COMP 3331 NOTES

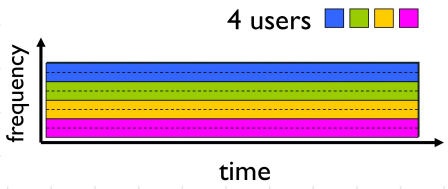
1.3. Network core: Packet/ Circuit Switching

✗ Circuit switch:

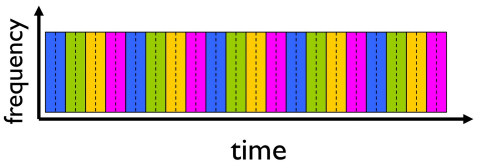
Used in legacy telephone networks.
Used dedicated network. (No sharing)

Con: · Inefficiency due to dedicated network.
· Fixed data rate. (but dynamic usage)
· Connecting state maintenance. (call, received, etc).

Frequency Division Multiplexing (FDM)



Time Division Multiplexing (TDM).

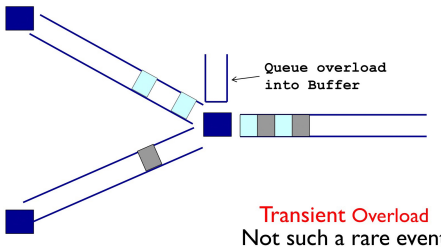


✓ Packet Switch: Host break msg into packets.

- Data is sent as chunks of formatted bits (Packets)
- Packets consist of a "header" and "payload"
- Switches "forward" packets based on their headers
- Each packet travels independently
- No link resources are reserved. Instead, packet switching leverages statistical multiplexing

→ Allow more users to use network. 高利用率.

Pipe view:



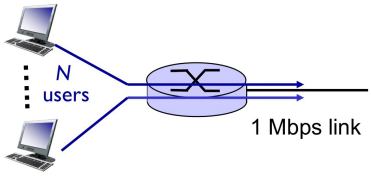
Transient Overload
Not such a rare event

Statistical multiplexing relies on the assumption that not all flows burst at the same time

↳ What's the probability of clashing?

↳ If buffer is full, packet loss.

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



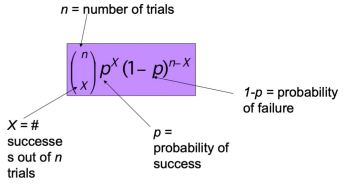
- *circuit-switching:*
  - 10 users
- *packet switching:*
  - with 35 users, probability > 10 active at same time is less than .0004

Q: how did we get value 0.0004?

Q: what happens if > 35 users say 70?

Binomial Distribution

Note the general pattern emerging → if you have only two possible outcomes (call them 1/0 or yes/no or success/failure) in *n* independent trials, then the probability of exactly *X* "successes"=



Since we know Binomial Distribution.

The probability of 10 users are online at the same time:

$$P(\text{User-active} > 10) = \binom{35}{10} (0.1)^{10} (0.9)^{25}$$

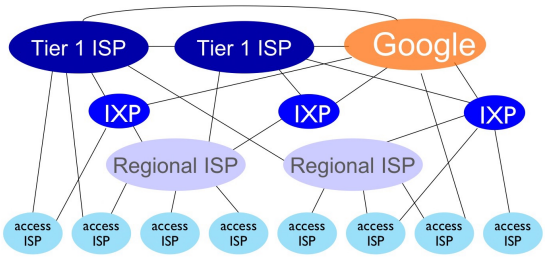
$$= C(35,10)(0.1)^{10}(0.9)^{25}$$

$$= \frac{35!}{10!(35-10)!}(0.1)^{10}(0.9)^{25}$$

$$\approx 0.0004 \quad (0.04\%).$$

$$C(n,r) = \frac{n!}{r!(n-r)!}$$

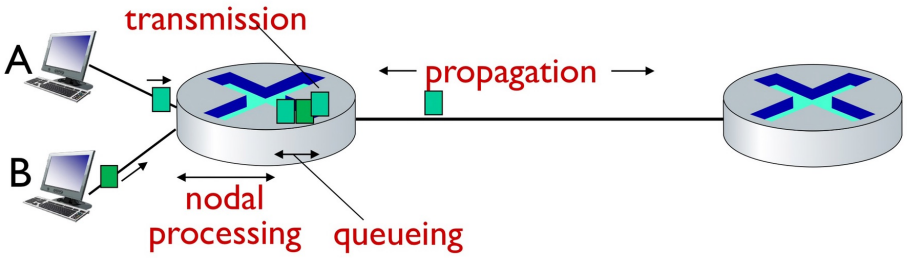
Network Core: Internet Structure.



At "center": small # of well-connected large networks

- "tier-1" commercial ISPs (e.g., Level 3, Sprint, AT&T, NTT), national & international coverage
- content provider networks (e.g., Google, Facebook): private network that connects its data centers to Internet, often bypassing tier-1, regional ISPs

1.4. Network Performance: Loss, Delay, Throughput.



$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

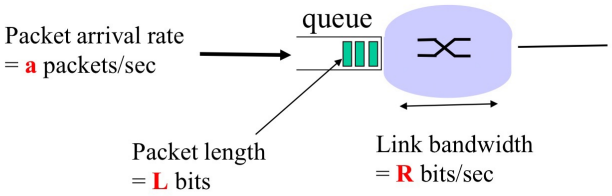
d_{proc}: Nodal Process: { Check bit errors; Determine output link.

- d_{queue}: Queueing Delay

d_{trans}: Transmission Delay: $\frac{\text{Packet Length}}{\text{Transmission Rate}}$ (Time for SENDING)

d_{prop}: Propagation Delay: $\frac{\text{Length of Physical Link}}{\text{Propagation Speed}}$ (3×10^8 m/s) (Travel Time).

→ Queueing Delay:

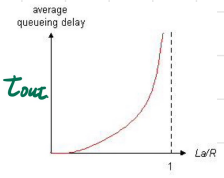


- Every second: aL bits arrive to queue
- Every second: R bits leave the router
- Question: what happens if aL > R ?
- Answer: queue will fill up, and packets will get dropped!!

aL/R is called traffic intensity

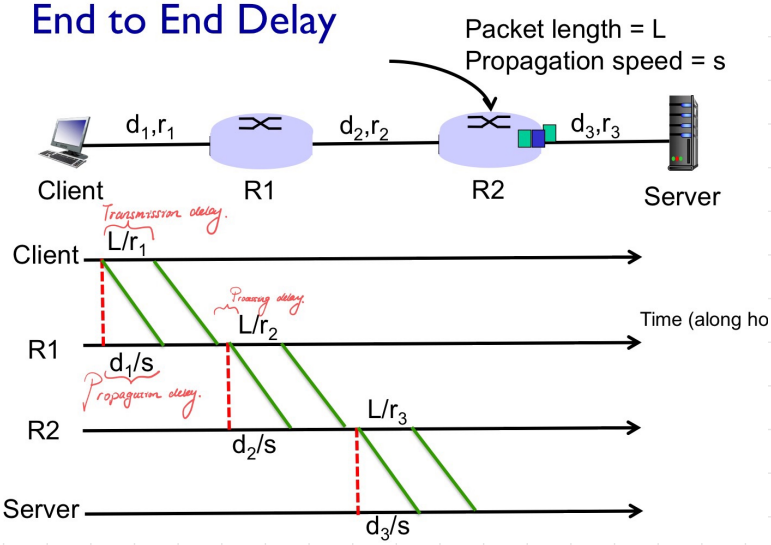
$\frac{aL}{R} = \frac{\text{Rate}_{input}}{\text{Rate}_{output}}$

- La/R ~ 0: avg. queueing delay small
- La/R -> 1: delays become large $t_{in} = t_{out}$
- La/R > 1: more "work" than can be serviced, average delay infinite! (this is when a is random!)



End to End Delay:

End to End Delay



Sum of all chodal on the path.

Traceroute: Send probes first through 3 diff path. get the speed/connectivity.

traceroute: gaia.cs.umass.edu to www.eurecom.fr

3 delay measurements from gaia.cs.umass.edu to cs-gw.cs.umass.edu

```
1  cs-gw (128.119.240.254)  1 ms  1 ms  2 ms
2  border1-rt-fa5-1-0.gw.umass.edu (128.119.3.145)  1 ms  1 ms  2 ms
3  cht-vbns.gw.umass.edu (128.119.3.130)  6 ms 5 ms 5 ms
4  jn1-at1-0-0-19.wor.vbns.net (204.147.132.129)  16 ms 11 ms 13 ms
5  jn1-so7-0-0-0.wae.vbns.net (204.147.136.136)  21 ms 18 ms 18 ms
6  abilene-vbns.abilene.ucaid.edu (198.32.11.9)  22 ms  18 ms  22 ms
7  nycm-wash.abilene.ucaid.edu (198.32.8.46)  22 ms  22 ms  22 ms
8  62.40.103.253 (62.40.103.253)  104 ms 109 ms 106 ms
9  de2-1.de1.de.geant.net (62.40.96.129)  109 ms 102 ms 104 ms
10  de.fr1.fr.geant.net (62.40.96.50)  113 ms 121 ms 114 ms
11  renater-gw.fr1.fr.geant.net (62.40.103.54)  112 ms  114 ms  112 ms
12  nio-n2.cssi.renater.fr (193.51.206.13)  111 ms  114 ms  116 ms
13  nice.cssi.renater.fr (195.220.98.102)  123 ms  125 ms  124 ms
14  r3t2-nice.cssi.renater.fr (195.220.98.110)  126 ms  126 ms  124 ms
15  eurecom-valbonne.r3t2.ft.net (193.48.50.54)  135 ms  128 ms  133 ms
16  194.214.211.25 (194.214.211.25)  126 ms  128 ms  126 ms
17  * * *
18  * * *
19  fantasia.eurecom.fr (193.55.113.142)  132 ms  128 ms  136 ms
```

3 delay measurements to border1-rt-fa5-1-0.gw.umass.edu

trans-oceanic link

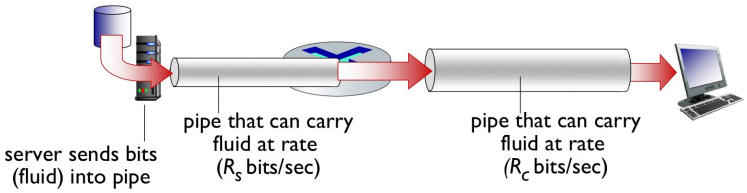
looks like delays *decrease*! Why?

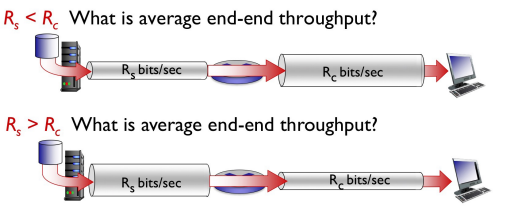
* means no response (probe lost, router not replying)

Throughput.

Throughput Bandwidth

- *throughput:* rate (bits/time unit) at which bits are being sent from sender to receiver
  - *instantaneous:* rate at given point in time
  - *average:* rate over longer period of time → Depends on min speed.





106

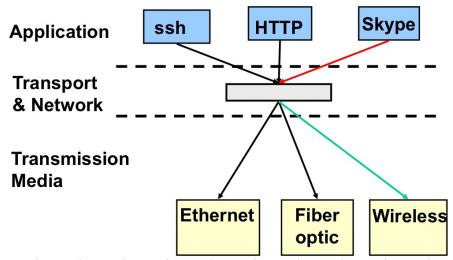
bottleneck link

link on end-end path that constrains end-end throughput

1.5. Protocol layers, service model.

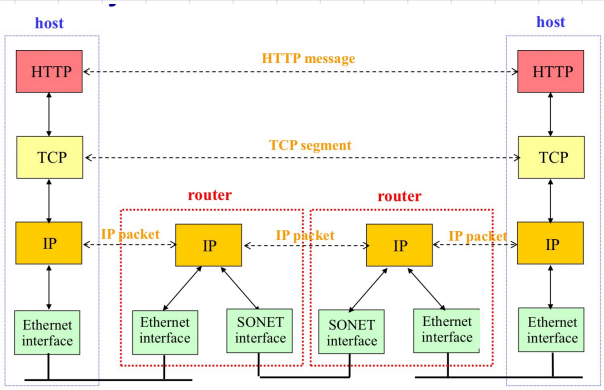
An Example: Benefit of Layering

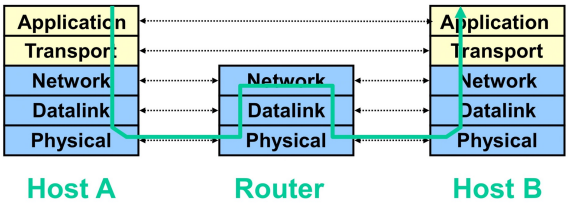
- Introducing an intermediate layer provides a common abstraction for various network technologies



Is Layering Harmful? Disadvantage.

- Layer N may duplicate lower-level functionality
  - E.g., error recovery to retransmit lost data
- Information hiding may hurt performance
  - E.g., packet loss due to corruption vs. congestion
- Headers start to get large
  - E.g., typically, TCP + IP + Ethernet headers add up to 54 bytes
- Layer violations when the gains too great to resist
  - E.g., Network Address Translation (NAT – to be covered in Network Layer)
- Layer violations when network doesn't trust ends
  - E.g., Firewalls (Security)





2. APPLICATION LAYER

2.1 Principles of Network Application.

TCP vs. UDP:

TCP service:

- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security
- *connection-oriented:* setup required between client and server processes

UDP service:

- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

Q: why bother? *Why* is there a UDP? *Fast.*

2.2 Web and HTTP.

Stateless:

HTTP uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains *no* information about past client requests

aside

protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

Maintain state by Cookie.

Maintaining user/server state: cookies

Web sites and client browser use *cookies* to maintain some state between transactions

four components:

1) cookie header line of HTTP *response* message
2) cookie header line in next HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

Example:

- Susan uses browser on laptop, visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID (aka "cookie")
  - entry in backend database for ID
- subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to "identify" Susan

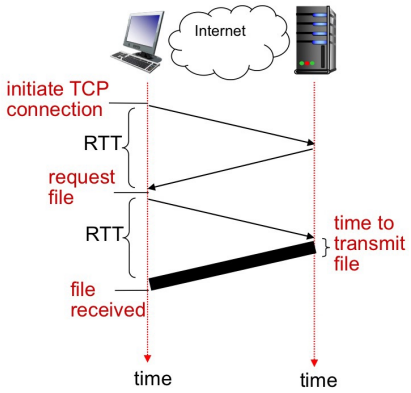
Non-persistend HTTP : At most one object sent over TCP connection each time.

Non-persistent HTTP: response time

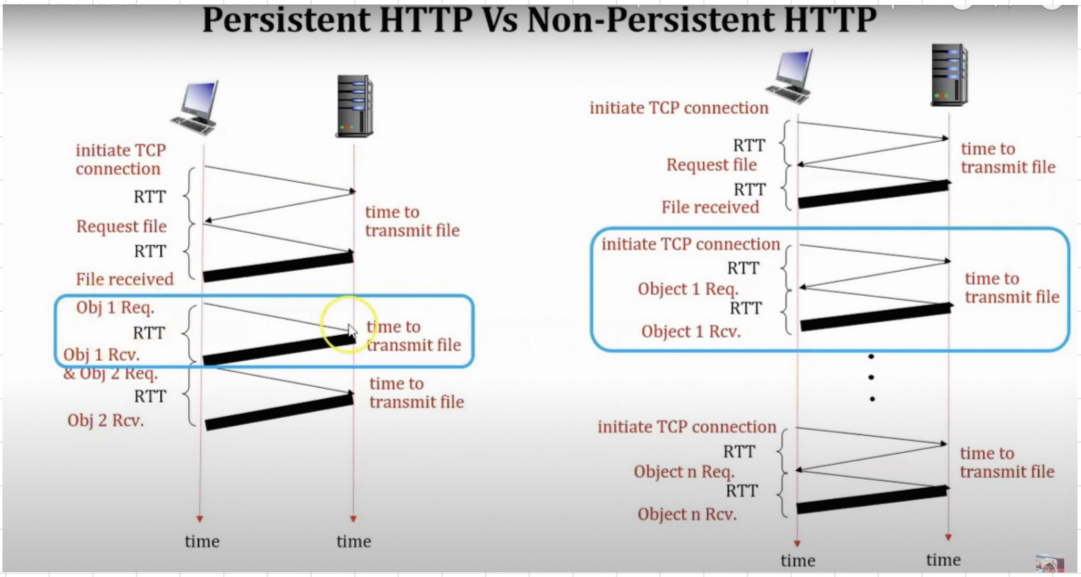
RTT (definition): time for a small packet to travel from client to server and back

HTTP response time:

- one RTT to initiate TCP connection (approximate 3-way handshake)
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time
- non-persistent HTTP response time = 2RTT+ file transmission time



Persistent vs. Non-persistent.



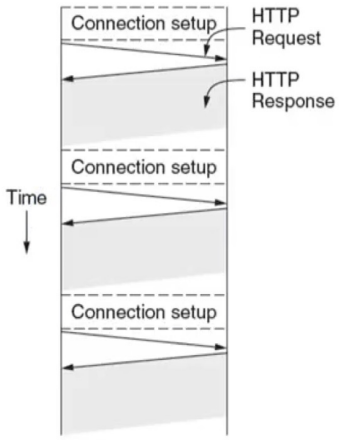
No need to establish every time.

Establish every time again.

HTTP 1.0

HTTP/1.0

- Non-Persistent: One TCP connection to fetch one web resource
- Fairly poor PLT
- 2 Scenarios
  - Multiple TCP connections setups to the same server
  - Sequential request/responses even when resources are located on different servers
- Multiple TCP slow-start phases (more in lecture on TCP)



HTTP 1.1 / Persistent HTTP

Persistent HTTP (HTTP/1.1)

Persistent HTTP

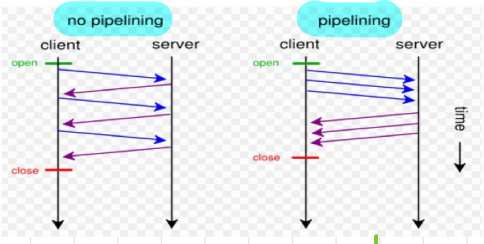
- server leaves TCP connection open after sending response
- subsequent HTTP messages between same client/server are sent over the same TCP connection
- Allow TCP to learn more accurate RTT estimate (APPARENT LATER IN THE COURSE)
- Allow TCP congestion window to increase (APPARENT LATER)
- i.e., leverage previously discovered bandwidth (APPARENT LATER)

Persistent without pipelining:

- client issues new request only when previous response has been received
- one RTT for each referenced object

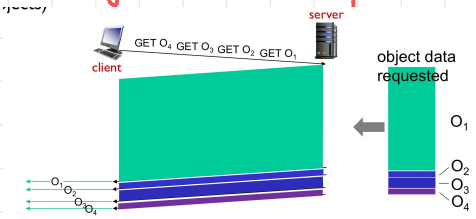
Persistent with pipelining:

- introduced in HTTP/1.1
- client sends requests as soon as it encounters a referenced object WHAT IS A REF OBJ?
- as little as one RTT for all the referenced objects



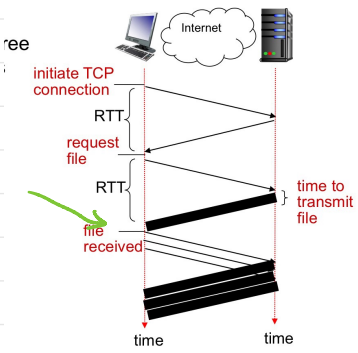
Con:

- First-in-first-serve → small object wait long time behind big object.



objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

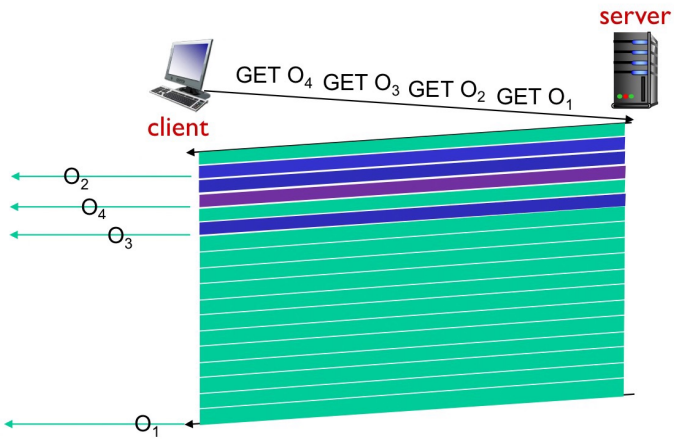
Ref obj.



- Loss recovery stalls object transmission.

HTTP 2 : Increased flexibility / decreased delay.

HTTP/2: objects divided into frames, frame transmission interleaved



unordered.

O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

2.3 Electronic Mail (SMTP, IMAP)

Simple Mail Transfer Protocol (Send only)

Internet Mail Access Protocol (Retrival, deletion, storage)

SMTP: closing observations

comparison with HTTP:

- HTTP: pull
- SMTP: push send email.
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response message
- SMTP: multiple objects sent in multipart message

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF.CRLF to determine end of message

2.4. DNS (Domain Name System)

DNS services

- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing @unsw.edu.au
- load distribution ← DNS ← Diff. IP addresses.
  - replicated Web servers: many IP addresses correspond to one name

Q: Why not centralize DNS?

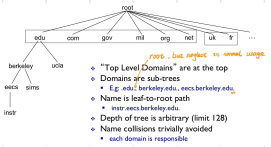
- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: doesn't scale!

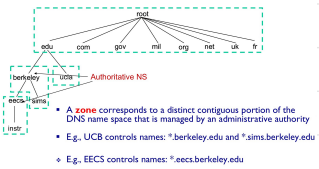
- Comcast DNS servers alone: 600B DNS queries per day

3 Intertwined Hierarchy

- Hierarchical Namespace.

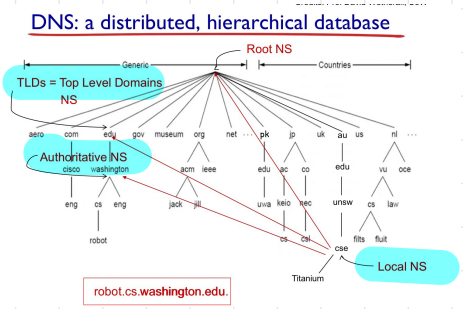


- Hierarchically administered.



- Distributed hierarchy of servers.

Explained below:



Top Level Domain (TLD) Servers:

Top-Level Domain (TLD) servers:

- responsible for .com, .org, .net, .edu, .aero, .jobs, .museums, and all top-level country domains, e.g.: .cn, .uk, .fr, .ca, .jp
- Network Solutions: authoritative registry for .com, .net TLD
- Educause: .edu TLD

Authoritative DNS Server:

Authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

Local DNS Server:

Local DNS name servers

ASK LOCAL SERVER FIRST, IF NO RESULT, THEN ASK ROOT.

- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- Hosts learn about the local DNS server via a host configuration protocol (e.g., DHCP)
- Client application
  - Obtain hostname (e.g., from URL)
  - Do **gethostbyname()** to trigger DNS request to its local DNS server
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

16

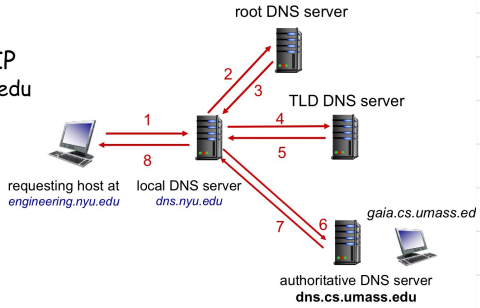
DNS Solutions

Iterated Query (More case).

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

Iterated query:

- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"

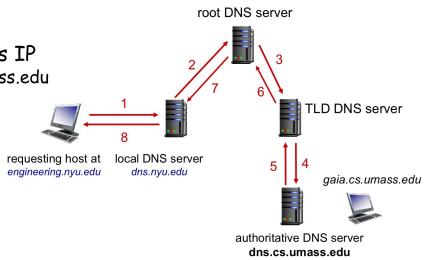


Recursive Query (Rarely used).

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

Recursive query:

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?



18

DNS Record

DNS records

DNS: distributed database storing resource records (RR)

RR format: (`name, value, type, ttl`)

type=A

- `name` is hostname
- `value` is IP address

type=NS

- `name` is domain (e.g., foo.com)
- `value` is hostname of authoritative name server for this domain

type=CNAME

- `name` is alias name for some "canonical" (the real) name
- www.ibm.com is really servereast.backup2.ibm.com
- `value` is canonical name

real

type=MX

- `value` is name of mailserver associated with `name`

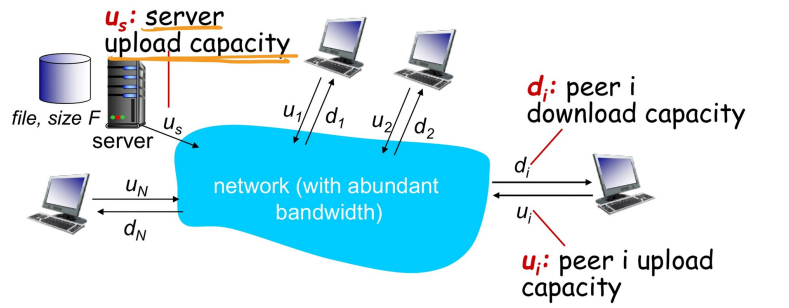
2.5. P2P Application.

Pro: Self scalability - new peers: < new capacity / new element.

Con: · Not always on
· Complex management.

Example: Bit Torren, Bitcoin.

Time cost between client-service and P2P:



File size: F

Server-client: Must sequentially upload N files (N is user request).

Server upload one copy: F/u_s

Server upload N copies: NF/u_s

Client slowest download rate: F/d_{min} (All users download simultaneously)

Hence, Time cost = $\max\{NF/u_s, F/d_{min}\}$.

P2P: Must upload at lease one copy:

Server upload one copy: F/u_s

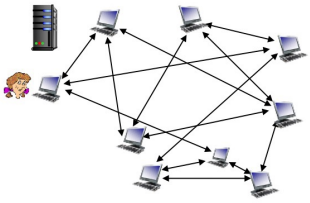
Client slowest download rate: F/d_{min}

☆ Client need to become server, max upload rate: $u_s + \sum u_i$

Time cost = $\max\{F/u_s, F/d_{min}, NF/(u_s+\sum u_i)\}$

Bit Torrent:

P2P file distribution: BitTorrent 种子 传输



- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

Requesting chunks:

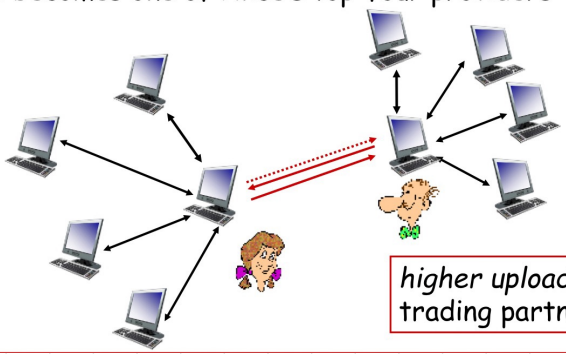
- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first (why?)

Sending chunks: tit-for-tat

- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4

BitTorrent: tit-for-tat

(1) Alice "optimistically unchokes" Bob
(2) Alice becomes one of Bob's top-four providers; Bob reciprocates
(3) Bob becomes one of Alice's top-four providers



higher upload rate: find better trading partners, get file faster !

2.6. CDNs

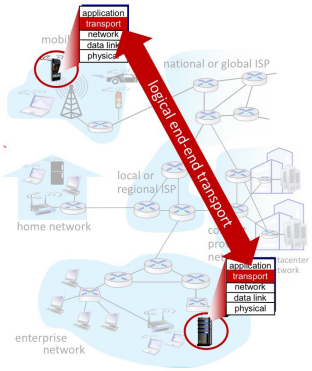
2.7. Socket Programming with UDP and TCP.

3. Transport Layer. (Final Exam Content Score).

3.1. Transport-Layer Service.

Two principal Internet transport protocols

- **TCP:** Transmission Control Protocol
  - reliable, in-order delivery
  - congestion control
  - flow control — Adjust speed rate based on receiver's capacity.
  - connection setup
- **UDP:** User Datagram Protocol
  - unreliable, unordered delivery
  - no-frills extension of "best-effort" IP
- services not available:
  - delay guarantees
  - bandwidth guarantees

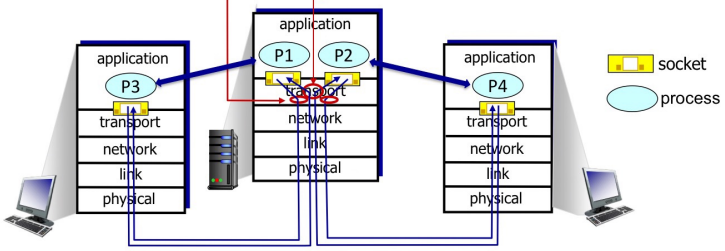


3.2. Multiplexing and Demultiplexing.

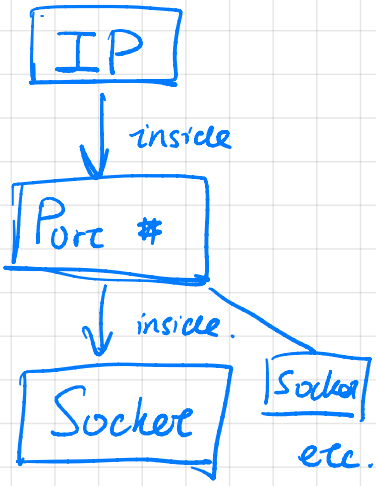
Multiplexing/demultiplexing

multiplexing at sender:
handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:
use header info to deliver received segments to correct socket



Note: The network is a shared resource. It does not care about your applications, sockets, etc.



Hosts use IP address and port numbers for direction.

UDP vs. TCP

Connectionless demultiplexing (UDP)

Recall:

- when creating socket, either specify *host-local* port # (or let OS pick random available port):

```
DatagramSocket mySocket1 = new
DatagramSocket(12534);
```

- when creating datagram to send into UDP socket, must specify
  - destination IP address
  - destination port #

MATOY DIFF.

when receiving host receives *UDP* segment:

- checks destination port # in segment
- directs UDP segment to socket with that port #

IP/UDP datagrams with *same dest. port #,* but different source IP addresses and/or source port numbers will be directed to *same socket* at receiving host

Regardless of source port, into same socket.

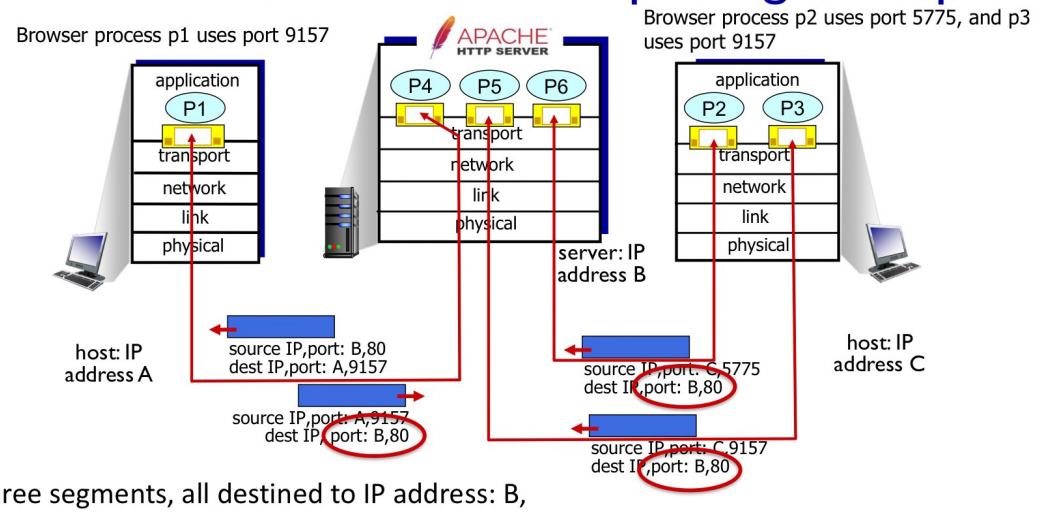
Connection-oriented demultiplexing (TCP)

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses *all four values (4-tuple)* to direct segment to appropriate socket

- server may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
  - each socket associated with a different connecting client

1 socket ↔ 1 client/application.

Connection-oriented demultiplexing: example



Three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

3.3. Connectionless Transport: UDP. (User Datagram Protocol).

UDP: User Datagram Protocol

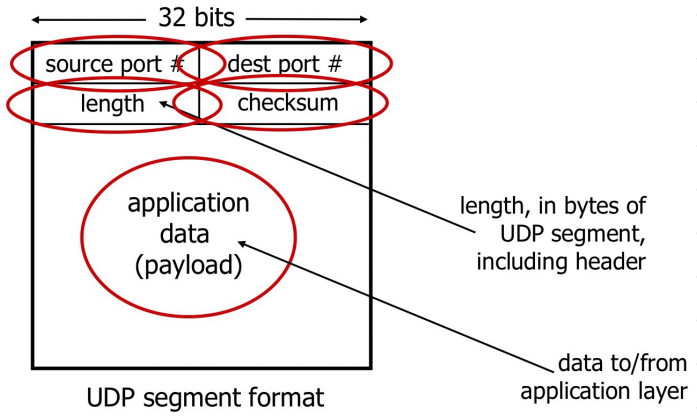
- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add RTT delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control
  - UDP can blast away as fast as desired!
  - can function in the face of congestion

Can be used on streaming

UDP Segment



UDP Checksum: Ensure the integrity of info.

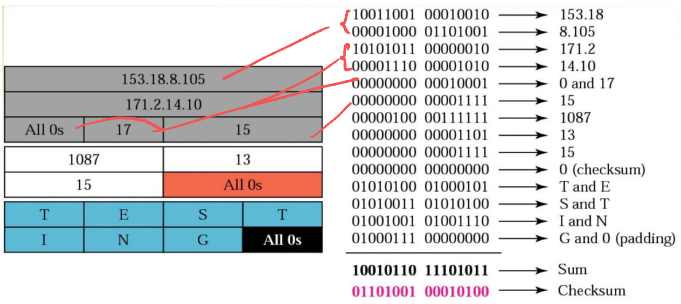
example: add two 16-bit integers

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

) fliped.

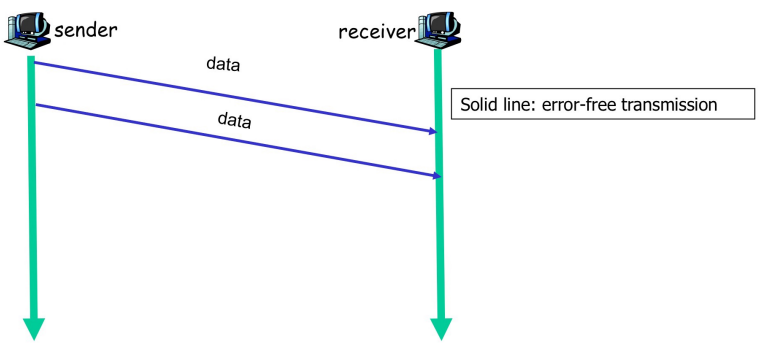
Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

Checksum: example



3.4. Principle of Reliable Data Transfer. (RDT).

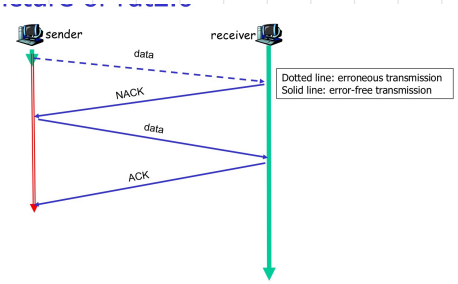
3.4.1. RDT1.0



3.4.2 RDT 2.0

rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
  - checksum to detect bit errors
- *the* question: how to recover from errors?
  - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - sender *retransmits* pkt on receipt of NAK
- new mechanisms in **rdt2.0** (beyond **rdt1.0**):
  - error detection
  - feedback: control msgs (ACK,NAK) from receiver to sender
  - retransmission



Con:
ACK/NACK can disrupted.
→ endless wait.

stop and wait
sender sends one packet, then waits for receiver response

3.4.3. RDT 2.1

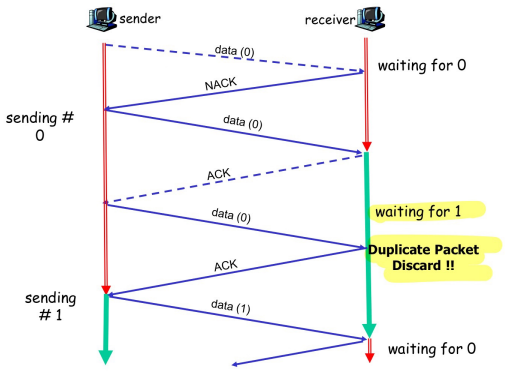
sender:

- seq # added to pkt
- two seq. #s (0,1) will suffice. Why?
- must check if received ACK/NAK corrupted
- twice as many states
  - state must "remember" whether "expected" pkt should have seq # of 0 or 1

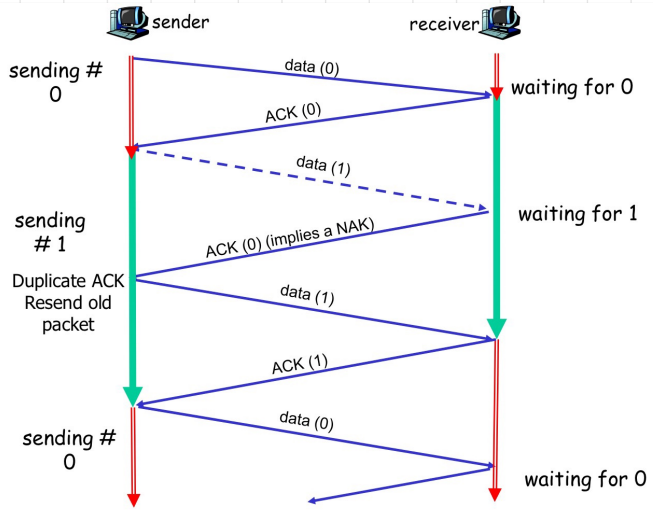
receiver:

- must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can *not* know if its last ACK/NAK received OK at sender

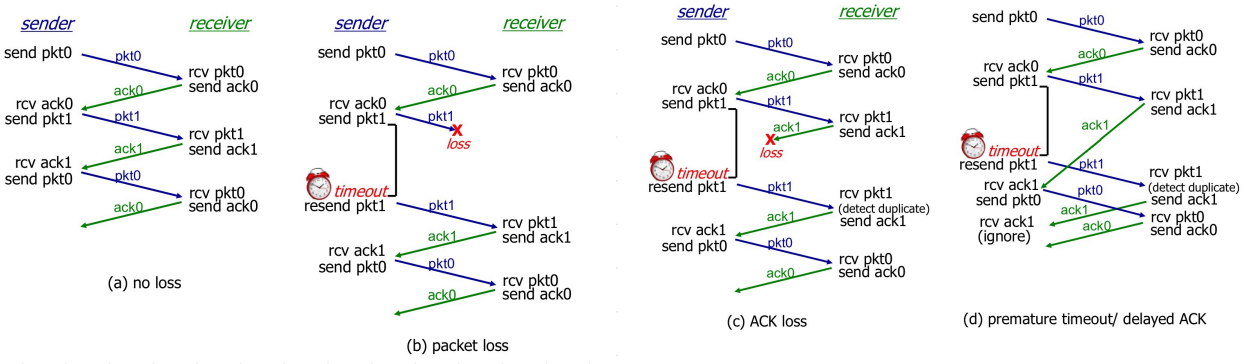
New Measures: Sequence Numbers, Checksum for ACK/NACK, Duplicate detection



3.4.4. RDT 2.2 : Only use ACK (0/1).



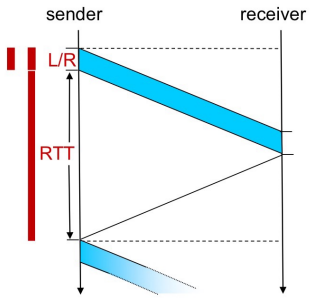
3.4.5. RDT 3.0 : Add Timer



Stop-and-wait Operation.

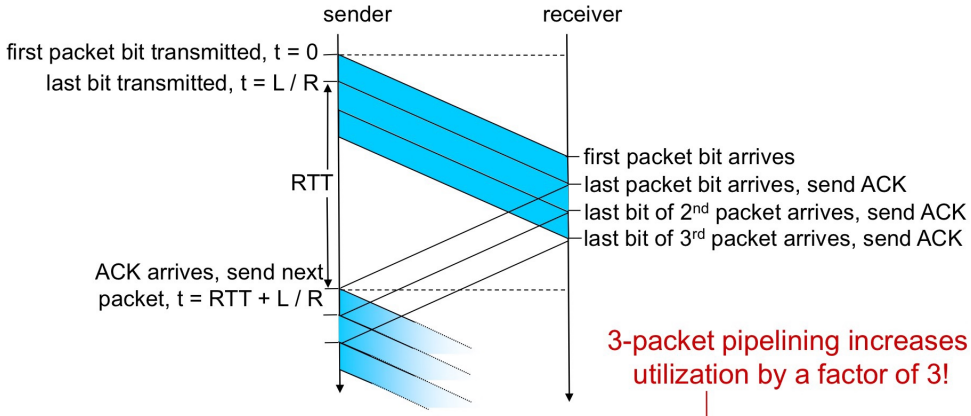
$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

Time fraction of sender busy sending.



- rdt 3.0 protocol performance is very poor!
- Protocol limits performance of underlying infrastructure (channel)

Pipelined protocols operation. (Allow multiple "in-flight", yet-to-acknowledge packet).



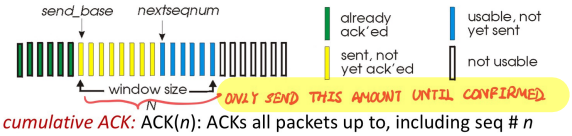
3-packet pipelining increases utilization by a factor of 3!

$$U_{sender} = \frac{3L/R}{RTT + L/R} = \frac{.0024}{30.008} = 0.00081$$

Go-Back-N: (GBN)

Go-Back-N: sender

- sender: "window" of up to N, consecutive transmitted but unACKed pkts
  - k-bit seq # in pkt header



- *cumulative ACK:* ACK(*n*): ACKs all packets up to, including seq # *n*
  - on receiving ACK(*n*): move window forward to begin at *n+1*
- timer for oldest in-flight packet
- *timeout(n):* retransmit packet n and all higher seq # packets in window

Applets: http://media.pearsoncmg.com/aw/aw_kurose_network_2/applets/go-back-n/go-back-n.html
http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/

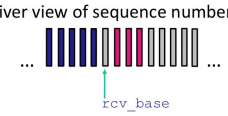
Go-Back-N: receiver

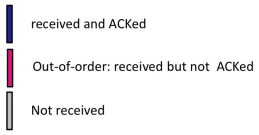
- ACK-only: always send ACK for correctly-received packet so far, with highest *in-order* seq #
  - may generate duplicate ACKs
  - need only remember `rcv_base`
- on receipt of out-of-order packet:
  - can discard (don't buffer) or buffer: an implementation decision
  - re-ACK pkt with highest in-order seq #

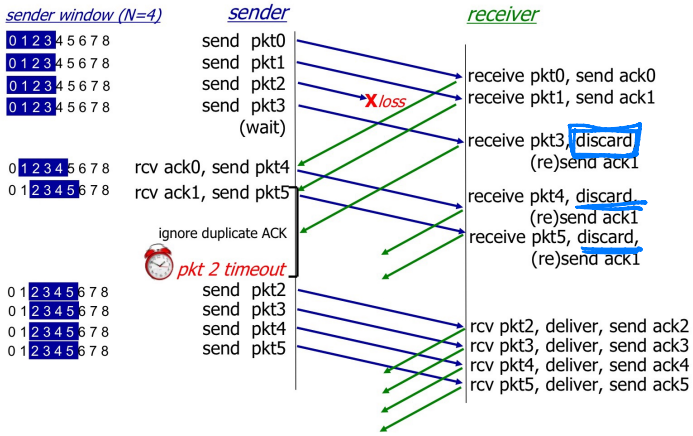
1 ✓ 2 ✓ 4 ✗ 5 ✓ 6 ✓ } Discard

send ACK for 2: then from 3 4 5 ecc.

Receiver view of sequence number space:

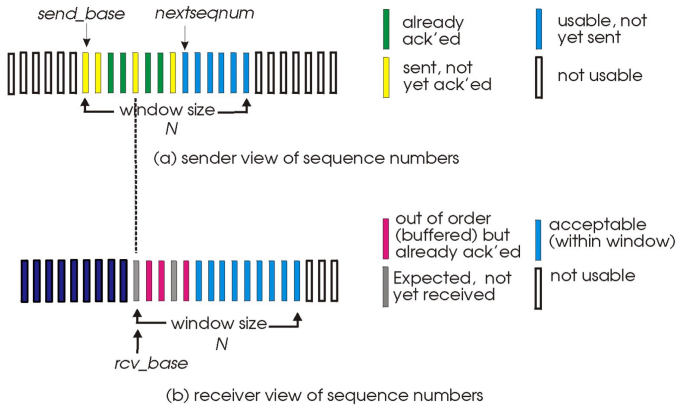


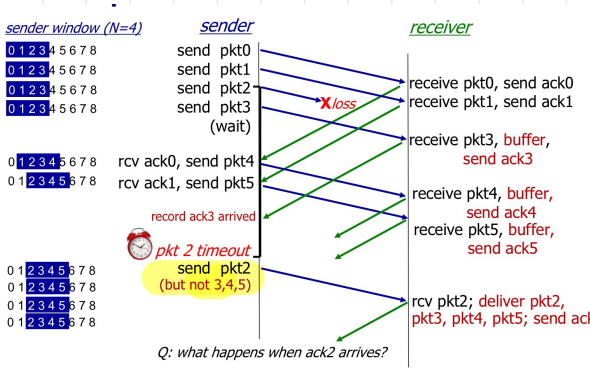




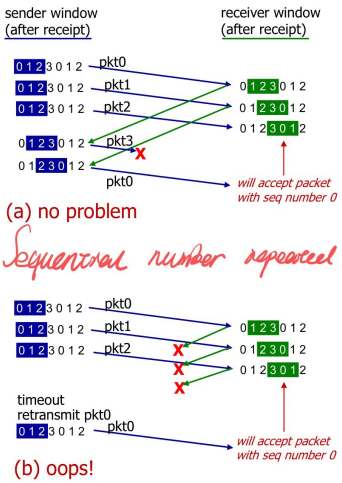
Even received, still discard.

Selective Repeat (SR): ACK individually, only retransmit unACK packets.





Howerer, there is a possible dilemma.



Selective repeat: a dilemma!

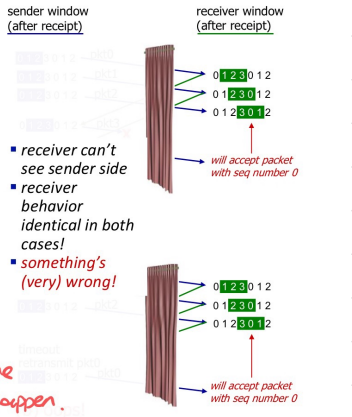
example:

- seq #s: 0, 1, 2, 3 (base 4 counting)
- window size=3

Q: what relationship is needed between sequence # size and window size to avoid problem in scenario (b)?

A: Sender window size <= ½ of Sequence number space

Make sure never happen.



81

3.5. Connection-oriented transport: TCP

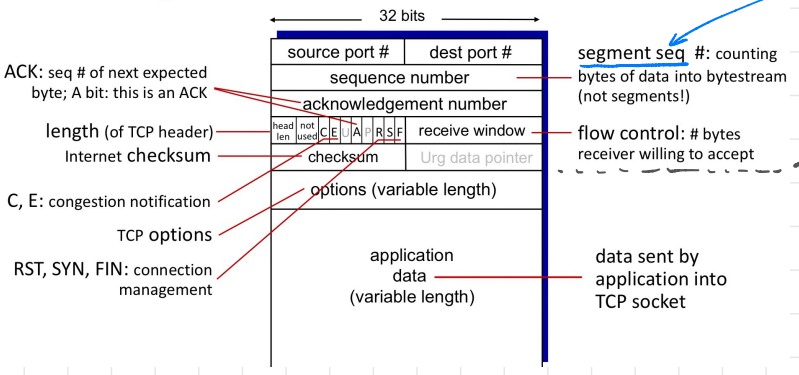
TCP: overview RFCs: 793,1122, 2018, 5681, 7323

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte stream:*
  - no "message boundaries"
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- cumulative ACKs
- pipelining:
  - TCP congestion and flow control set window size
- connection-oriented:
  - handshaking (exchange of control messages) initializes sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

87

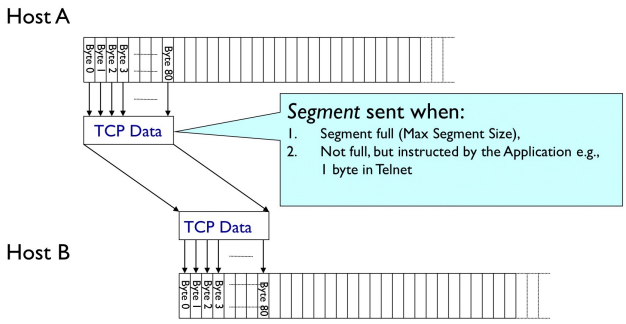
TCP segment structure

Discuss below

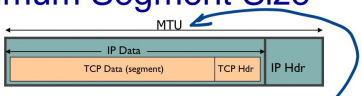


20 bytes, UDP has 8.

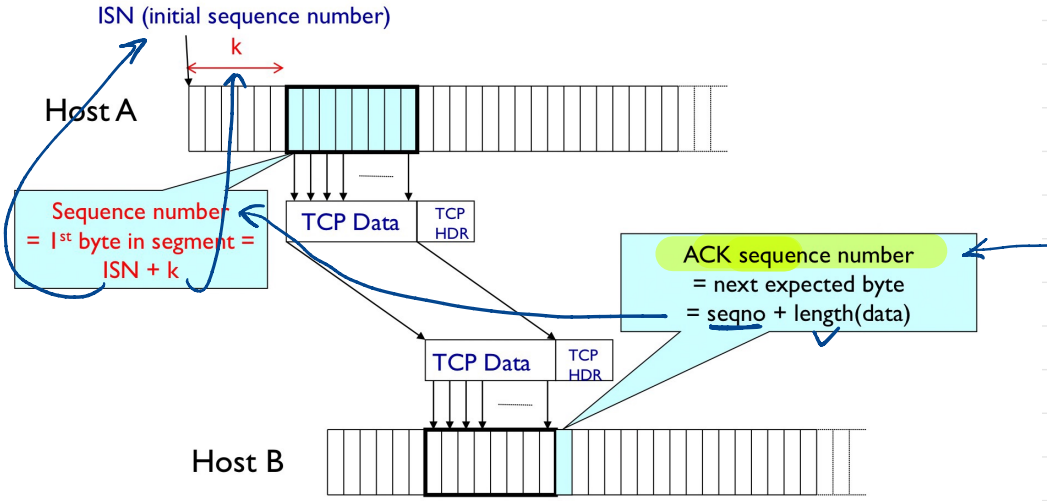
3.5.1 Sequence Number.



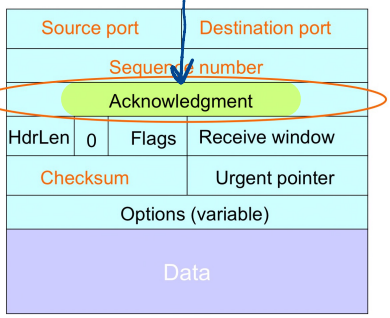
TCP Maximum Segment Size



- IP packet — WHOLE GREEN PART.
  - No bigger than Maximum Transmission Unit (MTU) of link layer
  - E.g., up to 1500 bytes with Ethernet
- TCP packet — WHOLE ORANGE PART.
  - IP packet with a TCP header and data inside
  - TCP header ≥ 20 bytes long
- TCP **segment** — LEFT ORANGE PART
  - No more than Maximum Segment Size (MSS) bytes
  - E.g., up to 1460 consecutive bytes from the stream
  - MSS = MTU – 20 (minimum IP header) – 20 (minimum TCP header)



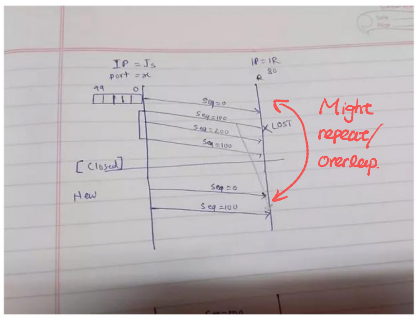
Acknowledgment gives seqno just beyond highest seqno received **in order** (*"What Byte is Next"*)



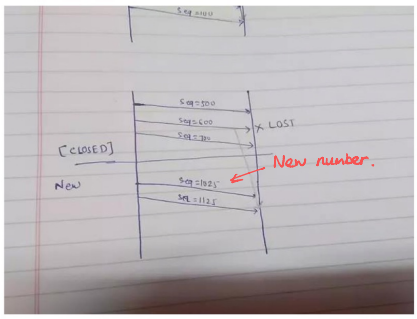
NEED CHECK.

Also ISN is chosen randomly.

- Avoids ambiguity with back-to-back connections between same end-points



(a) When ISN=0



(b) When ISN is random

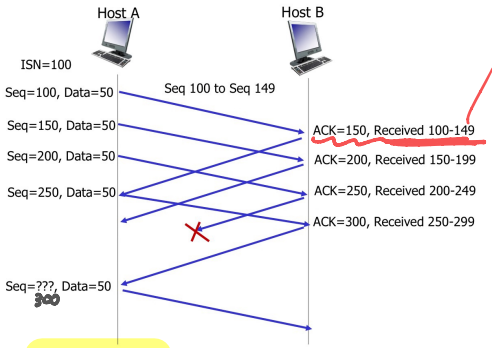
- Potential security issue if the ISN is known

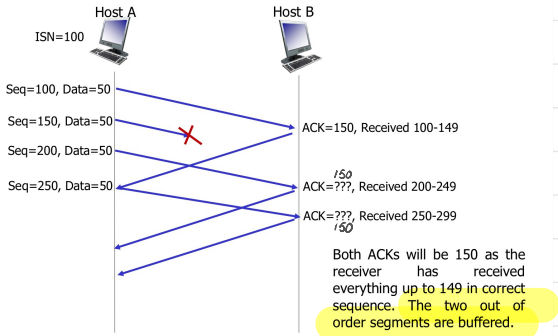
Each side has their own sequence number.

3.5.2. ACKing and sequence number:

- Sender sends packet
  - Data starts with sequence number X
  - Packet contains B bytes [X, X+1, X+2, ...X+B-1]
- Upon receipt of packet, receiver sends an ACK
  - If all data prior to X already received:
    - ACK acknowledges X+B (because that is next expected byte)
  - If highest in-order byte received is Y such that (Y+1) < X
    - ACK acknowledges Y+1
    - Even if this has been ACKed before

Seq = 100, data = 50, not starting from 101, but 100 hence eventually 149 ⇒ But ACK?

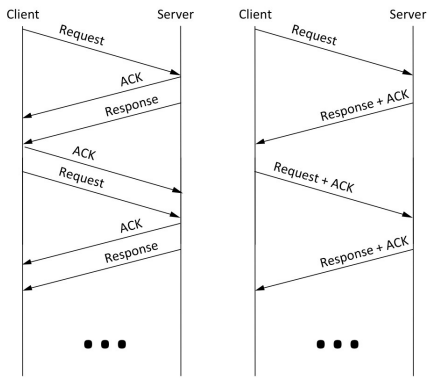




Both ACKs will be 150 as the receiver has received everything up to 149 in correct sequence. The two out of order segments are buffered.

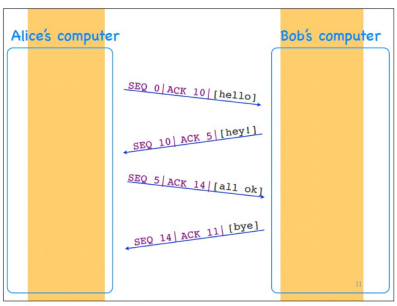
Seq = 300 (new segment)
Since TCP uses cumulative ACKs, the receipt of ACK 300 before a timeout (for seg with sequence number 200) implies the receiver has received all 4 segments sent above

3.5.3. Piggybacking. (ACK combined with seq).

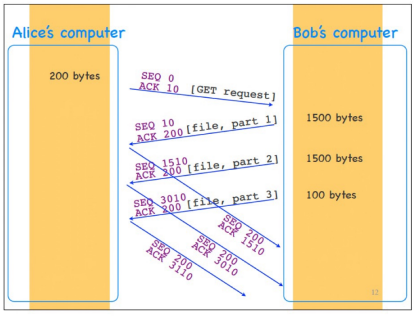


Without Piggybacking

With Piggybacking



Note: Connection establishment not shown. Alice's end point selects the initial sequence number as 0 while Bob's end point selects the initial sequence number as 10

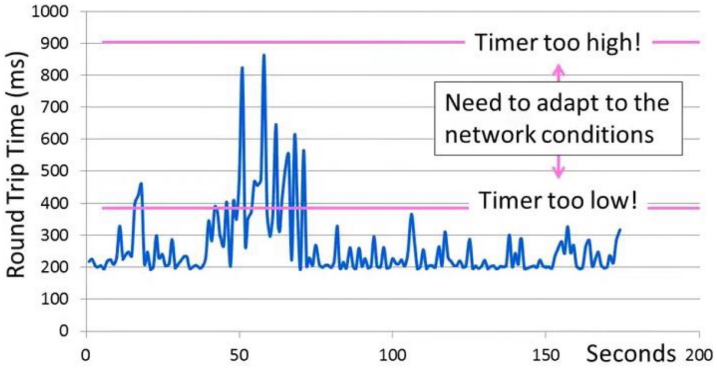


Note: Connection establishment not shown. Alice's end point selects the initial sequence number as 0 while Bob's end point selects the initial sequence number as 10

HTTP response split into 3 segments (MSS = 1500 bytes)

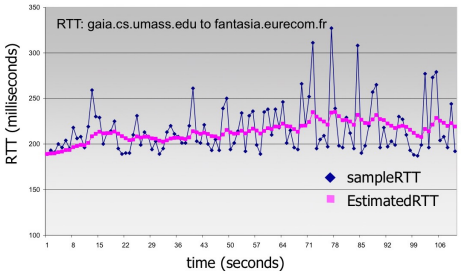
3.5.4 Buffer out-of-seq packets.

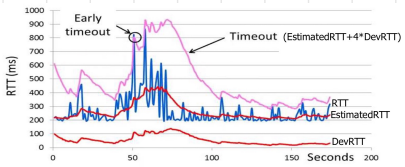
★ 3.5.5 Adjustable/dynamic timeout. 必考.



`EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT`

- exponential weighted moving average (EWMA)
- influence of past sample decreases exponentially fast
- typical value: α = 0.125





`TimeoutInterval = EstimatedRTT + 4*DevRTT`

estimated RTT "safety margin"

- timeout interval: **`EstimatedRTT`** plus "safety margin"
  - large variation in **`EstimatedRTT`**: want a larger safety margin

`TimeoutInterval = EstimatedRTT + 4*DevRTT`

estimated RTT "safety margin"

- **`DevRTT`**: EWMA of **`SampleRTT`** deviation from **`EstimatedRTT`**:

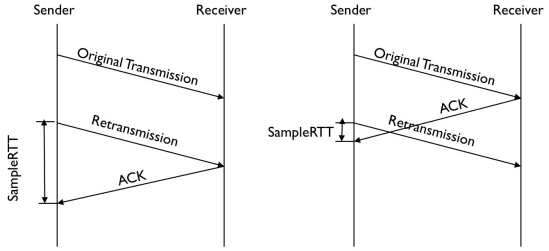
`DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT|`

(typically, β = 0.25)

Practice Problem:
http://wps.pearsoned.com/ecs_kurose_compnetw_6/216/55463/14198700.cw/index.html

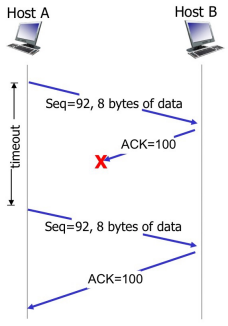
Also we exclude retransmit RTT since:

- How do we differentiate between the real ACK, and ACK of the retransmitted packet?
- Sender cannot differentiate between the two scenarios shown below

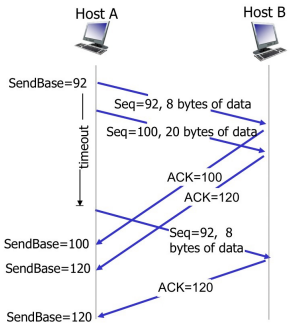


Event Conclusion:

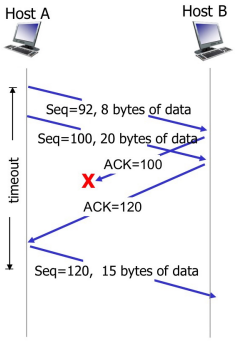
event at receiver	TCP receiver action
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately 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



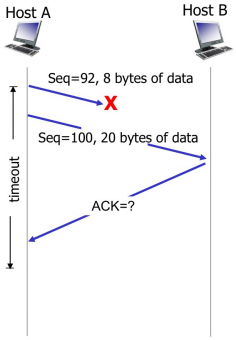
lost ACK scenario



premature timeout



cumulative ACK



cumulative ACK

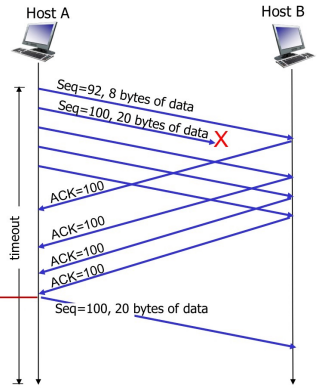
3.5.6. TCP Fast Retransmit.

TCP fast retransmit

if sender receives 3 additional ACKs for same data ("triple duplicate ACKs"), resend unACKed segment with smallest seq #

- likely that unACKed segment lost, so don't wait for timeout

Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!



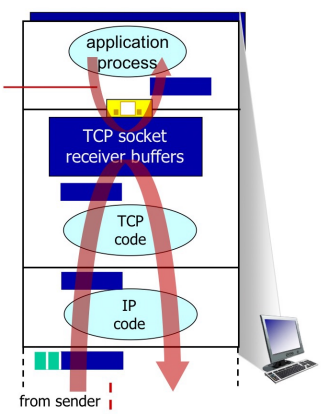
Normally we wait for timeout, since seq=120 might on the route. however, 3 msg later than seq=120 have arrived ⇒ sure resend.

3.5.7 Flow Control.

TCP flow control

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?

Application removing data from TCP socket buffers



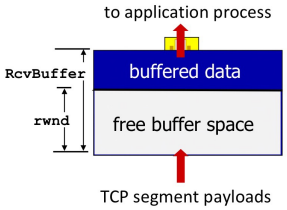
receiver protocol stack

flow control

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

Will lost packages without flow control.

- TCP receiver "advertises" free buffer space in **rwnd** field in TCP header
  - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **RcvBuffer**
- sender limits amount of unACKed ("in-flight") data to received **rwnd**
- guarantees receive buffer will not overflow



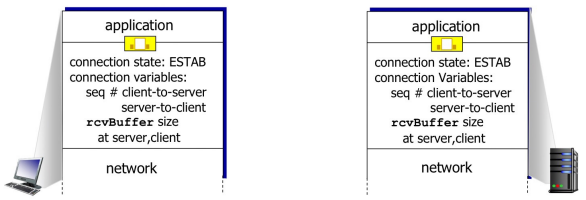
TCP receiver-side buffering

How sender knows the adjusted buffer space?

- What if **rwnd = 0**?
  - Sender would stop sending data
  - Eventually the receive buffer would have space when the application process reads some bytes
  - But how does the receiver advertise the new **rwnd** to the sender?
- Sender keeps sending TCP segments with one data byte to the receiver 你发我你发，
- These segments are dropped but acknowledged by the receiver with a zero-window size Ack no space.
- Eventually when the buffer empties, non-zero window is advertised

3.5.8 TCP Connection Management. (经典三次握手!).

before exchanging data, sender/receiver "handshake":
- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters (e.g., starting seq #s)



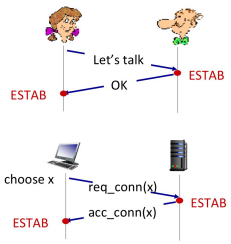

```
Socket clientSocket =
  newSocket("hostname","port number");
```

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

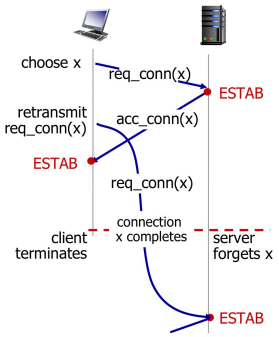
49

3.5.8.1 Two-way handshake.

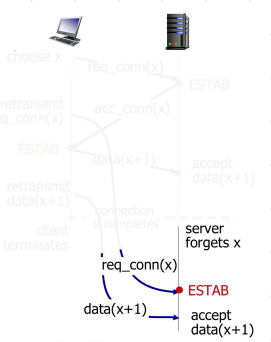
2-way handshake:



Q: will 2-way handshake always work in network?
- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

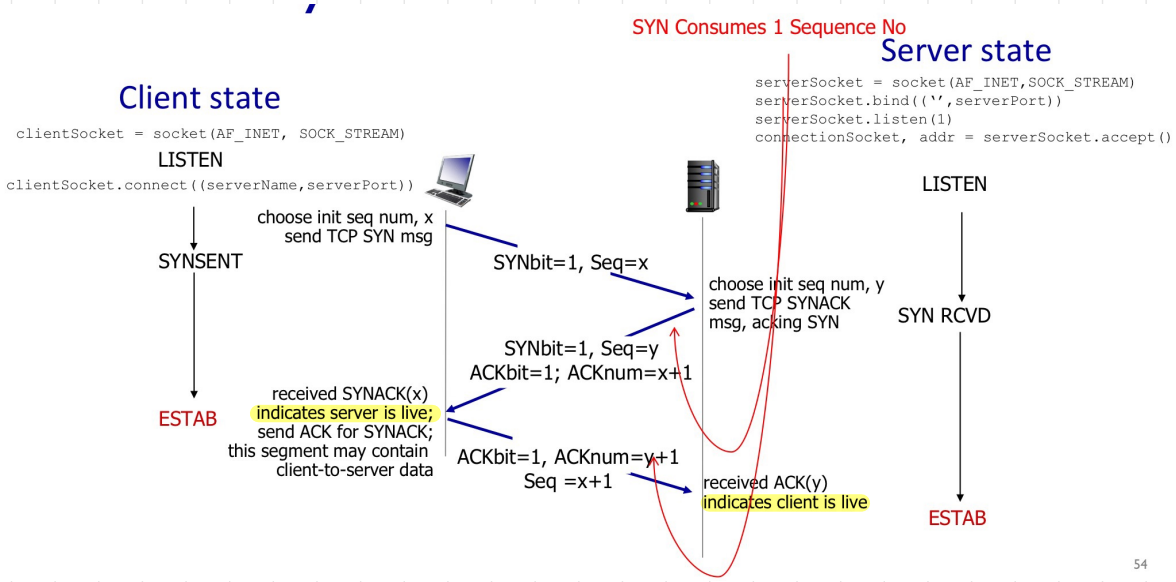


Problem: half open connection! (no client)



Problem: dup data accepted!

3.6.8.2 Three-way handshake.



54

How 3-way can different to 2-way handshake?

What if the SYN Packet Gets Lost?

- Suppose the SYN packet gets lost
  - Packet is lost inside the network, or:
  - Server discards the packet (e.g., it's too busy)
- Eventually, no SYN-ACK arrives
  - Sender sets a timer and waits for the SYN-ACK
  - … and retransmits the SYN if needed
- How should the TCP sender set the timer?
  - Sender has no idea how far away the receiver is
  - Hard to guess a reasonable length of time to wait
  - **SHOULD** (RFCs 1122,2988) use default of 3 second, RFC 6298 use default of 1 second

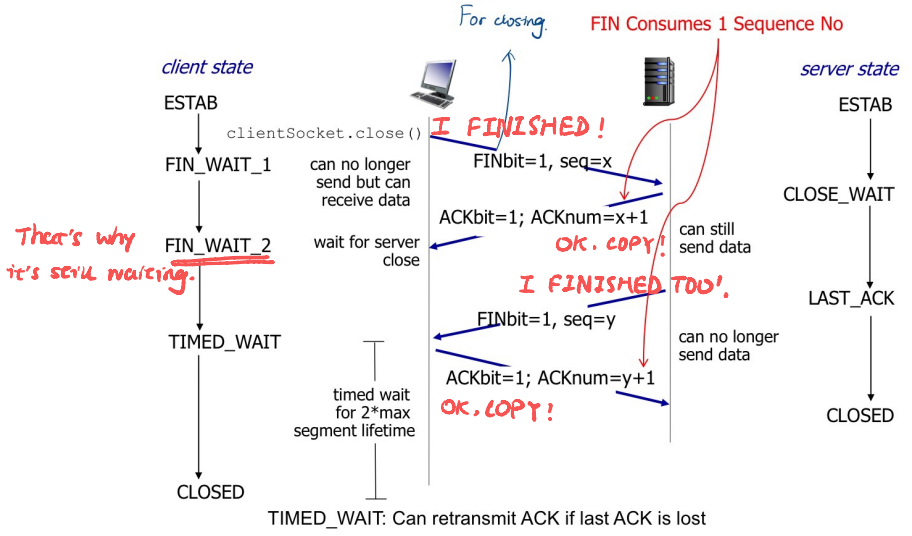
And close connection:

TCP: closing a connection

- client, server each close their side of connection
  - 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

The process of closing the connection:

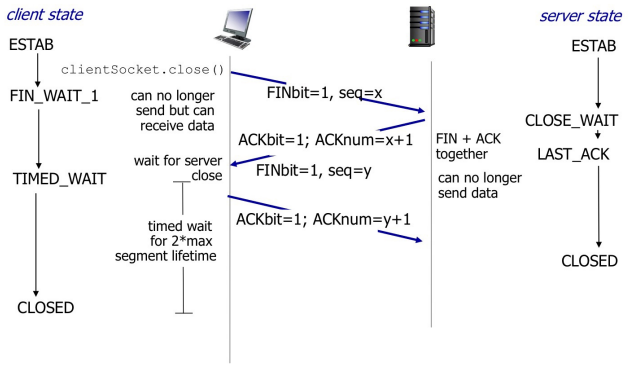
Normal Termination, One at a Time



TIMED_WAIT: Can retransmit ACK if last ACK is lost

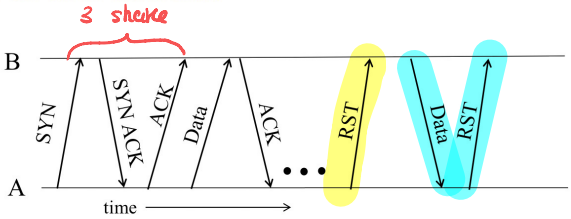
Or server can combined the "OK. COPY" & "I FINISHED TOO!" together:

Normal Termination, Both Together



If one side crashed, it will send RST:

Abrupt Termination

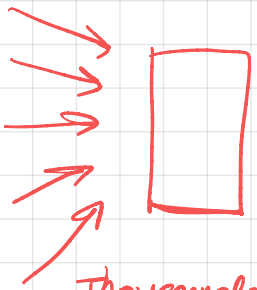


- A sends a RESET (**RST**) to B
  - E.g., because application process on A crashed
- That's it
  - B does not ack the **RST**
  - Thus, **RST** is not delivered reliably
  - And: any data in flight is lost
  - But: if B sends anything more, will elicit another **RST**

DDoS Attack.

TCP SYN Attack (SYN flooding)

- Miscreant creates a fake SYN packet
  - Destination is IP address of victim host (usually some server)
  - Source is some spoofed IP address
- Victim host on receiving creates a TCP connection state i.e allocates buffers, creates variables, etc and sends SYN ACK to the spoofed address (half-open connection)
- ACK never comes back
- After a timeout connection state is freed
- However for this duration the connection state is unnecessarily created
- Further miscreant sends large number of fake SYNs
  - Can easily overwhelm the victim
- Solutions:
  - Increase size of connection queue
  - Decrease timeout wait for the 3-way handshake
  - Firewalls: list of known bad source IP addresses
  - TCP SYN Cookies (explained on next slide)



Thousands of SYN

Server assigned buffer for these SYN, making itself no capacity.

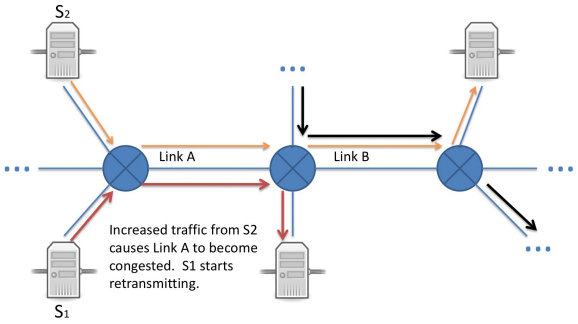
TCP SYN Cookie

- On receipt of SYN, server does not create connection state
- It creates an initial sequence number (*init_seq*) that is a hash of source & dest IP address and port number of SYN packet (secret key used for hash)
  - Replies back with SYN ACK containing *init_seq*
  - Server does not need to store this sequence number
- If original SYN is genuine, an ACK will come back
  - Same hash function run on the same header fields to get the initial sequence number (*init_seq*)
  - Checks if the ACK is equal to (*init_seq*+1)
  - Only create connection state if above is true
- If fake SYN, no harm done since no state was created

3.6. Principle of Congestion Control.

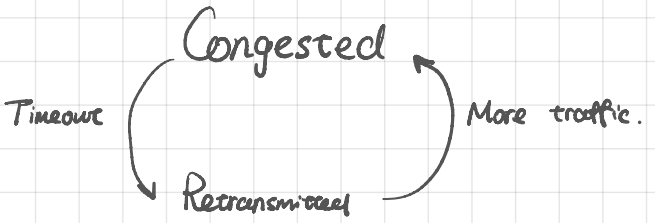
congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!



Result:

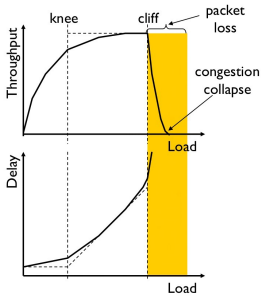
- Increases delays
  - If delays > RTO, sender retransmits
- Increases loss rate
  - Dropped packets also retransmitted
- Increases retransmissions, many unnecessary
  - Wastes capacity of traffic that is never delivered
  - Increase in load results in decrease in useful work done
- Increases congestion, cycle continues …



Cost:

Cost of Congestion

- Knee – point after which
  - Throughput increases slowly
  - Delay increases fast
- Cliff – point after which
  - Throughput starts to drop to zero (congestion collapse)
  - Delay approaches infinity



Crashed

Approach:

two broad approaches towards congestion control:

Server/client.

end-end congestion control:

- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- approach taken by TCP

✓ In practice

Router.

network-assisted congestion control:

- routers provide feedback to end systems
  - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - explicit rate for sender to send at

⊗

↓ Discuss next.

3.7. TCP Congestion Control.

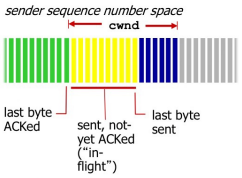
TCP's Approach in a Nutshell

- TCP connection maintains a window
  - Controls number of packets in flight

} Reduce traffic

- *TCP sending rate:*
  - *roughly:* send cwnd bytes, wait RTT for ACKs, then send more bytes

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

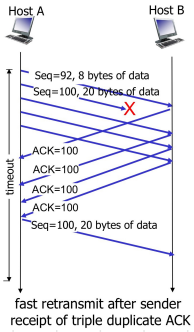


- Vary window size to control sending rate

- Congestion Window: CWND
  - How many bytes can be sent without overflowing routers
  - Computed by the sender using congestion control algorithm Varies.
    In the unit of MSS (Max Segment Size)
- Flow control window: Advertised / Receive Window (RWND)
  - How many bytes can be sent without overflowing receiver's buffers
  - Determined by the receiver and reported to the sender
    Sender Receiver.
- Sender-side window = minimum{CWND, RWND}
  - Assume for this discussion that RWND >> CWND
    Receiver size >> Sender size.

How sender detects congestion: Infer loss.

- Duplicate ACKs: isolated loss
  - dup ACKs indicate network capable of delivering some segments
- Timeout: much more serious
  - Not enough dup ACKs
  - Must have suffered several losses
- Will adjust rate differently for each case ⚠



If 3 consecutive prev ACK received, no need to wait timeout, then faster.

How does the sender detect congestion?

- Basic structure:
  - Upon receipt of ACK (of new data): increase rate
  - Upon detection of loss: decrease rate

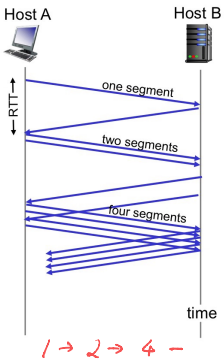
As we know.
more traffic → Queue ↑ → lost
Hence,
Decrease rate → reduce traffic.

- How we increase/decrease the rate depends on the phase of congestion control we're in:
  - Discovering available bottleneck bandwidth *vs.*
  - Adjusting to bandwidth variations

TCP Slow Start (Bandwidth discovery)

- when connection begins, increase rate exponentially until first loss event:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT (all ACKs)
  - Simpler implementation achieved by incrementing **cwnd** for every ACK received
    - cwnd += 1 for each ACK
- summary: initial rate is slow but ramps up exponentially fast

This means +1 window, equivalent to ×2



1 → 2 → 4 –

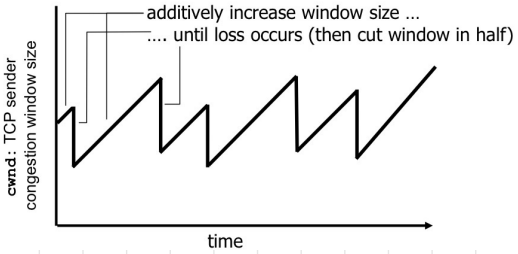
Adjusting to Varying Bandwidth

- Slow start gave an estimate of available bandwidth
- Now, want to track variations in this available bandwidth, oscillating around its current value
  - Repeated probing (rate increase) and back-off (rate decrease)
  - Known as Congestion Avoidance (CA)
- TCP uses: "Additive Increase Multiplicative Decrease" (AIMD)

AIMD

- *approach:* sender increases transmission rate (window size), probing for usable bandwidth, until another congestion event occurs
  - *additive increase:* increase **cwnd** by 1 MSS every RTT until loss detected
    - For each successful RTT (all ACKs), **cwnd = cwnd +1** (in multiples of MSS) → += 1 个格子 Rather double.
    - Simple implementation: for each ACK, **cwnd = cwnd + 1/cwnd** (since there are **cwnd/MSS** packets in a window) → $\frac{1}{window}$ + window
  - *multiplicative decrease:* cut **cwnd** in half after loss → Cut half

AIMD saw tooth behavior: probing for bandwidth



Slow Start vs. AIMD.

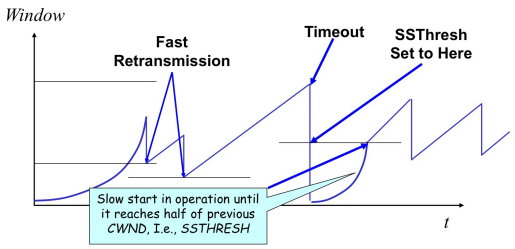
- Introduce a "slow start threshold" (ssthresh)
  - Initialized to a large value
- Convert to CA when cwnd = ssthresh, sender switches from slow-start to AIMD-style increase
  - On loss, ssthresh = CWND/2 → is also changing.

Congestion window = Threshold

Convert.

For example:

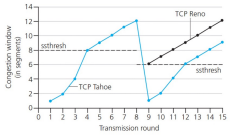
- State at sender
  - CWND (initialized to a small constant)
    - the slides use multiple of MSS
  - ssthresh (initialized to a large constant)
  - [Also dupACKcount and timer, as before]
- Events
  - ACK (new data)
  - dupACK (duplicate ACK for old data)
  - Timeout



Slow-start restart: Go back to CWND = 1 MSS, but take advantage of knowing the previous value of CWND

TCP Flavours

- TCP-Tahoe
  - cwnd =1 on triple dup ACK & timeout
- TCP-Reno
  - cwnd =1 on timeout
  - cwnd = cwnd/2 on triple dup ACK



For new ACK (new data).

- If CWND < ssthresh
  - CWND += 1 (window +1 (Double))
  - Hence after one RTT (All ACKs with no drops): CWND = 2xCWND
- If CWND < ssthresh
  - CWND += 1

  Slow start phase
- Else
  - CWND = CWND + 1/CWND

  "Congestion Avoidance" phase (additive increase)
  - Hence after one RTT (All ACKs with no drops): CWND = CWND + 1

For dupACK

- dupACKcount ++
- If dupACKcount = 3 /* fast retransmit */
  - ssthresh = CWND/2
  - CWND = CWND/2

For timeout.

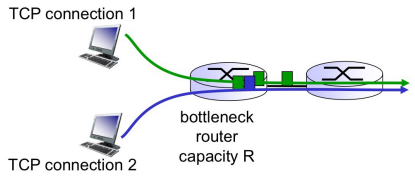
- On Timeout
  - ssthresh ← CWND/2
  - CWND ← 1

← Drop back to 1 window.

TCP Fairness

TCP Fairness

fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K

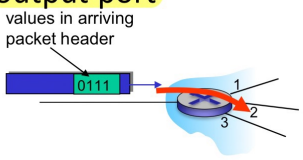


4. Network Layer : Data Plane.

4.1. Overview of Network Layer

Data plane:

- *local*, per-router function
- determines how datagram arriving on router input port is forwarded to router output port



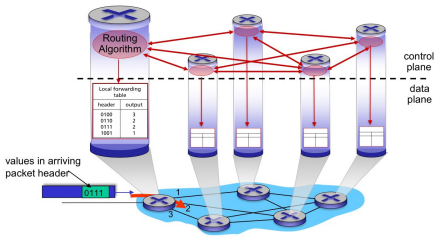
Control plane

- *network-wide* logic
- determines how datagram is routed among routers along end-end path from source host to destination host
- two control-plane approaches:
  - *traditional routing algorithms:* implemented in routers
  - *software-defined networking (SDN)*: implemented in (remote) servers

Global Planner.

Per-router control plane

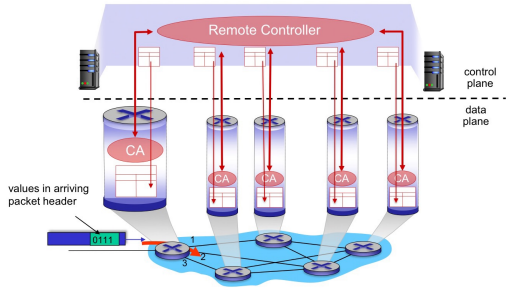
Individual routing algorithm components *in each and every router* interact in the control plane



Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers

NOT ON EXAM



"Best efford" service model.

Q: What *service model* for "channel" transporting datagrams from sender to receiver?

Internet "best effort" service model

No guarantees on:

i. successful datagram delivery to destination
ii. timing or order of delivery
iii. bandwidth available to end-end flow

Reflections on best-effort service:

- simplicity of mechanism has allowed Internet to be widely deployed adopted
- sufficient provisioning of bandwidth allows performance of real-time applications (e.g., interactive voice, video) to be "good enough" for "most of the time"
- replicated, application-layer distributed services (datacenters, content distribution networks) connecting close to clients' networks, allow services to be provided from multiple locations
- congestion control of "elastic" services helps

It's hard to argue with success of best-effort service model

4.3 Internet Protocol.

4.3.1 Datagram Format.

Special handling.
- Allow for special needs, e.g. low delay for audio.

Version of IP protocol.
Typically "4" for IPv4.

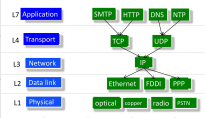
Number of bytes in the packet.
Max size is 65535 bytes ($2^{16}-1$)

4-bit Version	4-bit Header Length	8-bit Type of Service (TOS)	16-bit Total Length (Bytes)
16-bit Identification		3-bit Flags	13-bit Fragment Offset
8-bit Time to Live (TTL)		8-bit Protocol	16-bit Header Checksum
32-bit Source IP Address			
32-bit Destination IP Address			
Options (if any)			
Payload			

check HEADER only.

Fragmentation discuss below.

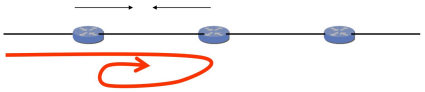
Way to demultiplexing at receiving host.



E.g.
Record timestamp,
Record route taken,
Visit specific list of routers.

TTL: Preventing Loop

- Forwarding loops cause packets to cycle for a long time
  - As these accumulate, eventually consume **all** capacity

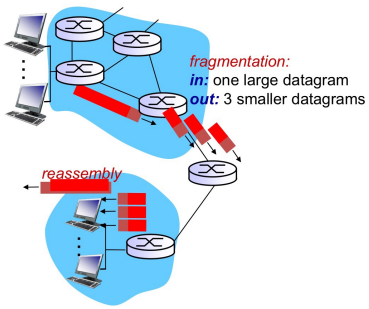


- Time-to-Live (TTL) Field (8 bits)
  - Decremented at each hop, packet discarded if reaches 0
  - …and "time exceeded" message is sent to the source
  - Recommended default value is 64

It's not time,
it's the number of routers can travel.

4.3.2 Fragmentation

- network links have MTU (max.transfer size) - largest possible link-level frame
  - different link types, different MTUs
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at final destination
  - IP header bits used to identify, order related fragments



Note: Offset is expressed as multiple of 8 bytes

example:
- 4000-byte datagram
- MTU = 1500 bytes

length	ID	MF flag	offset
=4000	=x	=0	=0

More Fragment (coming).

one large datagram becomes several smaller datagrams

File starting byte /8.

Data = 4000 – 20 (IP header)
=3980
F1=1480
F2=1480
F3=1020

1480 bytes in data field

length	ID	MF flag	offset
=1500	=x	=1	=0

offset = 1480/8

length	ID	MF flag	offset
=1500	=x	=1	=185

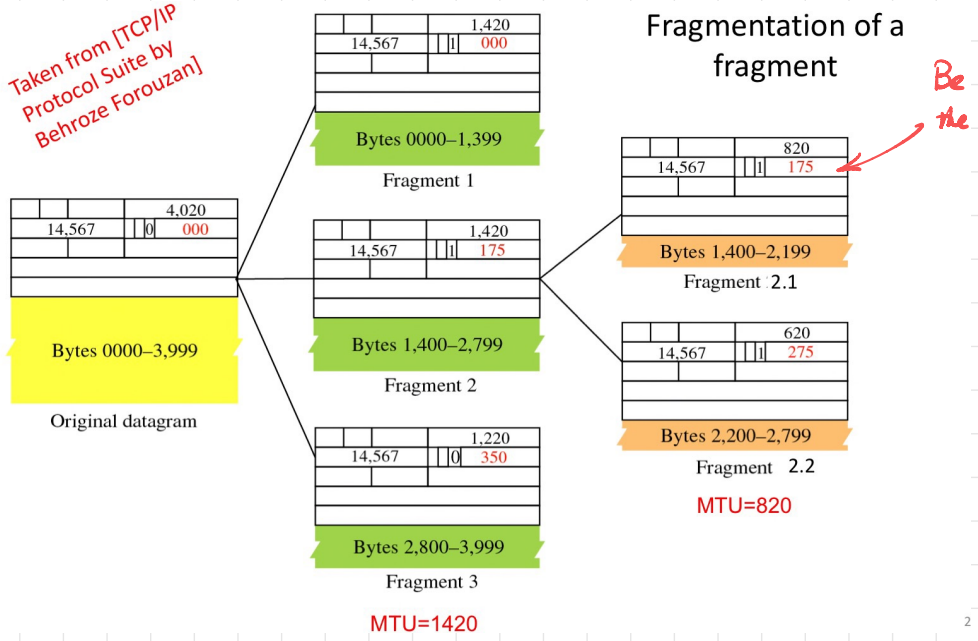
Starting at 1480, hence offset: 185

length	ID	MF flag	offset
=1040	=x	=0	=370

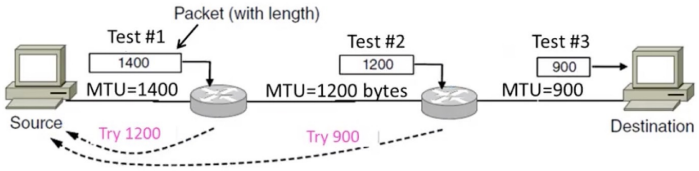
Starting at 1480×2=2960
hence offset = 2960/8 = 370.

Fragmentation of a fragment:

When encountering smaller and smaller MTU.



Path MTU Discovery procedure

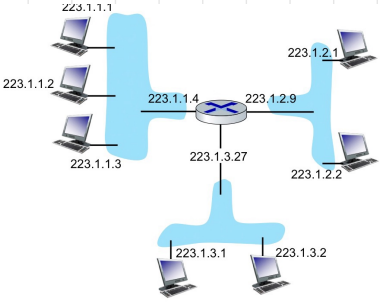


- Host
  - Sends a big packet to test whether all routers in path to the destination can support or not
  - Set DF (Do not fragment) flag
- Routers
  - Drops the packet if it is too large (as DF is set)
  - Provides feedback to Host with ICMP message telling the maximum supported size

4.3.3. IPv4 Addressing.

IP Address and interface.

- IP address: 32-bit identifier associated with each host or router *interface*
- interface: connection between host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)



dotted-decimal IP address notation:

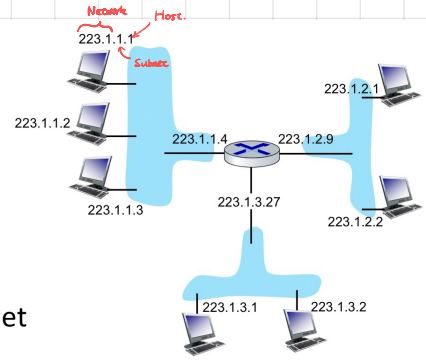
223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1

Network. Host (Subnet)

Subnets

- *What's a subnet ?*
  - device interfaces that can physically reach each other without passing through an intervening router
- IP addresses have structure:
  - subnet part: devices in same subnet have common high order bits
  - host part: remaining low order bits

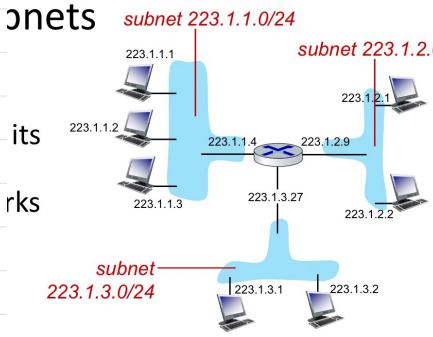


network consisting of 3 subnets

ɔnets

its

rks

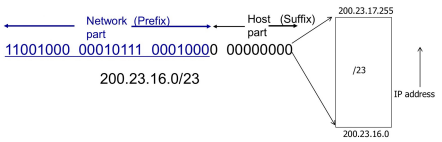


subnet mask: /24
(high-order 24 bits: subnet part of IP address)

2^32 bits.
CIDN Notation.

The number of bits belong to network out of 32 bits.
↳ so in this case, first 24 bits out of 32 bits.
3 bytes.
↳ 223. 1. 2 is net

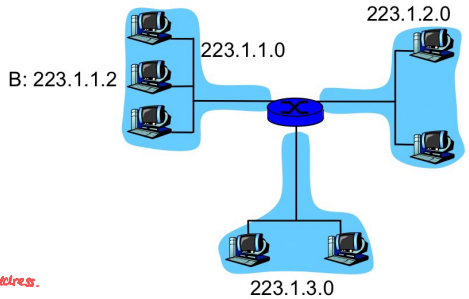
CIDR: Classless InterDomain Routing
- network portion of address of arbitrary length
- address format: a.b.c.d/x, where x is # bits in network portion of address



Mask: Used to filter out < Network. / Host > out of IP address.

Network Mask

- Mask
  - Used in conjunction with the network address to indicate how many higher order bits are used for the network part of the address
    - Bit-wise AND
  - 223.1.1.0 with mask 255.255.255.0
- Broadcast Address ← Send msg to all host in this address.
  - host part is all 111's
  - E.g., 223.1.1.255 (111 111)
- Network Address
  - Host part is all 0000's
  - E.g., 223.1.1.0
- Both are typically not assigned to any host



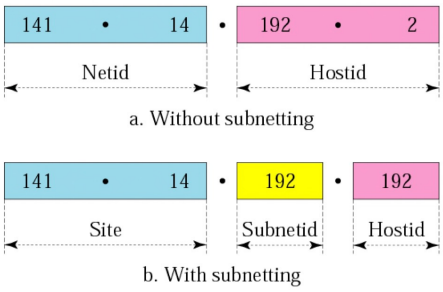
Host B	Dot-decimal address	Binary
IP address	223.1.1.2	11011111.00000001.00000001.00000010
Mask	255.255.255.0	11111111.11111111.11111111.00000000
Network Part	223.1.1.0	11011111.00000001.00000001.00000000
Host Part	0.0.0.2	00000000.00000000.00000000.00000010

Host : 1 ~ 254.
0 is network address 255 is broadcast address.

Subnetting: Dividing the class of your network into more manageable chucks.

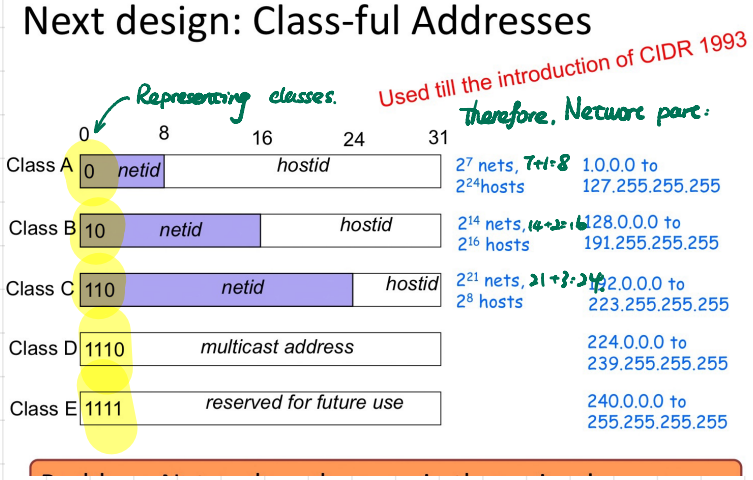
- Subnetting allows 3 levels of hierarchy
  - netid, subnetid, hostid
- Original netid remains the same and designates the site
- Subnetting remains transparent outside the site

- The process of subnetting simply extends the point where the 1's of Mask stop and 0's start
- You are sacrificing some host ID bits to gain Network ID bits



Class B Network

Next design: Class-ful Addresses

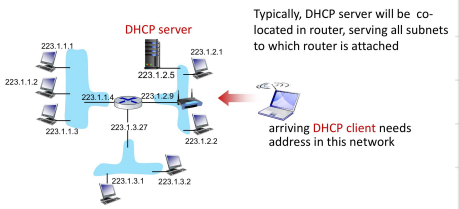


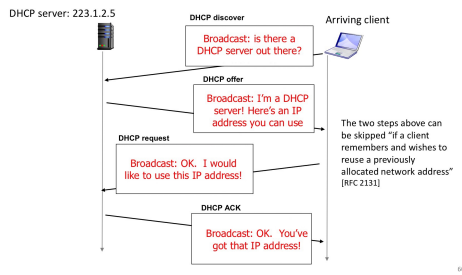
DHCP: IP 分配系统.

DHCP: Dynamic Host Configuration Protocol

goal: host *dynamically* obtains 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/on)
- support for mobile users who join/leave network

DHCP 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

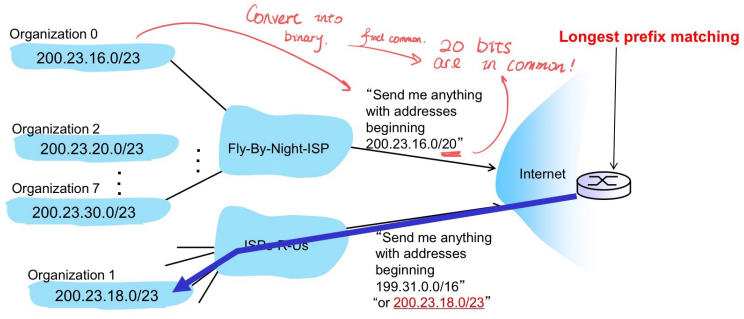




Route Aggregation.

Hierarchical addressing: more specific routes

- Organization 1 moves from Fly-By-Night-ISP to ISPs-R-Us
- ISPs-R-Us now advertises a more specific route to Organization 1



e.g.

10.1.2.0/26
10.1.2.64/26
10.1.2.128/26
10.1.2.192/26

00001010.00000001.00000010.00000000
00001010.00000001.00000010.01000000
00001010.00000001.00000010.10000000
00001010.00000001.00000010.11000000

00001010.00000001.00000010.00000000
10.1.2.0/24

192.168.5.130/25
How many addresses are in this block? 2^(32-25)=2^7=128
Network address:
Broadcast address:
Addresses can allocated to the users:

192.168.5.130
11000000.10101000.00000101.10000010
11111111.11111111.11111111.10000000
---------- (ANDing)
11000000.10101000.00000101.10000000
192.168.5.128/25 =Network address

11000000.10101000.00000101.11111111
192.168.5.255 Broadcast address

Total address=128

Network address and broadcast address cannot be allocated to the host.

Number of address to be allocated to the host=128-2=126
Range =192.168.2.129/25 to 192.168.2.254/25

Rule for Longest Prefix Matching.

Longest prefix matching

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

goes Link 0 since matched.

examples:

DA: 11001000 00010111 00010110 10100001 which interface?

DA: 11001000 00010111 00011000 10101010 which interface?

Match for both Link 1 & 2, We have to find longest prefix, hence Link 1.

4.3.4 Network Address Translation (NAT)

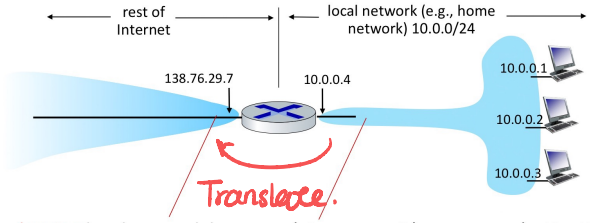
IP Address
- Private address (Cannot connect internet)
  - Defined in RFC 1918:
    - 10.0.0.0/8 (16,777,216 hosts)
    - -172.16.0.0/12 (1,048,576 hosts)
    - -192.168.0.0/16 (65536 hosts)

  } Range for private.

  - These addresses cannot be routed
    - Anyone can use them in a private network
    - -Typically used for NAT
- Public address

NAT translate private to public for connecting internet.

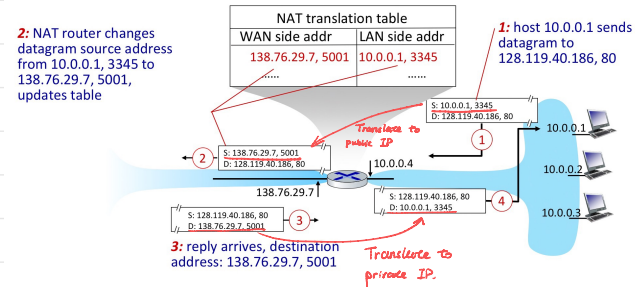
NAT: all devices in local network share just one IPv4 address as far as outside world is concerned



all datagrams *leaving* local network have *same* source NAT IP address: 138.76.29.7, but *different* source port numbers

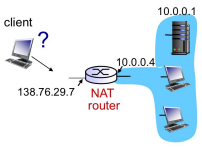
datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

advantages:
- just one IP address needed from provider ISP for *all* devices
- can change addresses of host in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- security: devices inside local net not directly addressable, visible by outside world



Disadvantage:
- NAT modifies port # and IP address
  - *Requires recalculation of TCP and IP checksum*
- Some applications embed IP address or port numbers in their message payloads
  - DNS, FTP (PORT command), SIP, H.323
  - For legacy protocols, NAT must look into these packets and translate the embedded IP addresses/port numbers
  - Duh, What if these fields are encrypted ?? (SSL/TLS, IPSEC, etc.)
  - **Q: In some cases, why may NAT need to change TCP sequence number?? (Discussion Question on Website)**
- If applications change port numbers periodically, the NAT must be aware of this

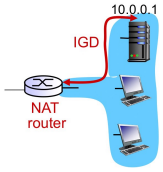
Solutions:
- client wants to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can't use it as destination addr)
  - only one externally visible NATed address: 138.76.29.7
- *Solution1:* Inbound-NAT Statically configure NAT to forward incoming connection requests at given port to server
  - e.g., (138.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000



pattern.

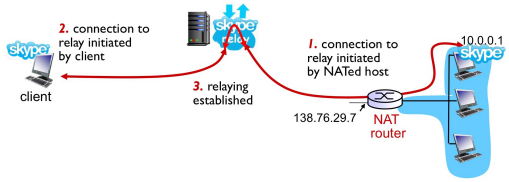
- *solution 2:* Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATed host to:
  - learn public IP address (138.76.29.7)
  - add/remove port mappings (with lease times)

  i.e., automate static NAT port map configuration



提前沟通.

- *solution 3:* relaying (used in Skype)
  - NATed client establishes connection to relay
  - external client connects to relay
  - relay bridges packets between to connections

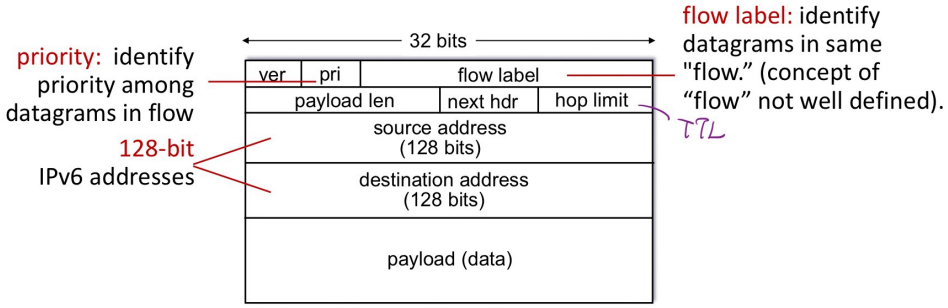


4.3.5 IPv6

IPv6: motivation → 128 bit → 2^{128}

- initial motivation: 32-bit IPv4 address space would be completely allocated 2^{32}
- additional motivation:
  - speed processing/forwarding: 40-byte fixed length header
  - enable different network-layer treatment of "flows"

IPv6 datagram format



What's missing (compared with IPv4):

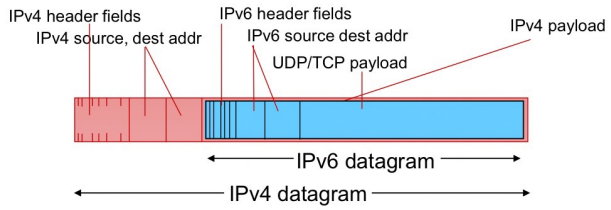
- no checksum (to speed processing at routers)
- no fragmentation/reassembly
- no options (available as upper-layer, next-header protocol at router)

19

Transition from IPv4 to IPv6

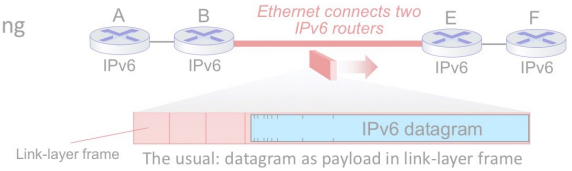
Compatible for both IPv4 and IPv6 routers.

- not all routers can be upgraded simultaneously
  - no "flag days"
  - how will network operate with mixed IPv4 and IPv6 routers?
- tunneling: IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers ("packet within a packet")
  - tunneling used extensively in other contexts (4G/5G)

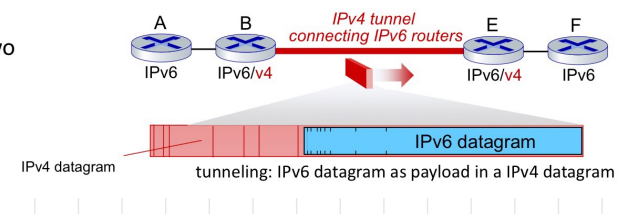


Tunneling and encapsulation

Ethernet connecting two IPv6 routers:

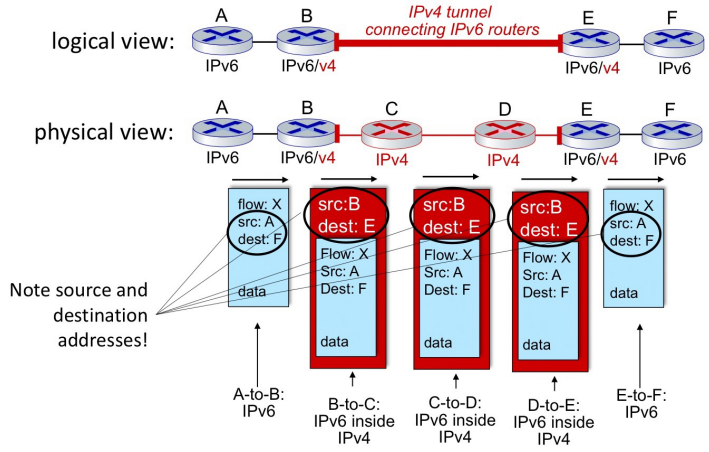


IPv4 tunnel connecting two IPv6 routers



Here,

Tunneling

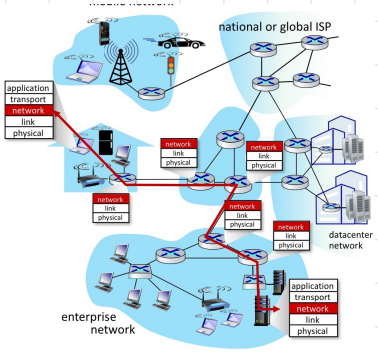


5. Control Plane (Routing).

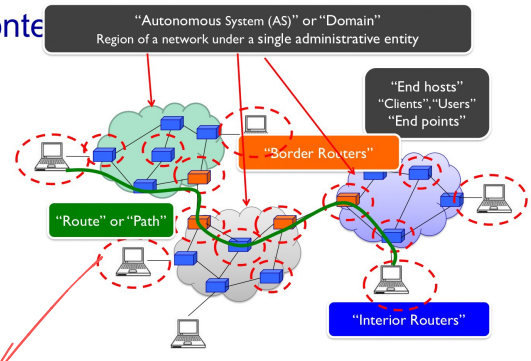
5.1 Intro

Routing protocol goal: determine "good" paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- path: sequence of routers packets traverse from given initial source host to final destination host
- "good": least "cost", "fastest", "least congested"
- routing: a "top-10" networking challenge!



Conte



Internet Routing

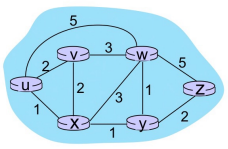
Communicate b/t blue only

- Internet Routing works at two levels
- Each AS runs an intra-domain routing protocol that establishes routes within its domain
  - AS -- region of network under a single administrative entity
  - Link State, e.g., Open Shortest Path First (OSPF)
  - Distance Vector, e.g., Routing Information Protocol (RIP)
- ASes participate in an inter-domain routing protocol that establishes routes between domains
  - Path Vector, e.g., Border Gateway Protocol (BGP)

Broader ↔ Broader (Orange)

Graph abstraction: link costs

can be calculated if directly connected, otherwise



$c_{a,b}$: cost of *direct* link connecting *a* and *b*
e.g., $c_{w,z} = 5$, $c_{u,z} = \infty$

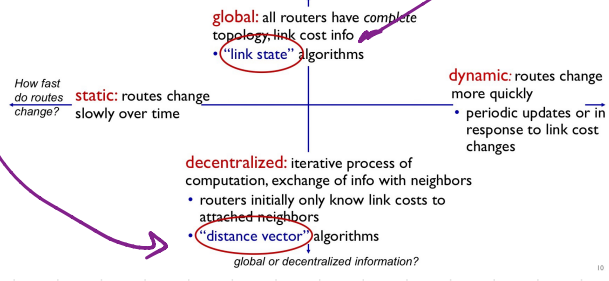
cost defined by network operator: could always be 1, or inversely related to bandwidth, or inversely related to congestion

graph: $G = (N,E)$

N: set of routers = { *u, v, w, x, y, z* }

E: set of links ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

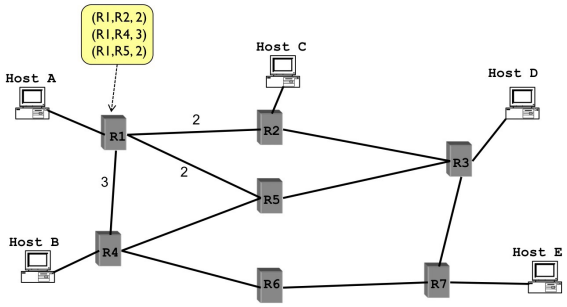
Routing algorithm classification



5.2 Routing Protocol.

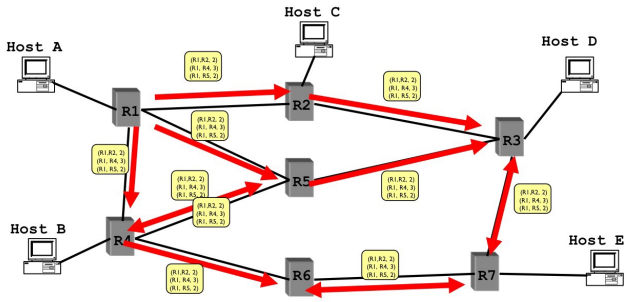
5.2.1 Link State.

- Each node maintains its local "link state" (LS)
  - i.e., a list of its directly attached links and their costs



Then: Update state thro the network.

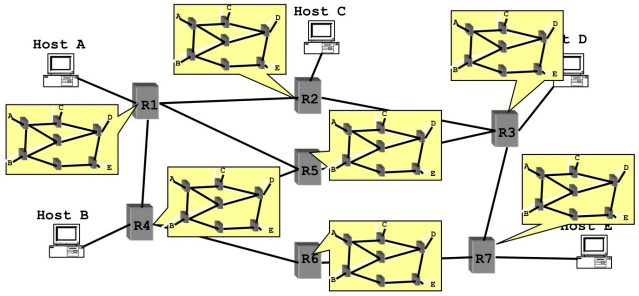
- Each node maintains its local "link state" (LS)
- Each node floods its local link state
  - on receiving a new LS message, a router forwards the message to all its neighbors other than the one it received the message from
    = ALL



- Routers transmit Link State Advertisement (LSA) on links
  - A neighbouring router forwards out on all links except incoming
  - Keep a copy locally; don't forward previously-seen LSAs
- Challenges
  - Packet loss
  - Out of order arrival
- Solutions
  - Acknowledgements and retransmissions
  - Sequence numbers
  - Time-to-live for each packet

Eventually,

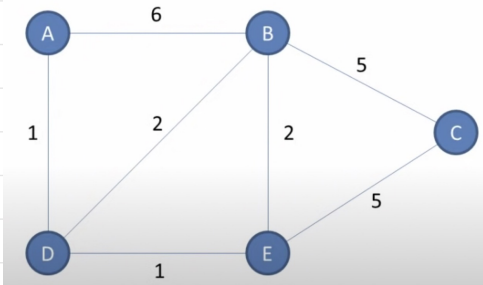
- Each node maintains its local "link state" (LS)
- Each node floods its local link state
- Eventually, each node learns the entire network topology
  - Can use Dijkstra's to compute the shortest paths between nodes



- **centralized:** network topology, link costs known to *all* nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- computes least cost paths from one node ("source") to all other nodes
  - gives *forwarding table* for that node
- **iterative:** after *k* iterations, know least cost path to *k* destinations

notation

- $c_{x,y}$: direct link cost from node *x* to *y*; = ∞ if not direct neighbors
- *D(v):* *current* estimate of cost of least-cost-path from source to destination *v*
- *p(v):* predecessor node along path from source to *v*
- *N'*: set of nodes whose least-cost-path *definitively* known



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B, C] Unvisited = []

Complexity of Dijkstra's Algorithm.

algorithm complexity: *n* nodes

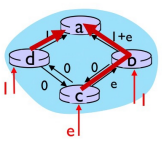
- each of *n* iteration: need to check all nodes, *w*, not in *N*
- $n(n+1)/2$ comparisons: $O(n^2)$ complexity
- more efficient implementations possible: $O(n\log n)$

message complexity:

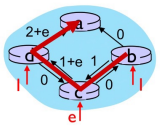
- each router must *broadcast* its link state information to other *n* routers
- efficient (and interesting!) broadcast algorithms: $O(n)$ link crossings to disseminate a broadcast message from one source
- each router's message crosses $O(n)$ links: overall message complexity: $O(n^2)$

Oscillation

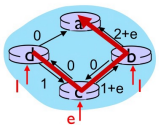
- when link costs depend on traffic volume, route oscillations possible
- sample scenario:
  - routing to destination a, traffic entering at d, c, e with rates 1, e (<1), 1
  - link costs are directional, and volume-dependent



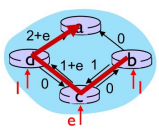
initially



given these costs, find new routing…. resulting in new costs



given these costs, find new routing…. resulting in new costs



given these costs, find new routing…. resulting in new costs

Always new route based on evolving routes.

5.2.2 Distance Vector. (rarely in reality)

Based on *Bellman-Ford* (BF) equation (dynamic programming):

Bellman-Ford equation

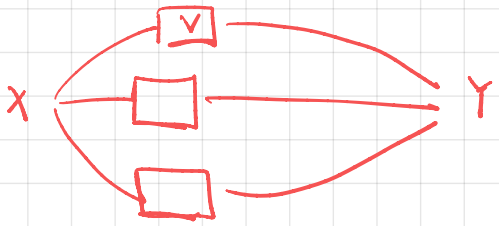
Let $D_x(y)$: cost of least-cost path from x to y.
Then:

$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

- v's estimated least-cost-path cost to y
- *min* taken over all neighbors v of x
- direct cost of link from x to v

Routing decision is made based on the distance, not cost as discussed in 5.2.1. (bandwidth)

Known cost + predicted cost.



D: shortest distance to z.

key idea:

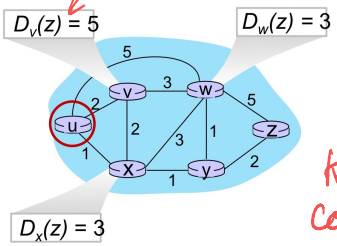
- from time-to-time, each node sends its own distance vector estimate to neighbors (Every 30s interval.)
- when x receives new DV estimate from any neighbor, it updates its own DV using B-F equation:

  $D_x(y) \leftarrow \min_v\{c_{x,v} + D_v(y)\}$ for each node $y \in N$

- under minor, natural conditions, the estimate $D_x(y)$ *converge to the actual least cost* $d_x(y)$

Only immediate neighbors.

Suppose that u's neighboring nodes, x,v,w, know that for destination z:



Bellman-Ford equation says:

$$D_u(z) = \min \{ c_{u,v} + D_v(z), \; c_{u,x} + D_x(z), \; c_{u,w} + D_w(z) \}$$
$$= \min \{ 2 + 5, \; 1 + 3, \; 5 + 3 \} = 4$$

Known Condition.

node achieving minimum (x) is next hop on estimated least-cost path to destination (z)

Disadvantage:

③ Max 15 hops, will treated as ∞ if over 15.

Problems with Distance Vector

- A number of problems can occur in a network using distance vector algorithm
- Most of these problems are caused by slow convergence or routers converging on incorrect information
- Convergence is the time during which all routers come to an agreement about the best paths through the internetwork
  - whenever topology changes there is a period of instability in the network as the routers converge
- Reacts rapidly to good news, but leisurely to bad news

① Then everyone will use this route.

② Too slow/long (unacceptable) to pass these info. across different routers.

2A. Good news: Cost less. ⇒ Pass rapidly
Bad news: Cost ↑ ⇒ Time ↑.

Count to Infinity Problem 时间差问题

- R3 disconnected with R4.
- R3 captured this change. and going to inform R2, however, since they "talk" every 30s. R3 need to wait.
- When they talk, R2 told R3: "I can access R4 with 2 cost" then R3 recompute $\min\{2, \infty\}$ update → R3 to R4 with 3 cost, then pass it to R2.
- R2 update to 4 cost. then pass back to R3.

..... Infinity loop

"Poisoned - Reverse" Rule. (Solution)

The "Poisoned Reverse" Rule

- Heuristic to avoid count-to-infinity (Looping)
- If B routes via C to get to A:
  - B tells C its (B's) distance to A is infinite (so C won't route to A via B)

Pass the information immediately to neighbor.

↳ False information, which can represents ∞ distance. Since max hop is 15, anything above 15 is ∞. So we could send $D(R_2 \to R_4) = 16$.

Comparison between Link State and Distance Vector.

message complexity

LS: n routers, $O(n^2)$ messages sent

DV: exchange between neighbors; convergence time varies

speed of convergence

LS: $O(n^2)$ algorithm, $O(n^2)$ messages
- may have oscillations

DV: convergence time varies
- may have routing loops
- count-to-infinity problem

robustness: what happens if router malfunctions, or is compromised?

LS:
- router can advertise incorrect *link* cost
- each router computes only its *own* table

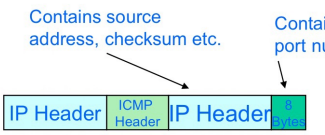
DV:
- DV router can advertise incorrect *path* cost ("I have a *really* low cost path to everywhere"): black-holing
- each router's table used by others: error propagate thru network

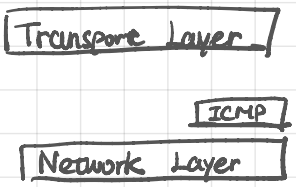
Protocol Used in Reality.

Link State	*Distance Vector*
Open Shortest Path First (OSPF) 90%	Routing Information Protocol (RIP)
Intermediate system to intermediate system (IS-IS)	Interior Gateway Routing Protocol (IGRP-Cisco)
	Border Gateway Protocol (BGP) - variant

5.6. ICMP : Internet Control Message Protocol.

- Used by hosts & routers to communicate network level infromation (· Test the connectivity)
  - Error reporting: unreachable host, network, port (· Report/pass error msg.)
  - Echo request/reply (used by ping)
- Works above IP layer
  - ICMP messages carried in IP datagrams
- ICMP message: type, code plus IP header and first 8 bytes of IP datagram payload causing error





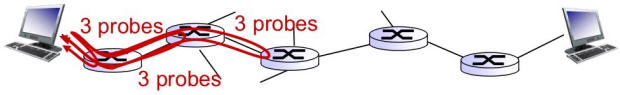
Type	Code	Description
0	0	echo reply(ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	3	dest port unreachable
3	4	frag needed; DF set
8	0	echo request(ping)
11	0	TTL expired
11	1	frag reassembly time exceeded
12	0	bad IP header

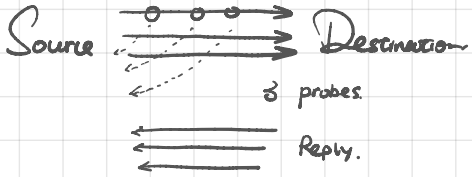
Traceroute and ICMP

- Source sends series of ~~UDP~~ ICMP protocol segments to dest
  - first set has TTL =1
  - second set has TTL=2, etc.
  - unlikely port number
- When *n*th set of datagrams arrives to nth router:
  - router discards datagrams
  - and sends source ICMP messages (type 11, code 0)
  - ICMP messages includes IP address of router
- when ICMP messages arrives, source records RTTs

stopping criteria:

- UDP segment eventually arrives at destination host
- destination returns ICMP "port unreachable" message (type 3, code 3)
- source stops





Calculate the TTL time.

6. Data Link Layer

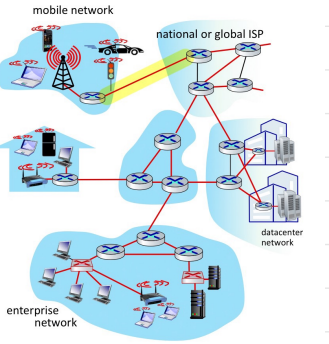
6.1 Introduction

terminology: Equivalent.

- hosts and routers: nodes
- communication channels that connect adjacent nodes along communication path: links
  - wired
  - wireless
  - LANs
- layer-2 packet: frame, encapsulates datagram

Header | Eth II | Datagram | ← frame. →

link layer has responsibility of transferring datagram from one node to physically adjacent node over a link

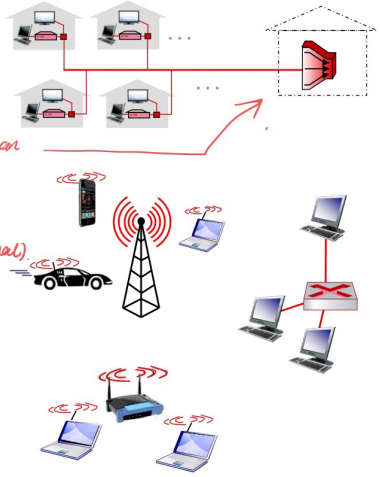


8

Link layer: services

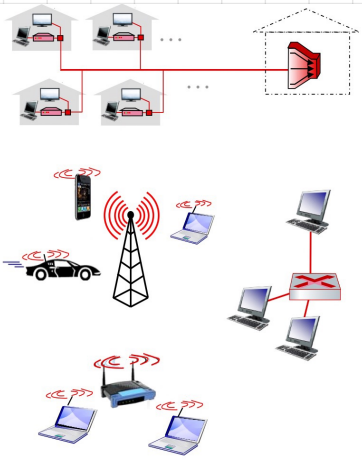
- framing, link access:
  - encapsulate datagram into frame, adding header, trailer → At the end.
  - channel access if shared medium SO only one user can send bit a time.
  - "MAC" addresses in frame headers identify source, destination (different from IP address!)
- reliable delivery between adjacent nodes (Optional)
  - we already know how to do this!
  - seldom used on low bit-error links
  - wireless links: high error rates
    - Q: why both link-level and end-end reliability?

From start of link to end of link.



- flow control: (Option)
  - pacing between adjacent sending and receiving nodes
- error detection: (Must).
  - errors caused by signal attenuation, noise.
  - receiver detects errors, signals retransmission, or drops frame
- error correction:
  - receiver identifies and corrects bit error(s) without retransmission
- half-duplex and full-duplex:
  - with half duplex, nodes at both ends of link can transmit, but not at same time

↳ Cannot send while receiving, vice-versa.

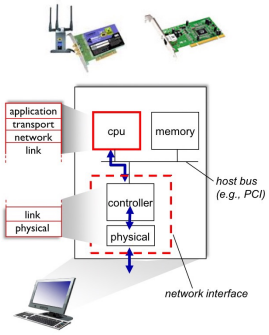


11

Network Interface Card

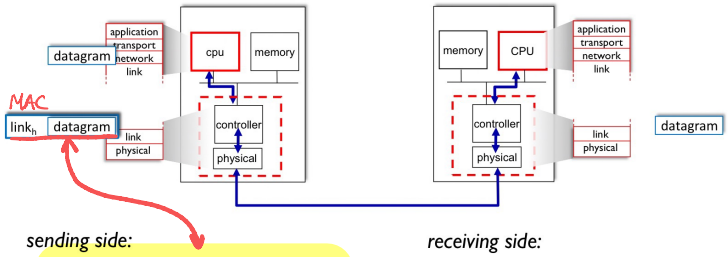
Where is the link layer implemented?

- in each-and-every host
- link layer implemented in network interface card (NIC) or on a chip
  - Ethernet, WiFi card or chip
  - implements link, physical layer
- attaches into host's system buses
- combination of hardware, software, firmware



Interface Communicating.

Interfaces communicating

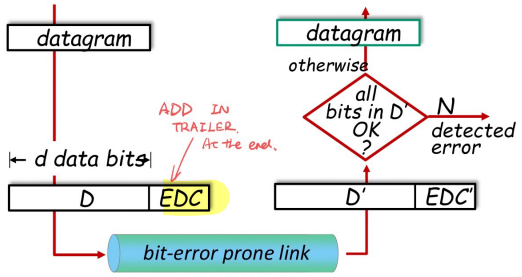


sending side:
- *encapsulates datagram in frame*
- *adds error checking bits, reliable data transfer, flow control, etc.*

receiving side:
- *looks for errors, reliable data transfer, flow control, etc.*
- *extracts datagram, passes to upper layer at receiving side*

6.2. Error Detection, Correction.

EDC: error detection and correction bits (e.g., redundancy)
D: data protected by error checking, may include header fields



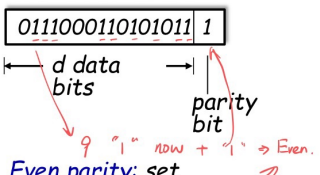
Error detection not 100% reliable!
- protocol may miss some errors, but rarely
- larger EDC field yields better detection and correction

There are several algo for EDC

Parity checking

UPGRADE.

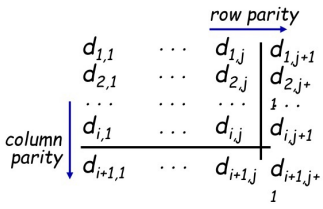
single bit parity:
- detect single bit errors



Even Parity.
Odd Parity.

However, if two bits flip, this cannot be detected.

two-dimensional bit parity:
- detect *and correct* single bit errors



no errors:
1 0 1 0 1 | 1
1 1 1 1 0 | 0
0 1 1 1 0 | 1
0 0 1 0 1 | 0

detected and correctable single-bit error:
1 0 1 0 1 1
1 0 1 1 0 0 ← parity error
0 1 1 1 0 1
0 0 1 0 1 0
parity error

Not only detect, But also Able to Locate the error

However, we wish to save space:

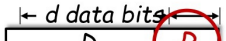
hence,

Cyclic Redundancy Check (CRC).

Cyclic Redundancy Check (CRC) BEST.

- more powerful error-detection coding
- D: data bits (given, think of these as a binary number)
- G: bit pattern (generator), of r+1 bits (given) $G = r+1$

r CRC bits



bit pattern

<D,R> = D* 2^r XOR R — formula for bit pattern

尾部长度 尾部长度+1. [Data + (尾部长度+1)] % G = 0.

goal: choose r CRC bits, R, such that <D,R> exactly divisible by G (mod 2)

- receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
- can detect all burst errors less than r+1 bits
- widely used in practice (Ethernet, 802.11 WiFi)

CRC example

same ⇒ 0
diff ⇒ 1

want:

$D \cdot 2^r$ XOR R = nG

Division.

equivalently:

$D \cdot 2^r$ = nG XOR R

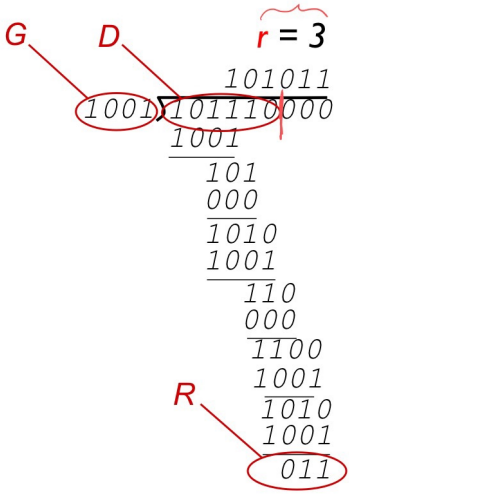
equivalently:

if we divide $D \cdot 2^r$ by G, want remainder R to satisfy:

$$R = \text{remainder}\left[\frac{D \cdot 2^r}{G}\right]$$

At the sender

Error checksum 3 bits.



Sender sends <D, R> into the channel

6.3. Multiple Access Protocols.

two types of "links":

- point-to-point
  - point-to-point link between Ethernet switch, host
  - PPP for dial-up access
- broadcast (shared wire or medium)
  - old-fashioned Ethernet
  - upstream HFC in cable-based access network
  - 802.11 wireless LAN, 4G/4G. satellite

- single shared broadcast channel
- two or more simultaneous transmissions by nodes: interference
  - collision if node receives two or more 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!
  - no out-of-band channel for coordination

The most ideal situation.

given: multiple access channel (MAC) of rate R bps

desired properties:

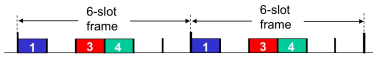
1. when one node wants to transmit, it can send at rate R.
2. when M nodes want to transmit, each can send at average rate R/M
3. fully decentralized:
   - no special node to coordinate transmissions
   - no synchronization of clocks, slots
4. simple

There are 3 broad classes for solving this issue
- Channel Partitioning.
- Random Access
- Taking Turns.

6.3.1. Channel Partitioning.

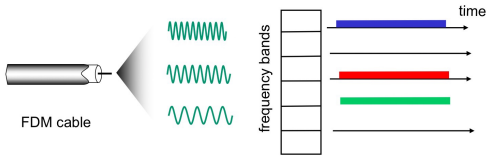
TDMA: time division multiple access Each user get 1/6. not ideal.

- access to channel in "rounds"
- each node gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-node LAN, 1,3,4 have packets to send, slots 2,5,6 idle



FDMA: frequency division multiple access if idle, but space has allocated, overall efficiency ↓.

- channel spectrum divided into frequency bands
- each node assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: 6-node LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle



Only ideal ① can be satisfied.

6.3.2. Random Access : CSMA (Carrier Sense Multiple Access)

simple **CSMA:** listen before transmit:

- *if channel sensed idle:* transmit entire frame
- *if channel sensed busy:* defer transmission

When 2 users check at the same time, found idle. ↓ Collision.

- human analogy: don't interrupt others!

CSMA/CD: CSMA with collision detection

- collisions detected within short time
- colliding transmissions aborted, reducing channel wastage
- collision detection easy in wired, difficult with wireless

Then wait random amount of time.

- human analogy: the polite conversationalist

Ethernet CSMA/CD algorithm

1. NIC receives datagram from network layer, creates frame
2. If NIC senses channel:
   - if idle: start frame transmission.
   - if busy: wait until channel idle, then transmit
3. If NIC transmits entire frame without collision, NIC is done with frame !
4. If NIC detects another transmission while sending: abort, send jam signal — Let everyone know collision happens.
5. After aborting, NIC enters binary (exponential) backoff: random wait.
   - 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
   - more collisions: longer backoff interval

NIC = Network Interface Card

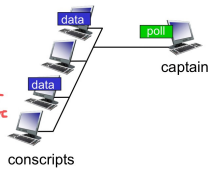
Good, only ideal ② cannot be satisfied.
↳ Each can send at average rate of R/M.

6.3.3. Taking Turns.

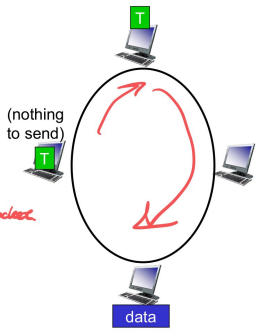
"Taking turns" MAC protocols

Controller regulate rules.

polling:
- captain node "invites" other nodes to transmit in turn
- typically used with "dumb" devices
- concerns:
  - polling overhead — captain need to inform conscripts first.
  - latency — wait for my turn
  - single point of failure (captain) — possible fail.



token passing: (IBM)
- control token passed from one node to next sequentially.
- token message
- concerns:
  - token overhead — Extra control packet
  - latency
  - single point of failure (token)



Only ideal ③ cannot satisfied,
↳ Fully decentralized.

6.4. LANs

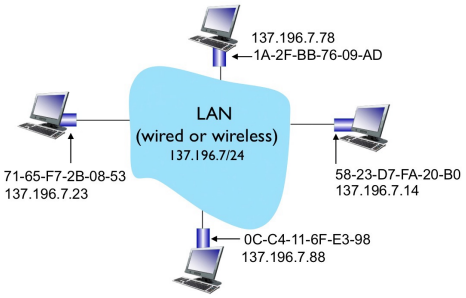
6.4.1. Addressing, ARP

IP Address vs. Mac Address

- 32-bit IP address:
  - network-layer address for interface
  - used for layer 3 (network layer) forwarding
  - e.g.: 128.119.40.136
- MAC (or LAN or physical or Ethernet) address:
  - function: used "locally" to get frame from one interface to another physically-connected interface (same subnet, in IP-addressing sense)
  - 48-bit MAC address (for most LANs) burned in NIC ROM, also sometimes software settable
  - e.g.: 1A-2F-BB-76-09-AD — hexadecimal (base 16) notation (each "numeral" represents 4 bits)

- MAC addresses (used in link-layer)
  - Hard-coded in read-only memory when adapter is built
  - Flat name space of 48 bits (e.g., 00-0E-9B-6E-49-76)
  - Portable, and can stay the same as the host moves
  - Used to get packet between interfaces on same network
- IP addresses
  - learned dynamically
  - Hierarchical name space of 32 bits (e.g., 12.178.66.9)
  - Not portable, and depends on where the host is attached
  - Used to get a packet to destination IP subnet

each interface on LAN
- has unique 48-bit MAC address
- has a locally unique 32-bit IP address (as we've seen)



Layer	Examples	Structure	Configuration	Resolution Service
App. Layer	www.cse.unsw.edu.au	organizational hierarchy	~ manual	DNS
Network Layer	129.94.242.51	topological hierarchy	DHCP	
Link layer	45-CC-4E-12-F0-97	vendor (flat)	hard-coded	ARP

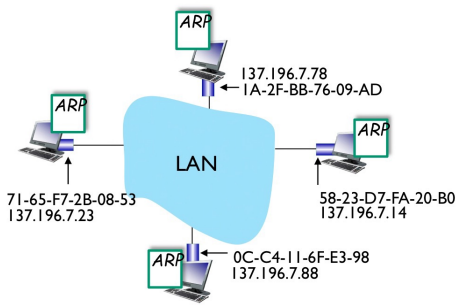
Transport Layer: port number.

Convert web domain → IP

Convert IP → MAC

ARP: Address Resolution Protocol

Question: how to determine interface's MAC address, knowing its IP address?



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

Every host has table.

- IP/MAC address mappings for some LAN nodes:

  < IP address; MAC address; TTL>
- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

What if no this record in the table ?

example: A wants to send datagram to B

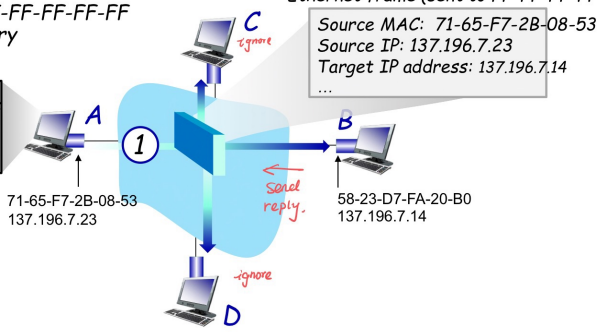
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address

A broadcasts ARP query, containing B's IP addr
- destination MAC address = FF-FF-FF-FF-FF-FF
- all nodes on LAN receive ARP query

(1)

ARP table in A

IP addr	MAC addr	TTL



Ethernet frame (sent to FF-FF-FF-FF-FF-FF)

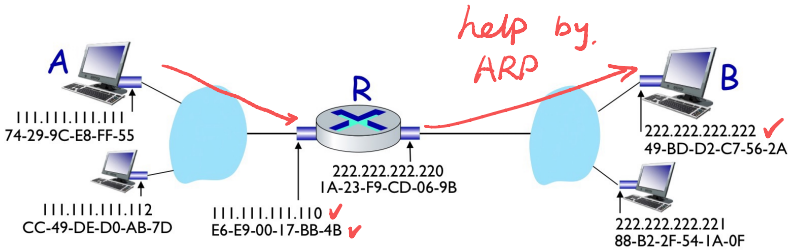
Source MAC: 71-65-F7-2B-08-53
Source IP: 137.196.7.23
Target IP address: 137.196.7.14
...

58

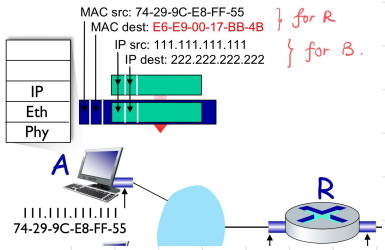
Another scenario: Routing to another subnet.

walkthrough: sending a datagram from A to B via R

- focus on addressing – at IP (datagram) and MAC layer (frame) levels
- assume that:
  - A knows B's IP address (how does A know that the next-hop is Router R?)
  - A knows IP address of first hop router, R (how?)
  - A knows R's MAC address (how?)



default gateway.
eg 192.168.1.1

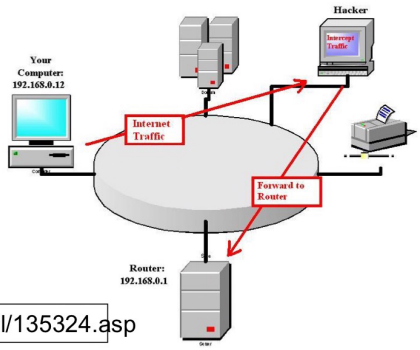


Security Issue: ARP Cache Poisoning.

Security Issues: ARP Cache Poisoning

- Denial of Service - Hacker replies back to an ARP query for a router NIC with a fake MAC address
- Man-in-the-middle attack - Hacker can insert his/her machine along the path between victim machine and gateway router
- Such attacks are generally hard to launch as hacker needs physical access to the networ

Solutions -
- Use Switched Ethernet with port security enabled (i.e., one host MAC address per switch port)
- Adopt static ARP configuration for small size networks
- Use ARP monitoring tools such as ARPWatch



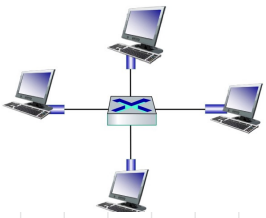
http://www.watchguard.com/infocenter/editorial/135324.asp

6.4.2 Ethernet

Dominate wired LAN technology:

- *bus:* popular through mid 90s
  - all nodes in same collision domain (can collide with each other)
- *switched:* prevails today
  - active link-layer 2 switch in center
  - each "spoke" runs a (separate) Ethernet protocol (nodes do not collide with each other)

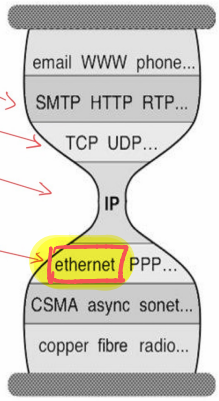
bus: coaxial cable



switched

Recall that:

- *application:* supporting network applications
  - FTP, SMTP, HTTP, Skype, ..
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.11(WiFi), PPP
- *physical:* bits "on the wire"

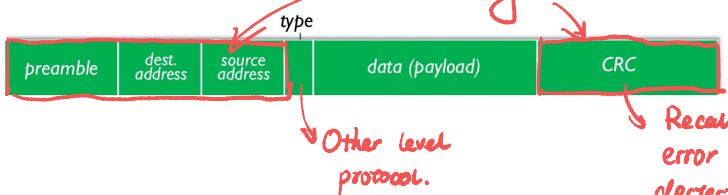


Within this layer,

Ethernet frame structure

sending interface encapsulates IP datagram (or other network layer protocol packet) in Ethernet frame

Adding.



Other level protocol.

Recall, error detection.

preamble:

- used to synchronize receiver, sender clock rates
- 7 bytes of 10101010 followed by one byte of 10101011

- *addresses:* 6 byte source, destination MAC addresses
  - if adapter receives frame with matching destination address, or with broadcast address (e.g., ARP packet), it passes data in frame to network layer protocol
  - otherwise, adapter discards frame
- *type:* indicates higher layer protocol
  - mostly IP but others possible, e.g., Novell IPX, AppleTalk
  - used to demultiplex up at receiver
- *CRC:* cyclic redundancy check at receiver
  - error detected: frame is dropped

Ethernet: unreliable, connectionless

- *connectionless:* no handshaking between sending and receiving NICs
- *unreliable:* receiving NIC doesn't send ACKs or NAKs to sending NIC
  - data in dropped frames recovered only if initial sender uses higher layer rdt (e.g., TCP), otherwise dropped data lost
- Ethernet's MAC protocol: unslotted CSMA/CD with binary backoff

6.4.3. Switches

- Switch is a link-layer device: it takes an active role
  - store, forward Ethernet frames
  - examine incoming frame's MAC address, selectively forward frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment

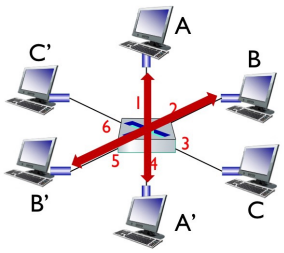
↗ learn new device.

- transparent: hosts unaware of presence of switches
- plug-and-play, self-learning
  - switches do not need to be configured

Properties.

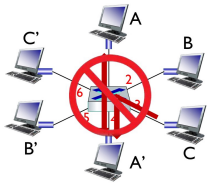
Switch: multiple simultaneous transmissions

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on each incoming link, so:
  - 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)

- switching: A-to-A' and B-to-B' can transmit simultaneously, without collisions
  - but A-to-A' and C to A' can not happen simultaneously



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

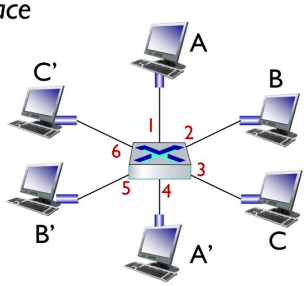
No 2 A's connections.

Switch forwarding table

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!

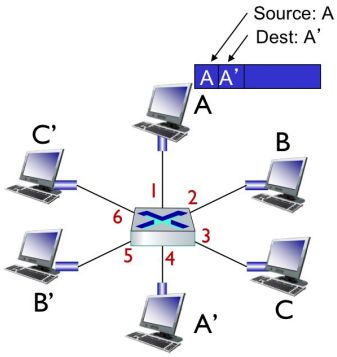


Q: how are entries created, maintained in switch table?

- something like a routing protocol?

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*



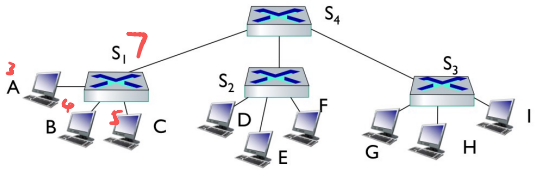
Broadcast desn : "where is A' ?"
↳ reply → learn its MAC.

MAC addr	interface	TTL
A	1	60

Switch table (initially empty)

Interconnecting switches

self-learning switches can be connected together:



MAC | Port Num | TTL
A | 3
B | 4
C | 5
D | 7
E | 7
F | 7
G | 7

Always the MAC of switch.

Q: sending from A to G - how does S_1 know to forward frame destined to G via S_4 and S_3?

- *A: self learning! (works exactly the same as in single-switch case!)*

83

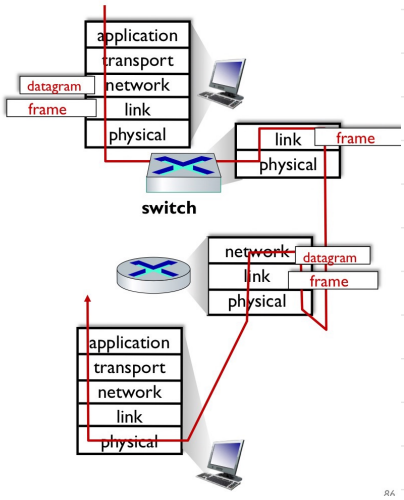
Switches vs. routers

both are store-and-forward:

- *routers: network-layer devices (examine network-layer headers)* Plan the route.
- *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* Go thro route.



84

Security Issues

- In a switched LAN once the switch table entries are established frames are not broadcast
  - Sniffing frames is harder than pure broadcast LANs
  - Note: attacker can still sniff broadcast frames and frames for which there are no entries (as they are broadcast)
- Switch Poisoning: Attacker fills up switch table with bogus entries by sending large # of frames with bogus source MAC addresses fause
- Since switch table is full, genuine packets frequently need to be broadcast as previous entries have been wiped out

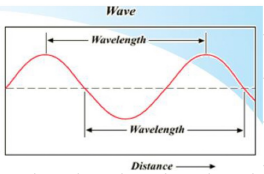
T. Wireless Network.
T.1. Introduction.

- **Frequency/Wave-Length –**

C is the speed of light
f is frequency
λ (lambda) is wavelength

Wavelength $\lambda = \frac{C}{f}$

Frequency $f = \frac{C}{\lambda}$



$f \uparrow \rightarrow$ Speed ↑.
(bandwidth)
↓
shorter distance.