

ACK Left	**Yes**	**No**	**No**
ACK K	**No**	**Yes**	**No**
One timer	**Yes - Resend all unacked**	**No - For each packet**	**Yes - Resend acked value and more if window allows**

Question 2:
Different algorithm of flow control vs congestion control

Question 3
Difference between Reno and Tahoe

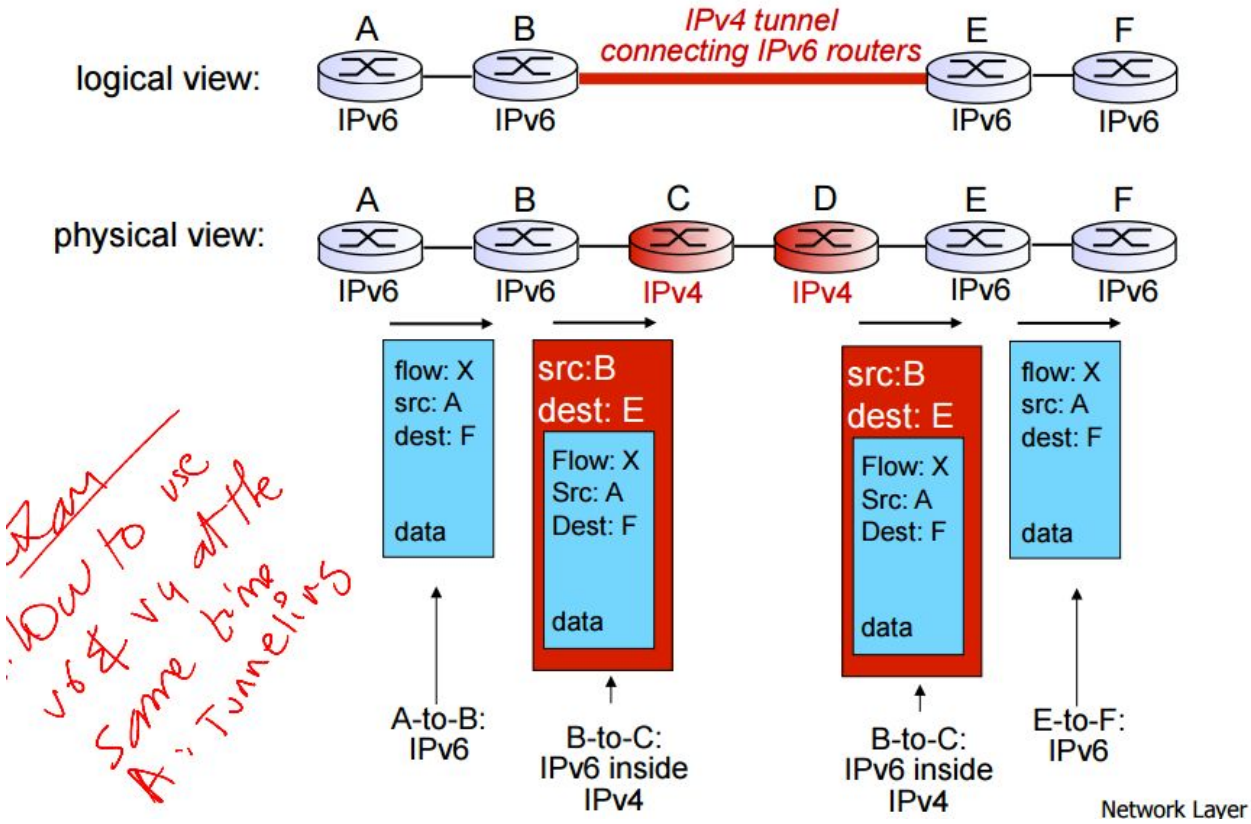
Chapter 4 -

Question 1 -
Difference between routing and forwarding

forwarding: move packets from router's input to appropriate router output

routing: determine route taken by packets from source to dest.

Question 2 -
How to use IPv6 and IPv4 at the same time? Answer: Tunnelling.



Question 3 -
Link State vs. Distance Vector

Global

- All routers have complete topology, link cost info
- **Link state** algorithms

Decentralized

- Router knows physically-connected neighbors, link costs to neighbour.
- Iterative process of computation, exchange of info with neighbors.
- **Distance Vector** algorithms.

Static

- Routes change slowly over time

Dynamic

- Routes change more quickly
- Periodic update, in response to link cost changes.

message complexity

- ***LS:*** with n nodes, E links, O(nE) msgs sent
- ***DV:*** exchange between neighbors only
  - convergence time varies

speed of convergence

- ***LS:*** $O(n^2)$ algorithm requires O (nE) msgs
  - may have oscillations
- ***DV:*** convergence time varies
  - may be routing loops
  - count-to-infinity problem

robustness: what happens if router malfunctions?

LS:

- node can advertise incorrect *link* cost
- each node computes only its *own* table

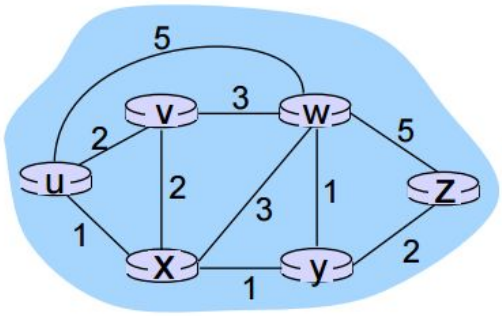
DV:

- DV node can advertise incorrect *path* cost
- each node's table used by others
  - error propagate thru network

Question 4 -

Link State - Dijkstra's algorithm

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



Question 5 -

Hierarchical Routing Information Protocols

Chapter 5 -

Question 1 -
Error Detection

Question 2 -
Cyclic Redundancy Check

Question 3 -
MAC Protocols: Taxonomy

Question 4 -
CSMA/CD - What is it? What layer: Data Link.

Question 5 -
MAC addresses and ARP

Question 6 -
Routers vs Switches.

Switches vs. routers

both are store-and-forward:

- *routers:* network-layer devices (examine network-layer headers)
- *switches:* link-layer devices (examine link-layer headers)

both have forwarding tables:

- *routers:* compute tables using routing algorithms, IP addresses
- *switches:* learn forwarding table using flooding, learning, MAC addresses