Computer Networking

Slide Set 1

Andrew W. Moore

Andrew.Moore@cl.cam.ac.uk

Computer Networking

Slide Set 1

Andrew W. Moore
Andrew.Moore@cl.cam.ac.uk

Topic 1 Foundation

- Administrivia
- Networks
- Channels
- Multiplexing
- Performance: loss, delay, throughput

Course Administration

Commonly Available Texts

- Computer Networks: A Systems Approach
  Peterson and Davie
  https://book.systemsapproach.org
  https://github.com/SystemsApproach/book
- Computer Networking : Principles, Protocols and Practice
  Olivier Bonaventure (and friends)
  Less GitHub but more practical exercises
  https://www.computer-networking.info/

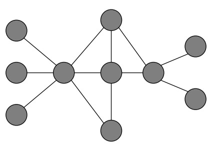
Other textbooks are available.

Thanks

- Slides are a fusion of material from
  to Stephen Strowes, Tilman Wolf & Mike Zink, Ashish Padalkar , Evangelia Kalyvianaki, Brad Smith, Ian Leslie, Richard Black, Jim Kurose, Keith Ross, Larry Peterson, Bruce Davie, Jen Rexford, Ion Stoica, Vern Paxson, Scott Shenker, Frank Kelly, Stefan Savage, Jon Crowcroft , Mark Handley, Sylvia Ratnasamy, Adam Greenhalgh, and Anastasia Courtney.
- Supervision material is drawn from
  Stephen Kell, Andy Rice, and the TA teams of 144 and 168
- Finally thanks to the fantastic past Part 1b students and Andrew Rice for all the tremendous feedback.

What is a network?

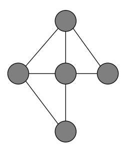
- A system of “links” that interconnect “nodes” in order to move “information” between nodes



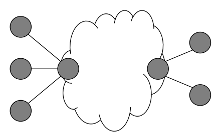
- Yes, this is all rather abstract

What is a network?

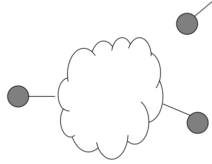
- We also talk about



or



or even



- Yes, abstract, vague, and under-defined….

There are *many* different types of networks

- Internet
- Telephone network
- Transportation networks
- Cellular networks
- Supervisory control and data acquisition networks
- Optical networks
- Sensor networks

We will focus almost exclusively on the Internet

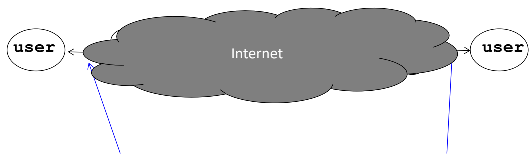
The Internet has transformed everything

- The way we do business
  - E-commerce, advertising, cloud-computing
- The way we have relationships
  - Facebook friends, E-mail, IM, virtual worlds
- The way we learn
  - Wikipedia, search engines
- The way we govern and view law
  - E-voting, censorship, copyright, cyber-attacks

A few defining characteristics of the Internet

A federated system

- The Internet ties together different networks
  - >20,000 ISP networks (the definition is fuzzy)



Tied together by IP -- the "Internet Protocol" : a single common interface between users and the network and between networks

A federated system

- The Internet ties together different networks
  - >20,000 ISP networks

- A single, common interface is great for interoperability…
- …but tricky for business

- Why does this matter?
  - ease of interoperability is the Internet's most important goal
  - practical realities of incentives, economics and real-world trust, drive topology, route selection and service evolution

Tremendous scale (2020 numbers – so some 'weird')

- 4.57 Billion users (58% of world population)
- 1.8 Billion web sites
  - 34.5% of which are powered by the WordPress!
- 4.88 Billion smartphones (45.4% of population)
- 500 Million Tweets a day
- 100 Billion WhatsApp messages per day
- 1 Billion hours of YouTube video watched per day
- 500 hours of Youtube video added per minute
- 2+ billion TikTok installs
- 60% video streaming
  - 12.5% of the Internet traffic is native Netflix

Tremendous scale (2020 numbers – so some ‘weird’)

“Internet Scale” refers to such systems

- 4.57 Billion users (58% of world populatio
- 1.8 Billion web sites
  - 34.5% of which are powere ress!
- 4.88 Billion smar % of population)
- 500 Mill day
- WhatsApp messages per day
- Billion hours of YouTube video watched per day
- 500 hours of Youtube video added per minute
- 2+ billion TikTok installs
- 60% video streaming
  - 12.5% of the Internet traffic is native Netflix

Enormous diversity and dynamic range

- Communication latency: nanoseconds to seconds (10^9)
- Bandwidth: 100bits/second to 400 Gigabits/second (10^9)
- Packet loss: 0 – 90%

- Technology: optical, wireless, satellite, copper

- Endpoint devices: from sensors and cell phones to datacenters and supercomputers
- Applications: social networking, file transfer, skype, live TV, gaming, remote medicine, backup, IM
- Users: the governing, governed, operators, malicious, naïve, savvy, embarrassed, paranoid, addicted, cheap …

Constant Evolution

1970s:
- 56kilobits/second “backbone” links
- <100 computers, a handful of sites in the US (and one UK)
- Telnet and file transfer are the “killer” applications

Today
- 400+Gigabits/second backbone links
- 40B+ devices, all over the globe
  - 27B+ IoT devices alone

Asynchronous Operation

- Fundamental constraint: **speed of light**
- Consider:
  - How many cycles does your 3GHz CPU in Cambridge execute before it can possibly get a response from a message it sends to a server in Palo Alto?
    - Cambridge to Palo Alto: 8,609 km
    - Traveling at 300,000 km/s: 28.70 milliseconds
    - Then back to Cambridge: 2 x 28.70 = 57.39 milliseconds
    - 3,000,000,000 cycles/sec * 0.05739 = 172,179,999 cycles!
- Thus, communication feedback is always *dated*

How much can change with 172 Million instructions

Prone to Failure

- To send a message, **all** components along a path must function correctly
  - software, wireless access point, firewall, links, network interface cards, switches,…
  - Including human operators
- Consider: 50 componentsin a system, each working correctly 99% of time → 39.5% chance communication will fail
- Plus, recall
  - scale → lots of components
  - asynchrony → takes a long time to hear (bad) news
  - federation (**inter**net) → hard to identify fault or assign blame

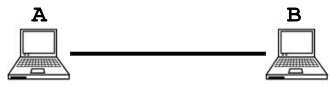
Recap: The Internet is…

- A complex federation
- Of enormous scale
- Dynamic range
- Diversity
- Constantly evolving
- Asynchronous in operation
- Failure prone
- Constrained by what’s practical to engineer
- Too complex for (simple) theoretical models
- “Working code” doesn’t mean much
- Performance benchmarks are too narrow

An Engineered System

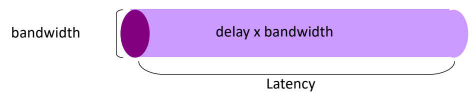
- Constrained by what technology is practical
  - Link bandwidths
  - Switch port counts
  - Bit error rates
  - Cost
  - …

Nodes and Links



Channels = Links
Peer entities = Nodes

Properties of Links (Channels)



- Bandwidth (capacity): “width” of the links
  - number of bits sent (or received) per unit time (bits/sec or bps)
- Latency (delay): “length” of the link
  - propagation time for data to travel along the link (seconds)
- Bandwidth-Delay Product (BDP): “volume” of the link
  - amount of data that can be “in flight” at any time
  - propagation delay × bits/time = total bits in link

Examples of Bandwidth-Delay

- Same city over a slow link:
  - BW~100Mbps
  - Latency~10msec
  - BDP ~ 10^6bits ~ 125KBytes

  17km * c = 56µs << 10ms
- To California over a fast link:
  - BW~10Gbps
  - Latency~140msec
  - BDP ~ 1.4×10^9bits ~ 175MBytes

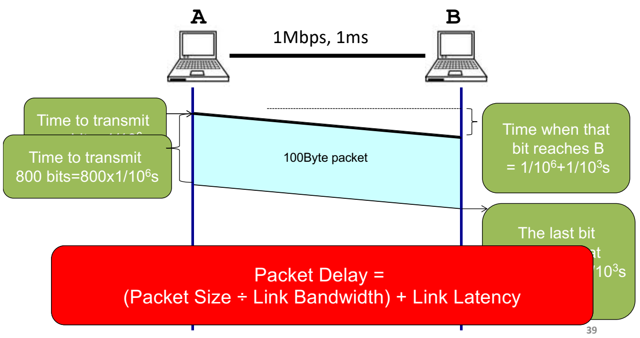
  9708km * c = 32ms << 140ms
- Intra Datacenter:
  - BW~100Gbps
  - Latency~30usec
  - BDP ~ 10^6bits ~ 375KBytes

  750m * c = 56µs ≅ 30µs
- Intra Host:
  - BW~100Gbps
  - Latency~16nsec
  - BDP ~ 1600bits ~ 200Bytes

  25cm * c = 83ps << 16ns

Packet Delay

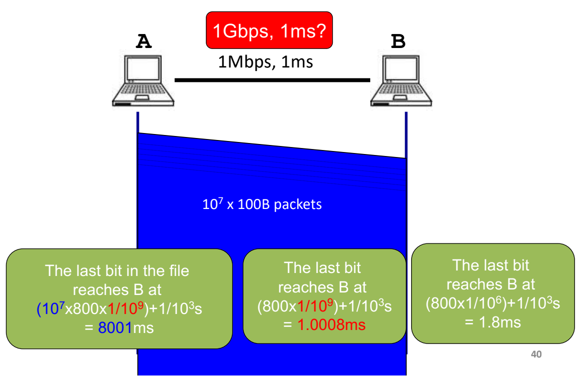
Sending a 100B packet from A to B?



Packet Delay =
(Packet Size ÷ Link Bandwidth) + Link Latency

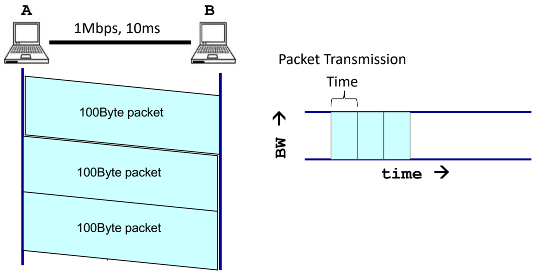
1GB file in 100B packets ay

Sending a 100B packet from A to B?



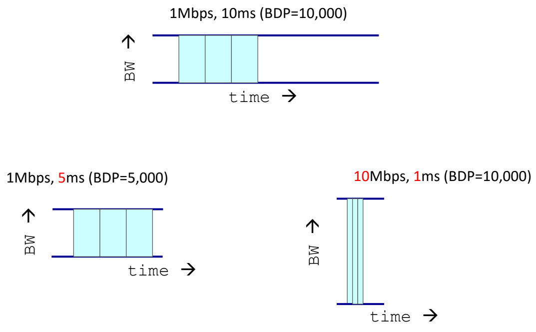
Packet Delay: The “pipe” view

Sending 100B packets from A to B?



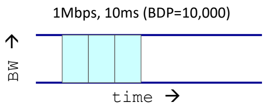
Packet Delay: The “pipe” view

Sending 100B packets from A to B?

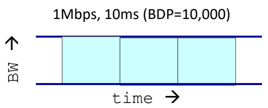


Packet Delay: The “pipe” view

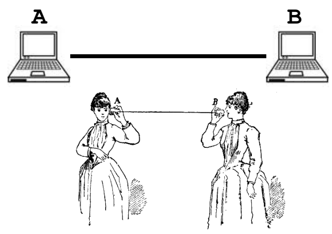
Sending 100B packets from A to B?



What if we used *200Byte packets??*

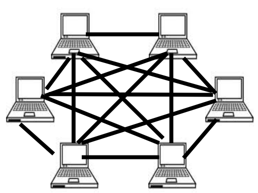


Recall Nodes and Links



What if we have more nodes?

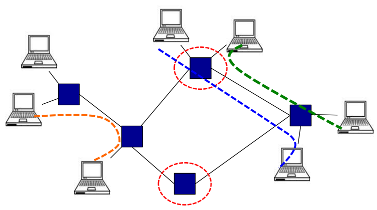
One link for every node?



Need a <u>scalable</u> way to interconnect nodes

Solution: A switched network

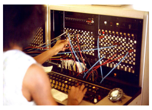
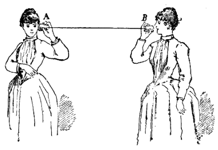
Nodes <u>share</u> network link resources



How is this sharing implemented?

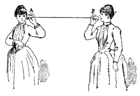
Two examples of switched networks

- Circuit switching (used in the *POTS*: Plain Old Telephone system)

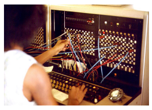



- Packet switching (used in the Internet)

Circuit switching

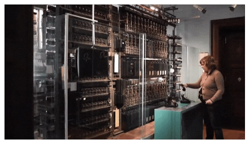



Telephone


Exchange

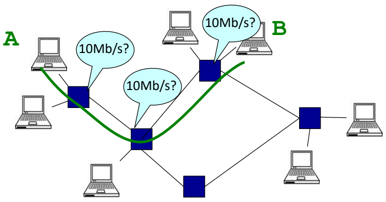

Exchange


Telephone

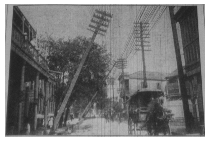
Circuit switching

Idea: source reserves network capacity along a path



(1) Node A sends a reservation request
(2) Interior switches establish a connection -- i.e., "circuit"
(3) A starts sending data
(4) A sends a "teardown circuit" message

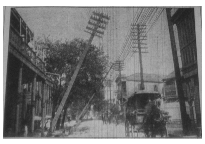
Multiplexing



Sharing makes things efficient (cost less)

- One airplane/train for 100's of people
- One telephone for many calls
- One lecture theatre for many classes
- One computer for many tasks
- One network for many computers
- One datacenter many applications

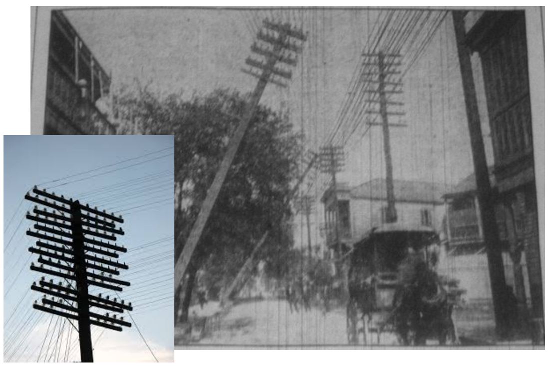
Multiplexing



Sharing makes things efficient (cost less)

- One airplane/train for 100's of people
- One telephone for many calls
- One ~~lecture theatre~~ Lecturer? for many classes
- One computer for many tasks
- One network for many computers
- One datacenter many applications

Old Time Multiplexing

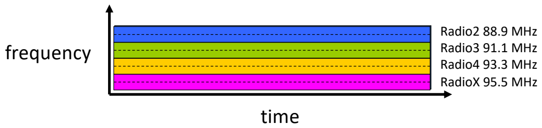


Sharing Circuit Switching: FDM and TDM

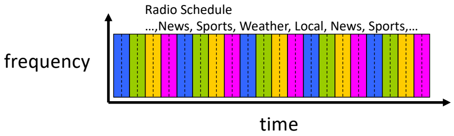
Frequency Division Multiplexing

Example:

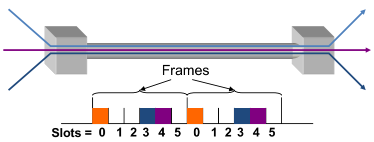
4 users



Time Division Multiplexing

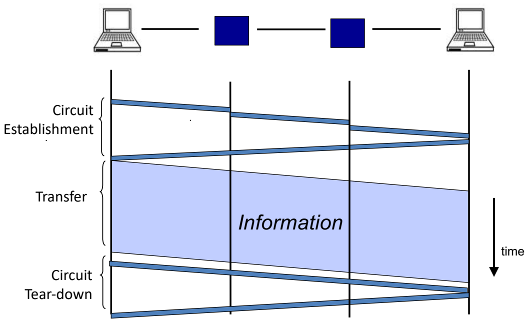


Time-Division Multiplexing/Demultiplexing



- Time divided into frames; frames into slots
- Relative slot position inside a frame determines to which conversation data belongs
  - e.g., slot 0 belongs to orange conversation
- Slots are reserved (released) during circuit setup (teardown)
- If a conversation does not use its circuit **capacity is lost!**

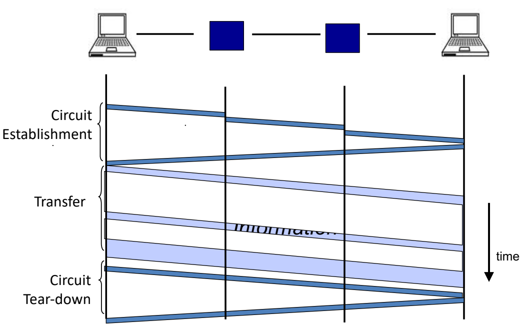
Timing in Circuit Switching



Circuit switching: pros and cons

- Pros
  - guaranteed performance
  - fast transfer (once circuit is established)
- Cons

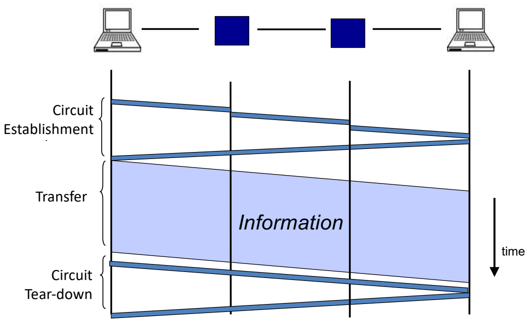
Timing in Circuit Switching



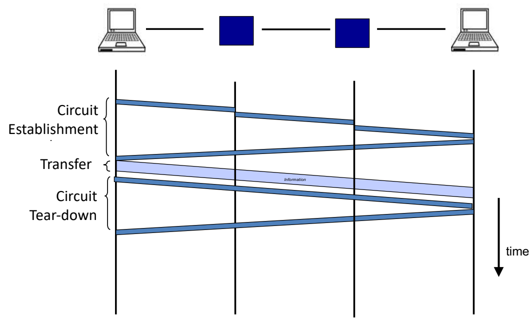
Circuit switching: pros and cons

- Pros
  - guaranteed performance
  - fast transfer (once circuit is established)
- Cons
  - **wastes bandwidth if traffic is “bursty”**

Timing in Circuit Switching



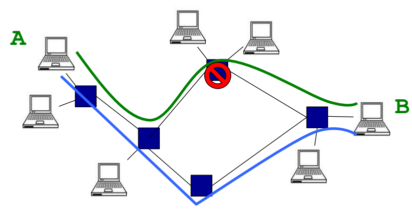
Timing in Circuit Switching



Circuit switching: pros and cons

- Pros
  - guaranteed performance
  - fast transfers (once circuit is established)
- Cons
  - wastes bandwidth if traffic is "bursty"
  - **connection setup time is overhead**

Circuit switching



Circuit switching doesn't "route around failure"

Circuit switching: pros and cons

- Pros
  - guaranteed performance
  - fast transfers (once circuit is established)
- Cons
  - wastes bandwidth if traffic is "bursty"
  - connection setup time is overhead
  - **recovery from failure is slow**

Numerical example

- How long does it take to send a file of 640,000 bits from host A to host B over a circuit-switched network?
  - All links are 1.536 Mbps
  - Each link uses TDM with 24 slots/sec
  - 500 msec to establish end-to-end circuit

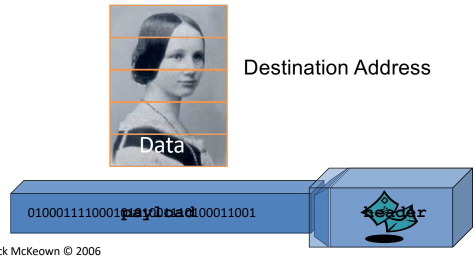
Let's work it out!

Two examples of switched networks

- Circuit switching (used in the *POTS*: Plain Old Telephone system)
- Packet switching (used in the Internet)

Packet Switching

- Data is sent as chunks of formatted bits (Packets)
- Packets consist of a "header" and "payload"*



After Nick McKeown © 2006

Packet Switching

- Data is sent as chunks of formatted bits (Packets)
- Packets consist of a "header" and "payload"*
  - payload is the data being carried
  - header holds instructions to the network for how to handle packet (think of the header as an API)
  - In this example, the header has a destination address
  - More complex headers may include
    - How this traffic should be handled? (first class, second class, etc)
    - Do I acknowledge this? Who signed for it?
    - Were the contents ok?

Packet Switching

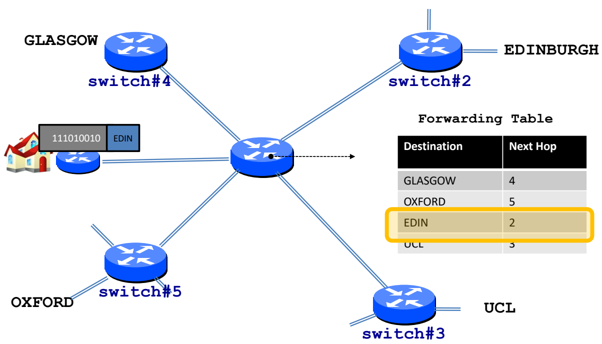
- Data is sent as chunks of formatted bits (Packets)
- Packets consist of a "header" and "payload"
- Switches "forward" packets based on their headers

A switch looks at the header and

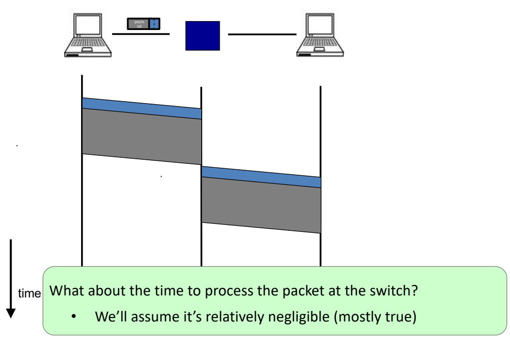
immediately decides which physical port

In a switch: address maps to port

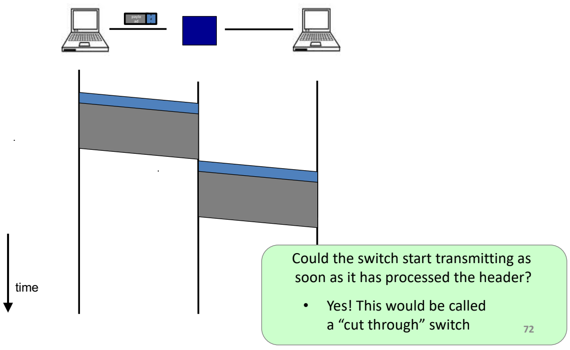
Switches forward packets



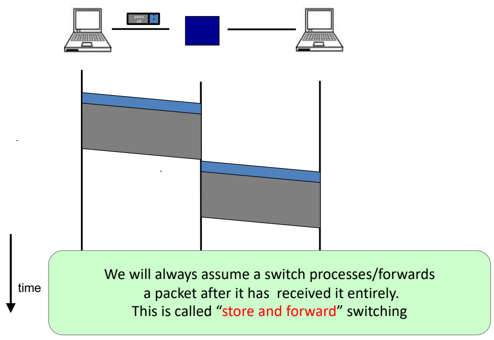
Timing in Packet Switching



Timing in Packet Switching



Timing in Packet Switching



Packet Switching

- Data is sent as chunks of formatted bits (Packets)
- Packets consist of a "header" and "payload"
- Switches "forward" packets based on their headers

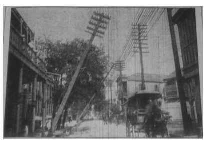
Packet Switching

- 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 notion of packets belonging to a "circuit"

Packet Switching

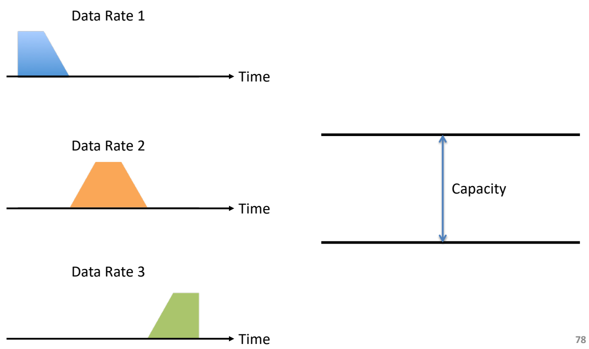
- Data is sent as chunks of formatted bits (Packets)
- Packets consist of a "header" and "payload"
- Switches "forward" packets based on their headers
- Each packet travels independently
- No link resources are reserved in advance. Instead packet switching leverages statistical multiplexing (stat muxing)

Multiplexing

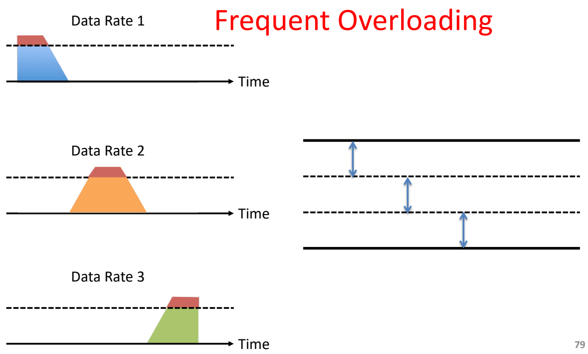


Sharing makes things efficient (cost less)
- One airplane/train for 100's of people
- One telephone for many calls
- One lecture theatre for many classes
- One computer for many tasks
- One network for many computers
- One datacenter many applications

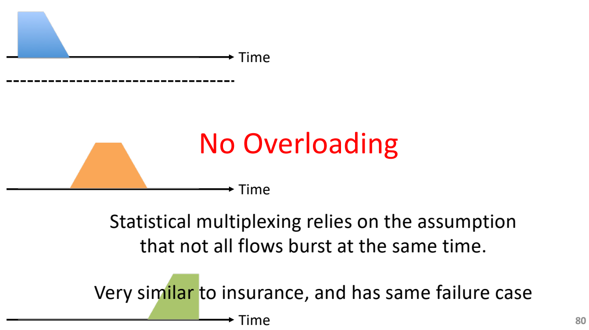
Three Flows with Bursty Traffic



When Each Flow Gets 1/3rd of Capacity



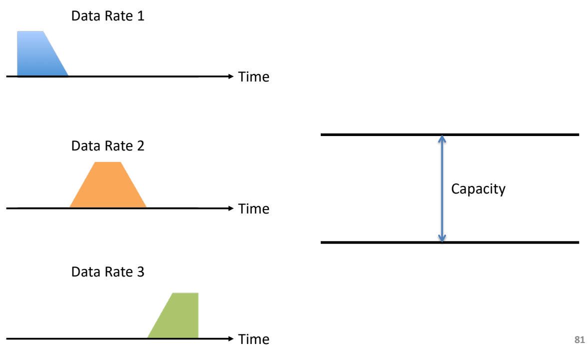
When Flows Share Total Capacity



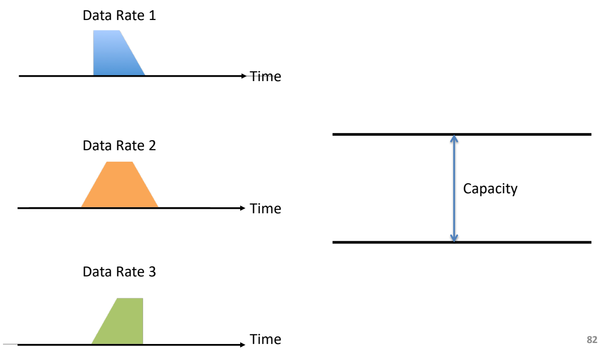
Statistical multiplexing relies on the assumption that not all flows burst at the same time.

Very similar to insurance, and has same failure case

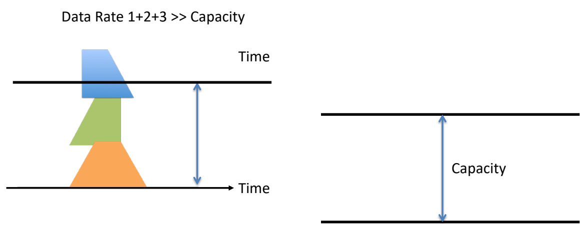
Three Flows with Bursty Traffic



Three Flows with Bursty Traffic

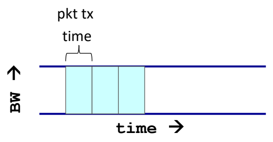


Three Flows with Bursty Traffic



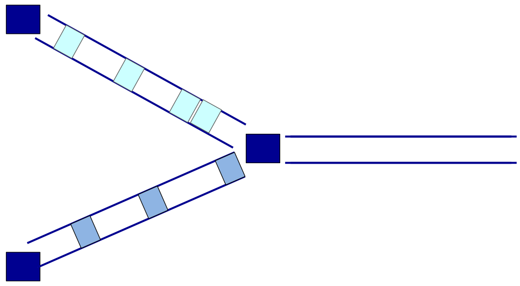
What do we do under overload?

Statistical multiplexing: pipe view



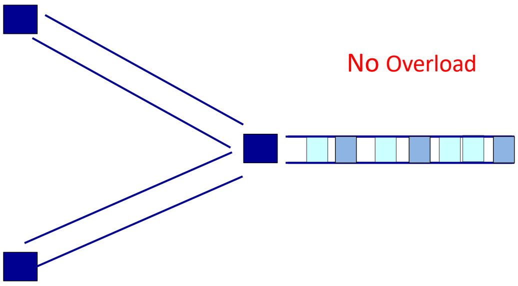
84

Statistical multiplexing: pipe view



85

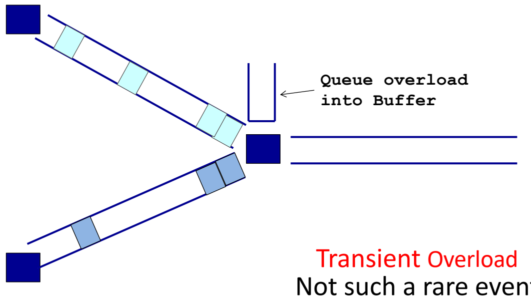
Statistical multiplexing: pipe view



No Overload

86

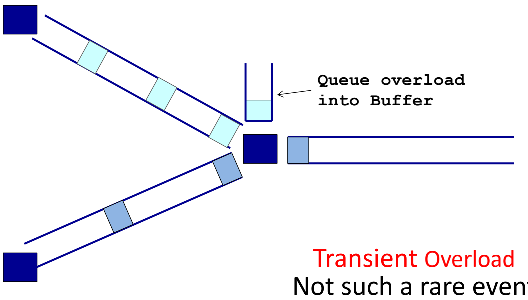
Statistical multiplexing: pipe view



Transient Overload
Not such a rare event

87

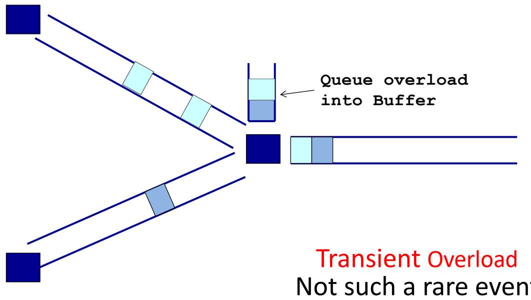
Statistical multiplexing: pipe view



Transient Overload
Not such a rare event

88

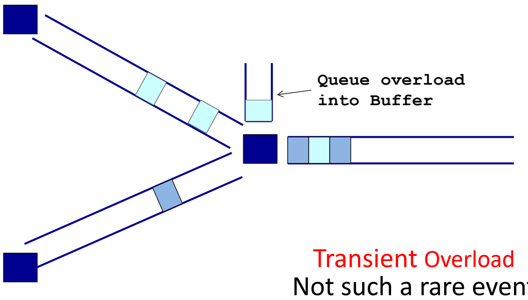
Statistical multiplexing: pipe view



Transient Overload
Not such a rare event

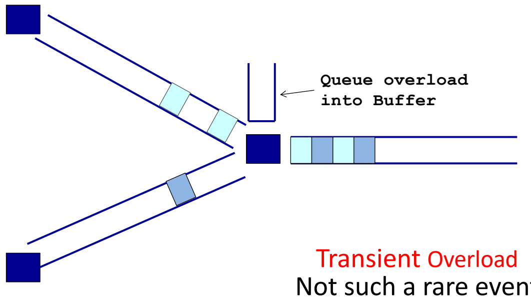
89

Statistical multiplexing: pipe view



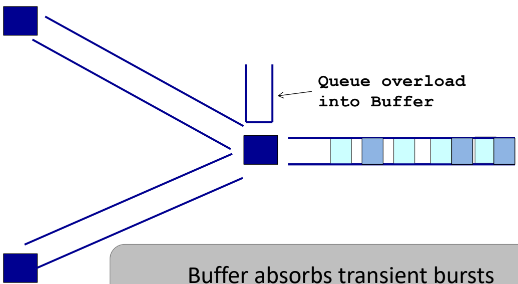
Transient Overload
Not such a rare event

Statistical multiplexing: pipe view



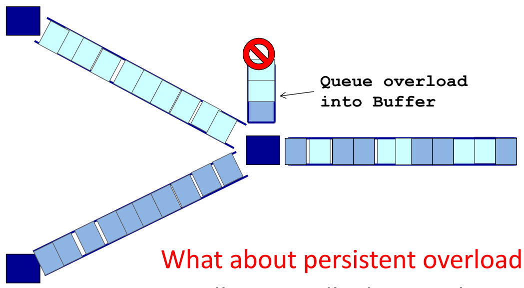
Transient Overload
Not such a rare event

Statistical multiplexing: pipe view



Buffer absorbs transient bursts
But *NOT* additional capacity

Statistical multiplexing: pipe view



What about persistent overload?
Will eventually drop packets

Queues introduce queuing delays

- Recall,

  packet delay = transmission delay + propagation delay (*)

- With queues (statistical multiplexing)

  packet delay = transmission delay + propagation delay + queuing delay (*)

- Queuing delay caused by "packet interference"

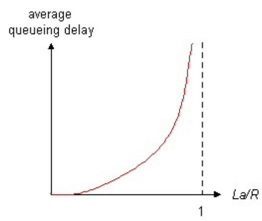
- Made worse at high load
  - less "idle time" to absorb bursts
  - think about traffic jams at rush hour or rail network failure

(* plus per-hop *processing* delay that we define as negligible)

Queuing delay extremes

- R=link bandwidth (bps)
- L=packet length (bits)
- a=average packet arrival rate

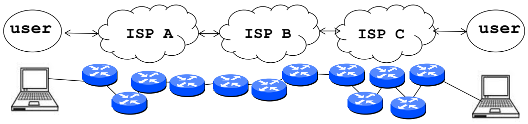
traffic intensity = La/R



- La/R ~ 0: average queuing delay small
- La/R -> 1: delays become large
- La/R > 1: more "work" arriving than can be serviced, average delay infinite – or data is lost (*dropped).*

Recall the Internet *federation*

- The Internet ties together different networks
  - >20,000 ISP networks



We can see (hints) of the nodes and links using traceroute…

"Real" Internet delays and routes

traceroute: rio.cl.cam.ac.uk to people.eng.unimelb.edu.au

(tracepath on winows is similar)

Three delay measurements from rio.cl.cam.ac.uk to gatwick.net.cl.cam.ac.uk

```
awm22@rio:~$ traceroute people.eng.unimelb.edu.au
traceroute to people.eng.unimelb.edu.au (128.250.59.37), 30 hops max, 60 byte packets
 1  vlan101.gatwick.net.cl.cam.ac.uk (128.232.32.2)  1.520 ms  1.822 ms  0.709 ms
 2  cl-wgb.d-mw.net.cam.ac.uk (193.60.89.5)  0.259 ms  0.256 ms  0.227 ms
 3  d-mw.c-ce.net.cam.ac.uk (131.111.6.53)  0.231 ms  0.381 ms  0.357 ms
 4  c-ce.b-ec.net.cam.ac.uk (131.111.6.82)  0.317 ms  0.481 ms  0.476 ms
 5  ae0.lowdss-ban1.ja.net (146.97.41.37)  2.842 ms  2.846 ms  2.821 ms
 6  ae26.lowdss-sbr1.ja.net (146.97.35.245)  2.877 ms  2.805 ms  2.795 ms
 7  ae28.londhx-sbr1.ja.net (146.97.33.17)  6.191 ms  6.109 ms  6.325 ms
 8  janet.mx1.lon.uk.geant.net (62.40.124.197)  6.319 ms  6.245 ms  6.258 ms
 9  138.44.226.6 (138.44.226.6)  169.704 ms  169.722 ms  169.682 ms
10  et-7-3-0.pe1.wmlb.vic.aarnet.net.au (113.197.15.28)  250.954 ms  251.163 ms  251.116 ms
11  * * *
12  4000v-eng-web-people-l.eng.unimelb.edu.au (128.250.59.37)  251.943 ms  251.952 ms  251.962 ms
13  4000v-eng-web-people-l.eng.unimelb.edu.au (128.250.59.37)  252.053 ms  252.018 ms  251.966 ms
14  * * *
15  4000v-eng-web-people-l.eng.unimelb.edu.au (128.250.59.37)  252.215 ms  252.088 ms  252.118 ms
16  4000v-eng-web-people-l.eng.unimelb.edu.au (128.250.59.37)  253.361 ms  253.109 ms  253.461 ms
17  4000v-eng-web-people-l.eng.unimelb.edu.au (128.250.59.37)  253.077 ms  253.832 ms  253.298 ms
18  * * *
….
29  * * *
30  * * *
```

Direct London-Perth

Australian link

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

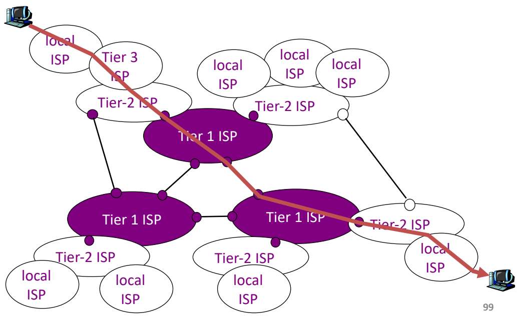
traceroute: rio.cl.cam.ac.uk to www.caida.org

```
rio:~$ traceroute --resolve-hostnames www.caida.org
traceroute to www.caida.org (192.172.226.122), 64 hops max
 1   128.232.64.2 (vlan398.gatwick.net.cl.cam.ac.uk)  3.760ms  2.060ms  1.226ms
 2   193.60.89.5 (cl-wgb.d-mw.net.cam.ac.uk)  53.777ms  67.458ms  0.556ms
 3   131.111.7.53 (d-mw.c-hi.net.cam.ac.uk)  0.638ms  0.621ms  0.658ms
 4   131.111.7.82 (c-hi.b-jc.net.cam.ac.uk)  0.353ms  0.346ms  0.338ms
 5   131.111.7.217 (ips-out.b-jc.net.cam.ac.uk)  0.582ms  0.441ms  0.397ms
 6   146.97.41.37 (ae0.lowdss-ban1.ja.net)  2.754ms  2.648ms  2.701ms
 7   146.97.35.245 (ae26.lowdss-sbr1.ja.net)  2.852ms  2.728ms  2.738ms
 8   146.97.33.25 (ae30.erdiss-sbr2.ja.net)  5.412ms  5.177ms  4.474ms
 9   146.97.33.21 (ae31.londpg-sbr2.ja.net)  8.408ms  8.213ms  8.293ms
10   62.40.125.57 (janet-bckp.mx1.lon2.uk.geant.net)  9.199ms  9.140ms  9.108ms
11   62.40.98.64 (ae2.mx1.lon.uk.geant.net)  10.119ms  9.818ms  9.756ms
12   62.40.124.45 (internet2-gw.mx1.lon.uk.geant.net)  95.065ms  95.962ms  95.434ms
13   163.253.1.120 (fourhundredge-0-0-0-0.4079.core2.ashb.net.internet2.edu)  152.834ms  153.562ms  154.448ms
14   163.253.1.139 (fourhundredge-0-0-0-1.4079.core2.clev.net.internet2.edu)  154.008ms  153.800ms  154.429ms
15   163.253.2.17 (fourhundredge-0-0-0-2.4079.core2.eqch.net.internet2.edu)  155.463ms  154.863ms  154.334ms
16   163.253.1.66 (fourhundredge-0-0-0-18.4079.core1.eqch.net.internet2.edu)  153.802ms  153.600ms  154.553ms
17   163.253.1.206 (fourhundredge-0-0-0-1.4079.core1.chic.net.internet2.edu)  154.783ms  154.926ms  154.796ms
18   163.253.2.29 (fourhundredge-0-0-0-1.4079.core2.kans.net.internet2.edu)  152.851ms  152.414ms  154.916ms
19   163.253.1.250 (fourhundredge-0-0-0-1.4079.core2.denv.net.internet2.edu)  155.571ms  155.047ms  154.572ms
20   163.253.1.169 (fourhundredge-0-0-0-3.4079.core2.salt.net.internet2.edu)  153.369ms  153.824ms  154.321ms
21   163.253.1.114 (fourhundredge-0-0-0-8.4079.core1.losa.net.internet2.edu)  153.786ms  153.549ms  154.839ms
22   137.164.26.200 (hpr-lax-agg10--i2.cenic.net)  152.552ms  153.465ms  152.493ms
23   137.164.25.89 (hpr-sdg-agg4--lax-agg10-100ge.cenic.net)  154.682ms  154.604ms  154.752ms
24   137.164.26.43 (hpr-sdsc-100ge--sdg-hpr3.cenic.net)  167.094ms  154.553ms  154.627ms
25   192.12.207.46 (medusa-mx960.sdsc.edu)  154.854ms  154.646ms  156.379ms
26   192.172.226.122 (proxy.caida.org)  154.581ms  154.390ms  154.477ms
```

A little more interesting because each hop resolves to a name (caida is in San Diego)

Internet structure: network of networks

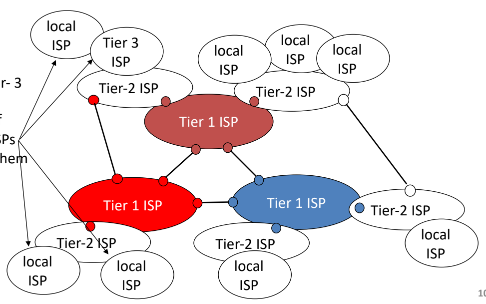
- a packet passes through many networks!



Internet structure: network of networks

- "Tier-3" ISPs and local ISPs
  - last hop ("access") network (closest to end systems)

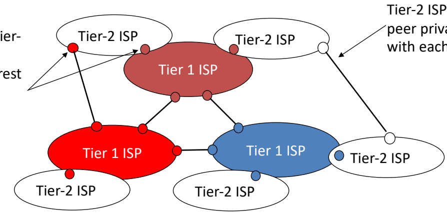
Local and tier- 3 ISPs are *customers* of higher tier ISPs connecting them to rest of Internet



Internet structure: network of networks

- "Tier-2" ISPs: smaller (often regional) ISPs
  - Connect to one or more tier-1 ISPs, possibly other tier-2 ISPs

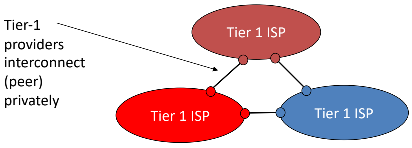
Tier-2 ISP pays tier-1 ISP for connectivity to rest of Internet
- tier-2 ISP is *customer* of tier-1 provider



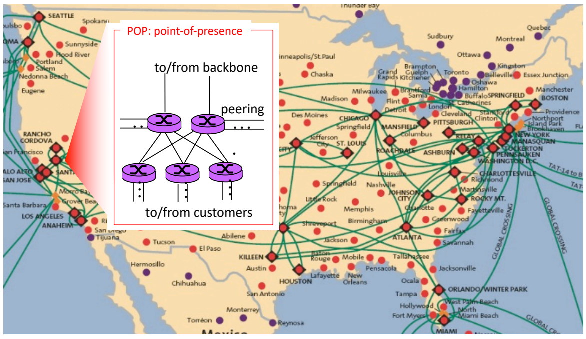
Tier-2 ISPs also peer privately with each other.

Internet structure: network of networks

- roughly hierarchical
- at center: "tier-1" ISPs (e.g., Verizon, Sprint, AT&T, Cable and Wireless), national/international coverage
  - treat each other as equals



Tier-1 ISP: e.g., Sprint



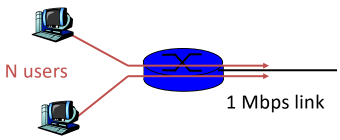
Packet Switching

- 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 in advance. Instead packet switching depends on statistical multiplexing
  - allows efficient use of resources
  - but introduces queues and queuing delays

Packet switching versus circuit switching

Packet switching may (does!) allow more users to use network

- 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?

Packet switching versus circuit switching

Q: how did we get value 0.0004?

- 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

Let U be number of users active
N the total users
P is 0.1 in our example to get 0.0004

$P(U=k) = \binom{n}{k} p^k (1-p)^{n-k}$

$\left[\therefore P(U \le K) = \sum_{k=0}^{K} \binom{n}{k} p^k (1-p)^{n-k}\right] \therefore \left[P(U>K) = 1 - \sum_{k=0}^{K} \binom{n}{k} p^k (1-p)^{n-k}\right]$

for $n = 35$, $K = 10$

$P(U \le 10) = \sum_{k=0}^{10} \binom{35}{k} p^k (1-p)^{35-k}$

where $p = 0.1$:

$P(U \le 10) = 0.99958$

$\therefore P(U > 10) = 0.00042$

Circuit switching: pros and cons

- Pros
  - guaranteed performance
  - fast transfers (once circuit is established)
- Cons
  - wastes bandwidth if traffic is "bursty"
  - connection setup adds delay
  - recovery from failure is slow

Packet switching: pros and cons

- Pros
  - efficient use of bandwidth (stat. muxing)
  - no overhead due to connection setup
  - resilient -- can `route around trouble'
- Cons
  - no guaranteed performance
  - header overhead per packet
  - queues and queuing delays

Summary

- A sense of how the basic `plumbing' works
  - links and switches
  - packet delays= transmission + propagation + queuing + (negligible) per-switch processing
  - statistical multiplexing and queues
  - circuit vs. packet switching

Topic 2 – Architecture and Philosophy

- Abstraction
- Layering
- Layers and Communications
- Entities and Peers
- What is a protocol?
- Protocol Standardization
- The architects process
  - How to break system into modules
  - Where modules are implemented
  - Where is state stored
- Internet Philosophy and Tensions

TRIGGER WARNING

- Philosophy,
- Bad Analogies, and
- RANTS verging on POLEMIC

Will follow.....

Abstraction Concept

A mechanism for breaking down a problem

what not *how*

- eg Specification *versus* implementation
- eg Modules in programs

Allows replacement of implementations without affecting system behavior

Vertical versus *Horizontal*

"Vertical" what happens in a box "How does it attach to the network?"

"Horizontal" the communications paths running through the system

Hint: paths are built ("layered") on top of other paths

Computer System Modularity

Partition system into modules & abstractions:

- Well-defined interfaces give flexibility
  - ***Hides*** implementation - can be freely changed
  - Extend functionality of system by adding new modules
- E.g., libraries encapsulating set of functionality
- E.g., programming language + compiler abstracts away how the particular CPU works …

Computer System Modularity (cnt'd)

- Well-defined interfaces hide information
  - Isolate assumptions
  - Present high-level abstractions
- **But can impair performance!**
- Ease of implementation vs worse performance

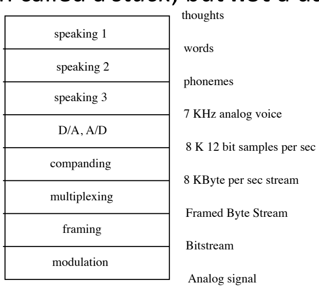
Network System Modularity

Like software modularity, but:

- Implementation is distributed across many machines (routers and hosts)
- Must decide:
  - How to break system into modules
    - **Layering**
  - Where modules are implemented
    - **End-to-End Principle**
  - Where state is stored
    - **Fate-sharing**

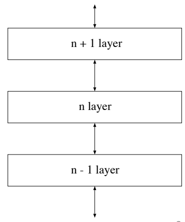
Layering Concept

- A restricted form of abstraction: system functions are divided into layers, one built upon another
- Often called a *stack*; but **not** a data structure!



Layers and Communications

- Interaction only between adjacent layers
- *layer n* uses services provided by *layer n-1*
- *layer n* provides service to *layer n+1*
- Bottom layer is physical media
- Top layer is application



Entities and Peers

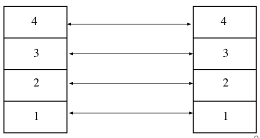
Entity – a *thing* (an independent existence)

Entities *interact* with the layers above and below

Entities *communicate* with *peer* entities

- same level but different place (eg different person, different box, different host)

Communications between peers is supported by entities at the lower layers



Entities and Peers

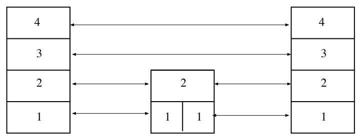
Entities usually do something useful

- Encryption – Error correction – Reliable Delivery
- Nothing at all is also reasonable

Not all communications is end-to-end

Examples for things in the middle

- IP Router – Mobile Phone Cell Tower
- Person translating French to English



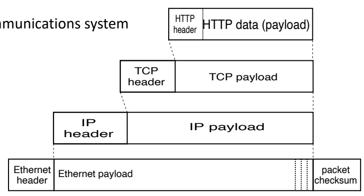
Layering and Embedding

In Computer Networks we often see higher-layer information embedded within lower-layer information

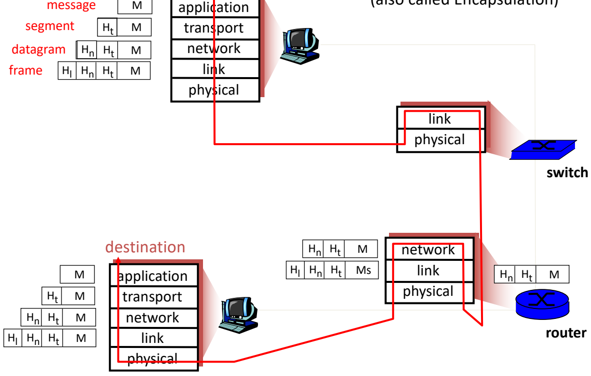
- Such embedding can be considered a form of layering
- Higher layer information is generated by stripping off headers and trailers of the current layer
- eg an IP entity only looks at the IP headers

BUT embedding is not the only form of layering

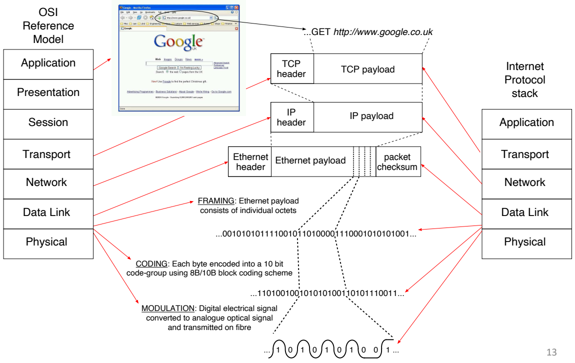
Layering is to help understand a communications system
NOT
determine implementation strategy



Example Embedding
(also called Encapsulation)



Internet protocol stack *versus* OSI Reference Model

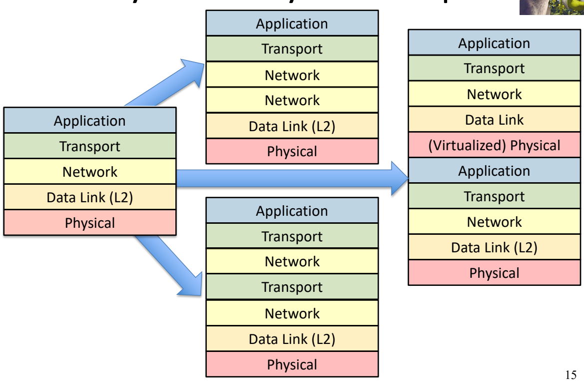


ISO/OSI reference model

- presentation: allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions
- *session:* synchronization, checkpointing, recovery of data exchange
- Internet stack "missing" these layers!
  - these services, *if needed,* must be implemented in application

application
presentation
session
transport
network
link
physical

Layers on Layers examples



What is a protocol?

<u>human protocols:</u>
- "what' s the time?"
- "I have a question"
- introductions

… specific msgs sent

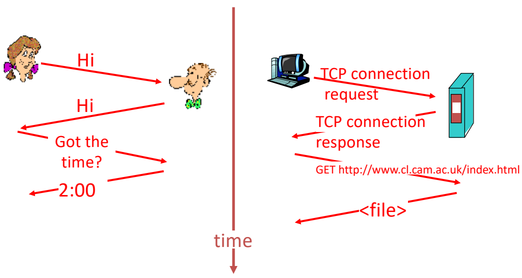
… specific actions taken when msgs received, or other events

<u>network protocols:</u>
- machines rather than humans
- all communication activity in Internet governed by protocols

protocols define format, order of msgs sent and received among network entities, and actions taken on msg transmission, receipt

What is a protocol?

a human protocol and a computer network protocol:



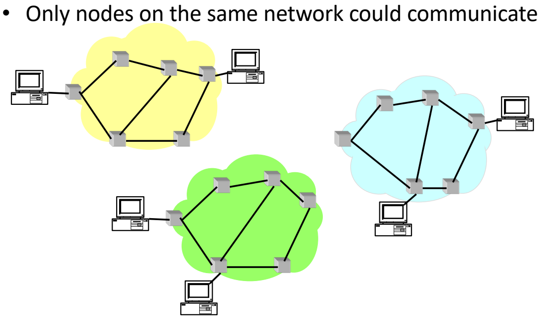
<u>Q:</u> Other human protocols?

Protocol Standardization

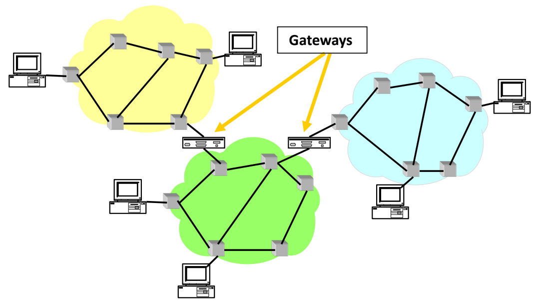
- All hosts must follow same protocol
  - Very small modifications can make a big difference
  - Or prevent it from working altogether
- This is why we have standards
  - Can have multiple implementations of protocol
- Internet Engineering Task Force (IETF)
  - Based on working groups that focus on specific issues
  - Produces "Request For Comments" (RFCs)
  - IETF Web site is ***http://www.ietf.org***
  - RFCs archived at ***http://www.rfc-editor.org***

So many Standards Problem

- Many different packet-switching networks
- Each with its own Protocol
- Only nodes on the same network could communicate



INTERnet Solution



Internet Design Goals (Clark '88)

- **Connect existing networks**
- Robust in face of failures
- Support multiple types of delivery services
- Accommodate a variety of networks
- Allow distributed management
- Easy host attachment
- Cost effective
- Allow resource accountability

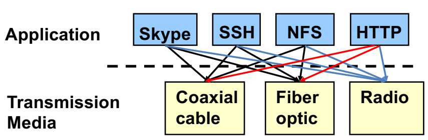
Real Goals

Internet Motto

We reject kings , presidents, and voting. We believe in rough consensus and running code." – David Clark

- **Build something that works!**
- Connect existing networks
- Robust in face of failures
- Support multiple types of delivery services
- Accommodate a variety of networks
- Allow distributed management
- Easy host attachment
- Cost effective
- ~~Allow resource accountability~~

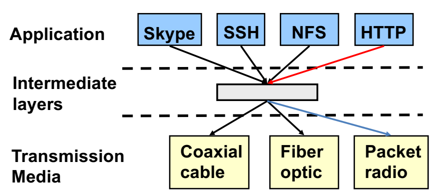
A Multitude of Apps Problem



- Re-implement every application for every technology?
- No! But how does the Internet design avoid this?

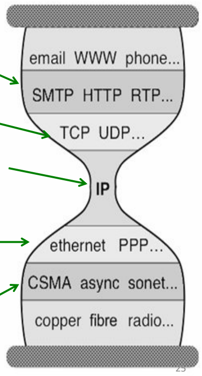
Solution: Intermediate Layers

- Introduce intermediate layers that provide set of abstractions for various network functionality and technologies
  - A new app/media implemented only once
  - Variation on "add another level of indirection"



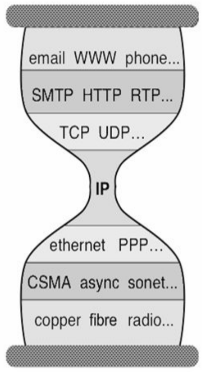
In the context of the Internet

Applications
...built on...
Reliable (or unreliable) transport
...built on...
Best-effort global packet delivery
...built on...
Best-effort local packet delivery
...built on...
Physical transfer of bits



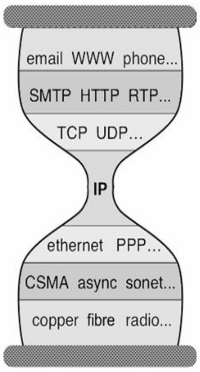
Three Observations

- Each layer:
  - Depends on layer below
  - Supports layer above
  - Independent of others
- Multiple versions in layer
  - Interfaces differ somewhat
  - Components pick which lower-level protocol to use
- But only one IP layer
  - Unifying protocol



Layering Crucial to Internet's Success

- Reuse
- Hides underlying detail
- Innovation at each level can proceed in parallel
- Pursued by very different communities



What are some of the drawbacks of protocols and layering?

Drawbacks of Layering

- Layer N may duplicate lower layer 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 really big
  - e.g., typical TCP+IP+Ethernet is 54 bytes
- Layer violations when the gains too great to resist
  - e.g., TCP-over-wireless
- Layer violations when network doesn't trust ends
  - e.g., firewalls

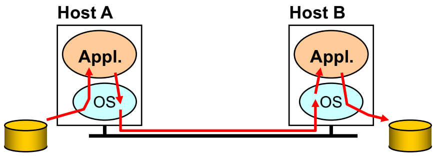
Placing Network Functionality

- Hugely influential paper: "End-to-End Arguments in System Design" by Saltzer, Reed, and Clark ('84)
  - articulated as the "End-to-End Principle" (E2E)
- Endless debate over what it means
- Everyone cites it as supporting their position
  (regardless of the position!)

Basic Observation

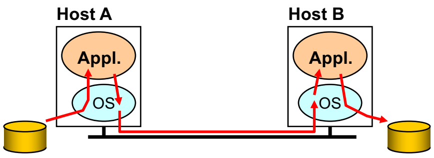
- Some application requirements can only be correctly implemented **end-to-end**
  - reliability, security, *etc.*
- Implementing these in the network is hard
  - every step along the way must be fail proof
- Hosts
  - **Can** satisfy the requirement without network's help
  - **Will/must** do so, since they can't rely on the network

Example: Reliable File Transfer



- Solution 1: make each step reliable, and string them together to make reliable end-to-end process

Example: Reliable File Transfer



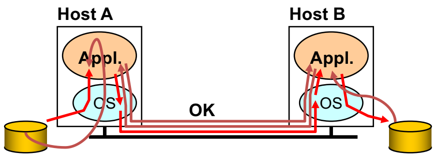
- Solution 1: make each step reliable, and string them together to make reliable end-to-end process

So what is the problem?

each component is 0.9 reliable

leads to total system failure of >0.4*

Example: Reliable File Transfer



- Solution 1: make each step reliable, and string them together to make reliable end-to-end process
- Solution 2: end-to-end **check** and retry

Discussion

- Solution 1 is incomplete
  - What happens if any network element misbehaves?
  - Receiver has to do the check anyway!
- Solution 2 is complete
  - Full functionality can be entirely implemented at application layer with no need for reliability from lower layers
- Is there any need to implement reliability at lower layers?

Summary of End-to-End Principle

- Implementing functionality (e.g., reliability) in the network
  - Doesn't reduce host implementation complexity
  - Does increase network complexity
  - Probably increases delay and overhead on all applications even if they don't need the functionality (e.g. VoIP)
- However, implementing in the network can improve performance in some cases
  - e.g., consider a very lossy link

“Only-if-Sufficient” Interpretation

- Don’t implement a function at the lower levels of the system unless it can be completely implemented at this level

- *Unless you can relieve the burden from hosts, don’t bother*

“Only-if-Necessary” Interpretation

- Don’t implement *anything* in the network that can be implemented correctly by the hosts

- Make network layer absolutely minimal
  - This E2E interpretation trumps performance issues
  - Increases flexibility, since lower layers stay **simple**

“Only-if-Useful” Interpretation

- If hosts can implement functionality correctly, implement it in a lower layer only as a performance enhancement
- But do so only if it does not impose burden on applications that do not require that functionality

We have some tools:

- Abstraction
- Layering
- Layers and Communications
- Entities and Peers
- Protocol as motivation
- Examples of the architects process
- Internet Philosophy and Tensions

Distributing Layers Across Network

- Layers are simple if only on a single machine
  - Just stack of modules interacting with those above/below

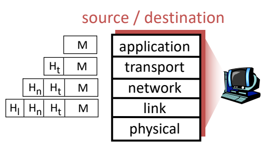
- But we need to implement layers across machines
  - Hosts
  - Routers (switches)

- What gets implemented where?

What Gets Implemented on Host?

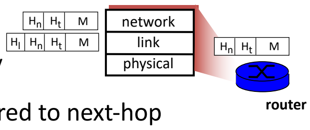
- Bits arrive on wire, must make it up to application

- Therefore, all layers must exist at the host



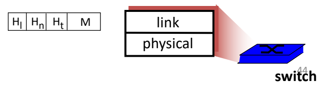
What Gets Implemented on a Router?

- Bits arrive on wire
  - Physical layer necessary
- Packets must be delivered to next-hop
  - Datalink layer necessary
- Routers participate in global delivery
  - Network layer necessary
- Routers don't support reliable delivery
  - Transport layer (and above) ***not*** supported

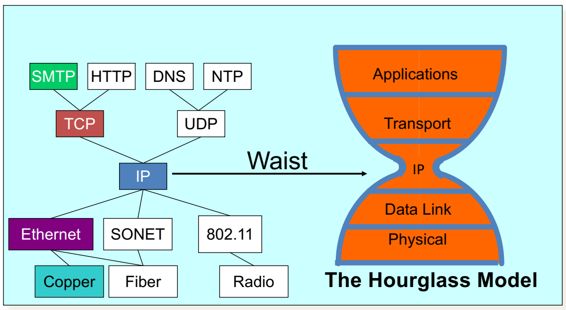


What Gets Implemented on Switches?

- Switches do what routers do, except they don't participate in global delivery, just local delivery
- They only need to support Physical and Datalink
  - Don't need to support Network layer
- Won't focus on the router/switch distinction
  - Almost all boxes support network layer these days
  - Routers have switches but switches do not have routers

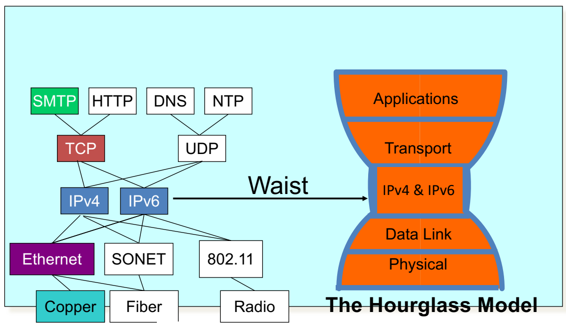


The Internet *Hourglass*



There is just one network-layer protocol, **IP**.
The "narrow waist" facilitates interoperability.

The middle-age Internet *Hourglass*



There is just ~~one~~ TWO network-layer protocol, **IP**v4 + v6
The "narrow waist" facilitates interoperability(???)

Protocol Standardization (Redux)

- All hosts must follow same protocol
  - Very small modifications can make a big difference
  - Or prevent it from working altogether
- This is why we have standards
  - Can have multiple implementations of protocol
- Internet Engineering Task Force (IETF)
  - Based on working groups that focus on specific issues
  - Produces "Request For Comments" (RFCs)
  - IETF Web site is ***http://www.ietf.org***
  - RFCs archived at ***http://www.rfc-editor.org***

Alternative to Standardization?

- Have one implementation used by everyone
- Open-source projects
  - Which has had more impact, Linux or POSIX?
- Or just sole-sourced implementation
  - zoom, Signal, FaceTime, etc.

Topic 3.0: The Physical Layer

Our goals:

- Understand physical channel fundamentals
  - Physical channels can carry data in proportion to the signal and inversely in proportion to noise
  - Modulation represents Digital data in analog channels
  - Baseband vs. Broadband
  - Synchronous vs. Asynchronous

Physical Channels / The Physical Layer

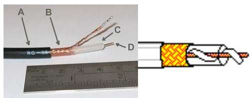
these example physical channels are also known as *Physical Media*

Twisted Pair (TP)

- two insulated copper wires
  - Category 3: traditional phone wires, 10 Mbps Ethernet
  - Category 8: 25Gbps Ethernet
- Shielded (STP)
- Unshielded (UTP)

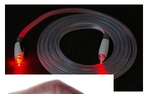
Coaxial cable:

- two concentric copper conductors
- bidirectional
- baseband:
  - single channel on cable
  - legacy Ethernet
- broadband:
  - multiple channels on cable
  - HFC (Hybrid Fiber Coax)



Fiber optic cable:

- high-speed operation
- point-to-point transmission
- (10's-100's Gbps)
- low error rate
- immune to electromagnetic noise



More Physical media: Radio

- Bidirectional and multiple access
- propagation environment effects:
  - reflection
  - obstruction by objects
  - interference

Radio link types:

- terrestrial microwave
  - e.g. 90 Mbps channels
- LAN (e.g., Wifi)
  - 11Mbps, 54 Mbps, 600 Mbps
- wide-area (e.g., cellular)
  - 5G cellular: ~ 40 Mbps - 10Gbps
- satellite
  - 27-50MHz typical bandwidth
  - geosynchronous versus low altitude
  - For geosync - 270 msec end-end delay to orbit

Physical Channel Characteristics - Fundamental Limits -

symbol type: generally, an analog waveform — voltage, current, photo intensity etc.

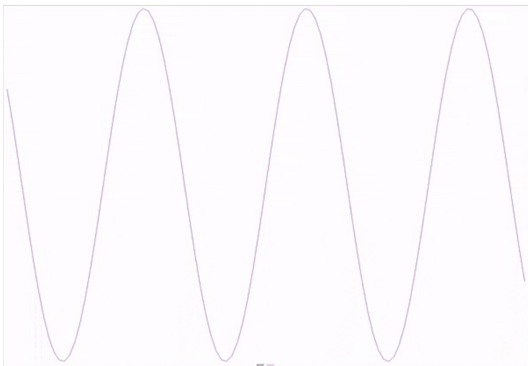
capacity: bandwidth

delay: speed of light in medium and distance travelled

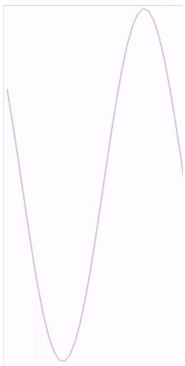
fidelity: signal to noise ratio

- measure of the range of frequencies of sinusoidal signal that channel supports
- E.g., a channel that supports sinusoids from 1 MHz to 1.1 MHz has a bandwidth of 100 KHz
- "supports" in this context means "comes out the other end of the channel"
- some frequencies supported better than others
- analysing what happens to an arbitrary waveform is done by examining what happens to its component sinusoids → Fourier analysis
- bandwidth is a resource

Analog meet Digital



Analog meet Digital



Square waves have high frequency components in them

Channels attenuate frequencies irregularly:
changing the shape of the signal

Receiver signal is related to the transmitted signal + noise

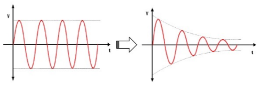
Noise may be systematic or random

Systematic noise from interfering equipment
can in principle be eliminated (not always convenient)

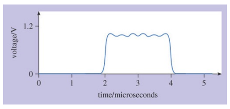
Random noise caused by thermal vibration (thermal noise)

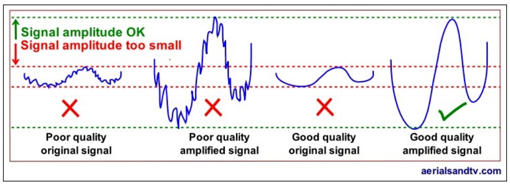
"White" noise is evenly distributed across frequencies
signal to noise ratio *S/N*
more distance more noise

Noise: Enemy of Communications



Attenuation, External Noise, Systematic, non-systematic, digitization, interference, reflection, ….





Bandwidth vs Signal to Noise

what's better: high bandwidth or low signal to noise?

- channels subject to white noise have information capacity *C* measured in bits per second, of a channel

$$C = Blog_2(1 + S/N)$$

B is the bandwidth of the channel *S/N* is the ratio of received signal power to received noise power.

- channels with no noise have information capacity determined only by bandwidth
- channels with any signal have nonzero information capacity
- channels with signal to noise ratio of unity have an information capacity in bits per second equal to its bandwidth in hertz
- (This is actually NOT the definition of information capacity; it is derived from the definition)

(Digital) Channels

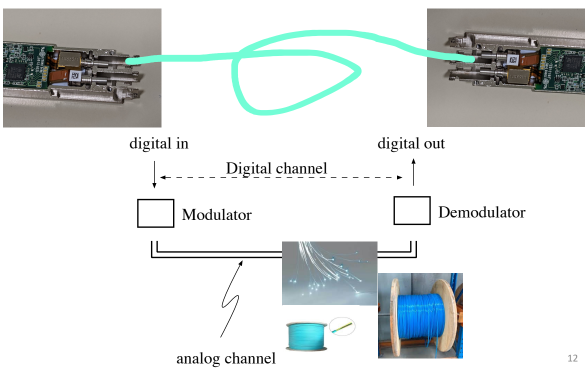
- Physical layer provides a channel
- Fixed rate for now
- Symbols are discrete values sent on the channel at fixed rate
- Symbols need not be binary
- Fidelity of the channel usually measured as a bit error rate — the probability that a bit sent as a 1 was interpreted as a 0 by the receiver or vice versa.
- Baud rate is the rate at which symbols can be transmitted
- Data rate (or bit rate) is the equivalent number of binary digits which can be sent
- E.g., if symbols represent with rate R then the data rate is 2 × R.

Modulation

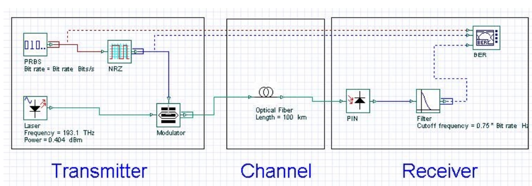
Two definitions:

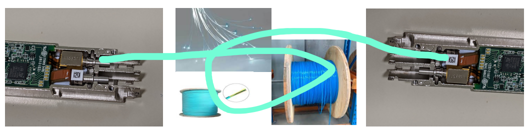
- Transform an information signal into a signal more appropriate for transmission on a physical medium
- The systematic alteration of a carrier waveform by an information signal

In general, we mean the first here
(which encompasses the second).



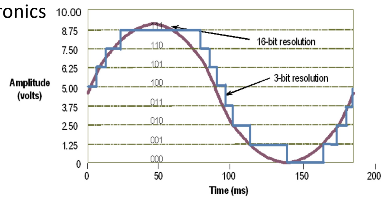
Communications





Analog/Digital Digital/Analog

Recall from Digital Electronics



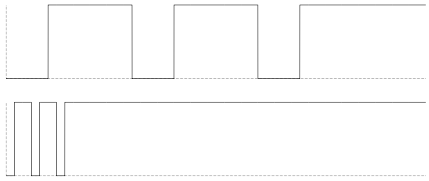
Conversion errors can occur in both directions

e.g.

Noise leads to incorrect digitization

Insufficient digitization resolution leads to information loss

More Challenges



Where are the bits?

WHEN are the bits?

Bit boundaries can be asynchronous or synchronous

Asynchronous versus Synchronous

- Transmission is sporadic, divided into frames
- Receiver and transmitter have oscillators which are close in frequency producing tx clocks and rx clock
- Receiver synchronises the phase of the rx clock with the tx clock by looking at one or more bit transitions
- RX clock drifts with respect to the tx clock but stays within a fraction of a bit of tx clock throughout the duration of a frame
- Transmission time is limited by accuracy of oscillators

- Transmission is continuous
- Receiver continually adjusts its frequency to track clock from incoming signal
- Requires bit transitions to inform clock
- Phase locked loop: rx clock predicts when incoming clock will change and corrects slightly when wrong.

Asynchronous versus Synchronous

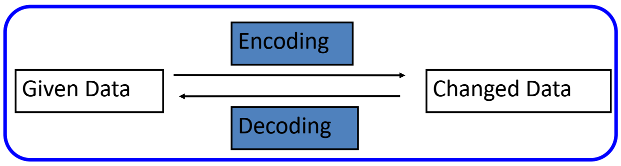
- Transmission is sporadic, divided into frames
- Receiver and transmitter have oscillators which are close in frequency producing tx clocks and rx clock
- Receiver synchronises the phase of the rx clock with the tx clock by looking at one or more bit transitions
- RX clock drifts with respect to the tx clock but stays within a fraction of a bit of tx clock throughout the duration of a frame
- Transmission time is limited by accuracy of oscillators

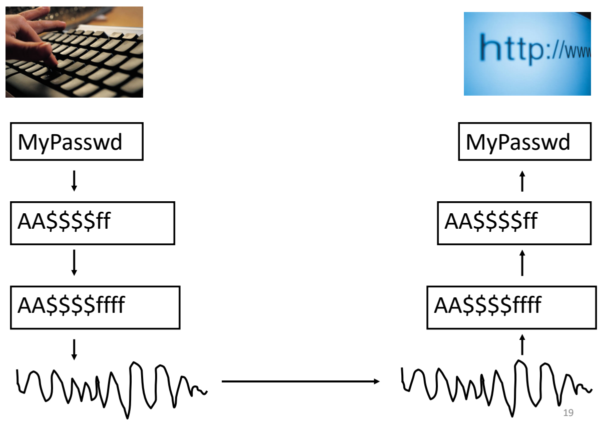
- Transmission is continuous
- Receiver continually adjusts its frequency to track clock from incoming signal
- Requires bit transitions to inform clock
- Phase locked loop: rx clock predicts when incoming clock will change and corrects slightly when wrong.

Bit transitions are critical

Coding – a channel function

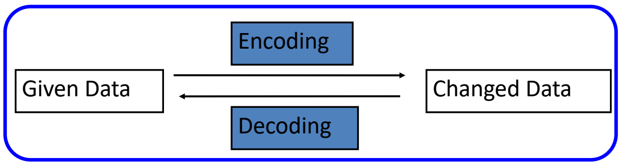
Change the representation of data.





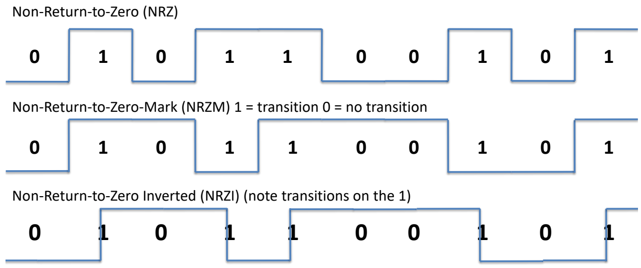
Coding

Change the representation of data.

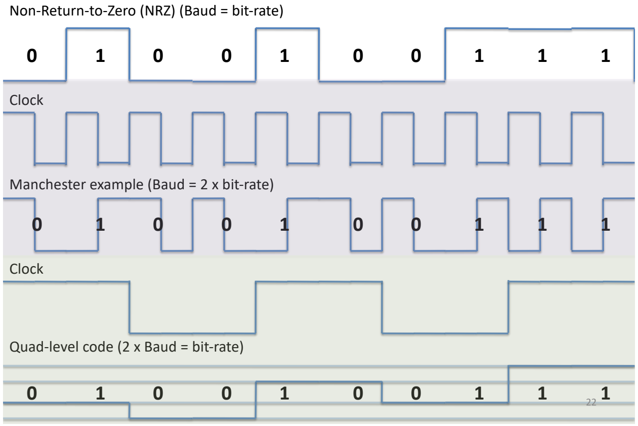


1. Encryption: MyPasswd <-> AA$$$$ff
2. Error Detection: AA$$$$ff <-> AA$$$$ffff
3. Compression: AA$$$$ffff <-> A2$4f4
4. Analog: A2$4f4 <-> 

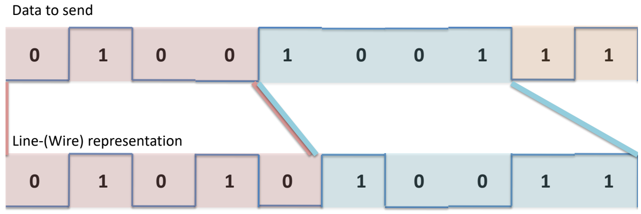
Line Coding Examples where Baud=bit-rate



Line Coding Examples



Line Coding – Block Code example



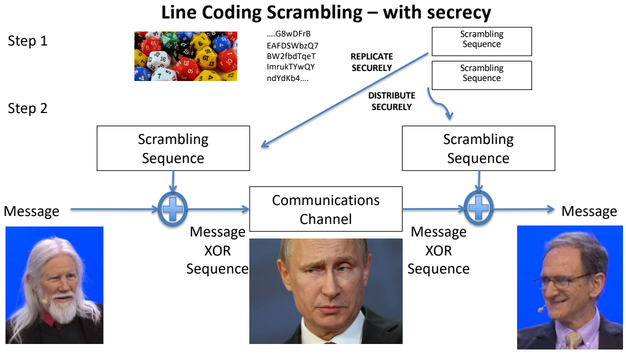
Name	4b	5b	Description	Name	4b	5b	Description
0	0000	11110	hex data 0	Q	-NONE-	00000	Quiet
1	0001	01001	hex data 1	I	-NONE-	11111	Idle
2	0010	10100	hex data 2	J	-NONE-	11000	SSD #1
3	0011	10101	hex data 3	K	-NONE-	10001	SSD #2
4	0100	01010	hex data 4	T	-NONE-	01101	ESD #1
5	0101	01011	hex data 5	R	-NONE-	00111	ESD #2
6	0110	01110	hex data 6	H	-NONE-	00100	Halt
7	0111	01111	hex data 7				
8	1000	10010	hex data 8				
9	1001	10011	hex data 9				
A	1010	10110	hex data A				
B	1011	10111	hex data B				
C	1100	11010	hex data C				
D	1101	11011	hex data D				
E	1110	11100	hex data E				
F	1111	11101	hex data F				

Block coding transfers data with a fixed overhead: 20% less information per Baud in the case of 4B/5B

So to send data at 100Mbps; the line rate (the Baud rate) must be 125Mbps.

1Gbps uses an 8b/10b codec; encoding entire bytes at a time but with 25% overhead

Line Coding Scrambling – with secrecy

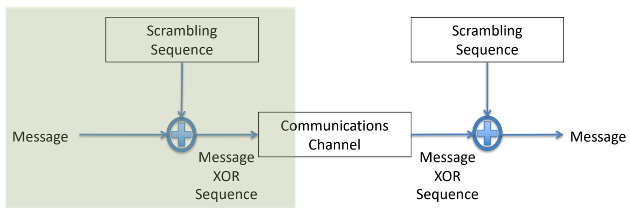


Step 3 Don't ever reuse Scrambling sequence, ever. <<< **this is quite important**

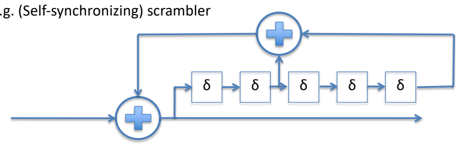
Whitfield Diffie

Martin Hellman

Line Coding Scrambling– no secrecy



e.g. (Self-synchronizing) scrambler



Line Coding Examples (Hybrid)

…1001111011010100010001011001110100010100101101010010011101011110100…

…100111101101010001010001011001110100010100101101010010011101011110100…

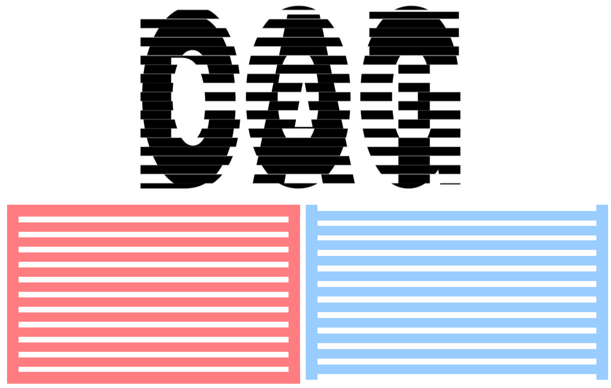
Inserted bits marking “start of frame/block/sequence”

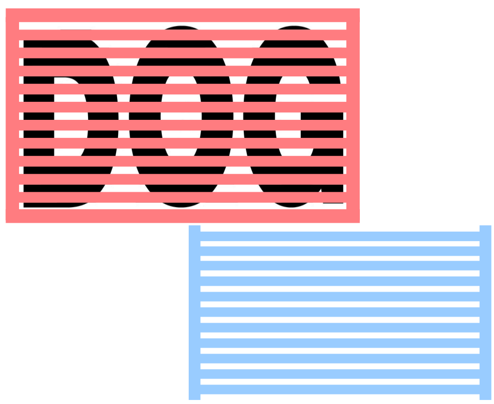
Scramble / Transmit / Unscramble

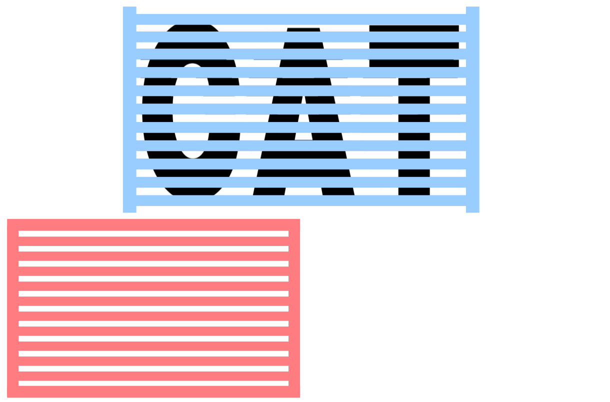
…01000101100111010001010010110101001001110101110100100101110111011111000…

Identify (and remove) “start of frame/block/sequence”
This gives you the Byte-delineations for *free*

64b/66b combines a scrambler and a framer. The start of frame is a pair of bits 01 or 10: 01 means “this frame is data” 10 means “this frame contains data and control” – control could be configuration information, length of encoded data or simply “this line is idle” (no data at all)



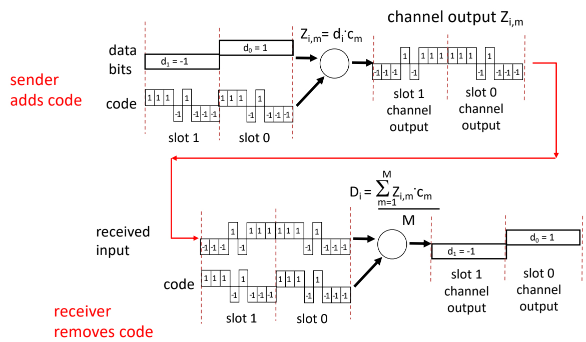




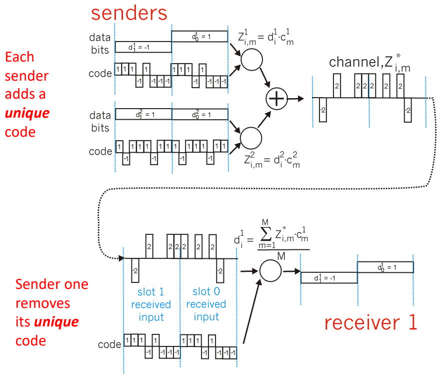
Code Division Multiple Access (CDMA) (not to be confused with CSMA!)

- used in several wireless broadcast channels (cellular, satellite, etc) standards
- unique “code” assigned to each user; i.e., code set partitioning
- all users share same frequency, but each user has own “chipping” sequence (i.e., code) to encode data
- *encoded signal* = (original data) XOR (chipping sequence)
- *decoding:* inner-product of encoded signal and chipping sequence
- allows multiple users to “coexist” and transmit simultaneously with minimal interference (if codes are “orthogonal”)

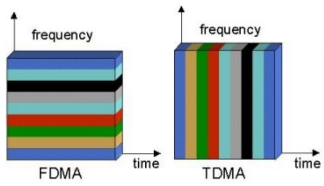
CDMA Encode/Decode



CDMA: two-sender interference



Multiple Access Mechanisms



Each dimension is orthogonal (so may be trivially combined)
Other dimensions are also available…

Coding Examples summary

- Common Wired coding
  - Block codecs: table-lookups
    - fixed overhead, inline control signals
  - Scramblers: shift registers
    - overhead free

Like earlier coding schemes and error correction/detection; you can combine these
- e.g, 10Gb/s Ethernet may use a hybrid

CDMA (Code Division Multiple Access)
- coping intelligently with competing sources
- Mobile phones

Error Detection and Correction

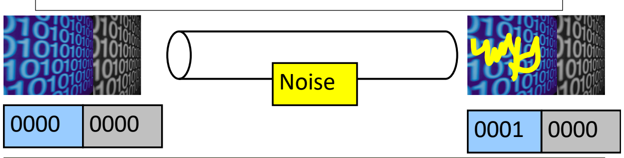
Transmission media are not perfect and cause signal impairments:

1. Attenuation
   - Loss of energy to overcome medium's resistance
2. Distortion
   - The signal changes its form or shape, caused in composite signals
3. Noise
   - Thermal noise, induced noise, crosstalk, impulse noise

Interference can change the shape or timing of a signal:
0 → 1 or 1 → 0

Error Detection and Correction

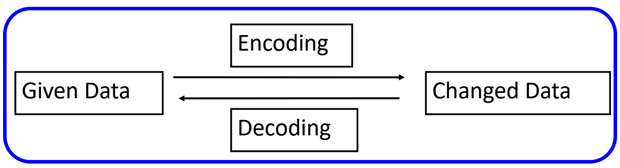
How to use coding to deal with errors in data communication?

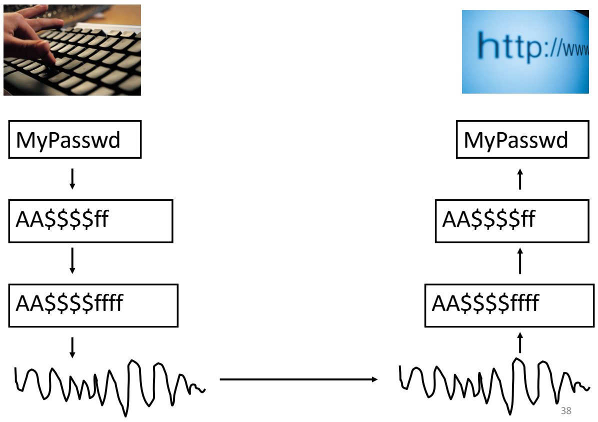


Basic Idea :
1. Add additional information (redundancy) to a message.
2. Detect an error and discard
   Or, fix an error in the received message.

Coding – a channel function

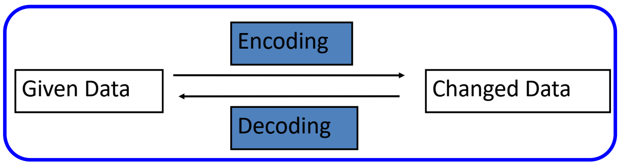
Change the representation of data.



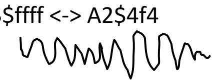


Coding Examples

Changig the representation of data.

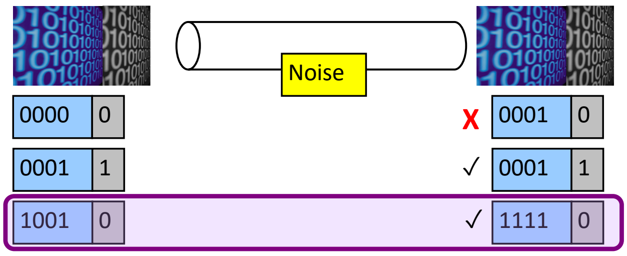


1. Encryption: MyPasswd <-> AA$$$$ff
2. Error Detection: AA$$$$ff <-> AA$$$$ffff
3. Compression: AA$$$$ffff <-> A2$4f4
4. Analog: A2$4f4 <->



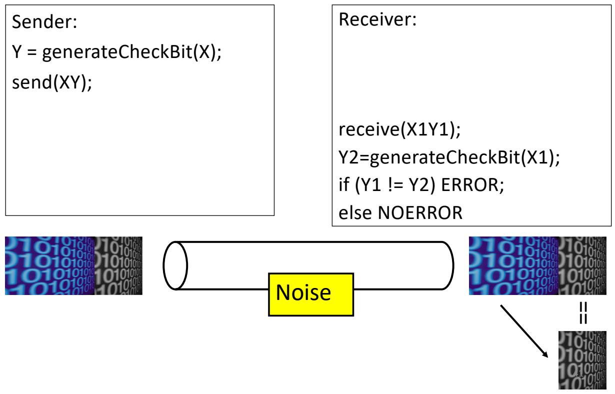
Error Detection Code: Parity

Add one bit, such that the number of all 1's is even.



Problem: This simple parity cannot detect two-bit errors.

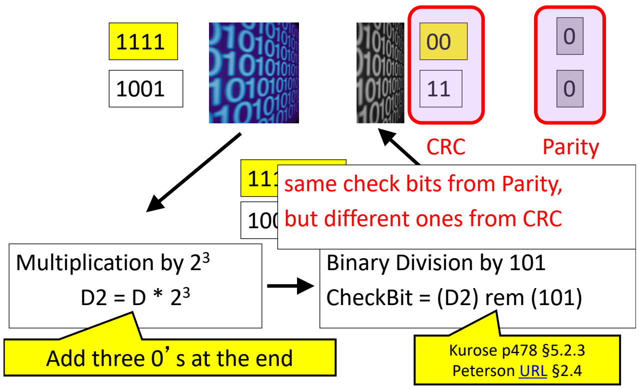
Error Detection Code



Error Detection Code: CRC

- CRC means "Cyclic Redundancy Check".
- *"A sequence of redundant bits, called CRC, is appended to the end of data so that the resulting data becomes exactly divisible by a second, predetermined binary number."*
- *CRC:= remainder (data ÷ predetermined divisor)*
- More powerful than parity.
  - It can detect various kinds of errors, including 2-bit errors.
- More complex: multiplication, binary division.
- Parameterized by n-bit divisor P.
  - Example: 3-bit divisor 101.
  - Choosing good P is crucial.

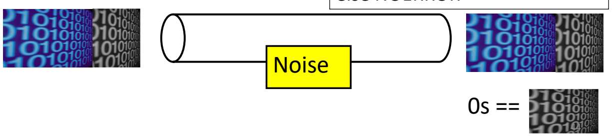
CRC with 3-bit Divisor 101



Error Detection Code

Sender:
Y = generateCRC(X div P);
send(X);
send(Y);

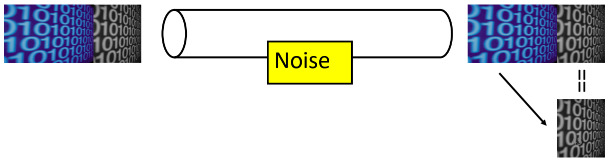
Receiver:
receive(X1);
receive(Y1);
Y2=generateCRC(X1Y1 div P);
if (Y2 != 0s) ERROR;
else NOERROR



Transforming Error Detection to…

Sender:
Y = generateCheckBit(X);
send(XY);

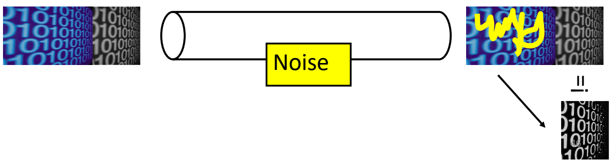
Receiver:
receive(X1Y1);
Y2=generateCheckBit(X1);
if (Y1 != Y2) ERROR;
else NOERROR



Forward Error Correction (FEC)

Sender:
Y = generateCheckBit(X);
send(XY);

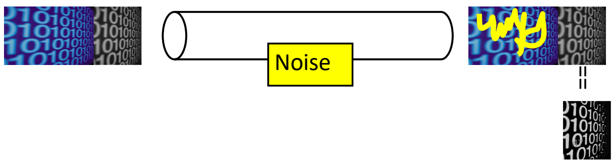
Receiver:
receive(X1Y1);
Y2=generateCheckBit(X1);
if (Y1 != Y2) FIXERROR(X1Y1);
else NOERROR



Forward Error Correction (FEC)

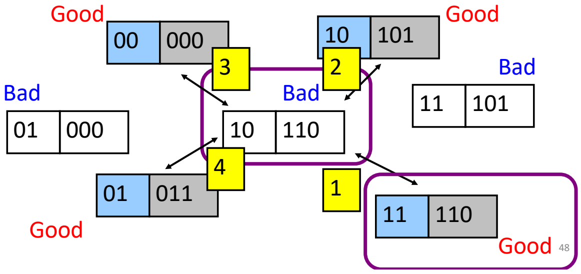
Sender:
Y = generateCheckBit(X);
send(XY);

Receiver:
receive(X1Y1);
Y2=generateCheckBit(X1);
if (Y1 != Y2) FIXERROR(X1Y1);
else NOERROR



Basic Idea of Forward Error Correction

Replace erroneous data
by its "closest" error-free data.



Error Detection vs Correction

Error Correction:
- Cons: More check bits. False recovery.
- Pros: No need to re-send.

Error Detection:
- Cons: Need to re-send.
- Pros: Less check bits.

Usage:
- Correction: A lot of noise. Expensive to re-send.
- Detection: Less noise. Easy to re-send.
- Can be used together.

Topic 3: The Data Link Layer

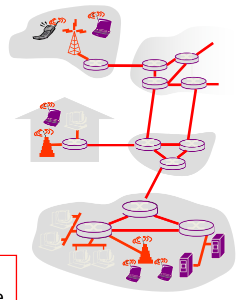
Our goals:

- understand principles behind data link layer services:
  (these are methods & mechanisms in your networking toolbox)
  - error detection, correction
  - sharing a broadcast channel: multiple access
  - link layer addressing
  - reliable data transfer, flow control
- instantiation and implementation of various link layer technologies
  - Wired Ethernet (aka 802.3)
  - Wireless Ethernet (aka 802.11 WiFi)
- Algorithms
  - Binary Exponential Back-off
  - Spanning Tree (Dijkstra)
- General knowledge
  - Random numbers are important and hard

Link Layer: Introduction

Some reminder-terminology:

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



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

Link Layer (Channel) Services - 1/2

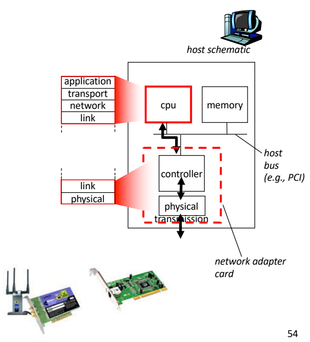
- *framing, physical addressing:*
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - "MAC" addresses used in frame headers to identify source, destination
    - This is **not** an IP address!
- *reliable delivery between adjacent nodes*
  - we revisit this again in the Transport Topic
  - seldom used on low bit-error link (fiber, some twisted pair)
  - wireless links: high error rates

Link Layer (Channel) Services – 2/2

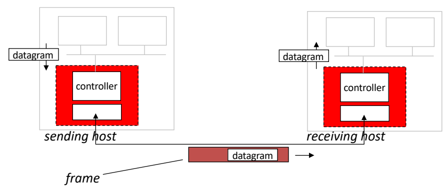
- *flow control:*
  - pacing between adjacent sending and receiving nodes
- *error control:*
  - *error detection:*
  - errors caused by signal attenuation, noise.
  - receiver detects presence of errors:
    - signals sender for retransmission or drops frame
  - error correction:
  - receiver identifies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

Where is the link layer implemented?

- in each and every host
- link layer implemented in "adaptor" (aka *network interface card* NIC)
  - Ethernet card, PCMCI card, 802.11 card
  - implements link, physical layer
- attaches into host's system buses
- combination of hardware, software, firmware



Adaptors Communicating

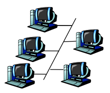


- sending side:
  - encapsulates datagram in frame
  - encodes data for the physical layer
  - adds error checking bits, provide reliability, flow control, etc.
- receiving side
  - decodes data from the physical layer
  - looks for errors, provide reliability, flow control, etc
  - extracts datagram, passes to upper layer at receiving side

Multiple Access Links and Protocols

Two types of “links”:

- point-to-point
  - point-to-point link between Ethernet switch and host
- broadcast (shared wire or medium)
  - old-fashioned wired Ethernet (*here be dinosaurs* – extinct)
  - upstream HFC (Hybrid Fiber-Coax – the Coax may be broadcast)
  - Home plug / Powerline networking
  - 802.11 wireless LAN



shared wire (e.g., Coax cabled Ethernet)

shared RF (e.g., 802.11 WiFi)

shared RF (satellite)

humans at a cocktail party (shared air, acoustical)

Multiple Access protocols

- 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

Ideal Multiple Access Protocol

Broadcast channel of rate *R* bps

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

MAC Protocols: a taxonomy

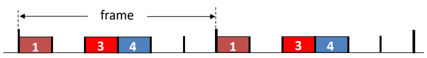
Three broad classes:

- Channel Partitioning
  - divide channel into smaller “pieces” (time slots, frequency, code)
  - allocate piece to node for exclusive use
- Random Access
  - channel not divided, allow collisions
  - “recover” from collisions
- “Taking turns”
  - nodes take turns, but nodes with more to send can take longer turns

Channel Partitioning MAC protocols: TDMA
(we discussed this earlier)

TDMA: time division multiple access

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

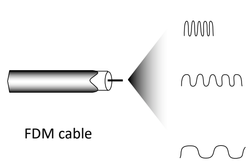


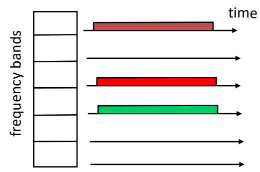
6 slots

Channel Partitioning MAC protocols: FDMA
(we discussed this earlier)

FDMA: frequency division multiple access

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





“Taking Turns” MAC protocols

channel partitioning MAC protocols:

– share channel *efficiently* and *fairly* at high load

– inefficient at low load: delay in channel access, 1/N bandwidth allocated even if only 1 active node!

random access MAC protocols:

– efficient at low load: single node can fully utilize channel

– high load: collision overhead

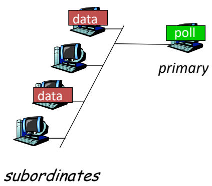
“taking turns” protocols:

look for best of both worlds!

“Taking Turns” MAC protocols

Polling:

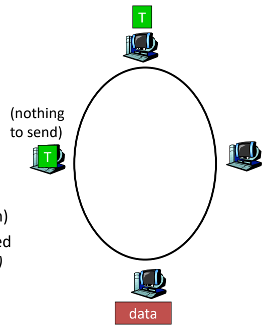
- Primary node “invites” subordinates nodes to transmit in turn
- typically used with simpler subordinate devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (primary)



“Taking Turns” MAC protocols

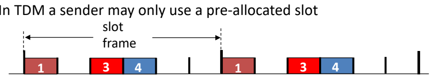
Token passing:

- control **token** passed from one node to next sequentially.
- token message
- concerns:
  - token overhead
  - latency
  - single point of failure (token)
- concerns fixed in part by a slotted ring (many simultaneous *tokens)*



ATM

In TDM a sender may only use a pre-allocated slot



In ATM a sender transmits labeled cells whenever necessary

ATM = Asynchronous Transfer Mode – an ugly expression
think of it as ATDM – Asynchronous Time Division Multiplexing

That’s a variant of **PACKET SWITCHING** to the rest of us – just like Ethernet
but using fixed length slots/packets/cells

Use the media when you need it, but
ATM had virtual circuits and these needed setup….

“Taking Turns” MAC protocols

channel partitioning MAC protocols:

– share channel *efficiently* and *fairly* at high load

– inefficient at low load: delay in channel access, 1/N bandwidth allocated even if only 1 active node!

random access MAC protocols:

– efficient at low load: single node can fully utilize channel

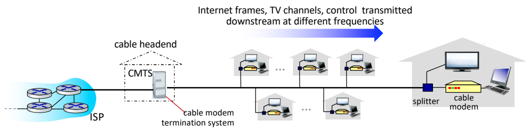
– high load: collision overhead

“taking turns” protocols:

look for best of both worlds!

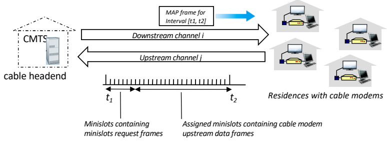
Recall…..

Cable access network: FDM, TDM *and* random access!



- multiple downstream (broadcast) FDM channels: up to 1.6 Gbps/channel
  - single CMTS transmits into channels
- multiple upstream channels (up to 1 Gbps/channel)
  - multiple access: all users contend (random access) for certain upstream channel time slots; others assigned TDM

Cable access network:



DOCSIS: data over cable service interface specification
- FDM over upstream, downstream frequency channels
- TDM upstream: some slots assigned, some have contention
  - downstream MAP frame: assigns upstream slots
  - request for upstream slots (and data) transmitted random access (binary backoff) in selected slots

Random Access MAC Protocols

- When node has packet to send
  - Transmit at full channel data rate
  - No *a priori* coordination among nodes
- Two or more transmitting nodes ⇒ collision
  - Data lost
- Random access MAC protocol specifies:
  - How to detect collisions
  - How to recover from collisions
- Examples
  - ALOHA and Slotted ALOHA
  - CSMA, CSMA/CD, CSMA/CA (wireless)

Key Ideas of Random Access

- Carrier sense
  - *Listen before speaking, and don't interrupt*
  - Checking if someone else is already sending data
  - … and waiting till the other node is done
- Collision detection
  - *If someone else starts talking at the same time, stop*
  - Realizing when two nodes are transmitting at once
  - …by detecting that the data on the wire is garbled
- Randomness
  - *Don't start talking again right away*
  - Waiting for a random time before trying again

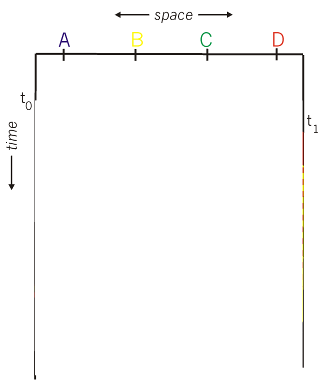
CSMA (Carrier Sense Multiple Access)

- CSMA: listen before transmit
  - If channel sensed idle: transmit entire frame
  - If channel sensed busy, defer transmission
- Human analogy: don't interrupt others!
- Does this eliminate all collisions?
  - No, because of nonzero propagation delay

CSMA Collisions

Propagation delay: two nodes may not hear each other's before sending.

Would slots hurt or help?



CSMA reduces but does not eliminate collisions

Biggest remaining problem?

Collisions still take full slot!
How do you fix that?

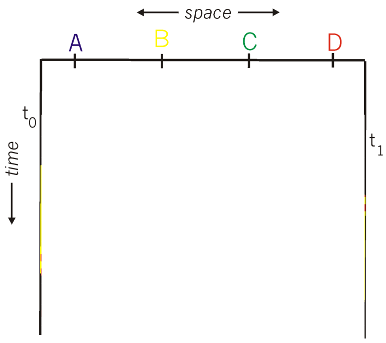
CSMA/CD (Collision Detection)

- CSMA/CD: carrier sensing, deferral as in CSMA
  - **Collisions detected within short time**
  - Colliding transmissions aborted, reducing wastage
- Collision detection easy in wired LANs:
  - Compare transmitted, received signals
- Collision detection difficult in wireless LANs:
  - Reception shut off while transmitting (well, perhaps not)
  - Not perfect broadcast (limited range) so collisions local
  - Leads to use of *collision avoidance* instead (later)

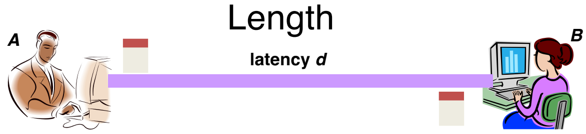
CSMA/CD Collision Detection

B and D can tell that collision occurred.

Note: for this to work, need restrictions on minimum frame size and maximum distance. Why?



Limits on CSMA/CD Network Length



- Latency depends on physical length of link
  - Time to propagate a packet from one end to the other
- Suppose *A* sends a packet at time ***t***
  - And *B* sees an idle line at a time just before ***t+d***
  - … so *B* happily starts transmitting a packet
- *B* detects a collision, and sends jamming signal
  - But *A* can't see collision until ***t+2d***

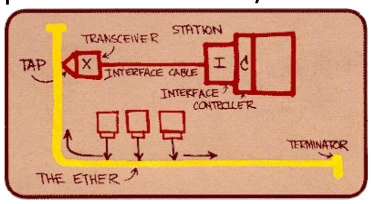
Performance of CSMA/CD

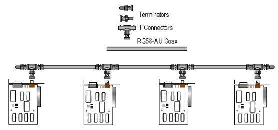
- Time wasted in collisions
  - Proportional to distance d
- Time spend transmitting a packet
  - Packet length p divided by bandwidth b
- Rough estimate for efficiency (K some constant)

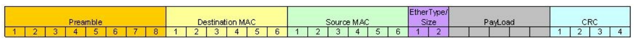
$$E \sim \frac{\frac{p}{b}}{\frac{p}{b} + Kd}$$

- Note:
  - For large packets, small distances, E ~ 1
  - As bandwidth increases, E decreases
  - That is why high-speed LANs are all switched aka packets are sent via a switch - (any d is bad)

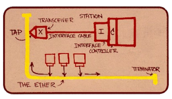
Ethernet…
yet another product of XEROX/PARC







Ethernet: CSMA/CD Protocol



- **Carrier sense**: wait for link to be idle
- **Collision detection**: listen while transmitting
  - No collision: transmission is complete
  - Collision: abort transmission & send **jam** signal
- **Random access**: binary exponential back-off
  - After collision, wait a random time before trying again
  - After m^{th} collision, choose K randomly from {0, …, 2^m-1}
  - … and wait for K*512 bit times before trying again
    - Using min packet size as "slot"
    - **If transmission occurring when ready to send, wait until end of transmission (CSMA)**

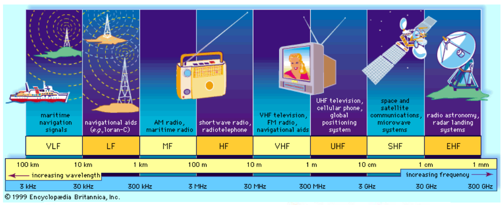
Benefits of Ethernet

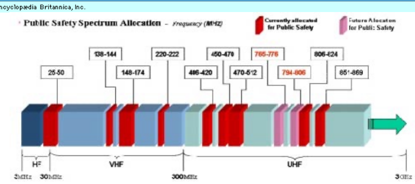
- Easy to administer and maintain
- Inexpensive
- Increasingly higher speed
- Evolvable!

Evolution of Ethernet

- Changed everything except the frame format
  - From single coaxial cable to hub-based star
  - From shared media to switches
  - From electrical signaling to optical
- Lesson #1
  - The right interface can accommodate many changes
  - Implementation is hidden behind interface
- Lesson #2
  - Really hard to displace the dominant technology
  - Slight performance improvements are not enough

The Wireless Spectrum





Metrics for evaluation / comparison of wireless technologies

- Bitrate or Bandwidth
- Range - PAN, LAN, MAN, WAN
- Two-way / One-way
- Multi-Access / Point-to-Point
- Digital / Analog
- Applications and industries
- Frequency – Affects most physical properties:
  - Distance (free-space loss)
  - Penetration, Reflection, Absorption
  - Energy proportionality
  - Policy: Licensed / Deregulated
  - Line of Sight (Fresnel zone)
  - Size of antenna
- ➢ Determined by wavelength – $\lambda = \frac{v}{f}$,)

Wireless Communication Standards

- Cellular (800/900/*1700*/1800/1900Mhz):
  - 2G: GSM / CDMA / GPRS /EDGE
  - 3G: CDMA2000/UMTS/HSDPA/EVDO
  - 4G: LTE, WiMax
- IEEE 802.11 (aka WiFi): (some examples)
  - b: 2.4Ghz band, 11Mbps (~*4.5 Mbps operating rate*)
  - g: 2.4Ghz, 54-108Mbps (~*19 Mbps operating rate*)
  - a: 5.0Ghz band, 54-108Mbps (~*25 Mbps operating rate*)
  - n: 2.4/5Ghz, 150-600Mbps (4x4 mimo)
  - ac: 2.4/5Ghz, 433-1300Mbps (improved coding 256-QAM)
  - ad: 60Ghz, 7Gbps
  - af: 54/790Mhz, 26-35Mbps (TV whitespace)
- IEEE 802.15 – lower power wireless:
  - 802.15.1: 2.4Ghz, 2.1 Mbps (Bluetooth)
  - 802.15.4: 2.4Ghz, 250 Kbps (Sensor Networks)

What Makes Wireless Different?

- Broadcast and multi-access medium…
  - err, so….
- BUT, Signals sent by sender don't always end up at receiver intact
  - Complicated physics involved, which we won't discuss
  - But what can go wrong?

Lets focus on 802.11

aka - WiFi …
What makes it special?

Deregulation > Innovation > Adoption > Lower cost = Ubiquitous technology

JUST LIKE ETHERNET – not lovely but sufficient

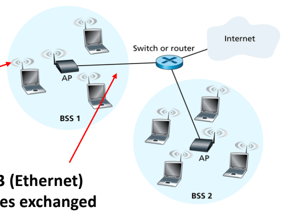
IEEE 802.11 Wireless LAN

IEEE 802.11 standard	Year	Max data rate	Range	Frequency
802.11b	1999	11 Mbps	30 m	2.4 Ghz
802.11g	2003	54 Mbps	30m	2.4 Ghz
802.11n (WiFi 4)	2009	600	70m	2.4, 5 Ghz
802.11ac (WiFi 5)	2013	3.47Gpbs	70m	5 Ghz
802.11ax (WiFi 6)	2020 (exp.)	14 Gbps	70m	2.4, 5 Ghz
802.11af	2014	35 – 560 Mbps	1 Km	unused TV bands (54-790 MHz)
802.11ah	2017	347Mbps	1 Km	900 Mhz

- all use CSMA/CA for multiple access, and have base-station and ad-hoc network versions

802.11 Architecture

802.11 frames exchanges



802.3 (Ethernet) frames exchanged

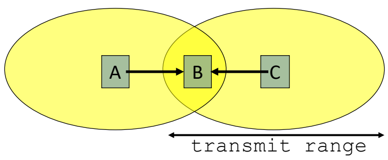
Figure 6.7 ♦ IEEE 802.11 LAN architecture

- Designed for limited area
- AP's (Access Points) set to specific channel
- Broadcast beacon messages with SSID (Service Set Identifier) and MAC Address periodically
- Hosts scan all the channels to discover the AP's
  - Host associates with AP

Wireless Multiple Access Technique?

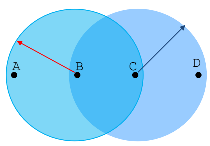
- Carrier Sense?
  - Sender can listen before sending
  - What does that tell the sender?
- Collision Detection?
  - Where do collisions occur?
  - How can you detect them?

Hidden Terminals



- A and C can both send to B but can't hear each other
  - A is a *hidden terminal* for C and vice versa
- Carrier Sense will be ineffective

Exposed Terminals



- Exposed node: B sends a packet to A; C hears this and decides not to send a packet to D (despite the fact that this will not cause interference)!
- Carrier sense would prevent a successful transmission.

Key Points

- No concept of a global collision
  - Different receivers hear different signals
  - Different senders reach different receivers
- Collisions are at receiver, not sender
  - Only care if receiver can hear the sender clearly
  - It does not matter if sender can hear someone else
  - As long as that signal does not interfere with receiver
- Goal of protocol:
  - Detect if receiver can hear sender
  - Tell senders who might interfere with receiver to shut up

Basic Collision Avoidance

- Since can't detect collisions, we try to *avoid* them
- Carrier sense:
  - When medium busy, choose random interval
  - Wait that many **idle** timeslots to pass before sending
- When a collision is inferred, retransmit with binary exponential backoff (like Ethernet)
  - Use ACK from receiver to infer "no collision"
  - Use exponential backoff to adapt contention window

IEEE 802.11 MAC Protocol: CSMA/CA

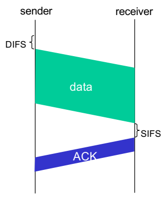
802.11 sender

1 if sense channel idle for **DIFS** then
 transmit entire frame (no CD)

2 if sense channel busy then
 start random backoff time
 timer counts down while channel idle
 transmit when timer expires
 if no ACK, increase random backoff interval, repeat 2

802.11 receiver

if frame received OK
 return ACK after **SIFS** (ACK needed due to hidden terminal problem)

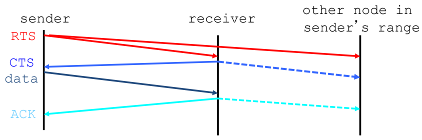


Avoiding collisions

idea: sender "reserves" channel use for data frames using small reservation packets

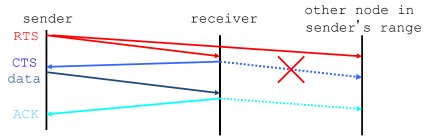
- sender first transmits *small* request-to-send (RTS) packet to BS using CSMA
  - RTSs may still collide with each other (but they're short)
- BS broadcasts clear-to-send CTS in response to RTS
- CTS heard by all nodes
  - sender transmits data frame
  - other stations defer transmissions

CSMA/CA – and in this case RTS/CTS



- Before every data transmission
  - Sender sends a Request to Send (RTS) frame containing the length of the transmission
  - Receiver respond with a Clear to Send (CTS) frame
  - Sender sends data
  - Receiver sends an ACK; now another sender can send data
- When sender doesn't get a CTS back, it assumes collision

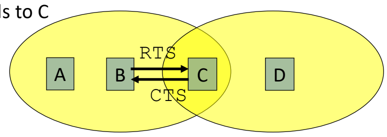
CSMA/CA, con't



- If other nodes hear RTS, but not CTS: send
  - Presumably, destination for first sender is out of node's range …
  - … Can cause problems when a CTS is lost
- When you hear a CTS, you keep quiet until scheduled transmission is over (hear ACK)

RTS / CTS Protocols (CSMA/CA)

B sends to C

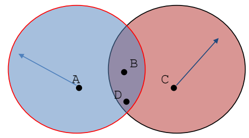


Overcome hidden terminal problems with contention-free protocol

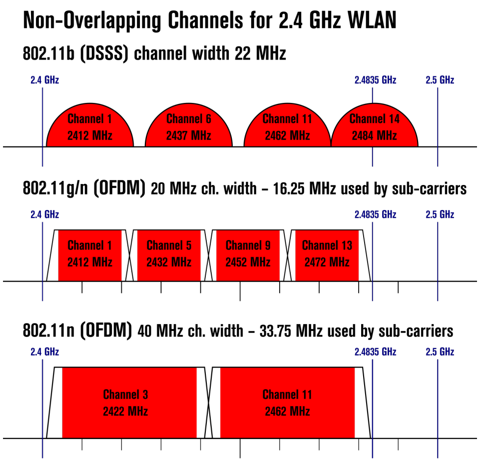
1. B sends to C Request To Send (RTS)
2. A hears RTS and defers (to allow C to answer)
3. C replies to B with Clear To Send (CTS)
4. D hears CTS and defers to allow the data
5. B sends to C

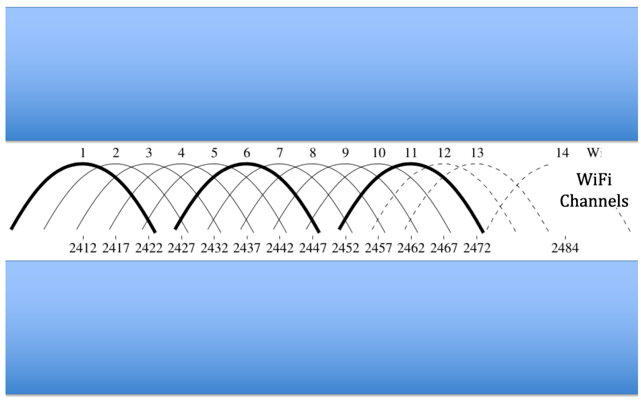
Preventing Collisions Altogether

- Frequency Spectrum partitioned into several channels
  - Nodes within interference range can use separate channels



  - Now A and C can send without any interference!
- Most cards have only 1 transceiver
  - **Not Full Duplex: Cannot send and receive at the same time**
  - Aggregate Network throughput doubles



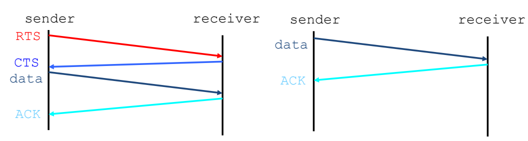


Wifi has been evolving!

Using dual band (2.4GHz + 5GHz), multiple channels, MIMO, Meshing WiFi

Outside this introduction but the state of the art is very fast and very flexible

CSMA/CA and RTS/CTS



RTS/CTS

- helps with hidden terminal
- good for high-traffic Access Points
- often turned on/off dynamically

Without RTS/CTS

- lower latency -> faster!
- reduces wasted b/w
  if the *Pr(collision)* is low
- good for when net is small and not *weird*
  eg no hidden/exposed terminals

CSMA/CD vs CSMA/CA (without RTS/CTS)

CD Collision Detect

wired – listen and talk

1. Listen for others
2. Busy? goto 1.
3. Send message (and listen)
4. Collision?
   a. JAM
   b. increase your BEB
   c. sleep
   d. goto 1.

CA Collision Avoidance

wireless – talk OR listen

1. Listen for others
2. Busy? goto 1.
3. Send message
4. Wait for ACK (*MAC ACK*)
5. Got No ACK from MAC?
   a. increase your BEB
   b. sleep
   c. goto 1.

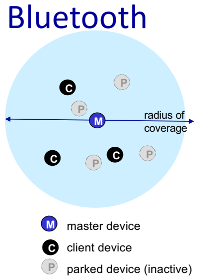
802.11: advanced capabilities

power management

- node-to-AP: "I am going to sleep until next beacon frame"
  - AP knows not to transmit frames to this node
  - node wakes up before next beacon frame
- beacon frame: contains list of mobiles with AP-to-mobile frames waiting to be sent
  - node will stay awake if AP-to-mobile frames to be sent; otherwise sleep again until next beacon frame

Personal area networks: Bluetooth

- TDM, 625 μsec sec. slot
- FDM: sender uses 79 frequency channels in known, pseudo-random order slot-to-slot (spread spectrum)
  - other devices/equipment not in piconet only interfere in some slots
- parked mode: clients can "go to sleep" (park) and later wakeup (to preserve battery)
- bootstrapping: nodes self-assemble (plug and play) into piconet



Summary of MAC protocols

- *channel partitioning,* by time, frequency or code
  - Time Division (TDMA), Frequency Division (FDMA), Code Division (CDMA)
- *random access* (dynamic),
  - ALOHA, S-ALOHA, CSMA, CSMA/CD
  - carrier sensing: easy in some technologies (wire), hard in others (wireless)
  - CSMA/CD used in (old-style, coax) Ethernet, and PowerLine
  - CSMA/CA used in 802.11
- *taking turns*
  - polling from central site, token passing
  - Bluetooth, FDDI, IBM Token Ring

MAC Addresses

- MAC (or LAN or physical or Ethernet) address:
  - function: *get frame from one interface to another physically-connected interface (same network)*
  - 48 bit MAC address (for most LANs)
    - *burned* in NIC ROM, nowadays usually software settable and set at boot time

```
awm22@rio:~$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:30:48:fe:c0:64
          inet addr:128.232.33.4  Bcast:128.232.47.255  Mask:255.255.240.0
          inet6 addr: fe80::230:48ff:fefe:c064/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:215084512 errors:252 dropped:25 overruns:0 frame:123
          TX packets:146711866 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:170815941033 (170.8 GB)  TX bytes:86755864270 (86.7 GB)
          Memory:f0000000-f0020000
```


LAN Address (more)

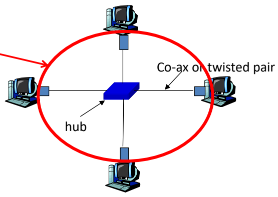
- MAC address allocation administered by IEEE
- manufacturer buys portion of MAC address space (to assure uniqueness)
- analogy:
  (a) MAC address: like a National Insurance Number
  (b) IP address: like a postal address
- MAC flat address ➜ portability
  - can move LAN card from one LAN to another
- IP hierarchical address NOT portable
  - address depends on IP subnet to which node is attached

Hubs

… physical-layer (“dumb”) repeaters:

- bits coming in one link go out *all* other links at same rate
- all nodes connected to hub can collide with one another
- no frame buffering
- no CSMA/CD at hub: host NICs detect collisions

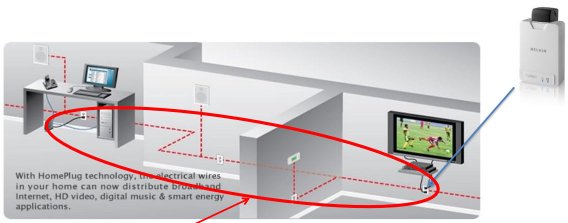
Collision Domain
in CSMA/CD *speak*



CSMA in our home

Home Plug Powerline Networking….

Home Plug and similar Powerline Networking….



Collision Domain
in CSMA *speak*

To secure network traffic on a specific HomePlug network, each set of adapters use an encryption key common to a specific HomePlug network

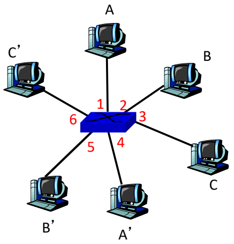
Switch (example: Ethernet Switch)

- link-layer device: smarter than hubs, take *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
- *transparent*
  - hosts are unaware of presence of switches
- *plug-and-play, self-learning*
  - switches do not need to be configured

If you want to connect different physical media
(optical – copper – coax – wireless - ….)
you **NEED** a switch.
Why? (Because each link, each media access protocol is specialised)

Switch: allows *multiple* simultaneous transmissions

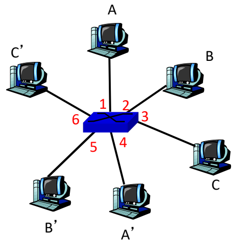
- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, but no collisions; full duplex
  - each link is its own collision domain
- *switching:* A-to-A’ and B-to-B’ simultaneously, without collisions
  - not possible with dumb hub



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

Switch Table

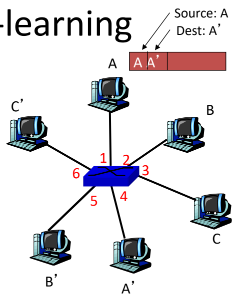
- *Q:* how does switch know that A’ reachable via interface 4, B’ reachable via interface 5?
- *A:* each switch has a switch table, each entry:
  - (MAC address of host, interface to reach host, time stamp)
- looks like a routing table!
- *Q:* how are entries created, maintained in switch table?
  - something like a routing protocol?



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

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



MAC addr	interface	TTL
A	*1*	*60*

Switch table (initially empty)

Switch: frame filtering/forwarding

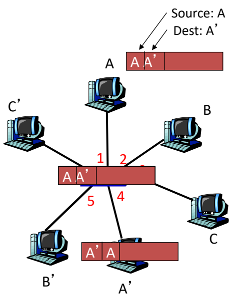
When frame received:

1. record link associated with sending host
2. index switch table using MAC dest address
3. **if** entry found for destination
   then {
   if dest on segment from which frame arrived
   then drop the frame
   else forward the frame on interface indicated
   }
   else flood

forward on all but the interface on which the frame arrived

Self-learning, forwarding: example

- frame destination unknown: *flood*
- destination A location known: selective send

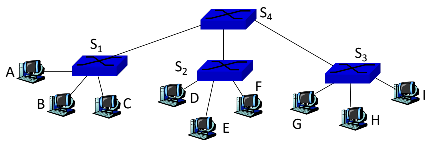


MAC addr	interface	TTL
A *A'*	*1* *4*	*60* *60*

Switch table (initially empty)

Interconnecting switches

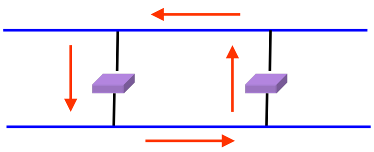
- switches can be connected together



- Q: sending from A to G - how does S_1 know to forward frame destined to F via S_4 and S_3?
- A: self learning! (works exactly the same as in single-switch case – flood/forward/drop)

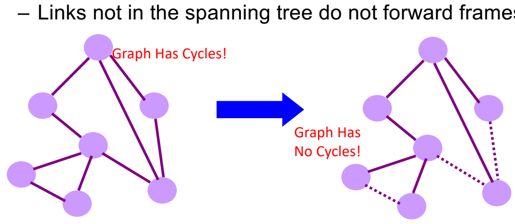
Flooding Can Lead to Loops

- Flooding can lead to forwarding loops
  - E.g., if the network contains a cycle of switches
  - "Broadcast storm"



Solution: Spanning Trees

- Ensure the forwarding topology has no loops
  - Avoid using some of the links when flooding
  - … to prevent loop from forming
- Spanning tree
  - Sub-graph that covers all vertices but *contains no cycles*
  - Links not in the spanning tree do not forward frames

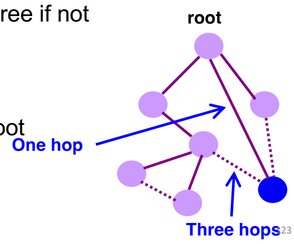


What Do We Know?

- *"Spanning tree algorithm is an algorithm to create a tree out of a graph that includes all nodes with a minimum number of edges connecting to vertices."*
- Shortest paths to (or from) a node form a tree
- So, algorithm has two aspects :
  - Pick a root
  - Compute shortest paths to it
- Only keep the links on shortest-path

Constructing a Spanning Tree

- Switches need to elect a root
  - The switch w/ smallest identifier (MAC addr)
- Each switch determines if each interface is on the shortest path from the root
  - Excludes it from the tree if not
- Messages (Y, d, X)
  - From node X
  - Proposing Y as the root
  - And the distance is d

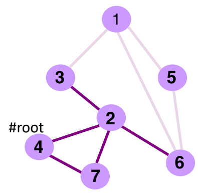


Steps in Spanning Tree Algorithm

- Initially, each switch proposes itself as the root
  - Switch sends a message out every interface
  - … proposing itself as the root with distance 0
  - Example: switch X announces (X, 0, X)
- Switches update their view of the root
  - Upon receiving message (Y, d, Z) from Z, check Y's id
  - If new id smaller, start viewing that switch as root
- Switches compute their distance from the root
  - Add 1 to the distance received from a neighbor
  - Identify interfaces not on shortest path to the root
  - … and exclude them from the spanning tree
- If root or shortest distance to it changed, "flood" updated message (Y, d+1, X)

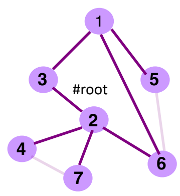
Example From Switch #4's Viewpoint

- Switch #4 thinks it is the root
  - Sends (4, 0, 4) message to 2 and 7
- Then, switch #4 hears from #2
  - Receives (2, 0, 2) message from 2
  - … and thinks that #2 is the root
  - And realizes it is just one hop away
- Then, switch #4 hears from #7
  - Receives (2, 1, 7) from 7
  - And realizes this is a longer path
  - So, prefers its own one-hop path
  - And removes 4-7 link from the tree



Example From Switch #4's Viewpoint

- Switch #2 hears about switch #1
  - Switch 2 hears (1, 1, 3) from 3
  - Switch 2 starts treating 1 as root
  - And sends (1, 2, 2) to neighbors
- Switch #4 hears from switch #2
  - Switch 4 starts treating 1 as root
  - And sends (1, 3, 4) to neighbors
- Switch #4 hears from switch #7
  - Switch 4 receives (1, 3, 7) from 7
  - And realizes this is a longer path
  - So, prefers its own three-hop path
  - And removes 4-7 link from the tree



Robust Spanning Tree Algorithm

- Algorithm must react to failures
  - Failure of the root node
    - Need to elect a new root, with the next lowest identifier
  - Failure of other switches and links
    - Need to recompute the spanning tree
- Root switch continues sending messages
  - Periodically reannouncing itself as the root (1, 0, 1)
  - Other switches continue forwarding messages
- Detecting failures through timeout (soft state)
  - If no word from root, times out and claims to be the root
  - Delay in reestablishing spanning tree is ***major problem***
  - Work on rapid spanning tree algorithms…

Given a switch-tree of a given size, link length, speed of computation, …

How long does a failure take to rectify?

Weirder “Data Link Layer” Networks

VLAN

Application
Transport
Network
Data Link (L2)
Data Link (L2)
Physical

VPN

Application
Transport
Network
Transport
Network
Data Link (L2)
Physical

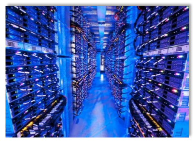
Datacenter

“so you think your LAN has a lot of computers….”

Datacenter networks

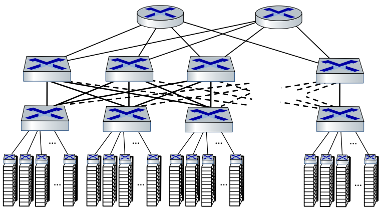
10’s to 100’s of thousands of hosts, often closely coupled, in close proximity:
- e-business (e.g. Amazon)
- content-servers (e.g., YouTube, Akamai, Apple, Microsoft)
- search engines, data mining (e.g., Google)

challenges:
- multiple applications, each serving massive numbers of clients
- reliability
- managing/balancing load, avoiding processing, networking, data bottlenecks



Inside a 40-ft Microsoft container, Chicago data center

Datacenter networks: network elements



Border routers
- connections outside datacenter

Tier-1 switches
- connecting to ~16 T-2s below

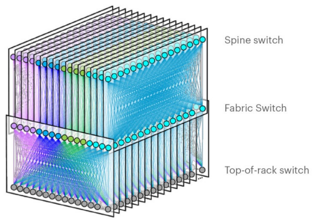
Tier-2 switches
- connecting to ~16 TORs below

Top of Rack (TOR) switch
- one per rack
- 40-100Gbps Ethernet to blades

Server racks
- 20- 40 server blades: hosts

Datacenter networks: network elements

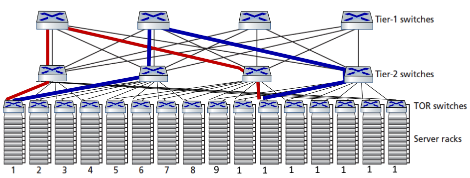
Facebook F16 data center network topology:



https://engineering.fb.com/data-center-engineering/f16-minipack/ (posted 3/2019)

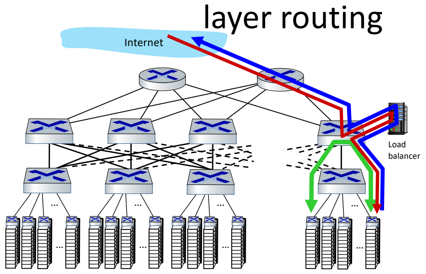
Datacenter networks: multipath

- rich interconnection among switches, racks:
  - increased throughput between racks (multiple routing paths possible)
  - increased reliability via redundancy



two disjoint paths highlighted between racks 1 and 11

Datacenter networks: application-layer routing



load balancer: application-layer routing
- receives external client requests
- directs workload within data center
- returns results to external client (hiding data center internals from client)

Summary

- principles behind data link layer services:
  - error detection, correction
  - sharing a broadcast channel: multiple access
  - link layer addressing
- instantiation and implementation of various link layer technologies
  - Ethernet
  - switched LANS
  - WiFi
- algorithms
  - Binary Exponential Backoff
  - Spanning Tree

Topic 4: Network Layer

Our goals:

- understand principles behind network layer services:
  - network layer service models
  - forwarding versus routing (versus switching)
  - how a router works
  - routing (path selection)
  - IPv6

For the most part, the Internet is our example – again.

Recall: Network layer is responsible for ***GLOBAL*** delivery

Name: a *something*

Address: Where is a *something*

Routing: How do I get to the *something*

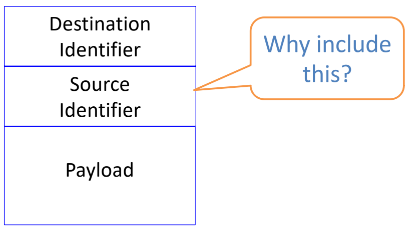
Forwarding: What path do I take next to get to the *something*

Addressing (at a conceptual level)

- Assume all hosts have unique IDs
- No particular structure to those IDs
- Later in topic I will talk about real IP addressing
- Do I route on location or identifier?
- If a host moves, should its address change?
  - If not, how can you build scalable Internet?
  - If so, then what good is an address for identification?

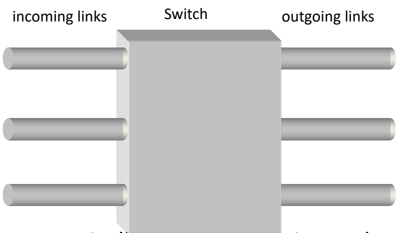
Packets (at a conceptual level)

- Assume packet headers contain:
  - Source ID, Destination ID, and perhaps other information



Switches/Routers

- Multiple ports (attached to other switches or hosts)



- Ports are typically duplex (incoming and outgoing)

A Variety of *(Internet Protocol-based)* Networks

- ISPs: carriers
  - Backbone
  - Edge
  - Border (to other ISPs)
- Enterprises: companies, universities
  - Core
  - Edge
  - Border (to outside)
- Datacenters: massive collections of machines
  - Top-of-Rack
  - Aggregation and Core
  - Border (to outside)

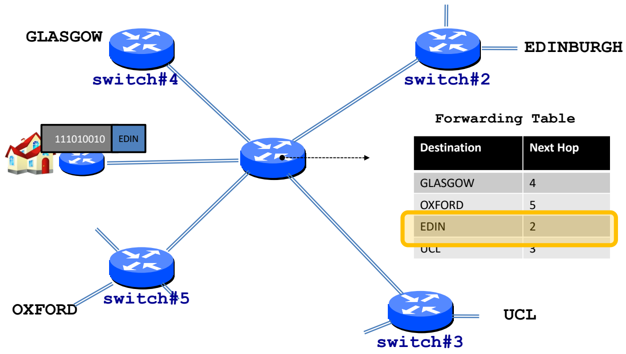
A Variety of *(Internet Protocol-based)* Routers

- ISPs: carriers
  - Backbone
  - Edge
  - Border (to other ISPs)
- Enterprises: companies, universities
  - Core
  - Edge
  - Border (to outside)
- Datacenters: massive collections of machines
  - Top-of-Rack
  - Aggregation and Core
  - Border (to outside)




Recall….

Switches forward packets



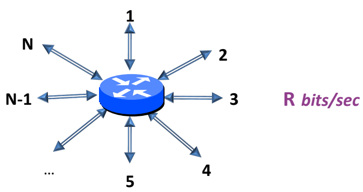
Forwarding Decisions

- When packet arrives..
  - Must decide which outgoing port to use
  - In single transmission time
  - Forwarding decisions must be ***simple***
- Routing state dictates where to forward packets
  - Assume decisions are **deterministic**
- *Global routing state* is the collection of routing state in each of the routers
  - Will focus on where this routing state comes from
  - But first, a few preliminaries….

Forwarding vs Routing

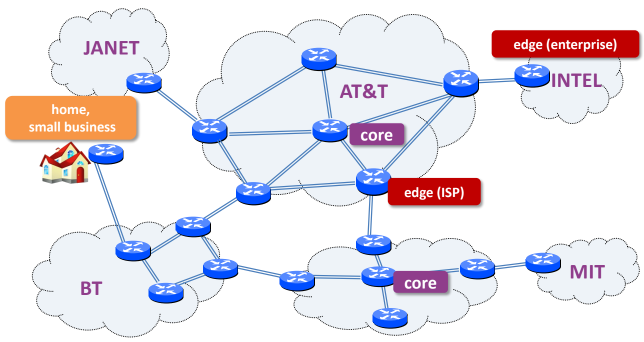
- Forwarding: “data plane”
  - Directing a data packet to an outgoing link
  - Individual router using routing state
- Routing: “control plane”
  - Computing paths the packets will follow
  - Routers talking amongst themselves
  - Jointly creating the routing state
- Two very different timescales….

Router definitions

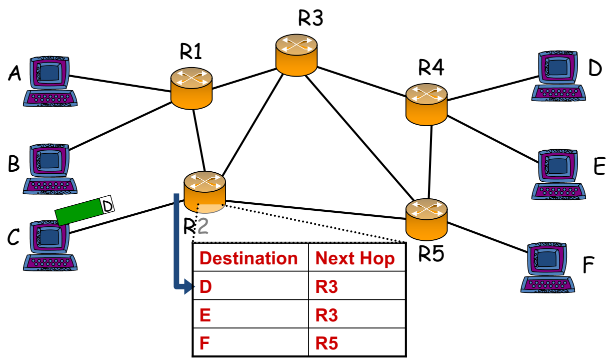


- **N = number of external router “ports”**
- **R = speed (“line rate”) of a port**
- **Router capacity = N x R**

Networks and routers

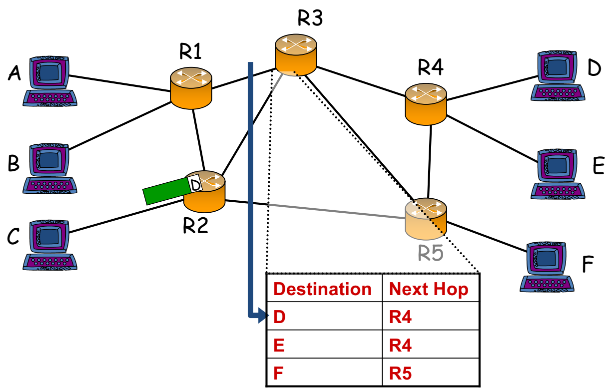


Basic Operation of Router



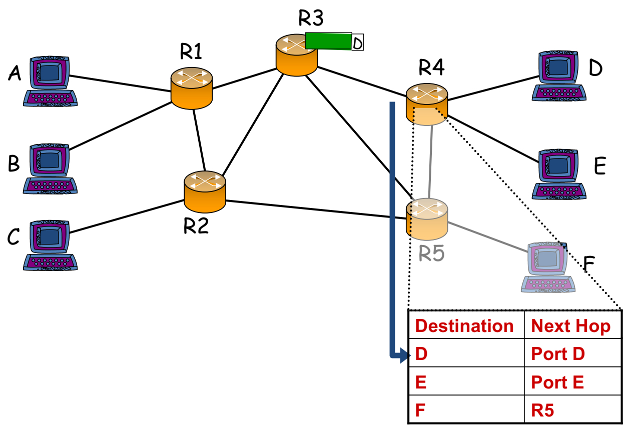
Destination	Next Hop
D	R3
E	R3
F	R5

Basic Operation of Router



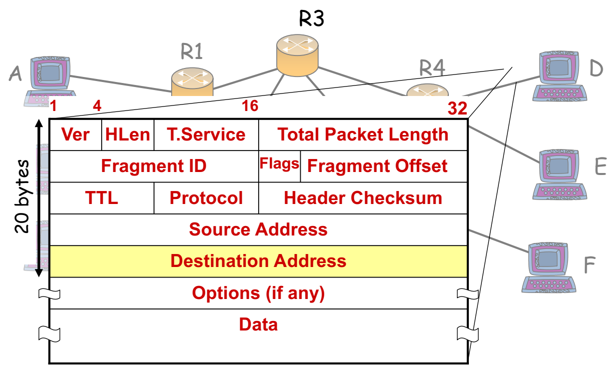
Destination	Next Hop
D	R4
E	R4
F	R5

Basic Operation of Router

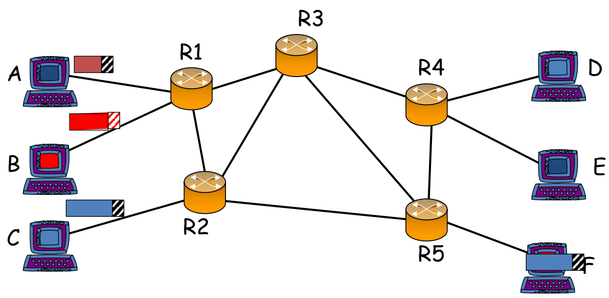


Destination	Next Hop
D	Port D
E	Port E
F	R5

What does a router do?

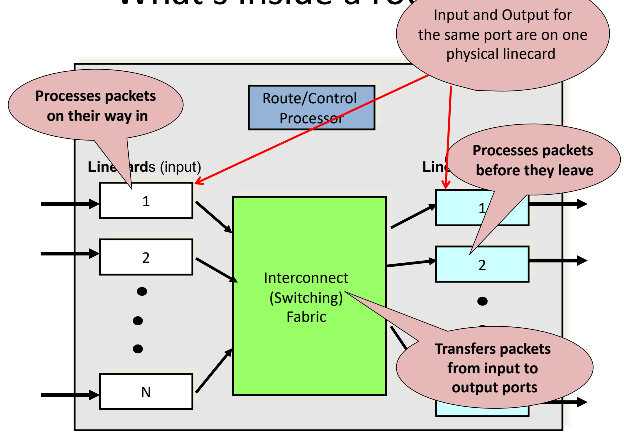


What does a router do?

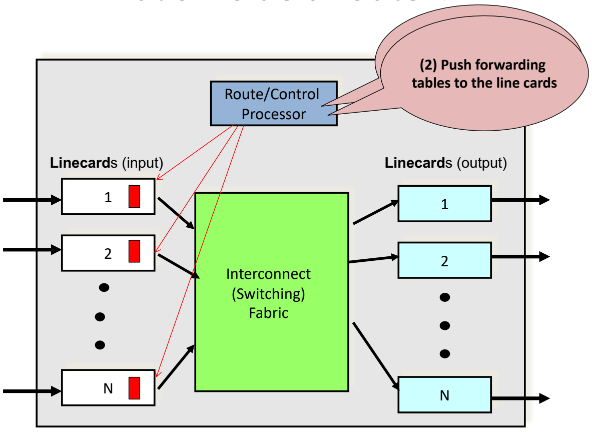


1. Every router performs a per-packet lookup for every packet
2. Each router performs a lookup in it's local lookup table
3. Each router performs lookups (ENTIRELY) independently of every other router

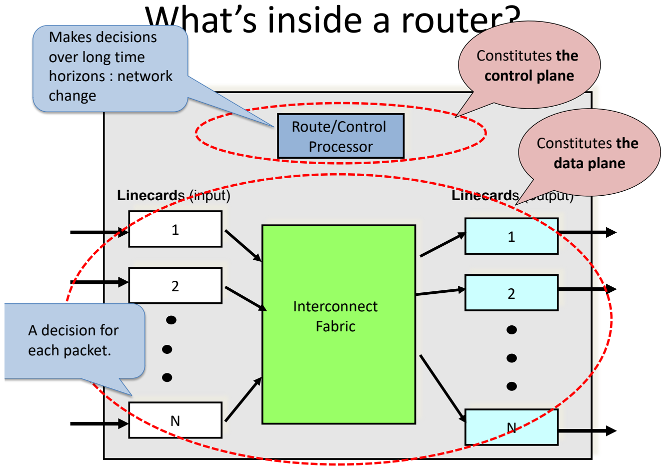
What's inside a rout

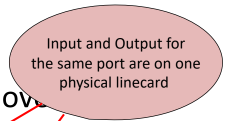


What's inside a router?



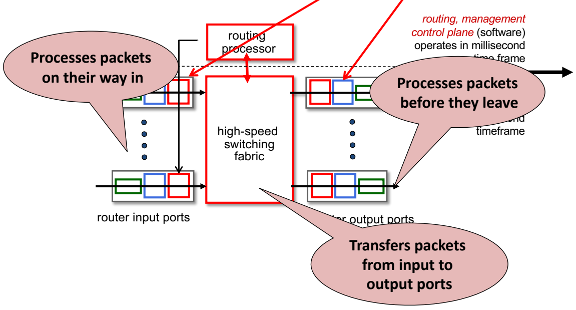
What's inside a router?



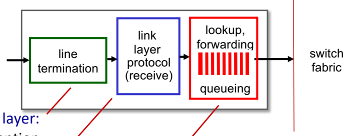


Router architecture ov

high-level view of generic router architecture:



Input port functions

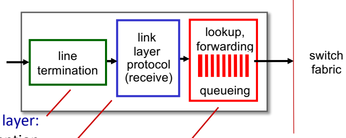


physical layer:
bit-level reception

link layer:
e.g., Ethernet
(Topic3)

decentralized switching:
- using header field values, lookup output port using forwarding table in input port memory *("match plus action")*
- goal: complete input port processing at 'line speed'
- input port queuing: if datagrams arrive faster than forwarding rate into switch fabric

Input port functions



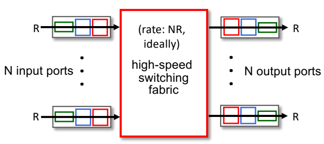
physical layer:
bit-level reception

link layer:
e.g., Ethernet
(chapter 6)

decentralized switching:
- using header field values, lookup output port using forwarding table in input port memory *("match plus action")*
- destination-based forwarding: forward based only on destination IP address (traditional)
- generalized forwarding: forward based on any set of header field values

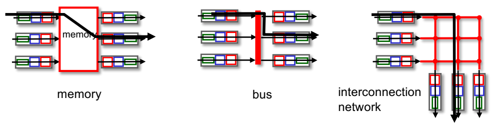
Switching fabrics

- transfer packet from input link to appropriate output link
- switching rate: rate at which packets can be transfer from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable



Switching fabrics

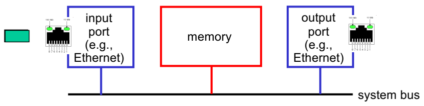
- transfer packet from input link to appropriate output link
- switching rate: rate at which packets can be transfer from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
- three major types of switching fabrics:



Switching via memory

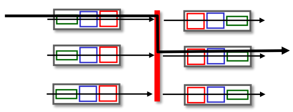
first generation routers:

- traditional computers with switching under direct control of CPU
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)



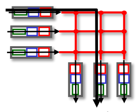
Switching via a bus

- datagram from input port memory to output port memory via a shared bus
- *bus contention:* switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access routers

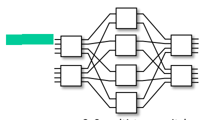


Switching via interconnection network

- Crossbar, Clos networks, other interconnection nets initially developed to connect processors in multiprocessor
- multistage switch: *nxn* switch from multiple stages of smaller switches
- exploiting parallelism:
  - fragment datagram into fixed length cells on entry
  - switch cells through the fabric, reassemble datagram at exit



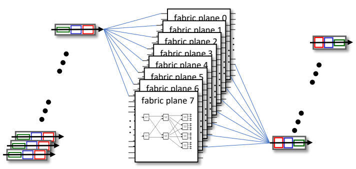
3x3 crossbar



8x8 multistage switch
built from smaller-sized switches

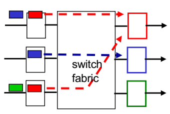
Switching via interconnection network

- scaling, using multiple switching "planes" in parallel:
  - speedup, scaleup via parallelism
- Cisco CRS router:
  - basic unit: 8 switching planes
  - each plane: 3-stage interconnection network
  - up to 100's Tbps switching capacity

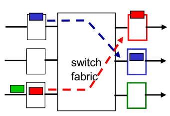


Input port queuing

- If switch fabric slower than input ports combined -> queueing may occur at input queues
  - queueing delay and loss due to input buffer overflow!
- Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward

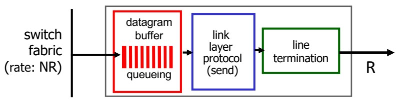


output port contention: only one red datagram can be transferred. lower red packet is *blocked*



one packet time later: green packet experiences HOL blocking

Output port queuing



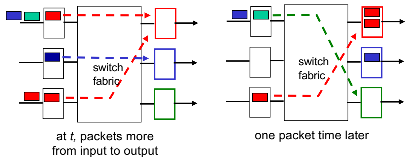
- *Buffering* required when datagrams arrive from fabric faster than link transmission rate. *Drop policy:* which datagrams to drop if no free buffers?

Datagrams can be lost due to congestion, lack of buffers

- *Scheduling discipline* chooses among queued datagrams for transmission

Priority scheduling – who gets best performance, network neutrality

Output port queuing



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

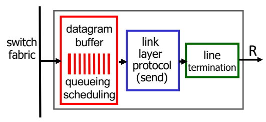
How much buffering? (related material in Topic 5)

- RFC 3439 rule of thumb: average buffering equal to "typical" RTT (say 250 msec) times link capacity C
  - e.g., C = 10 Gbps link: 2.5 Gbit buffer
- more recent recommendation: with *N* flows, buffering equal to

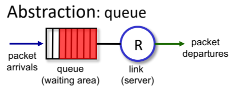
$$\frac{RTT \cdot C}{\sqrt{N}}$$

- but *too* much buffering can increase delays (particularly in home routers)
  - long RTTs: poor performance for realtime apps, sluggish TCP response
  - recall delay-based congestion control: "keep bottleneck link just full enough (busy) but no fuller"

Buffer Management



Abstraction: queue



buffer management:

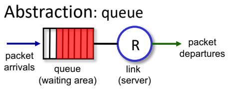
- drop: which packet to add, drop when buffers are full
  - tail drop: drop arriving packet
  - priority: drop/remove on priority basis
- marking: which packets to mark to signal congestion (ECN, RED)

Packet Scheduling: FCFS

packet scheduling: deciding which packet to send next on link

- first come, first served
- priority
- round robin
- weighted fair queueing

Abstraction: queue



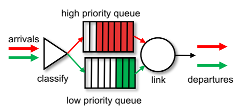
FCFS: packets transmitted in order of arrival to output port

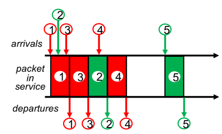
- also known as: First-in-first-out (FIFO)
- real world examples?

Scheduling policies: priority

Priority scheduling:

- arriving traffic classified, queued by class
  - any header fields can be used for classification
- send packet from highest priority queue that has buffered packets
  - FCFS within priority class

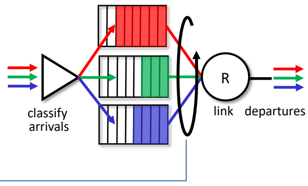




Scheduling policies: round robin

Round Robin (RR) scheduling:

- arriving traffic classified, queued by class
  - any header fields can be used for classification
- server cyclically, repeatedly scans class queues, sending one complete packet from each class (if available) in turn



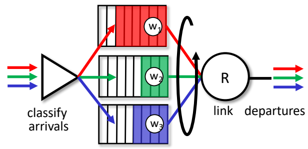
Scheduling policies: weighted fair queueing

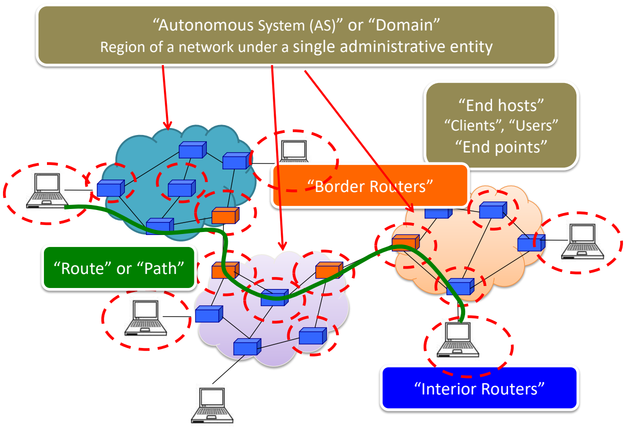
Weighted Fair Queuing (WFQ):

- generalized Round Robin
- each class, *i*, has weight, w_i, and gets weighted amount of service in each cycle:

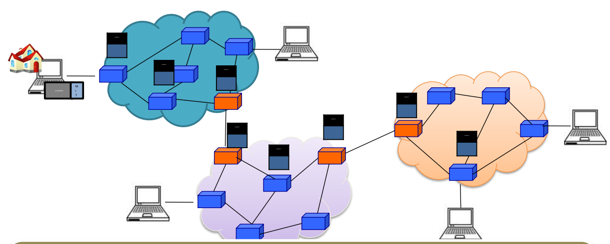
$$\frac{w_i}{\Sigma_j w_j}$$

- minimum bandwidth guarantee (per-traffic-class)





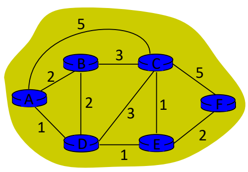
Context and Terminology



Internet routing protocols are responsible for constructing and updating the forwarding tables at routers

Routing Protocols

- Routing protocols implement the core function of a network
  - Establish paths between nodes
  - Part of the network's "control plane"
- Network modeled as a graph
  - Routers are graph vertices
  - Links are edges
  - Edges have an associated "cost"
    - e.g., distance, loss



- Goal: compute a "good" path from source to destination
  - "good" usually means the shortest (least cost) path

Internet Routing

- 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)

Addressing (to date)
- a reminder -

Recall each host has a unique ID (address)

- No particular structure to those IDs
  (e.g. *Ethernet*)
- IP addressing – in contrast – has implicit structure
  (Why???)

Outline

- Popular Routing Algorithms:
  – Link State Routing
  – Distance Vector Algorithm
- Routing: goals and metrics

Link-State Routing

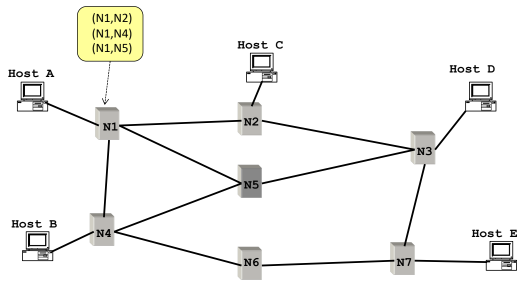
Examples:

Open Shortest Path First (**OSPF**) or
Intermediate System to Intermediate System
(written as **IS-IS**/**ISIS** and pronounced eye-esss-eye-esss)

The two common Intradomain routing or
interior gateway protocols (IGP)

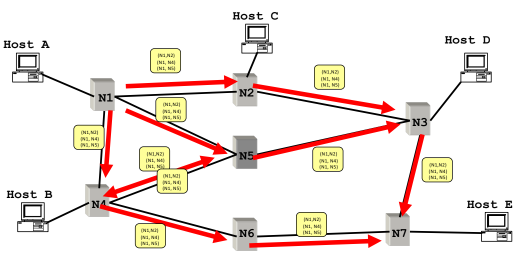
Link State Routing

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



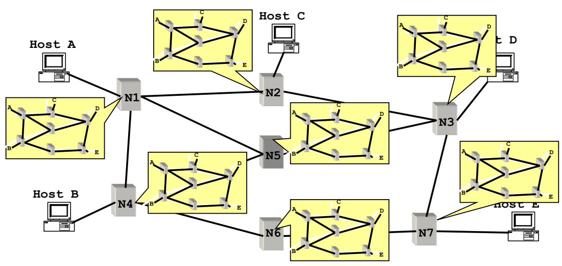
Link State Routing

- 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



Link State Routing

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

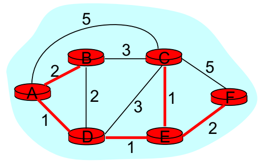


Dijkstra's Shortest Path Algorithm

- INPUT:
  – Network topology (graph), with link costs
- OUTPUT:
  – Least cost paths from one node to all other nodes
- Iterative: after *k* iterations, a node knows the least cost path to its *k* closest neighbors
- This is covered in Algorithms

The Forwarding Table

- Running Dijkstra at node A gives the shortest path from A to all destinations
- We then construct the *forwarding table*



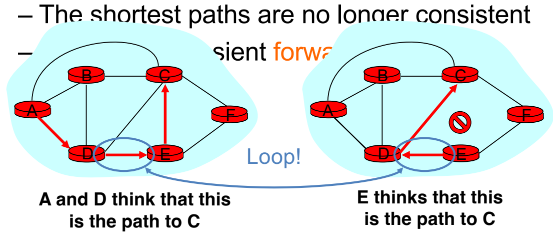
Destination	Link
B	(A,B)
C	(A,D)
D	(A,D)
E	(A,D)
F	(A,D)

Issue #1: Scalability

- How many messages needed to flood link state messages?
  – O(N x E), where N is #nodes; E is #edges in graph
- Processing complexity for Dijkstra's algorithm?
  – $O(N^2)$, because we check all nodes w not in S at each iteration and we have O(N) iterations
  – more efficient implementations: O(N log(N))
- How many entries in the LS topology database? O(E)
- How many entries in the forwarding table? O(N)

Issue#2: Transient Disruptions

- Inconsistent link-state database
  – Some routers know about failure before others
  – The shortest paths are no longer consistent
  – sient forwa



Distance Vector Routing

Learn-By-Doing

Let's try to collectively develop distance-vector routing from first principles

Experiment

- Your job: find the (route to) the youngest person in the room
- Ground Rules
  - **You may not** leave your seat, nor shout loudly across the class
  - **You may** talk with your immediate neighbors (N-S-E-W only)

  (hint: "exchange updates" with them)
- At the end of 5 minutes, I will pick a victim and ask:
  - who is the youngest person in the room? (date&name)
  - which one of your neighbors first told you this info.?

EQUIPMENT REQUIRED: PIECE OF PAPER and a PEN (or your emotional equivalent)

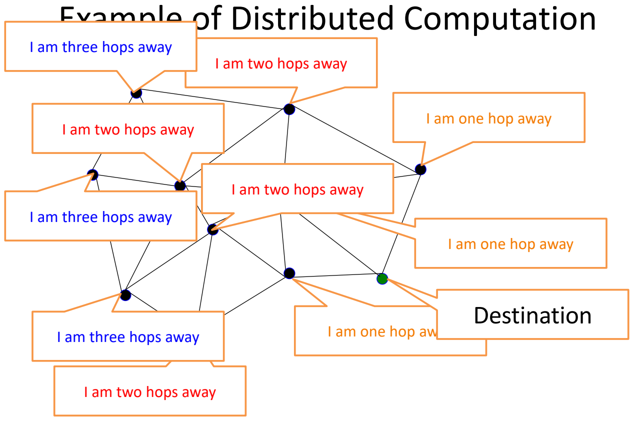
Go!

Distance-Vector Routing

Example:

Routing Information Protocol (RIP)

Example of Distributed Computation



Distance Vector Routing

Each router sends its knowledge about the "whole" network to its neighbors. Information sharing at regular intervals.

- Each router knows the links to its neighbors
  - Does *not* flood this information to the whole network
- Each router has provisional "shortest path" to every other router
  - E.g.: Router A: "I can get to router B with cost 11"
- Routers exchange this distance vector information with their neighboring routers
  - Vector because one entry per destination
- Routers look over the set of options offered by their neighbors and select the best one
- Iterative process converges to set of shortest paths

A few other inconvenient truths

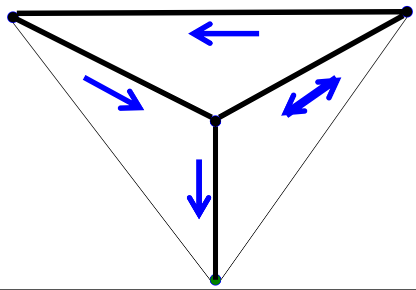
- What if we use a non-additive metric?
  - E.g., maximal capacity
- What if routers don't use the same metric?
  - I want low delay, you want low loss rate?
- What happens if nodes lie?

Can You Use Any Metric?

- I said that we can pick any metric. Really?
- What about maximizing capacity?

What Happens Here?

Problem: "cost" does not change around loop

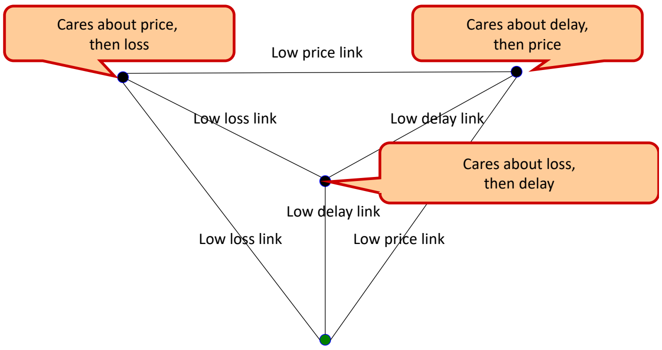


Additive measures avoid this problem!

No agreement on metrics?

- If the nodes choose their paths according to different criteria, then bad things might happen
- Example
  - Node A is minimizing latency
  - Node B is minimizing loss rate
  - Node C is minimizing price
- Any of those goals are fine, if globally adopted
  - Only a problem when nodes use different criteria
- Consider a routing algorithm where paths are described by delay, cost, loss

What Happens Here?



Must agree on loop-avoiding metric

- When all nodes minimize same metric
- And that metric increases around loops
- Then process is guaranteed to converge

What happens when routers lie?

- What if a router claims a 1-hop path to everywhere?
- All traffic from nearby routers gets sent there
- How can you tell if they are lying?
- Can this happen in real life?
  - It has, several times….

Link State vs. Distance Vector

- Core idea
  - LS: tell all nodes about your immediate neighbors
  - DV: tell your immediate neighbors about (your least cost distance to) all nodes

Link State vs. Distance Vector

- LS: each node learns the complete network map; each node computes shortest paths independently and in parallel
- DV: no node has the complete picture; nodes cooperate to compute shortest paths in a distributed manner

→LS has higher messaging overhead
→LS has higher processing complexity
→LS is less vulnerable to looping

Link State vs. Distance Vector

Message complexity
- LS: O(NxE) messages;
  - N is #nodes; E is #edges
- DV: O(#Iterations x E)
  - where #Iterations is ideally O(network diameter) but varies due to routing loops or the count-to-infinity problem

Processing complexity
- LS: O(N^2)
- DV: O(#Iterations x N)

Robustness: what happens if router malfunctions?
- LS:
  - node can advertise incorrect *link* cost
  - each node computes only its *own* table
- DV:
  - node can advertise incorrect *path* cost
  - each node's table used by others; error propagates through network

Routing: Just the Beginning

- Link state and distance-vector are the deployed routing paradigms for intra-domain routing
- Inter-domain routing (BGP)
  - more Part II (Principles of Communications)
  - A version of DV

What are desirable goals for a routing solution?

- "Good" paths (least cost)
- Fast convergence after change/failures
  - no/rare loops
- Scalable
  - #messages
  - table size
  - processing complexity
- Secure
- Policy
- Rich metrics (more later)

Delivery models

- What if a node wants to send to more than one destination?
  - broadcast: send to all
  - multicast: send to all members of a group
  - anycast: send to any member of a group
- What if a node wants to send along more than one path?

Metrics

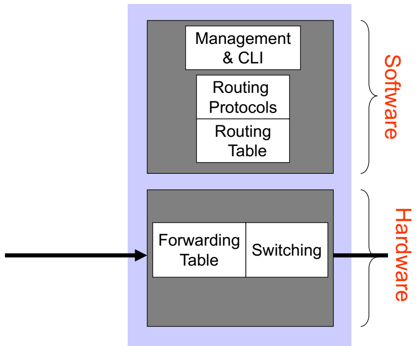
- Propagation delay
- Congestion
- Load balance
- Bandwidth (available, capacity, maximal, bbw)
- Price
- Reliability
- Loss rate
- Combinations of the above

In practice, operators set abstract “weights” (much like our costs); how exactly is a bit of a black art

From Routing back to Forwarding

- Routing: “control plane”
  - Computing paths the packets will follow
  - Routers talking amongst themselves
  - Jointly creating the routing state
- Forwarding: “data plane”
  - Directing a data packet to an outgoing link
  - Individual router using routing state
- Two very different timescales….

Basic Architectural Components of an IP Router



Control Plane
network-change processing

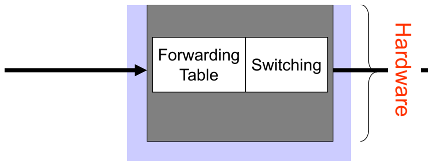
Datapath
per-packet processing

Independent operation!

If the control-plane **fails**…..

The data-path is **not affected**…
like a loyal pet it will keep going using the current (last) table update

This is a feature **not** a bug

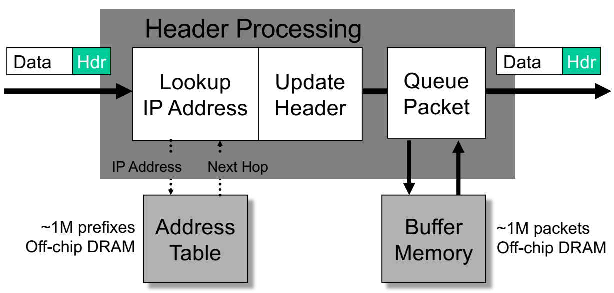


Datapath
per-packet processing

Per-packet processing in an IP Router

1. Accept packet arriving on an incoming link.
2. Lookup packet destination address in the forwarding table, to identify outgoing port(s).
3. Manipulate packet header: e.g., decrement TTL, update header checksum.
4. Send packet to the outgoing port(s).
5. Buffer packet in the queue.
6. Transmit packet onto outgoing link.

Generic Router Architecture



Forwarding tables

IP address }32 bits wide → ~ 4 billion unique address

Naïve approach:
One entry per address

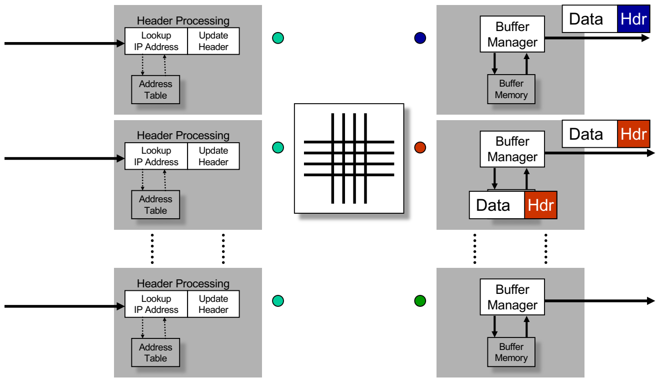
Entry	Destination	Port
1	0.0.0.0	1
2	0.0.0.1	2
⋮	⋮	⋮
2^{32}	255.255.255.255	12

} ~ **4 billion entries**

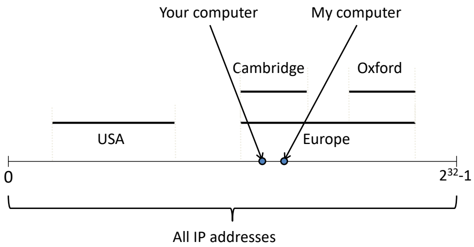
Improved approach:
Group (and ***SORT)*** entries to reduce table size

Entry	Destination	Port
1	0.0.0.0 – 127.255.255.255	1
2	128.0.0.1 – 128.255.255.255	2
⋮	⋮	⋮
50	248.0.0.0 – 255.255.255.255	12

Generic Router Architecture



IP addresses as a line



Entry	Destination	Port
1	Cambridge	1
2	Oxford	2
3	Europe	3
4	USA	4
5	Everywhere (default)	5

Longest Prefix Match (LPM)

Entry	Destination	Port
1	Cambridge	1
2	Oxford	2
3	Europe	3
4	USA	4
5	Everywhere (default)	5

} Universities
} Continents
Planet

Matching entries:
- Cambridge — Most specific
- Europe
- Everywhere

To: Cambridge	Data

Longest Prefix Match (LPM)

Entry	Destination	Port
1	Cambridge	1
2	Oxford	2
3	Europe	3
4	USA	4
5	Everywhere (default)	5

} Universities
} Continents
Planet

Matching entries:
- Europe — Most specific
- Everywhere

To: France	Data

Implementing Longest Prefix Match

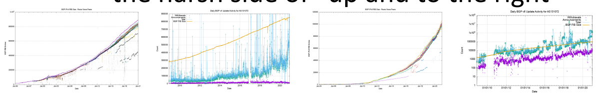
Entry	Destination	Port	
1	Cambridge	1	Searching
2	Oxford	2	
3	Europe	3	
4	USA	4	FOUND
5	Everywhere (default)	5	

Most specific
↓
Least specific

Forwarding table realities

- High Speed: Must be "packet-rate" lookup
  - about 200M lookups / second for 100Gbps
- Large (messy) tables – (BGP Jan 2021 stats)
  - 866,000+ routing prefix entries for IPv4
  - 104,000+ routing prefix entries for IPv6
- Changing and Growing

the harsh side of "up and to the right"

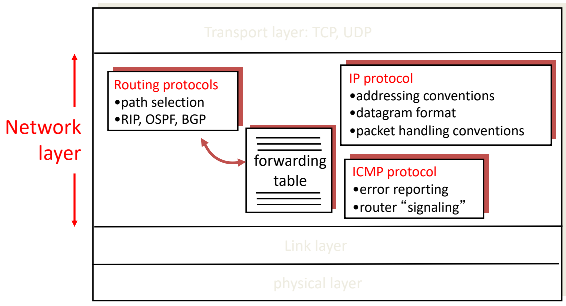


Open problems : continual growth is continual demand for innovation opportunities in control, algorithms, & network hardware

Hudson 2020 report https://blog.apnic.net/2021/01/05/bgp-in-2020-the-bgp-table/

The Internet version of a Network layer

Host, router network layer functions:



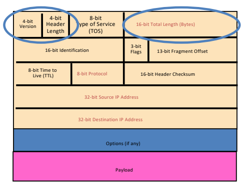
IPv4 Packet Structure
20 Bytes of Standard Header, then Options

<table>
<tr><td>4-bit Version</td><td>4-bit Header Length</td><td>8-bit Type of Service (TOS)</td><td colspan="2">16-bit Total Length (Bytes)</td></tr>
<tr><td colspan="3">16-bit Identification</td><td>3-bit Flags</td><td>13-bit Fragment Offset</td></tr>
<tr><td colspan="2">8-bit Time to Live (TTL)</td><td>8-bit Protocol</td><td colspan="2">16-bit Header Checksum</td></tr>
<tr><td colspan="5">32-bit Source IP Address</td></tr>
<tr><td colspan="5">32-bit Destination IP Address</td></tr>
<tr><td colspan="5">Options (if any)</td></tr>
<tr><td colspan="5">Payload</td></tr>
</table>

(Packet) Network Tasks One-by-One

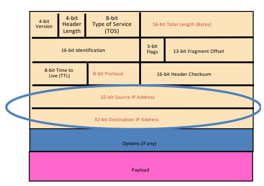
- Read packet correctly
- Get packet to the destination
- Get responses to the packet back to source
- Carry data
- Tell host what to do with packet once arrived
- Specify any special network handling of the packet
- Deal with problems that arise along the path

Reading Packet Correctly



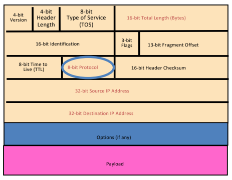
- Version number (4 bits)
  – Indicates the version of the IP protocol
  – Necessary to know what other fields to expect
  – Typically "4" (for IPv4), and sometimes "6" (for IPv6)
- Header length (4 bits)
  – Number of 32-bit words in the header
  – Typically "5" (for a 20-byte IPv4 header)
  – Can be more when IP options are used
- Total length (16 bits)
  – Number of bytes in the packet
  – Maximum size is 65,535 bytes (2^{16} -1)
  – … though underlying links may impose smaller limits

Getting Packet to Destination and Back



- Two IP addresses
  – Source IP address (32 bits)
  – Destination IP address (32 bits)
- Destination address
  – Unique identifier/locator for the receiving host
  – Allows each node to make forwarding decisions
- Source address
  – Unique identifier/locator for the sending host
  – Recipient can decide whether to accept packet
  – Enables recipient to send a reply back to source

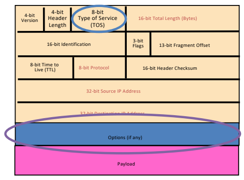
Telling Destination Host How to Handle Packet



- Protocol (8 bits)
  - Identifies the higher-level protocol
  - Important for demultiplexing at receiving host
- Most common examples
  - E.g., "6" for the Transmission Control Protocol (TCP)
  - E.g., "17" for the User Datagram Protocol (UDP)

protocol=6	protocol=17
IP header	IP header
TCP header	UDP header

Special Handling

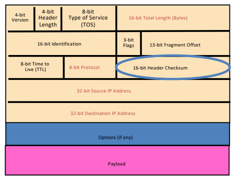


- Type-of-Service (8 bits)
  - Allow packets to be treated differently based on needs
  - E.g., low delay for audio, high bandwidth for bulk transfer
  - Has been redefined several times
  - Rarely actually used and never consistently.....
- Options

Potential Problems IPv4 solves

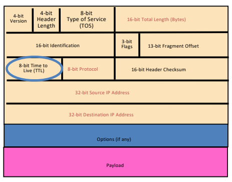
- Header Corrupted: **Checksum**
- Loop: **TTL**
- Packet too large: **Fragmentation**

Header Corruption

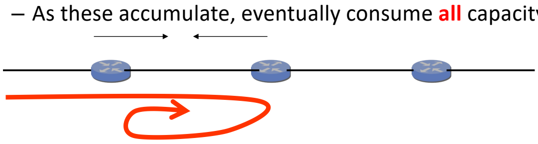


- Checksum (16 bits)
  - Particular form of checksum over packet header
- If not correct, router discards packets
  - So it doesn't act on bogus information
- Checksum recalculated at every router
  - **Why?**
  - **Why include TTL?**
  - **Why only header?**

Preventing Loops
(aka Internet Zombie plan)

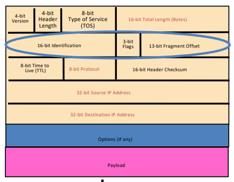


- Forwarding loops cause packets to cycle forever
  - As these accumulate, eventually consume **all** capacity



- 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
    - Using "ICMP" control message; basis for **traceroute**

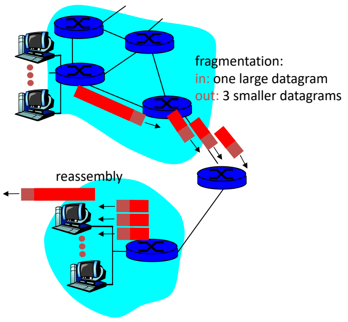
Fragmentation
(some assembly required)



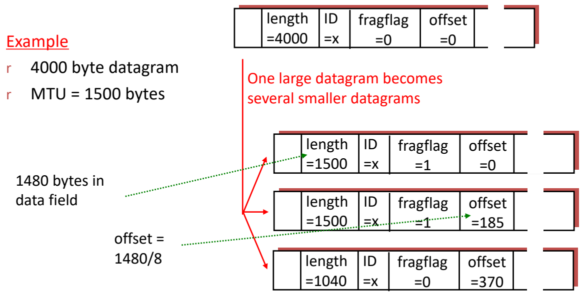
- Fragmentation: when forwarding a packet, an Internet router can split it into multiple pieces ("fragments") if too big for next hop link
- Must reassemble to recover original packet
  - Need fragmentation information (32 bits)
  - Packet identifier, flags, and fragment offset

IP Fragmentation & Reassembly

- 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
- IPv6 does things differently…

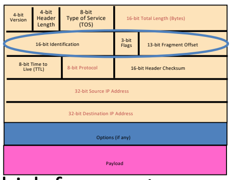


IP Fragmentation and Reassembly



Question: What happens when a fragment is lost?

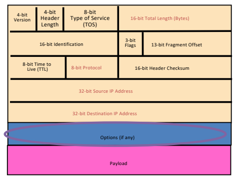
Fragmentation Details



- Identifier (16 bits): used to tell which fragments belong together
- Flags (3 bits):
  - Reserved **(RF):** unused bit
  - Don't Fragment **(DF):** instruct routers to **not** fragment the packet even if it won't fit
    - Instead, they **drop** the packet and send back a "Too Large" ICMP control message
    - Forms the basis for "Path MTU Discovery"
  - More (**MF**): this fragment is not the last one
- Offset (13 bits): what part of datagram this fragment covers in 8-byte units

Pop quiz question: Why do frags use offset and not a frag number?

Options

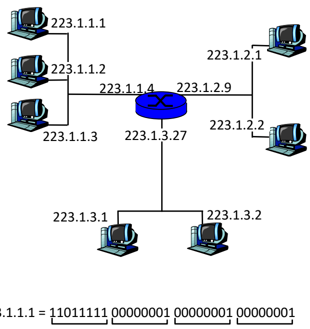


- End of Options List
- No Operation (padding between options)
- Record Route
- Strict Source Route
- Loose Source Route
- Timestamp
- Traceroute
- Router Alert
- …..

Few are used as each requires special handling in an IP router.

IP Addressing: introduction

- IP address: 32-bit identifier for host, router *interface*
- *interface:* connection between host/router and physical link
  - routers typically have multiple interfaces
  - host typically has one interface
  - IP addresses associated with each interface



223.1.1.1 = 11011111 00000001 00000001 00000001

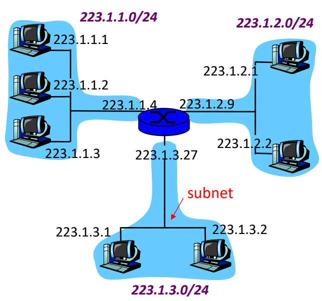
223 1 1 1

Subnets

- IP address:
  - subnet part (high order bits)
  - host part (low order bits)
- *What's a subnet ?*
  - device interfaces with same subnet part of IP address
  - can physically reach each other without intervening router

223.1.3.0/24

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



Subnet mask: /24

network consisting of 3 subnets

IP addresses: how to get one?

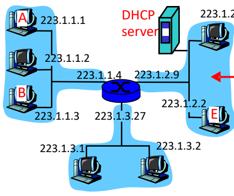
Q: How does a *host* get IP address?

- hard-coded by system admin in a file
  - Windows: control-panel->network->configuration->tcp/ip->properties
  - UNIX: /etc/rc.config (circa 1980's your mileage will vary)
- DHCP: Dynamic Host Configuration Protocol: dynamically get address from as server
  - "plug-and-play"

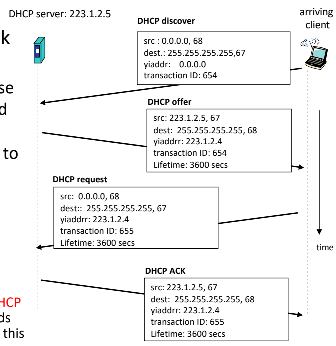
DHCP client-server scenario

Goal: allow host to *dynamically* obtain its IP address from network server when it joins network

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



arriving DHCP client needs address in this network



IP addresses: how to get one?

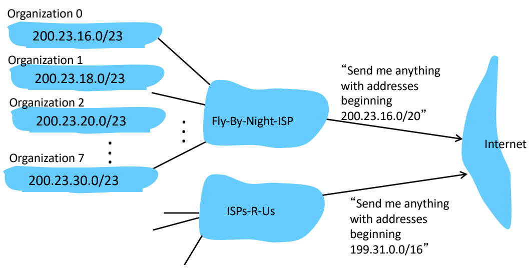
Q: How does *network* get subnet part of IP addr?

A: gets allocated portion of its provider ISP's address space

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

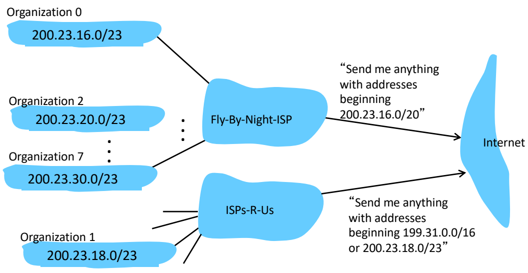
Hierarchical addressing: route aggregation

Hierarchical addressing allows efficient advertisement of routing information:



Hierarchical addressing: more specific routes

ISPs-R-Us has a more specific route to Organization 1



IP addressing: the last word...

Q: How does an ISP get a block of addresses?

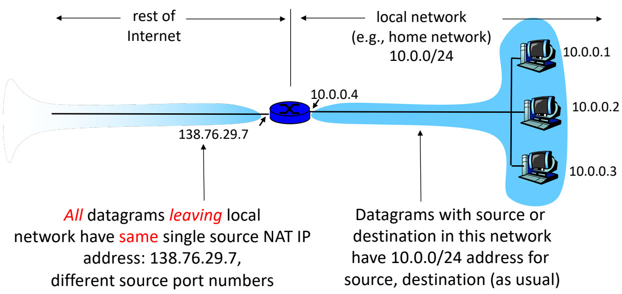
A: ICANN: Internet Corporation for Assigned Names and Numbers

- allocates addresses
- manages DNS
- assigns domain names, resolves disputes

There are regional subordinates but the (US location) of the ICANN dominates proceedings.....

Cant get more IPv4 addresses? well there is always…..

NAT: Network Address Translation



NAT: Network Address Translation

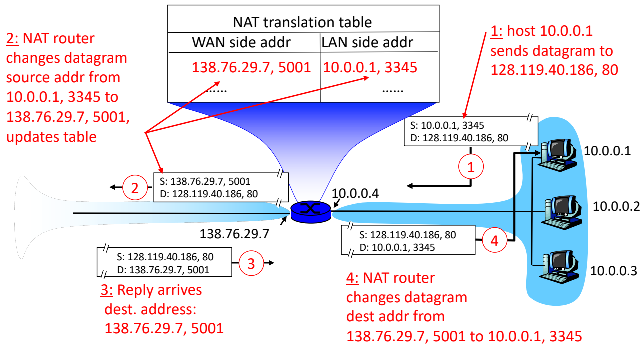
- Motivation: local network uses just one IP address as far as outside world is concerned:
  - range of addresses not needed from ISP: just one IP address for all devices
  - can change addresses of devices in local network without notifying outside world
  - can change ISP without changing addresses of devices in local network
  - devices inside local net not explicitly addressable, visible by outside world (a security plus).

NAT: Network Address Translation

Implementation: NAT router must:

- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
  - . . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr.
- *remember (in NAT translation table)* every (source IP address, port #) to (NAT IP address, new port #) translation pair
- *incoming datagrams: replace* (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

NAT: Network Address Translation

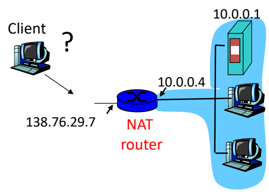


NAT: Network Address Translation

- 16-bit port-number field:
  - 60,000+ simultaneous connections with a single WAN-side address!
- NAT is controversial:
  - routers should only process up to layer 3
  - violates end-to-end argument (?)
    - NAT possibility must be taken into account by app designers, eg, P2P applications
  - address shortage *"should"* instead be solved by IPv6

NAT traversal problem

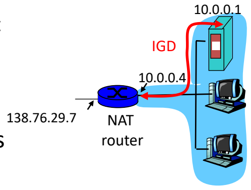
- client wants to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can't use it as destination addr)
  - only one externally visible NATted address: 138.76.29.7
- solution 1: statically configure NAT to forward incoming connection requests at given port to server
  - e.g., (138.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000



NAT traversal problem

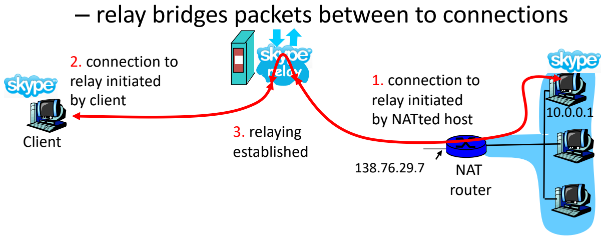
- solution 2: Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATted host to:
  - ❖learn public IP address (138.76.29.7)
  - ❖add/remove port mappings (with lease times)

i.e., automate static NAT port map configuration



NAT traversal problem

- solution 3: relaying (was used in (really old) Skype)
  - NATed client establishes connection to relay
  - External client connects to relay
  - relay bridges packets between to connections



Remember this? Traceroute at work…

traceroute: rio.cl.cam.ac.uk to munnari.oz.au

(tracepath on windows is similar)

Three delay measurements from rio.cl.cam.ac.uk to gatwick.net.cl.cam.ac.uk

trans-continent link

```
traceroute munnari.oz.au
traceroute to munnari.oz.au (202.29.151.3), 30 hops max, 60 byte packets
 1  gatwick.net.cl.cam.ac.uk (128.232.32.2)  0.416 ms  0.384 ms  0.427 ms
 2  cl-sby.route-nwest.net.cam.ac.uk (193.60.89.9)  0.393 ms  0.440 ms  0.494 ms
 3  route-nwest.route-mill.net.cam.ac.uk (192.84.5.137)  0.407 ms  0.448 ms  0.501 ms
 4  route-mill.route-enet.net.cam.ac.uk (192.84.5.94)  1.006 ms  1.091 ms  1.163 ms
 5  xe-11-3-0.camb-rbr1.eastern.ja.net (146.97.130.1)  0.300 ms  0.313 ms  0.350 ms
 6  ae24.lowdss-sbr1.ja.net (146.97.37.185)  2.679 ms  2.664 ms  2.712 ms
 7  ae28.londhx-sbr1.ja.net (146.97.33.17)  5.955 ms  5.953 ms  5.901 ms
 8  janet.mx1.lon.uk.geant.net (62.40.124.197)  6.059 ms  6.066 ms  6.052 ms
 9  ae0.mx1.par.fr.geant.net (62.40.98.77)  11.742 ms  11.779 ms  11.724 ms
10  ae1.mx1.mad.es.geant.net (62.40.98.64)  27.751 ms  27.734 ms  27.704 ms
11  mb-so-02-v4.bb.tein3.net (202.179.249.117)  138.296 ms  138.314 ms  138.282 ms
12  sg-so-04-v4.bb.tein3.net (202.179.249.53)  196.303 ms  196.293 ms  196.264 ms
13  th-pr-v4.bb.tein3.net (202.179.249.66)  225.153 ms  225.178 ms  225.196 ms
14  pyt-thairen-to-02-bdr-pyt.uni.net.th (202.29.12.10)  225.163 ms  223.343 ms  223.363 ms
15  202.28.227.126 (202.28.227.126)  241.038 ms  240.941 ms  240.834 ms
16  202.28.221.46 (202.28.221.46)  287.252 ms  287.306 ms  287.282 ms
17  * * *
18  * * *
19  * * *
20  coe-gw.psu.ac.th (202.29.149.70)  241.681 ms  241.715 ms  241.680 ms
21  munnari.OZ.AU (202.29.151.3)  241.610 ms  241.636 ms  241.537 ms
```

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

Traceroute and ICMP

- Source sends series of UDP segments to dest
  - First has TTL =1
  - Second has TTL=2, etc.
  - Unlikely port number
- When nth datagram arrives to nth router:
  - Router discards datagram
  - And sends to source an ICMP message (type 11, code 0)
  - Message includes name of router& IP address
- When ICMP message arrives, source calculates RTT
- Traceroute does this 3 times

Stopping criterion

- UDP segment eventually arrives at destination host
- Destination returns ICMP "host unreachable" packet (type 3, code 3)
- When source gets this ICMP, stops.

ICMP: Internet Control Message Protocol

- used by hosts & routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer "above" IP:
  - ICMP msgs carried in IP datagrams
- ICMP message: type, code plus first 8 bytes of IP datagram causing error

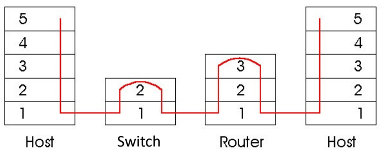
Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

Gluing it together:

How does my Network (address) interact with my Data-Link (address) ?

Switches vs. Routers Summary

- both (can be implemented as) store-and-forward devices
  - routers: network layer devices (**manipulate** network layer headers eg IP)
  - switches are link layer devices (**examine** Data-Link-Layer headers eg Ethernet)
- Routers: implement routing algorithms, maintain routing tables of the network – create network forwarding tables from routing tables
- Switches: implement learning algorithms, learn switch/DLL forwarding tables

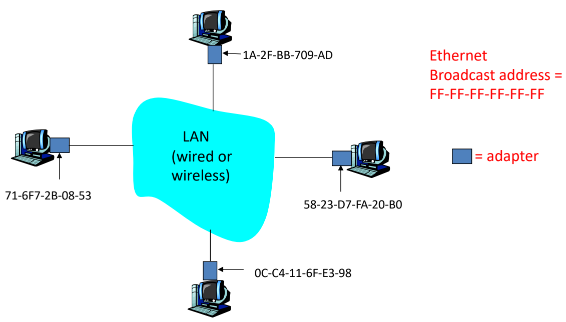


MAC Addresses (and IPv4 ARP)
or How do I glue my network to my data-link?

- 32-bit IP address:
  - *network-layer* address
  - used to get datagram to destination IP subnet
- MAC (or LAN or physical or Ethernet) address:
  - function: *get frame from one interface to another physically-connected interface (same network)*
  - 48 bit MAC address (for most LANs)
    - burned in NIC ROM, firmware, etc.

LAN Addresses and ARP

Each adapter on LAN has unique LAN (MAC) address

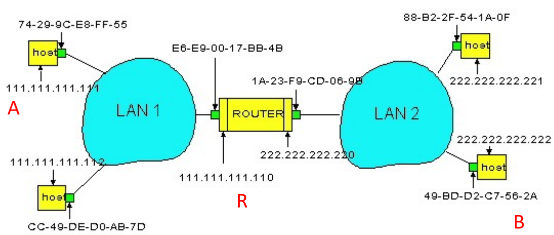


Address Resolution Protocol

- Every node maintains an ARP table
  - <IP address, MAC address> pair
- Consult the table when sending a packet
  - Map destination IP address to destination MAC address
  - Encapsulate and transmit the data packet
- But: what if IP address not in the table?
  - Sender broadcasts: "**Who has IP address 1.2.3.156**?"
  - Receiver responds: "**MAC address 58-23-D7-FA-20-B0**"
  - Sender caches result in its ARP table

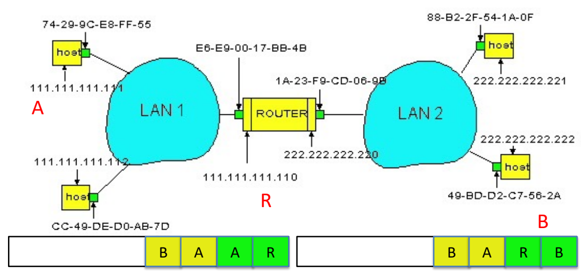
Example: A Sending a Packet to B

How does host A send an IP packet to host B?



Example: A Sending a Packet to B

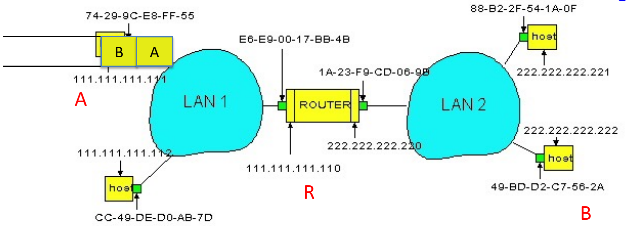
How does host A send an IP packet to host B?



1. **A sends packet to R.**
2. **R sends packet to B.**

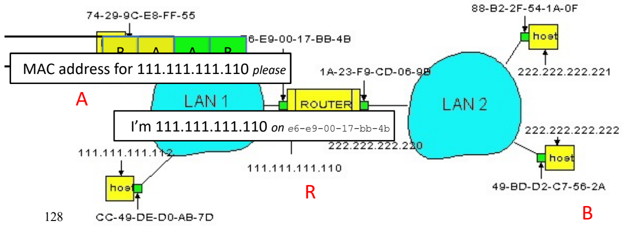
Host A Decides to Send Through R

- Host A constructs an IP packet to send to B
  - Source 111.111.111.111, destination 222.222.222.222
- Host A has a gateway router R
  - Used to reach destinations outside of 111.111.111.0/24
  - Address 111.111.111.110 for R learned via DHCP/config



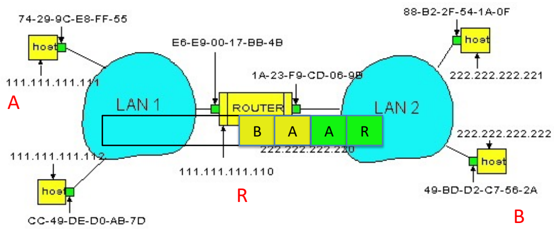
Host A Sends Packet Through R

- Host A learns the MAC address of R's interface
  - ARP request: broadcast request for 111.111.111.110
  - ARP response: R responds with E6-E9-00-17-BB-4B
- Host A encapsulates the packet and sends to R



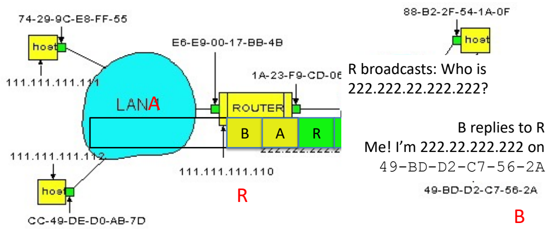
R Decides how to Forward Packet

- Router R's adaptor receives the packet
  - R extracts the IP packet from the Ethernet frame
  - R sees the IP packet is destined to 222.222.222.222
- Router R consults its forwarding table
  - Packet matches 222.222.222.0/24 via other adaptor



R Sends Packet to B

- Router R's learns the MAC address of host B
  - ARP request: broadcast request for 222.222.222.222
  - ARP response: B responds with 49-BD-D2-C7-52A
- Router R encapsulates the packet and sends to B



Security Analysis of ARP

- Impersonation
  - Any node that hears request can answer …
  - … and can say whatever they want
- Actual legit receiver never sees a problem
  - Because even though later packets carry its IP address, its NIC doesn't capture them since the (naughty) packets are not its MAC address

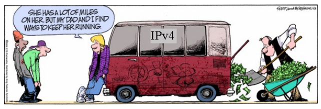
Key Ideas in Both ARP and DHCP

- Broadcasting: Can use broadcast to make contact
  - Scalable because of limited size
- Caching: remember the past for a while
  - Store the information you learn to reduce overhead
  - Remember your own address & other host's addresses
- Soft state: eventually forget the past
  - Associate a time-to-live field with the information
  - … and either refresh or discard the information
  - Key for robustness in the face of unpredictable change

Why Not Use DNS-Like Tables?

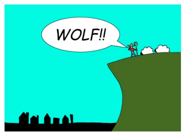
- When host arrives:
  - Assign it an IP address that will last as long it is present
  - Add an entry into a table in DNS-server that maps MAC to IP addresses
- Answer:
  - Names: explicit creation, and are plentiful
  - Hosts: come and go without informing network
    - Must do mapping on demand
  - Addresses: not plentiful, need to reuse and remap
    - Soft-state enables dynamic reuse

IPv6



prematurely

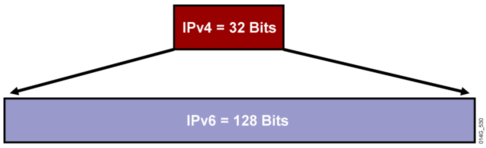
- Motivated by address exhaustion
  - addresses are larger
  - packet headers are laid out differently
  - address management and configuration are completely different
  - some DNS behavior changes
  - some sockets code changes
  - *everybody now has a hard time parsing IP addresses*
- Steve Deering focused on simplifying IP
  - Got rid of all fields that were not absolutely necessary
  - "Spring Cleaning" for IP
- Result is an elegant, if unambitious, protocol



IPv4	IPv6
Addresses are 32 bits (4 bytes) in length.	Addresses are 128 bits (16 bytes) in length
Address (A) resource records in DNS to map host names to IPv4 addresses.	Address (AAAA) resource records in DNS to map host names to IPv6 addresses.
Pointer (PTR) resource records in the IN-ADDR.ARPA DNS domain to map IPv4 addresses to host names.	Pointer (PTR) resource records in the IP6.ARPA DNS domain to map IPv6 addresses to host names.
IPSec is optional and should be supported externally	IPSec support is not optional
Header does not identify packet flow for QoS handling by routers	Header contains Flow Label field, which Identifies packet flow for QoS handling by router.
Both routers and the sending host fragment packets.	Routers do not support packet fragmentation. Sending host fragments packets
Header includes a checksum.	Header does not include a checksum.
Header includes options.	Optional data is supported as extension headers.
ARP uses broadcast ARP request to resolve IP to MAC/Hardware address.	Multicast Neighbor Solicitation messages resolve IP addresses to MAC addresses.
Internet Group Management Protocol (IGMP) manages membership in local subnet groups.	Multicast Listener Discovery (MLD) messages manage membership in local subnet groups.
Broadcast addresses are used to send traffic to all nodes on a subnet.	IPv6 uses a link-local scope all-nodes multicast address.
Configured either manually or through DHCP.	Does not require manual configuration or DHCP.
Must support a 576-byte packet size (possibly fragmented).	Must support a 1280-byte packet size (without fragmentation).

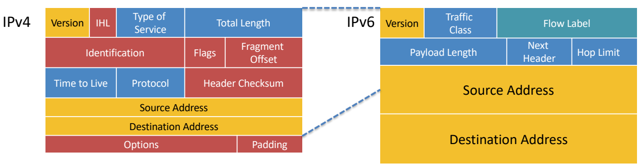
Larger Address Space

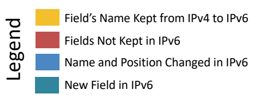
- IPv4 = 4,294,967,295 addresses
- IPv6 = 340,282,366,920,938,463,374,607,432,768,211,456 addresses
- 4x in number of bits translates to **huge** increase in address space!



Other Significant Protocol Changes - 1

- Increased minimum MTU from 576 to 1280
- No enroute fragmentation… fragmentation only at source
- Header changes (20bytes to 40bytes)
- Replace broadcast with multicast





Ok, so the default header got a little bit bigger…

Other Significant Protocol Changes - 2

operation is intended to be simpler within the network:

- no *in-network* fragmentation
- no checksums in IPv6 header
- UDP checksum required (wasn't in IPv4) rfc6936: **No more zero**
- optional state carried in *extension headers*
  - Extension headers notionally replace IP options
  - Each extension header indicates the type of the *following* header, so they can be chained
  - The final 'next header' either indicates there is no 'next', or escapes into a transport-layer header (e.g., TCP)

IPv6 Basic Address Structure

IPv6 addresses are split into two primary parts:

0	32	64	96	128
Routing Prefix		Interface Identifier		

- 64 bits is dedicated to an addressable interface (equivalent to the host, if it only has one interface)
- The network prefix allocated to a network by a registry can be up to 64-bits long
- An allocation of a /64 (i.e. a 64-bit network prefix) allows *one* subnet (it cannot be subdivided)
- A /63 allows two subnets; a /62 offers four, etc. /48s are common for older allocations (RFC 3177, obsoleted by RFC 6177).
- Longest-prefix matching operates as in IPv4.

IPv6 Address Representation (quick)

IPv6 addresses represented as eight 16-bit blocks (4 hex chars) separated by colons:

- `2001:4998:000c:0a06:0000:0000:0002:4011`

But we can condense the representation by removing leading zeros in each block:

- `2001:4998:c:a06:0:0:2:4011`

And by reducing the consecutive block of zeros to a "`::`"
(this double colon rule can only be applied once)

- `2001:4998:c:a06::2:4011`

IPv6 Address Families

The address space is carved, like v4, into certain categories [1]:

host-local : localhost; `::1` is equivalent to `127.0.0.1`
link-local : not routed: `fe80::/10` is equivalent to `169.254.0.0/16`
site-local : not routed *globally*: `fc00::/7` is equivalent to `192.168.0.0/16` or `10.0.0.0/8`
global unicast : `2000::/3` is basically any v4 address not reserved in some other way
multicast : `ff00::/8` is equivalent to `224.0.0.0/4`

[1] http://www.ripe.net/lir-services/new-lir/ipv6_reference_card.pdf

Problem with /64 Subnets

- Scanning a subnet becomes a DoS attack!
  - Creates IPv6 version of 2^{64} ARP entries in routers
  - Exhaust address-translation table space
- So now we have:

`ping6 ff02::1` All nodes in broadcast domain
`ping6 ff02::2` All routers in broadcast domain

- Solutions
  - RFC 6164 recommends use of /127 to protect router-router links
  - RFC 3756 suggest "clever cache management" to address more generally

Neighbour Discovery

- The Neighbour Discovery Protocol[2] specifies a set of ICMPv6 message types that allow hosts to discover other hosts or routing hardware on the network
  - neighbour solicitation
  - neighbour advertisement
  - router solicitation
  - router advertisement
  - redirect
- In short, a host can *solicit* neighbour (host) state to determine the layer-2 address of a host *or* to check whether an address is in use
- or it can solicit router state to learn more about the network configuration
- In both cases, the solicit message is sent to a well-known multicast address

[2] http://tools.ietf.org/html/rfc4861

IPv6 Dynamic Address Assignment

We have the two halves of the IPv6 address: the network component and the host component. Those are derived in different ways.

Network (top 64 bits):

- Router Advertisements (RAs)

Interface

Identifier (bottom 64 bits):

- Stateless, automatic: SLAAC
- Stateful, automatic: DHCPv6

SLAAC: overview

SLAAC is:

- ... intended to make network configuration easy without manual configuration *or even a DHCP server*
- ... an algorithm for hosts to automatically configure their network interfaces (set up addresses, learn routes) without intervention

SLAAC: overview

- When a host goes live or an interface comes up, the system wants to know more about its environment
- It *can* configure link-local addresses for its interfaces: it uses the interface identifier, the EUI-64
- It uses this to ask (solicit) router advertisements sooner than the next periodic announcements; ask the network for information

SLAAC: overview

The algorithm (assuming one interface):

1. Generate potential link-local address
2. Ask the network (multicast[4]) if that address is in use: *neighbour solicitation*
3. Assuming no responses, assign to interface

[4]https://tools.ietf.org/html/rfc2373

The EUI-64 Interface Identifier

- IEEE 64-bit Extended Unique Identifier (EUI-64)[3]
- There are various techniques to derive a 64-bit value, but often times we derive from the 48-bit MAC address

Company ID Vendor ID

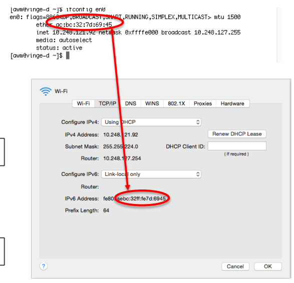
c8	e0	eb	18	75	87

↓

c8	e0	eb	ff	fe	18	75	87

the seventh bit from the left, or the universal/local (U/L) bit, needs to be inverted, 0 = local admin 1 = universal admin

ca	e0	eb	ff	fe	18	75	87



[3]http://tools.ietf.org/html/rfc2373

SLAAC: overview; Router Solicitation

Then,

- Once the host has a unique *link-local* address, it can send packets to anything else sharing that link substrate
  ... but the host doesn't yet know any routers, or public routes
  ... bootstrap: routers listen to a well-known multicast address

4. host asks the network (multicast) for router information: *router solicitation*

5. responses from the routers are sent directly (unicast) to the host that sent the router solicitation

6. the responses *may* indicate that the host should do more (e.g., use DHCP to get DNS information)

Router Advertisement

Without solicitation, regular router advertisements are generated by routing hardware.

Router Advertisements:

- nodes that forward traffic periodically advertise themselves to the network
- periodicity and expiry of the advertisement are configurable

Router Advertisement (RA), among other things, tells a host where to derive its network state with two flags: M(anaged) and O(ther info):

- M: "Managed Address Configuration", which means: use DHCPv6 to find your host address (and ignore option O)
- O: Other information is available via DHCPv6, such as DNS configuration

Uh-oh

What problem(s) arises from totally decentralised address configuration?

Concerns that arise from using an EUI-64:
- Privacy: SLAAC interface identifiers don't change over time, so a host can be identified across networks
- Security: embedding a MAC address into an IPv6 address will carry that vendor's ID(s)[5], a possible threat vector

[5]http://standards.ieee.org/develop/regauth/oui/public.html

Address Configuration: SLAAC Privacy Addresses

Privacy extensions for SLAAC[6]
- temporary addresses for initiating outgoing sessions
- generate one temporary address per prefix
- when they expire, they are not used for new sessions, but can continue to be used for existing sessions
- the addresses should appear random, such that they are difficult to predict
- lifetime is configurable; this OSX machine sets an 86,400s timer (1 day)

[6]https://tools.ietf.org/html/rfc4941

Address Configuration: SLAAC Privacy Addresses

The algorithm:
- Assume: a stored 64-bit input value from previous iterations, or a pseudo-randomly generated value

1. take that input value and append it to the EUI-64
2. compute the MD5 message digest of that value
3. set bit 6 to zero
4. compare the leftmost 64-bits against a list of reserved interface identifiers and those already assigned to an address on the local device. If the value is unacceptable, re-run using the rightmost 64 bits of the result instead of the historic input value in step 1
5. use the leftmost 64-bits as the randomised interface identifier
6. store the rightmost 64-bits as the history value to be used in the next iteration of the algorithm

IPv6: why has the transition taken so long?

IPv4 and IPv6 are not compatible:
- different packet formats
- different addressing schemes
- no flag days

as the Internet has grown bigger and accumulated many IPv4-only services, transition has proven ... Tricky

Incentive issues

e.g. Virgin Media policy in 2010

....When IPV6 is rolled out across the whole of the Internet then a lot of the ISP's will roll out IPV6, ...

Virgin Media are only now (late 2022) *"committing"* to IPv6

IPv6: why has the transition taken so long?

- IPv4 has/had the momentum
  - ... which led to CIDR
  - ... and encouraged RFC1918 space and NAT
- IPv4 NAT was covered earlier in this topic (reminder)
  - your ISP hands you only one IPv4 address
  - you share that across multiple devices in your household
  - The NAT handles all the translation between internal ("private") and external ("public") space

Transition tech: outline

- Tunnelling
- dual-stacked services, and happy eyeballs
- DNS64 and NAT64[8]
- 464XLAT
- DNS behaviour

[8]https://tools.ietf.org/html/rfc6146

Transition tech: outline

- Tunnelling

HURRICANE ELECTRIC
INTERNET SERVICES

Hurricane Electric Free IPv6 Tunnel Broker

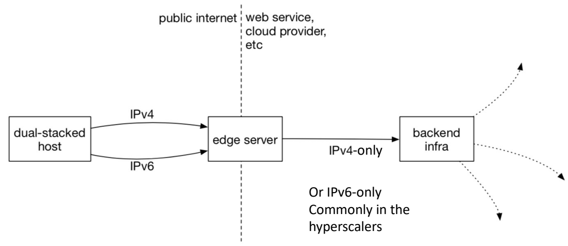
IPv6 Tunnel Broker

Think of it as an IPv6 VPN service; which is essentially what it is

[8] https://tools.ietf.org/html/rfc6146

Dual-Stack Services: Common Deployment

It's common for web services to play conservatively: dual-stack your edge services (e.g., load balancers), leaving some legacy infrastructure for later:



Dual-Stack Services: Common Deployment

Aim is to reduce the pain:

- You can dual-stack the edge hosts, and carry state in, say, HTTP headers indicating the user's IP address (common over v4 anyway)
- You can dual-stack the backend opportunistically, over a longer period of time
- You use DNS to enable/disable the v6 side last (if there is no AAAA record in DNS, no real users will connect to the IPv6 infrastructure

IPV6 sadness and DNS

- The introduction of IPv6 carried with it an obligation that applications attempt to use IPv6 before falling back to IPv4.
- What happens though if you try to connect to a host which doesn't exist?[9]
- But the presence of IPv6 modifies the behaviour of DNS responses and response preference[10]

[9] https://tools.ietf.org/html/rfc5461
[10] https://tools.ietf.org/html/rfc3484

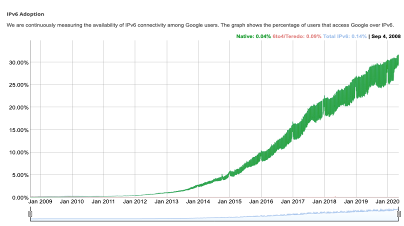
Happy Eyeballs

- Happy Eyeballs[11] was the proposed solution
  - the eyeballs in question are yours, or mine, or whoever is sitting in front of their browser getting mad that things are unresponsive
- Modifies application behaviour

[11] https://tools.ietf.org/html/rfc8305

IPv6: adoption

- Google[1]: ~ 30% of clients access services via IPv6
- NIST: 1/3 of all US government domains are IPv6 capable



[1] https://www.google.com/intl/en/ipv6/statistics.html

Improving on IPv4 and IPv6?

- Why include unverifiable source address?
  - Would like accountability ***and*** anonymity (now neither)
  - Return address can be communicated at higher layer
- Why packet header used at edge same as core?
  - Edge: host tells network what service it wants
  - Core: packet tells switch how to handle it
    - One is local to host, one is global to network
- Some kind of payment/responsibility field?
  - Who is responsible for paying for packet delivery?
  - Source, destination, other?
- Other ideas?

Summary Network Layer

- understand principles behind network layer services:
  - network layer service models
  - forwarding versus routing (versus switching)
  - how a switch & router works
  - routing (path selection)
  - IPv6
- Algorithms
  - Two routing approaches (LS vs DV)
  - One of these in detail (LS)
  - ARP
- Other Core ideas
  - Caching, soft-state, broadcast
  - Fate-sharing in practice….

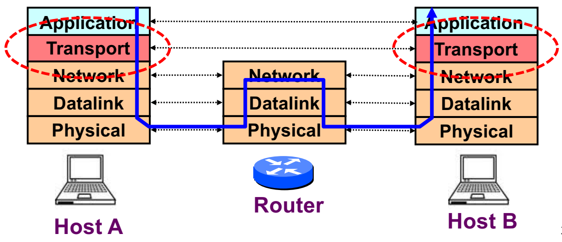
Topic 5 – Transport

Our goals:

- understand principles behind transport layer services:
  - multiplexing/demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
  - buffers
- learn about transport layer protocols in the Internet:
  - UDP: connectionless transport
  - TCP: connection-oriented transport
  - TCP congestion control
  - TCP flow control

Transport Layer

- Commonly a layer at end-hosts, between the application and network layer

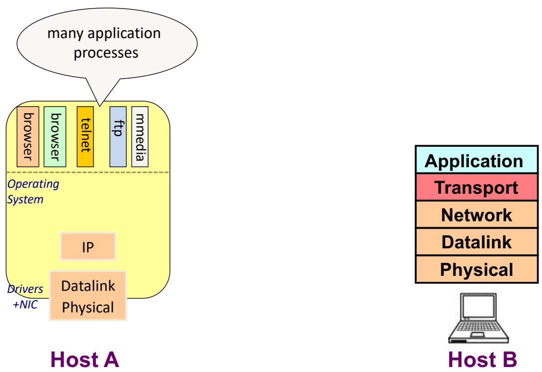


Why a transport layer?

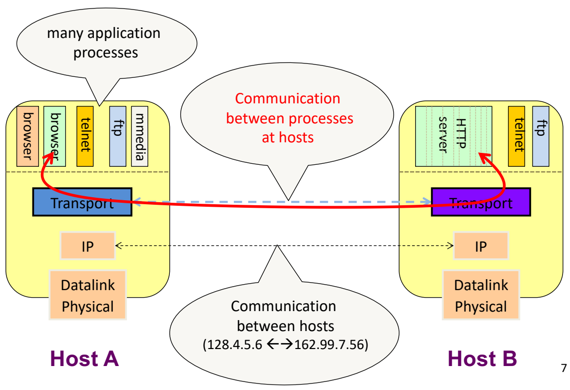
- IP packets are addressed to a host but end-to-end communication is between application/ processes/tasks at hosts
  - Need a way to decide which packets go to which applications (*more multiplexing*)

Why a transport layer?

Why a transport layer?



Why a transport layer?



Why a transport layer?

- IP packets are addressed to a host but end-to-end communication is between application processes at hosts
  - Need a way to decide which packets go to which applications (mux/demux)
- IP provides a weak service model (*best-effort*)
  - Packets can be corrupted, delayed, dropped, reordered, duplicated
  - No guidance on how much traffic to send and when
  - Dealing with this is tedious for application developers

Role of the Transport Layer

- Communication between application processes
  - Multiplexing between application processes
  - Implemented using *ports*

Role of the Transport Layer

- Communication between application processes
- Provide common end-to-end services for app layer [optional]
  - Reliable, in-order data delivery
  - Paced data delivery: flow and congestion-control
    - too fast may overwhelm the network
    - too slow is not efficient

(Just Like Computer Networking Lectures….)

Role of the Transport Layer

- Communication between processes
- Provide common end-to-end services for app layer [optional]
- TCP and UDP are the common transport protocols
  - also SCTP, MTCP, SST, RDP, DCCP, …

Role of the Transport Layer

- Communication between processes
- Provide common end-to-end services for app layer [optional]
- TCP and UDP are the common transport protocols
- UDP is a minimalist, no-frills transport protocol
  - only provides mux/demux capabilities

Role of the Transport Layer

- Communication between processes
- Provide common end-to-end services for app layer [optional]
- TCP and UDP are the common transport protocols
- UDP is a minimalist, no-frills transport protocol
- TCP is the *totus porcus* protocol
  - offers apps a reliable, in-order, byte-stream abstraction
  - with congestion control
  - but **no** performance (delay, bandwidth, ...) guarantees

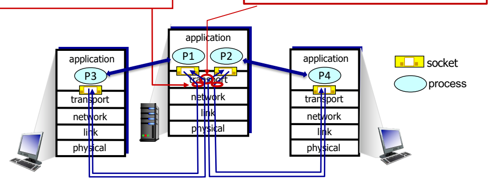
Role of the Transport Layer

- Communication between processes
  - mux/demux from and to application processes
  - implemented using ports

Multiplexing/demultiplexing

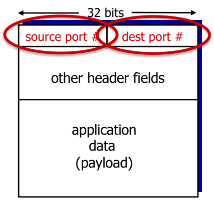
multiplexing as sender:
handle data from multiple sockets, add transport header (later used for demultiplexing)

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



How demultiplexing Works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- host uses *IP addresses & port numbers* to direct segment to appropriate socket



TCP/UDP segment format

Connectionless demultiplexing

- when creating socket, must specify *host-local* port #:

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

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

when receiving host receives *UDP* segment:
- checks destination port # in segment
- directs UDP segment to socket with that port #

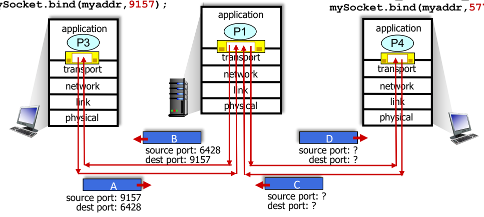
IP/UDP datagrams with *same dest. port #,* but different source IP addresses and/or source port numbers will be directed to *same socket* at receiving host

Connectionless demultiplexing: an example

```
mySocket =
  socket(AF_INET,SOCK_DGRAM)
mySocket.bind(myaddr,6428);
```

```
mySocket =
  socket(AF_INET,SOCK_STREAM)
mySocket.bind(myaddr,9157);
```

```
mySocket =
  socket(AF_INET,SOCK_STREAM)
mySocket.bind(myaddr,5775);
```



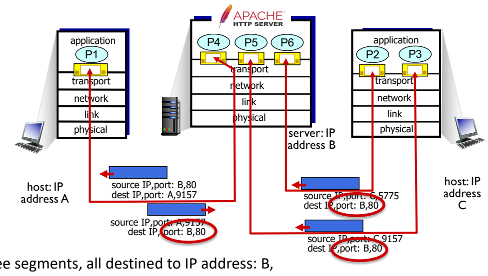
Connection-oriented demultiplexing

- 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

slight lie alert…. I should say that a common network tuple has FIVE values
- source IP address
- source port number
- dest IP address
- dest port number AND
- *protocol* e.g. TCP (6) or UDP (17)

Connection-oriented demultiplexing: example



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

Summary

- Multiplexing, demultiplexing: based on segment, datagram header field values
- **UDP:** demultiplexing using destination port number (only)
- **TCP:** demultiplexing using 4-tuple: source and destination IP addresses, and port numbers
- Multiplexing/demultiplexing can happen at *any* layer

More on Ports

- Separate 16-bit port address space for UDP and TCP
- “Well known” ports (0-1023): everyone agrees which services run on these ports
  - e.g., ssh:22, http:80, https:443
  - helps client know server’s port
- Ephemeral ports (most 1024-65535): dynamically selected: as the source port for a client process

UDP: User Datagram Protocol

- Lightweight communication between processes
  - Avoid overhead and delays of ordered, reliable delivery
- UDP described in RFC 768 – (1980!)
  - Destination IP address and port to support demultiplexing
  - Optional error checking on the packet contents
    - (checksum field of 0 means “don’t verify checksum”) ***not in IPv6!***
    - ((this idea of optional checksum is removed in IPv6))

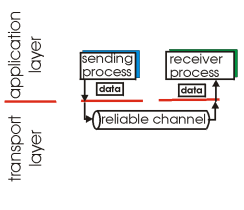
SRC port	DST port
checksum	length
DATA	

Why a transport layer?

- IP packets are addressed to a host but end-to-end communication is between application processes at hosts
  - Need a way to decide which packets go to which applications (mux/demux)
- IP provides a weak service model (*best-effort*)
  - Packets can be corrupted, delayed, dropped, reordered, duplicated

Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!



(a) provided service

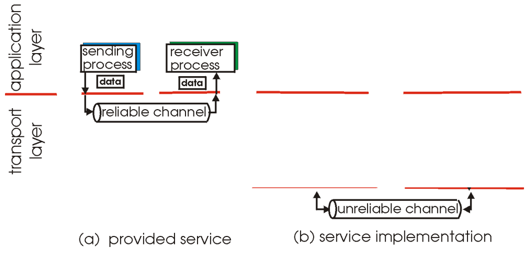
- In a perfect world, reliable transport is easy

But the Internet default is *best-effort*

- All the bad things best-effort can do
  - a packet is corrupted (bit errors)
  - a packet is lost
  - a packet is delayed (*why?*)
  - packets are reordered (*why?*)
  - a packet is duplicated (*why?*)

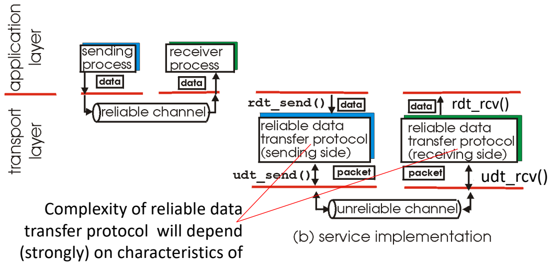
Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!



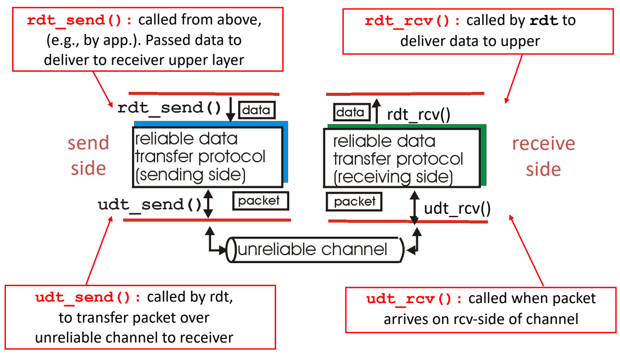
Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!



Complexity of reliable data transfer protocol will depend (strongly) on characteristics of unreliable channel (lose, corrupt, reorder data?)

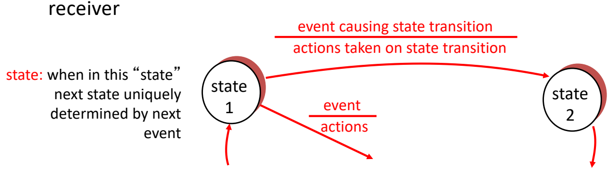
Reliable data transfer: getting started



Reliable data transfer: getting started

We' ll:

- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only unidirectional data transfer
  - but control info will flow on both directions!
- use finite state machines (FSM) to specify sender, receiver



KR state machines – a note.

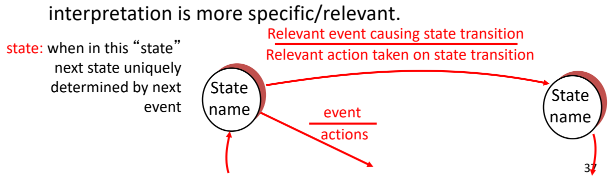
Beware

Kurose and Ross has a confusing/confused attitude to state-machines.

I've attempted to normalise the representation.

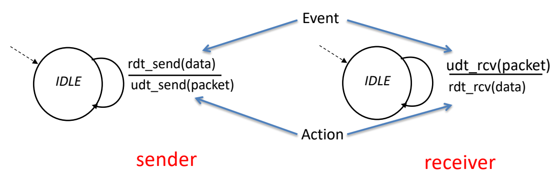
UPSHOT: these slides have differing information to the KR book (from which the RDT example is taken.)

in KR "actions taken" appear wide-ranging, my interpretation is more specific/relevant.



Rdt1.0: reliable transfer over a reliable channel

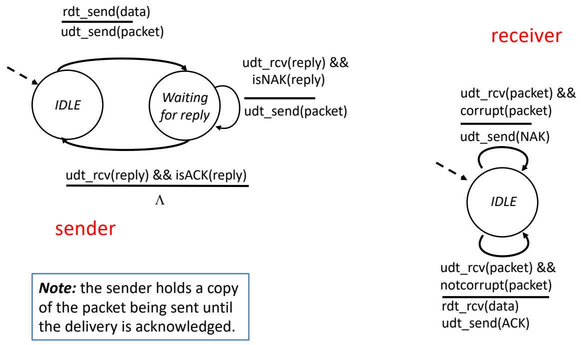
- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- separate FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver read data from underlying channel



Rdt2.0: channel with bit errors

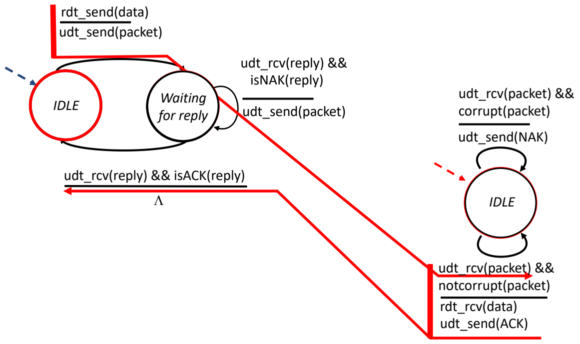
- underlying channel may flip bits in packet
  - checksum to detect bit errors
- *the* question: how to recover from errors:
  - *acknowledgements (ACKs):* receiver explicitly tells sender that packet received is OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that packet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

rdt2.0: FSM specification

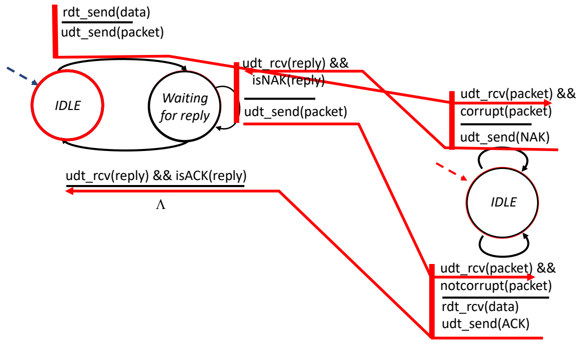


Note: the sender holds a copy of the packet being sent until the delivery is acknowledged.

rdt2.0: operation with no errors



rdt2.0: error scenario



rdt2.0 has a fatal flaw!

What happens if ACK/NAK corrupted?

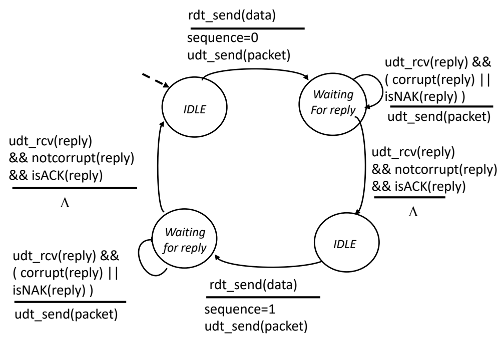
- sender doesn't know what happened at receiver!
- can't just retransmit: possible duplicate

Handling duplicates:

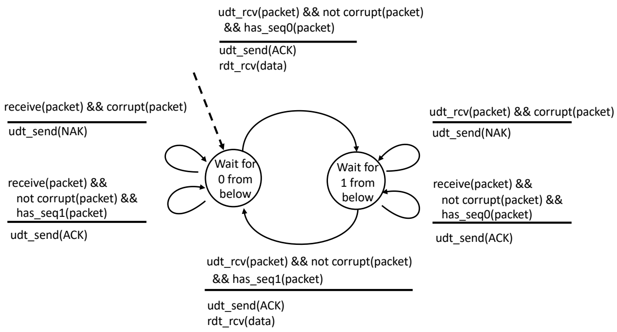
- sender retransmits current packet if ACK/NAK garbled
- sender adds *sequence number* to each packet
- receiver discards (doesn't deliver) duplicate packet

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

rdt2.1: sender, handles garbled ACK/NAKs



rdt2.1: receiver, handles garbled ACK/NAKs



rdt2.1: discussion

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 "current" pkt has a 0 or 1 sequence number

Receiver:

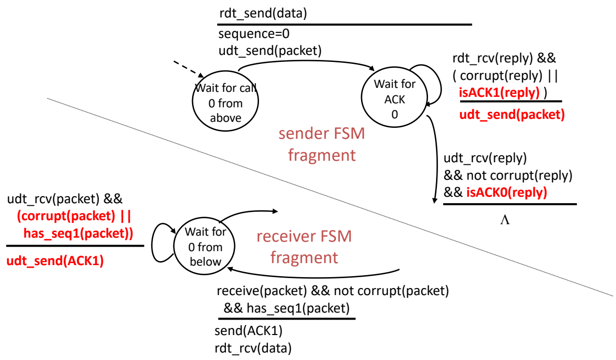
- must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can *not* know if its last ACK/NAK received OK at sender

rdt2.2: a NAK-free protocol

- same functionality as rdt2.1, using ACKs only
- instead of NAK, receiver sends ACK for last pkt received OK
  - receiver must *explicitly* include seq # of pkt being ACKed
- duplicate ACK at sender results in same action as NAK: *retransmit current pkt*

As we will see, TCP uses this approach to be NAK-free

rdt2.2: sender, receiver fragments



rdt3.0: channels with errors *and* loss

New channel assumption: underlying channel can also *lose* packets (data, ACKs)

- checksum, sequence #s, ACKs, retransmissions will be of help … but not quite enough

Q: How do *humans* handle lost sender-to-receiver words in conversation?

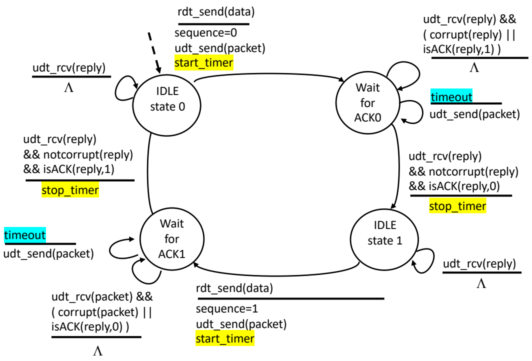
rdt3.0: channels with errors *and* loss

Approach: sender waits "reasonable" amount of time for ACK

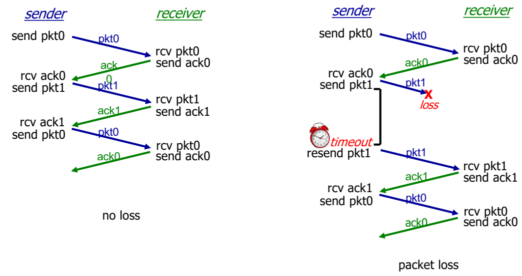
- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but seq #s already handles this!
  - receiver must specify seq # of packet being ACKed
- use countdown timer to interrupt after "reasonable" amount of time

timeout

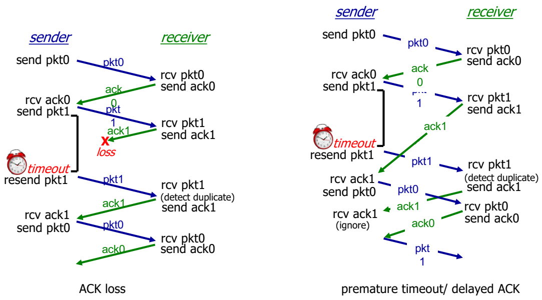
rdt3.0 sender



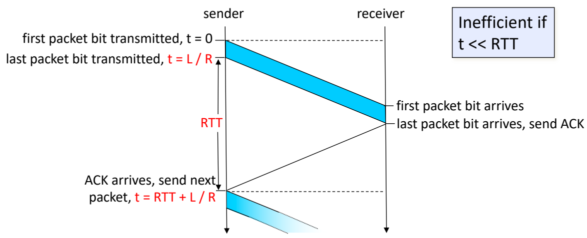
rdt3.0 in action



rdt3.0 in action



rdt3.0: stop-and-wait operation



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

Performance of rdt3.0 (stop-and-wait)

- rdt3.0 works, but performance stinks
- ex: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

$$d_{trans} = \frac{L}{R} = \frac{8000\text{bits}}{10^9\text{bps}} = 8\text{microseconds}$$

- U_{sender}: utilization – fraction of time sender busy sending

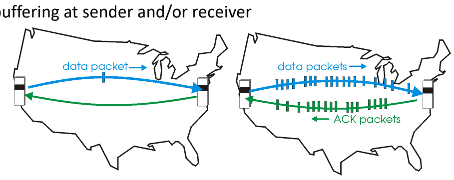
$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

- 1KB pkt every 30 msec -> 33kB/sec throughput over 1 Gbps link

- The network protocol limits use of physical resources!

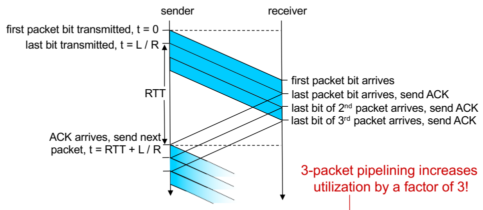
Pipelined (Packet-Window) protocols

Pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged pkts
- range of sequence numbers must be increased
- buffering at sender and/or receiver



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

Pipelining: increased utilization



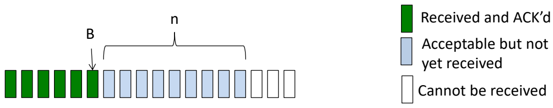
$$U_{sender} = \frac{3L/R}{RTT + L/R} = \frac{.0024}{30.008} = 0.00081$$

A Sliding Packet Window

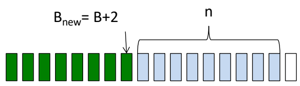
- window = set of adjacent sequence numbers
  - The size of the set is the window size; assume window size is *n*
- General idea: send up to *n* packets at a time
  - Sender can send packets in its window
  - Receiver can accept packets in its window
  - Window of acceptable packets “slides” on successful reception/acknowledgement

Acknowledgements (1)

- At receiver



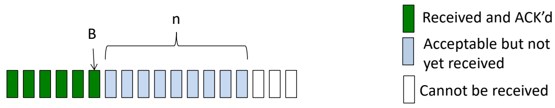
- After receiving B+1, B+2



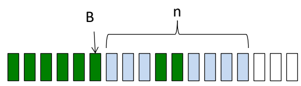
- Receiver sends ACK(B_{new}+1)

Acknowledgements (2)

- At receiver



- After receiving B+4, B+5



- Receiver sends ACK(B+???)

Oh…. how do we recover?

Acknowledgements w/ Sliding Window

Dealing with loss….

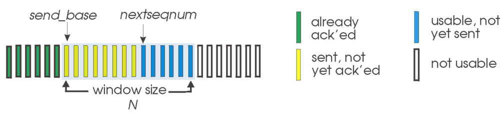
- Two common options
  - Go-Back-N (GBN)
  - Selective Repeat (SR)
    Also called Selective Acknowledgement (SACK)

Go-Back-N (GBN)

- Sender transmits up to *n* unacknowledged packets
- Receiver only accepts packets in order
  - discards out-of-order packets (i.e., packets other than *B+1*)
- Receiver uses cumulative acknowledgements
  - i.e., sequence# in ACK = next expected in-order sequence#
- Sender sets timer for 1st outstanding ack (A+1)
- If timeout, retransmit *A+1*, … , *A+n*

Go-Back-N: sender

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



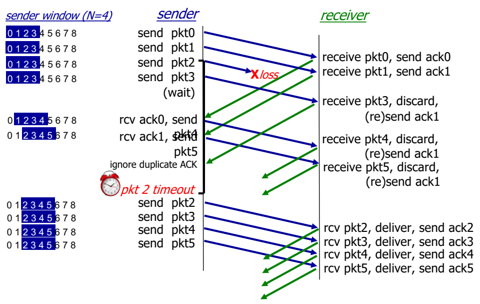
- *cumulative ACK:* ACK(*n*): ACKs all packets up to, including seq # *n*
  - on receiving ACK(*n*): move window forward to begin at *n+1*
- timer for oldest in-flight packet
- *timeout(n):* retransmit packet n and all higher seq # packets in window

Go-Back-N: receiver

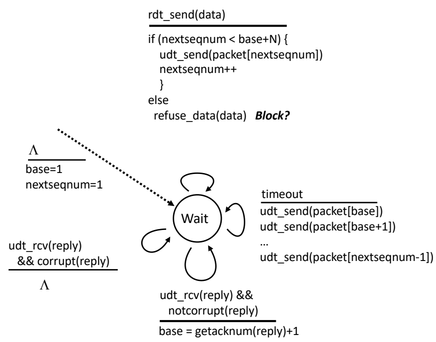
- ACK-only: always send ACK for correctly-received packet so far, with highest *in-order* seq #
  - may generate duplicate ACKs
  - need only remember `rcv_base`
- on receipt of out-of-order packet:
  - can discard (don't buffer) or buffer: an implementation decision
  - re-ACK pkt with highest in-order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

Receiver view of sequence number space:

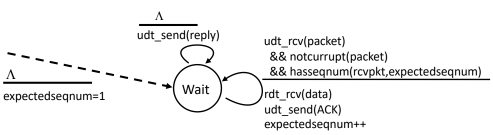
Go-Back-N in action



GBN: sender extended FSM



GBN: receiver extended FSM



ACK-only: always send an ACK for correctly-received packet with the highest *in-order* seq #

- may generate duplicate ACKs
- need only remember **`expectedseqnum`**

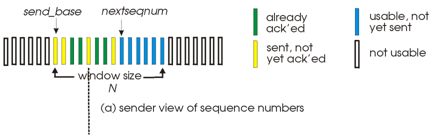
- out-of-order packet:
  - discard (don't buffer) -> no receiver buffering!
  - Re-ACK packet with highest in-order seq #

Selective repeat

- receiver *individually* acknowledges all correctly received packets
  - buffers packets, as needed, for eventual in-order delivery to upper layer
- sender times-out/retransmits individually for unACKed packets
  - sender maintains timer for each unACKed pkt
- sender window
  - *N* consecutive seq #s
  - limits seq #s of sent, unACKed packets

This is also known as Selective Acknowledgement or simply SACK

Selective repeat: sender, receiver windows



Selective repeat: sender and receiver

sender

data from above:
- if next available seq # in window, send packet

timeout(*n*):
- resend packet *n*, restart timer

ACK(*n*) in [sendbase,sendbase+N-1]:
- mark packet *n* as received
- if n smallest unACKed packet, advance window base to next unACKed seq #

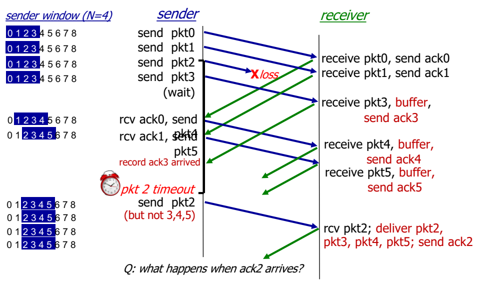
receiver

packet *n* in [rcvbase, rcvbase+N-1]
- send ACK(*n*)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order packets), advance window to next not-yet-received packet

packet *n* in [rcvbase-N,rcvbase-1]
- ACK(*n*)

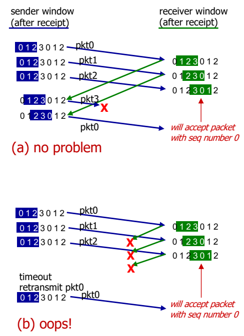
otherwise:
- ignore

Selective Repeat in action



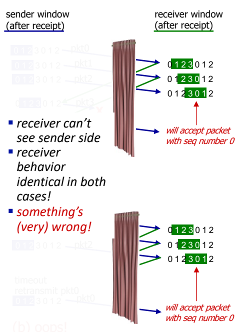
Selective repeat: a dilemma!

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



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)?

Solution:

maximum allowable window size = half the sequence number space.

Observations

- With sliding windows, it is possible to fully utilize a link, provided the window size (n) is large enough. Throughput is ~ (n/RTT)
  - Stop & Wait is like n = 1.
- Sender has to buffer all unacknowledged packets, because they may require retransmission
- Receiver may be able to accept out-of-order packets, but only up to its buffer limits
- Implementation complexity depends on protocol details (GBN vs. SR)

Recap: components of a solution

- Checksums (for error detection)
- Timers (for loss detection)
- Acknowledgments
  - cumulative
  - selective
- Sequence numbers (duplicates, windows)
- Sliding Windows (for efficiency)

- Reliability protocols use the above to decide when and what to retransmit or acknowledge

What does TCP do?

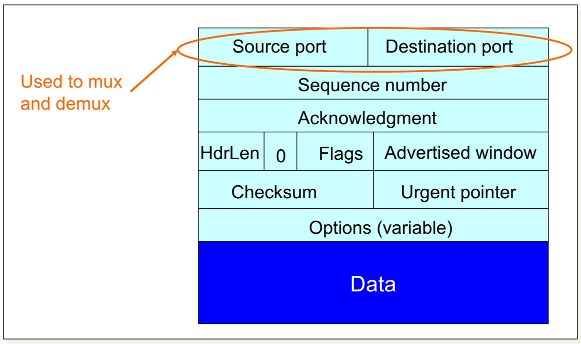
Most of our previous tricks + a few more beside

- Sequence numbers are byte offsets
- Sender and receiver maintain a sliding window
- Receiver sends cumulative acknowledgements (like GBN)
- Sender maintains a single retx. timer
- Receivers do not drop out-of-sequence packets (like SR)
- Introduces fast retransmit : optimization that uses duplicate ACKs to trigger early retx
- Introduces timeout estimation algorithms

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

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - 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

TCP Header

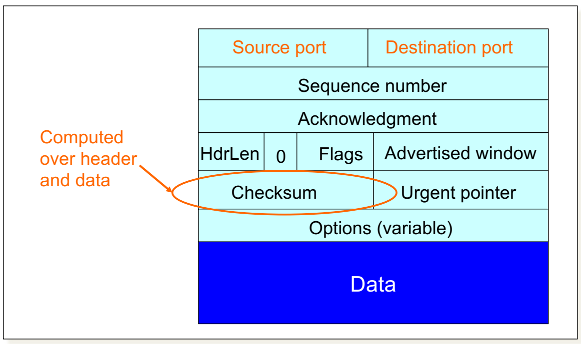


What does TCP do?

Many of our previous ideas, but some key differences

- Checksum

TCP Header



What does TCP do?

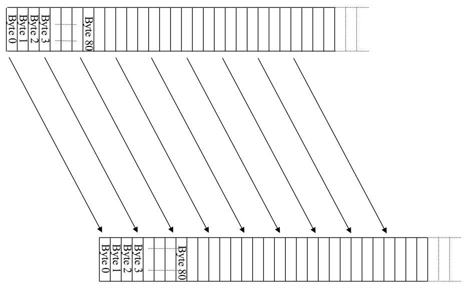
Many of our previous ideas, but some key differences

- Checksum
- **Sequence numbers are byte offsets**

TCP: Segments and Sequence Numbers

TCP "Stream of Bytes" Service…

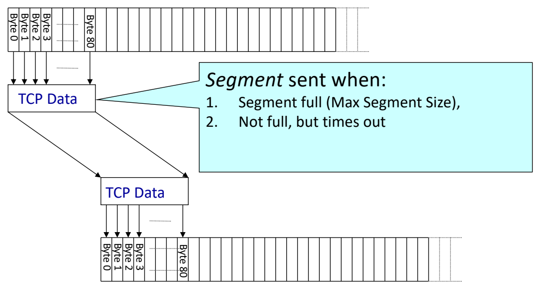
Application @ Host A



Application @ Host B

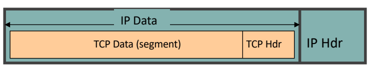
… Provided Using TCP "Segments"

Host A



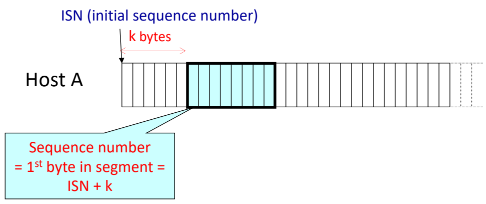
Host B

TCP Segment

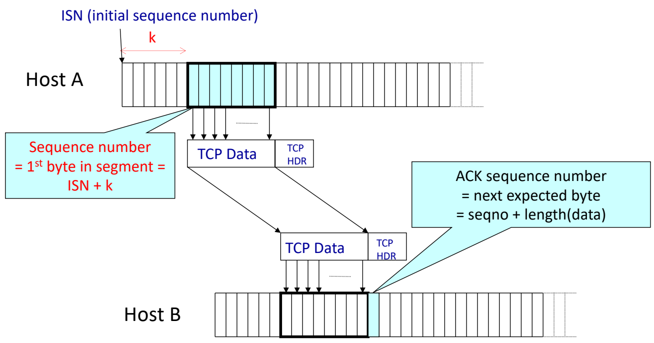


- IP packet
  - No bigger than Maximum Transmission Unit (MTU)
  - E.g., up to 1500 bytes with Ethernet
- TCP packet
  - IP packet with a TCP header and data inside
  - TCP header ≥ 20 bytes long
- TCP **segment**
  - No more than Maximum Segment Size (MSS) bytes
  - E.g., up to 1460 consecutive bytes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

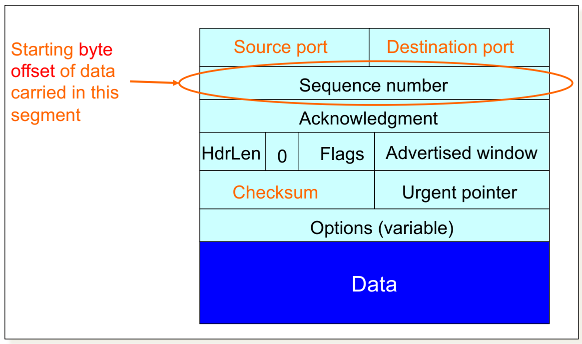
Sequence Numbers



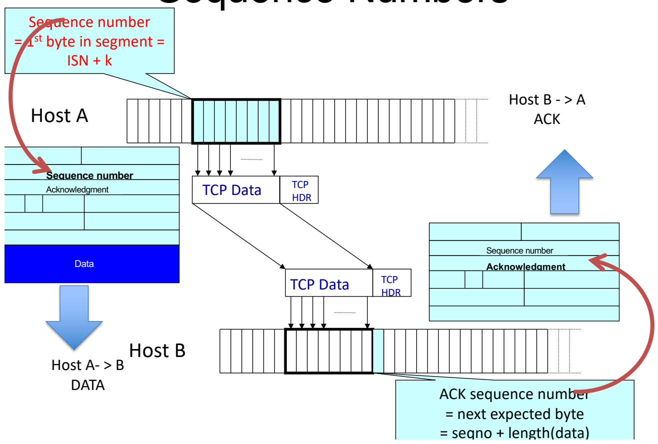
Sequence Numbers



TCP Header



Sequence Numbers



TCP Sequences and ACKS

TCP is full duplex by default

- two independently flows of sequence numbers

Sequence acknowledgement is given in terms of BYTES (not packets); the window is in terms of bytes.

number of packets = window size (bytes) / Segment Size

Servers and Clients are not Source and Destination

Piggybacking increases efficiency but many flows may only have data moving in one direction

What does TCP do?

Most of our previous tricks, but a few differences

- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)

ACKing and Sequence Numbers

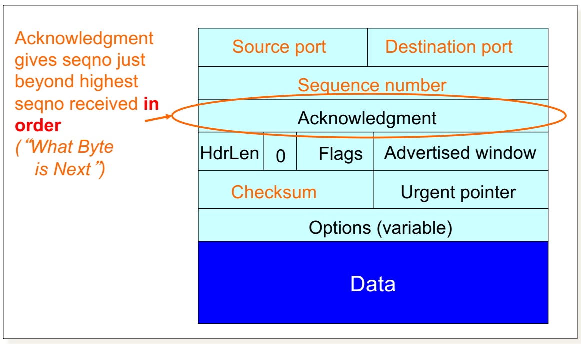
- Sender sends packet
  - Data starts with sequence number X
  - Packet contains B bytes [X, X+1, X+2, ...X+B-1]
- Upon receipt of packet, receiver sends an ACK
  - If all data prior to X already received:
    - ACK acknowledges X+B (because that is next expected byte)
  - If highest in-order byte received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

Normal Pattern

- Sender: seqno=X, length=B
- Receiver: ACK=X+B
- Sender: seqno=X+B, length=B
- Receiver: ACK=X+2B
- Sender: seqno=X+2B, length=B

- Seqno of next packet is same as last ACK field

TCP Header



What does TCP do?

Most of our previous tricks, but a few differences

- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)
- Receivers can buffer out-of-sequence packets (like SR)

Loss with cumulative ACKs

- Sender sends packets with 100B and seqnos.:
  – 100, 200, 300, 400, 500, 600, 700, 800, 900, …
- Assume the fifth packet (seqno 500) is lost, but no others
- Stream of ACKs will be:
  – 200, 300, 400, 500, 500, 500, 500,…

What does TCP do?

Most of our previous tricks, but a few differences

- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)
- Receivers may not drop out-of-sequence packets (like SR)
- Introduces fast retransmit: optimization that uses duplicate ACKs to trigger early retransmission

Loss with cumulative ACKs

- "Duplicate ACKs" are a sign of an <u>isolated</u> loss
  – The lack of ACK progress means 500 hasn't been delivered
  – Stream of ACKs means some packets are being delivered
- Therefore, could trigger resend upon receiving k duplicate ACKs
  - TCP uses k=3
- But response to loss is trickier….

Loss with cumulative ACKs

- Two choices:
  - Send missing packet and increase W by the number of dup ACKs
  - Send missing packet, and wait for ACK to increase W
- Which should TCP do?

What does TCP do?

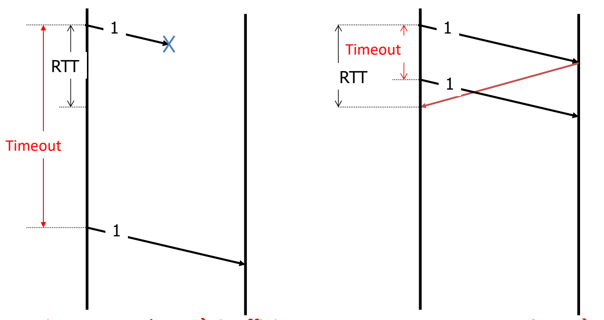
Most of our previous tricks, but a few differences

- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)
- Receivers do not drop out-of-sequence packets (like SR)
- Introduces fast retransmit: optimization that uses duplicate ACKs to trigger early retransmission
- Sender maintains a single retransmission timer (like GBN) and retransmits on timeout

Retransmission Timeout

- If the sender hasn't received an ACK by timeout, retransmit the first packet in the window
- How do we pick a timeout value?

Timing Illustration



Timeout too long → inefficient

Timeout too short → duplicate packets

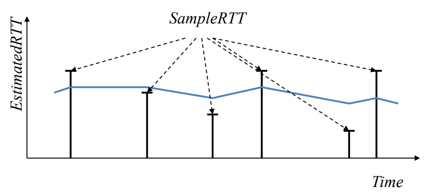
Retransmission Timeout

- If haven't received ack by timeout, retransmit the first packet in the window
- How to set timeout?
  - Too long: connection has low throughput
  - Too short: retransmit packet that was just delayed
- Solution: make timeout proportional to RTT
- But how do we measure RTT?

RTT Estimation

- Use exponential averaging of RTT samples

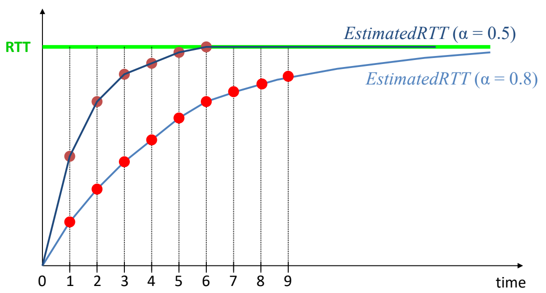
$$SampleRTT = AckRcvdTime - SendPacketTime$$
$$EstimatedRTT = \alpha \times EstimatedRTT + (1-\alpha) \times SampleRTT$$
$$0 < \alpha \le 1$$



Exponential Averaging Example

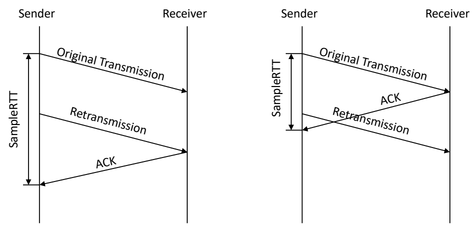
*EstimatedRTT = α*EstimatedRTT + (1 – α)*SampleRTT*

Assume RTT is constant → *SampleRTT* = RTT



Problem: Ambiguous Measurements

- How do we differentiate between the real ACK, and ACK of the retransmitted packet?



Karn/Partridge Algorithm

Discard junk measures

- Measure *SampleRTT* only for original transmissions
  - Once a segment has been retransmitted, do not use it for any further measurements
- Computes EstimatedRTT using *α = 0.875*
- Timeout value (RTO) = 2 × EstimatedRTT
- Employs exponential backoff
  - Every time RTO timer expires, set RTO ← 2·RTO
  - (Up to maximum ≥ 60 sec)
  - Every time new measurement comes in (= successful original transmission), collapse RTO back to 2 × EstimatedRTT

Jacobson/Karels Algorithm

Add a safety margin

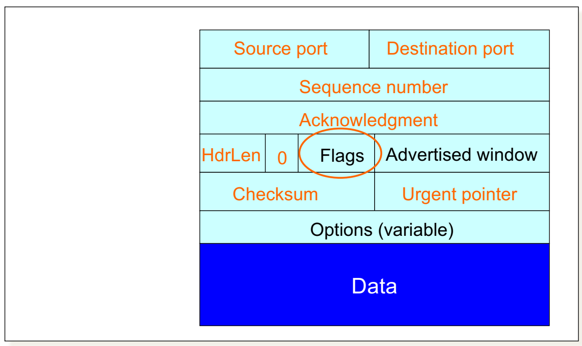
- Problem: need to better capture variability in RTT
  - Directly measure deviation
- Deviation = | SampleRTT – EstimatedRTT |
- EstimatedDeviation: exponential average of Deviation
- RTO = EstimatedRTT + 4 x EstimatedDeviation

What does TCP do?

Most of our previous ideas, but some key differences

- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)
- Receivers do not drop out-of-sequence packets (like SR)
- Introduces fast retransmit: optimization that uses duplicate ACKs to trigger early retransmission
- Sender maintains a single retransmission timer (like GBN) and retransmits on timeout

TCP Header: What's left?

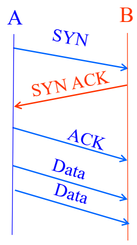


TCP Connection Establishment and Initial Sequence Numbers

Initial Sequence Number (ISN)

- Sequence number for the very first byte
- Why not just use ISN = 0?
- Practical issue
  - IP addresses and port #s uniquely identify a connection
  - Eventually, though, these port #s do get used again
  - … small chance an old packet is still in flight
- TCP therefore requires changing ISN
- Hosts exchange ISNs when they establish a connection

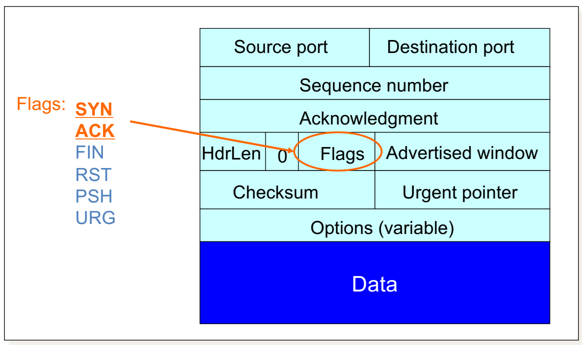
Establishing a TCP Connection



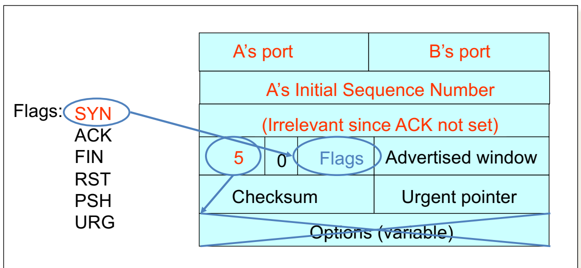
Each host tells its ISN to the other host.

- Three-way handshake to establish connection
  - Host A sends a **SYN** (open; "synchronize sequence numbers") to host B
  - Host B returns a SYN acknowledgment (**SYN ACK**)
  - Host A sends an **ACK** to acknowledge the SYN ACK

TCP Header

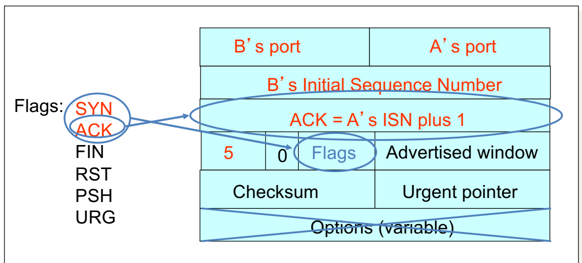


Step 1: A's Initial SYN Packet



A tells B it wants to open a connection…

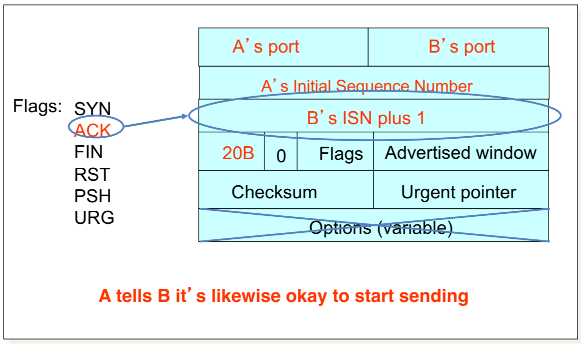
Step 2: B's SYN-ACK Packet



B tells A it accepts, and is ready to hear the next byte…

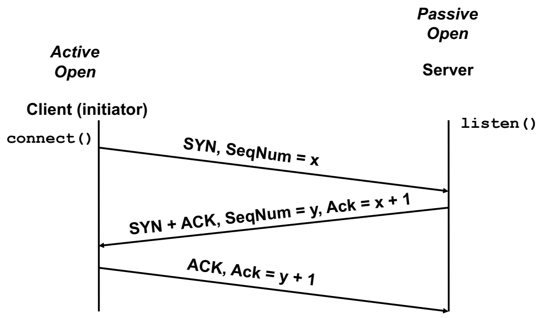
… upon receiving this packet, A can start sending data

Step 3: A’s ACK of the SYN-ACK



… upon receiving this packet, B can start sending data

Timing Diagram: 3-Way Handshaking

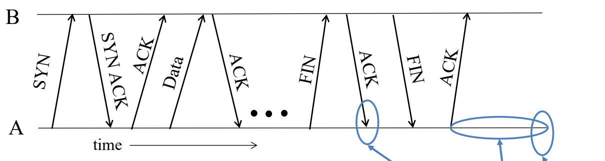


What if the SYN Packet Gets Lost?

- 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 seconds
    - Some implementations instead use 6 seconds

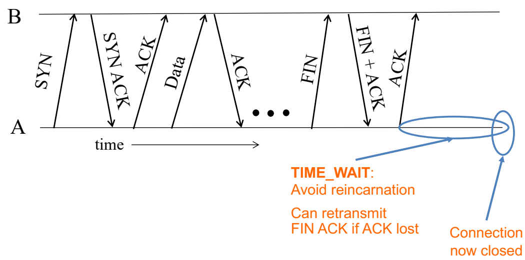
Tearing Down the Connection

Normal Termination, One Side At A Time



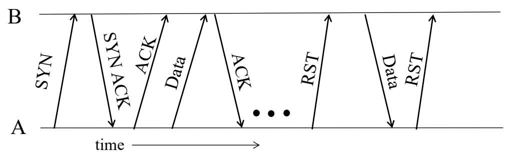
- Finish (**FIN**) to close and receive remaining bytes
  - **FIN** occupies one byte in the sequence space
- Other host acks the byte to confirm
- Closes A’s side of the connection, but not B’s
  - Until B likewise sends a **FIN**
  - Which A then acks

Normal Termination, Both Together



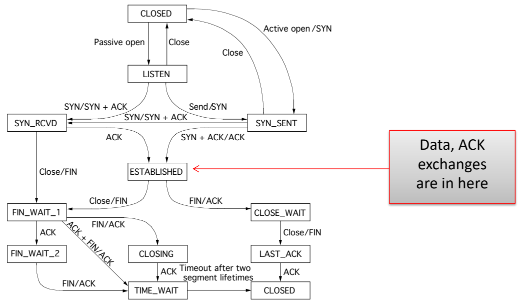
- Same as before, but B sets **FIN** with their ack of A’s **FIN**

Abrupt Termination

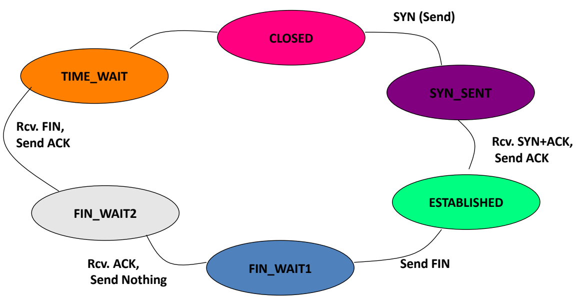


- A sends a RESET (**RST**) to B
  - E.g., because application process on A crashed
- That's it
  - B does not ack the **RST**
  - Thus, **RST** is not delivered reliably
  - And: any data in flight is lost
  - But: if B sends anything more, will elicit another **RST**

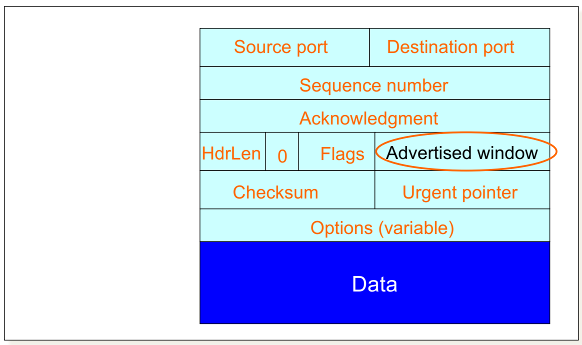
TCP State Transitions



An Simpler View of the Client Side



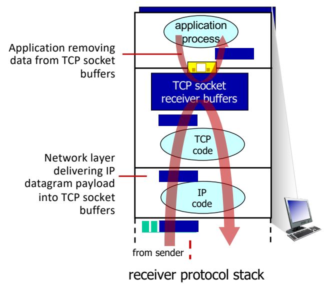
TCP Header



- What does TCP do?
  - ARQ windowing, set-up, tear-down
- Flow Control in TCP

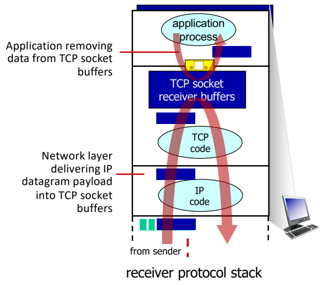
TCP flow control

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



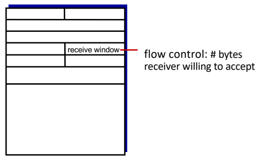
TCP flow control

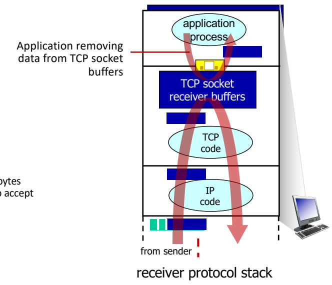
Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?



TCP flow control

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

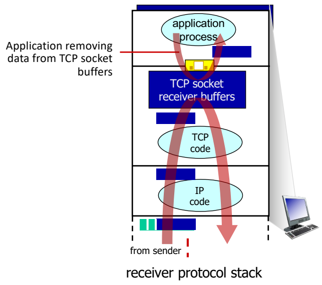




TCP flow control

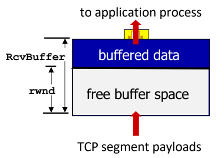
Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?

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



TCP flow control

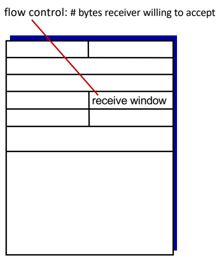
- TCP receiver "advertises" free buffer space in **`rwnd`** field in TCP header
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **`RcvBuffer`**
- sender limits amount of unACKed ("in-flight") data to received **`rwnd`**
- guarantees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



TCP receiver-side buffering

TCP 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 segment format

Advertised Window Limits Rate

- Sender can send no faster than W/RTT bytes/sec
- Receiver only advertises more space when it has consumed old arriving data
- In original TCP design, that was the **sole** protocol mechanism controlling sender's rate
- What's missing?

- What does TCP do?
  - ARQ windowing, set-up, tear-down
- Flow Control in TCP
- **Congestion Control in TCP**

We have seen:

- Flow control: adjusting the sending rate to keep from overwhelming a slow *receiver*

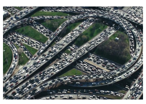
Now lets attend…

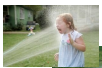
- Congestion control: adjusting the sending rate to keep from overloading the *network*

Principles of congestion control

Congestion:

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





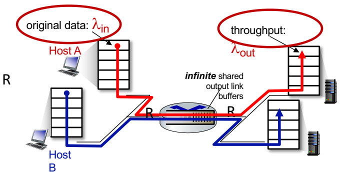
congestion control: too many senders, sending too fast

flow control: one sender too fast for one receiver

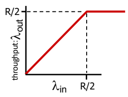
Causes/costs of congestion: scenario 1

Simplest scenario:

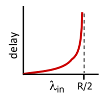
- one router, infinite buffers
- input, output link capacity: R
- two flows
- no retransmissions needed



Q: What happens as arrival rate λ_{in} approaches R/2?



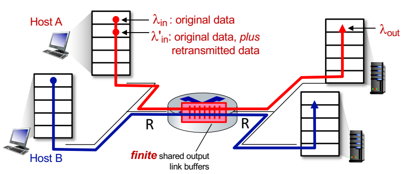
maximum per-connection throughput: R/2



large delays as arrival rate λ_{in} approaches capacity

Causes/costs of congestion: scenario 2

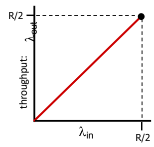
- one router, *finite* buffers
- sender retransmits lost, timed-out packet
  - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
  - transport-layer input includes *retransmissions* : $\lambda'_{in} \geq \lambda_{in}$

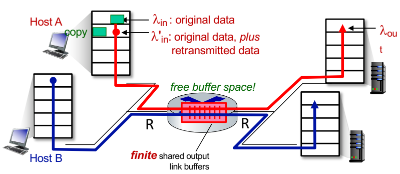


Causes/costs of congestion: scenario 2

Idealization: perfect knowledge

- sender sends only when router buffers available

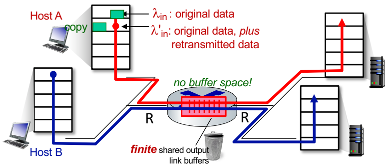




Causes/costs of congestion: scenario 2

Idealization: *some* perfect knowledge

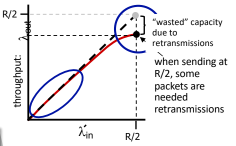
- packets can be lost (dropped at router) due to full buffers
- sender knows when packet has been dropped: only resends if packet *known* to be lost

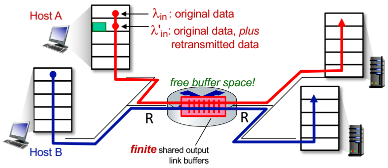


Causes/costs of congestion: scenario 2

Idealization: *some* perfect knowledge

- packets can be lost (dropped at router) due to full buffers
- sender knows when packet has been dropped: only resends if packet *known* to be lost

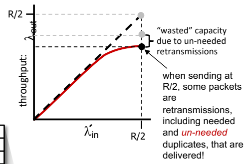


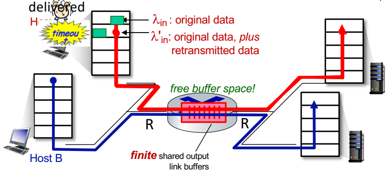


Causes/costs of congestion: scenario 2

Realistic scenario: *un-needed duplicates*

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

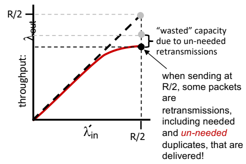




Causes/costs of congestion: scenario 2

Realistic scenario: *un-needed duplicates*

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



"costs" of congestion:

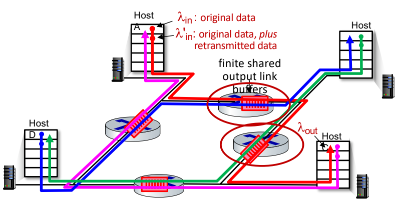
- more work (retransmission) for given receiver throughput
- unneeded retransmissions: link carries multiple copies of a packet
  - decreasing maximum achievable throughput

Causes/costs of congestion: scenario 3

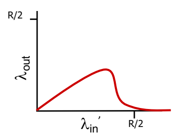
- *four* senders
- *multi-hop* paths
- timeout/retransmit

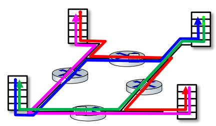
Q: what happens as λ_{in} and λ_{in}' increase ?

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



Causes/costs of congestion: scenario 3



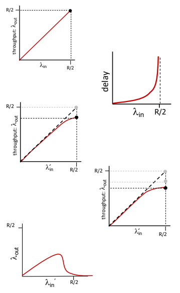


another "cost" of congestion:

- when packet dropped, any upstream transmission capacity and buffering used for that packet was wasted!

Causes/costs of congestion: insights

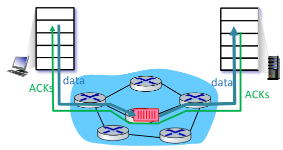
- throughput can never exceed capacity
- delay increases as capacity approached
- loss/retransmission decreases effective throughput
- un-needed duplicates further decreases effective throughput
- upstream transmission capacity / buffering wasted for packets lost downstream



Approaches towards congestion control

End-end congestion control:

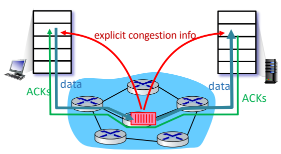
- no explicit feedback from network
- congestion *inferred* from observed loss, delay
- approach taken by TCP



Approaches towards congestion control

Network-assisted congestion control:

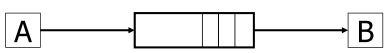
- routers provide *direct* feedback to sending/receiving hosts with flows passing through congested router
- may indicate congestion level or explicitly set sending rate
- TCP ECN, ATM, DECbit protocols



Three Issues to Consider

- Discovering the available (bottleneck) bandwidth
- Adjusting to variations in bandwidth
- Sharing bandwidth between flows

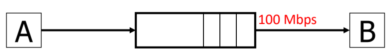
Abstract View



Sending Host Buffer in Router Receiving Host

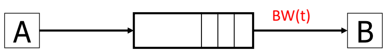
- Ignore internal structure of router and model it as having a single queue for a particular input-output pair

Discovering available bandwidth



- Pick sending rate to match bottleneck bandwidth
  - Without any *a priori* knowledge
  - Could be gigabit link, could be a modem

Adjusting to variations in bandwidth

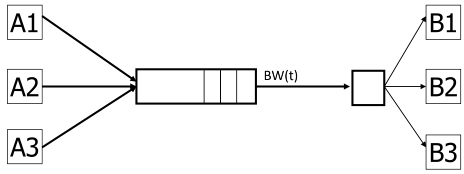


- Adjust rate to match instantaneous bandwidth
  – Assuming you have rough idea of bandwidth

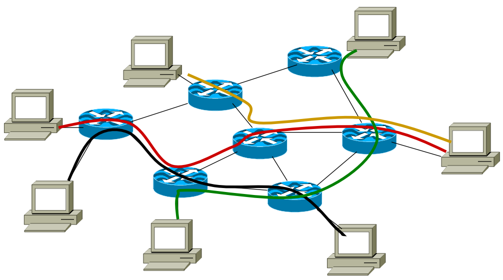
Multiple flows and sharing bandwidth

Two Issues:

- Adjust total sending rate to match bandwidth
- Allocation of bandwidth between flows



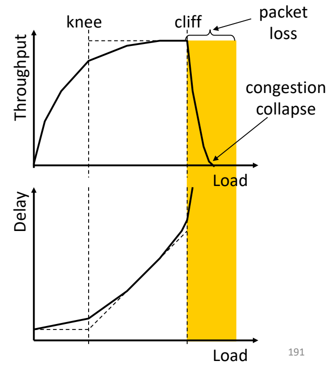
Reality



Congestion control is a resource allocation problem involving many flows, many links, and complicated global dynamics

View from a single flow

- 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



General Approaches

(0) Send without care

– Many packet drops

General Approaches

(0) Send without care

(1) Reservations

– Pre-arrange bandwidth allocations

– Requires negotiation before sending packets

– Low utilization

General Approaches

(0) Send without care

(1) Reservations

(2) Pricing

- Don't drop packets for the high-bidders
- Requires payment model

General Approaches

(0) Send without care

(1) Reservations

(2) Pricing

(3) Dynamic Adjustment

- Hosts probe network; infer level of congestion; adjust
- Network reports congestion level to hosts; hosts adjust
- Combinations of the above
- Simple to implement but suboptimal, messy dynamics

General Approaches

(0) Send without care

(1) Reservations

(2) Pricing

(3) Dynamic Adjustment

All three techniques have their place

- *Generality* of dynamic adjustment has proven powerful
- Doesn't presume business model, traffic characteristics, application requirements; does assume good citizenship

Who Takes Care of Congestion?

- Network? End hosts? Both?

- TCP's approach:
  - **End hosts** adjust sending rate
  - Based on **implicit feedback** from network

- Not the only approach
  - A consequence of history rather than planning

Some History: TCP in the 1980s

- Sending rate only limited by flow control
  - Packet drops → senders (repeatedly!) retransmit a full window's worth of packets

- Led to "congestion collapse" starting Oct. 1986
  - Throughput on the NSF network dropped from 32Kbits/s to 40bits/sec

- "Fixed" by Van Jacobson's development of TCP's congestion control (CC) algorithms

Jacobson's Approach

- Extend TCP's existing window-based protocol but adapt the window size in response to congestion
  - required no upgrades to routers or applications!
  - patch of a few lines of code to TCP implementations

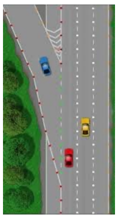
- A pragmatic and effective solution
  - but many other approaches exist

- Extensively improved on since
  - topic now sees less activity in ISP contexts
  - but is making a comeback in datacenter environments

TCP's Approach in a Nutshell

- TCP connection has window
  - Controls number of packets in flight
- Sending rate: ~Window/RTT
- Vary window size to control sending rate

Windows, Buffers, and TCP



Windows, Buffers, and TCP

- TCP connection has a window
  - Controls number of packets in flight;
    filling a channel to improve throughput, and
    vary window size to control sending rate
- Buffers adapt mis-matched channels
  - Buffers smooth bursts
  - Adapt (re-time) arrivals for multiplexing

Windows, Buffers, and TCP

Buffers & TCP can make link utilization 100%

but

Buffers add delay, **variable** delay

Sizing Buffers in Routers

- Packet loss
  - Queue overload, and subsequent packet loss
- End-to-end delay
  - Transmission, propagation, and queueing delay
  - The only variable part is queueing delay
- Router architecture
  - Board space, power consumption, and cost
  - On chip buffers: higher density, higher capacity

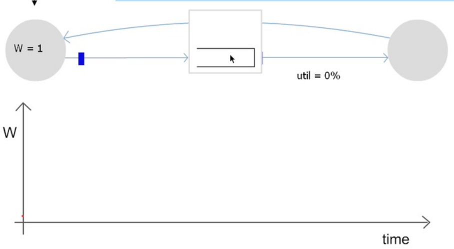
Buffer Sizing Story

	Rule-of-thumb
	$2T \times C$
# of packets	1,000,000
Intuition	TCP Sawtooth
Assume	Single TCP Flow, 100% Utilization
Evidence	Simulation, Emulation

Continuous ARQ (TCP) adapting to congestion

Only W packets may be outstanding

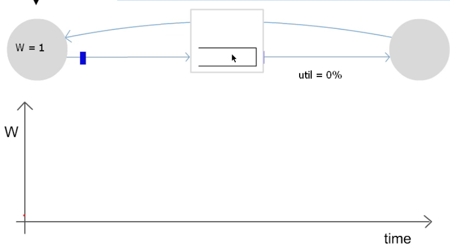
Rule for adjusting W
- If an ACK is received: W ← W+1/W
- If a packet is lost: W ← W/2



Continuous ARQ (TCP) adapting to congestion

Only W packets may be outstanding

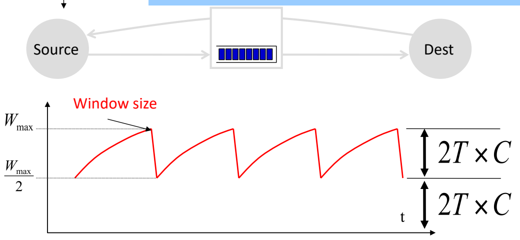
Rule for adjusting W
- If an ACK is received: W ← W+1/W
- If a packet is lost: W ← W/2



Rule-of-thumb – Intuition

Only W packets may be outstanding

Rule for adjusting W
- If an ACK is received: W ← W+1/W
- If a packet is lost: W ← W/2



Buffers in Routers

So how large should the buffers be?

Buffer size matters
- Packet loss
  - Queue overload, and subsequent packet loss
- End-to-end delay
  - Transmission, propagation, and queueing delay
  - The only variable part is queueing delay

Buffer Sizing Story

	Rule-of-thumb $2T \times C$	Small Buffers $\frac{2T \times C}{\sqrt{n}}$
# of packets	1,000,000	10,000
Intuition	TCP Sawtooth	Sawtooth Smoothing
Assume	Single TCP Flow, 100% Utilization	Many Flows, 100% Utilization
Evidence	Simulation, Emulation	Simulations, Test-bed and Real Network Experiments

Buffers in Routers

So how large should the buffers be?

Buffer size matters
- Packet loss
  - Queue overload, and subsequent packet loss
- End-to-end delay
  - Transmission, propagation, and queueing delay
  - The only variable part is queueing delay
- Router architecture
  - Board space, power consumption, and cost
  - On chip buffers: higher density, higher capacity

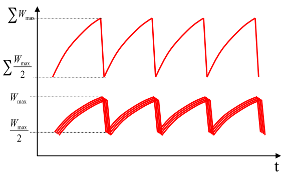
Small Buffers – Intuition

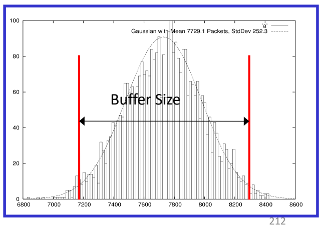
Synchronized Flows

- Aggregate window has same dynamics
- Therefore buffer occupancy has same dynamics
- Rule-of-thumb still holds.

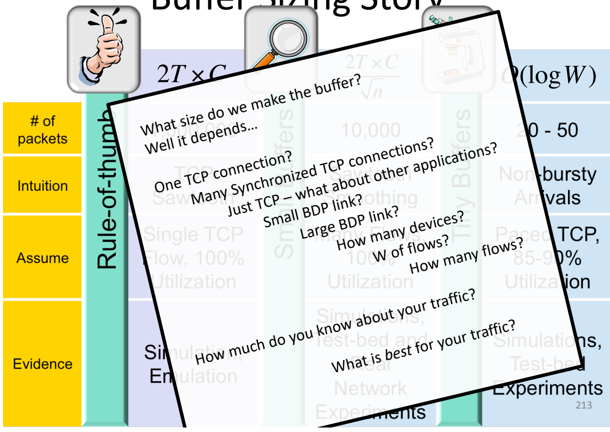
Many TCP Flows

- Independent, desynchronized
- Central limit theorem says the aggregate becomes Gaussian
- Variance (buffer size) decreases as N increases





Buffer Sizing Story



TCP's Approach in a Nutshell

- TCP connection has window
  - Controls number of packets in flight
- Sending rate: ~Window/RTT
- Vary window size to control sending rate

All These Windows…

- Congestion Window: CWND
  - How many bytes can be sent without overflowing routers
  - Computed by the sender using congestion control algorithm
- Flow control window: AdvertisedWindow (RWND)
  - How many bytes can be sent without overflowing receiver's buffers
  - Determined by the receiver and reported to the sender
- Sender-side window = minimum{CWND,RWND}
  - Assume for this material that RWND >> CWND

Note

- This lecture will talk about CWND in units of MSS
  - (Recall MSS: Maximum Segment Size, the amount of payload data in a TCP packet)
  - This is only for pedagogical purposes
- **In reality this is a SIMPLICATION:**
  Real implementations maintain CWND in bytes

Two Basic Questions

- How does the sender detect congestion?
- How does the sender adjust its sending rate?
  - To address three issues
    - Finding available bottleneck bandwidth
    - Adjusting to bandwidth variations
    - Sharing bandwidth

(Recall) Detecting Congestion

- Packet delays
  - Tricky: noisy signal (delay often varies considerably)
- Router tell end-hosts they're congested
- Packet loss
  - Fail-safe signal that TCP already has to detect
  - Complication: non-congestive loss (checksum errors)
- Two indicators of packet loss
  - No ACK after certain time interval: timeout
  - Multiple duplicate ACKs

Not All Losses the Same

- Duplicate ACKs: isolated loss
  - Still getting ACKs
- Timeout: much more serious
  - Not enough packets in progress to trigger duplicate-acks, OR
  - Suffered several losses
- We will adjust rate differently for each case

Rate Adjustment

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

Bandwidth Discovery with Slow Start

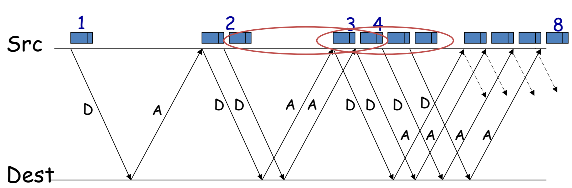
- Goal: estimate available bandwidth
  - start slow (for safety)
  - but ramp up quickly (for efficiency)
- Consider
  - RTT = 100ms, MSS=1000bytes
  - Window size to fill 1Mbps of BW = 12.5 packets
  - Window size to fill 1Gbps = 12,500 packets
  - Either is possible!

"Slow Start" Phase

- Sender starts at a slow rate but increases **exponentially** until first loss
- Start with a small congestion window
  - Initially, CWND = 1
  - So, initial sending rate is MSS/RTT
- Double the CWND for each RTT with no loss

Slow Start in Action

- For each RTT: double CWND
- Simpler implementation: for each ACK, CWND += 1



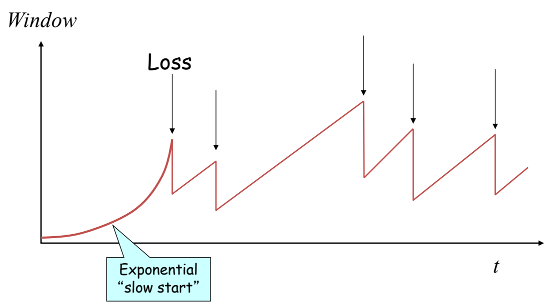
Adjusting to Varying Bandwidth

- 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 backoff (rate decrease)
- TCP uses: “Additive Increase Multiplicative Decrease” (AIMD)
  - We’ll see why shortly…

AIMD

- Additive increase
  - Window grows by one MSS for every RTT with no loss
  - For each successful RTT, CWND = CWND + 1
  - Simple implementation:
    - for each ACK, CWND = CWND+ 1/CWND
- Multiplicative decrease
  - On loss of packet, divide congestion window in **half**
  - On loss, CWND = CWND/2

Leads to the TCP “Sawtooth”



Slow-Start vs. AIMD

- When does a sender stop Slow-Start and start Additive Increase?
- Introduce a “slow start threshold” (ssthresh)
  - Initialized to a large value
  - On timeout, ssthresh = CWND/2
- When CWND = ssthresh, sender switches from slow-start to AIMD-style increase

- What does TCP do?
  - ARQ windowing, set-up, tear-down
- Flow Control in TCP
- Congestion Control in TCP
  - AIMD (slow-start, congestion avoidance)

- What does TCP do?
  - ARQ windowing, set-up, tear-down
- Flow Control in TCP
- Congestion Control in TCP
  - AIMD (slow-start, congestion avoidance) and Fast-Recovery

One Final Phase: Fast Recovery

- The problem: congestion avoidance too slow in recovering from an isolated loss

Example (in units of MSS, not bytes)

- Consider a TCP connection with:
  - CWND=10 packets
  - Last ACK was for packet # 101
    - i.e., receiver expecting next packet to have seq. no. 101
- 10 packets [101, 102, 103,..., 110] are in flight
  - Packet 101 is dropped
  - What ACKs do they generate?
  - And how does the sender respond?

The problem – A timeline

- ACK 101 (due to 102) cwnd=10 dupACK#1 (no xmit)
- ACK 101 (due to 103) cwnd=10 dupACK#2 (no xmit)
- ACK 101 (due to 104) cwnd=10 dupACK#3 (no xmit)
- RETRANSMIT 101 ssthresh=5 cwnd= 5
- ACK 101 (due to 105) cwnd=5 + 1/5 (no xmit)
- ACK 101 (due to 106) cwnd=5 + 2/5 (no xmit)
- ACK 101 (due to 107) cwnd=5 + 3/5 (no xmit)
- ACK 101 (due to 108) cwnd=5 + 4/5 (no xmit)
- ACK 101 (due to 109) cwnd=5 + 5/5 (no xmit)
- ACK 101 (due to 110) cwnd=6 + 1/5 (no xmit)
- ACK 111 (due to 101) 🡐 only now can we transmit new packets
- Plus no packets in flight so ACK "clocking" (to increase CWND) stalls for another RTT

Solution: Fast Recovery

Idea: Grant the sender temporary "credit" for each dupACK so as to keep packets in flight

- If dupACKcount = 3
  - ssthresh = cwnd/2
  - cwnd = ssthresh + 3
- While in fast recovery
  - cwnd = cwnd + 1 for each additional duplicate ACK
- Exit fast recovery after receiving new ACK
  - set cwnd = ssthresh

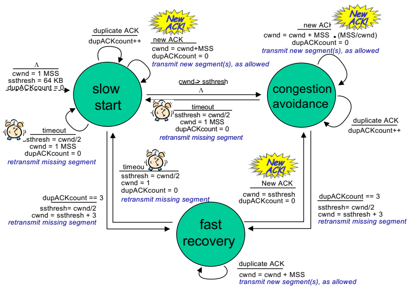
Example

- Consider a TCP connection with:
  - CWND=10 packets
  - Last ACK was for packet # 101
    - i.e., receiver expecting next packet to have seq. no. 101
- 10 packets [101, 102, 103,..., 110] are in flight
  - Packet 101 is dropped

Timeline

- ACK 101 (due to 102) cwnd=10 dup#1
- ACK 101 (due to 103) cwnd=10 dup#2
- ACK 101 (due to 104) cwnd=10 dup#3
- REXMIT 101 ssthresh=5 cwnd= 8 (5+3)
- ACK 101 (due to 105) cwnd= 9 (no xmit)
- ACK 101 (due to 106) cwnd=10 (no xmit)
- ACK 101 (due to 107) cwnd=11 (xmit 111)
- ACK 101 (due to 108) cwnd=12 (xmit 112)
- ACK 101 (due to 109) cwnd=13 (xmit 113)
- ACK 101 (due to 110) cwnd=14 (xmit 114)
- ACK 111 (due to 101) cwnd = 5 (xmit 115) 🡐 exiting fast recovery
- Packets 111-114 already in flight
- ACK 112 (due to 111) cwnd = 5 + 1/5 ← back in congestion avoidance

Summary: TCP congestion control



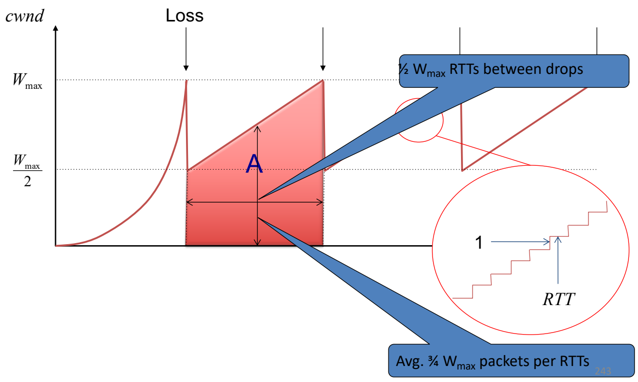
- What does TCP do?
  - ARQ windowing, set-up, tear-down
- Flow Control in TCP
- Congestion Control in TCP
  - AIMD (slow-start, congestion avoidance) and Fast-Recovery

Congestion avoidance algorithm has been a fertile field….

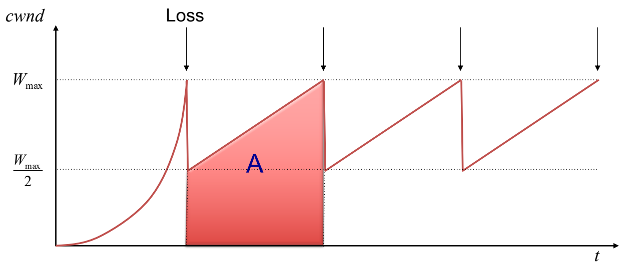
Variant	Feedback	Required changes	Benefits	Fairness
(New) Reno	Loss	—	—	Delay
Vegas	Delay	Sender	Less loss	Proportional
High Speed	Loss	Sender	High bandwidth	
BIC	Loss	Sender	High bandwidth	
CUBIC	Loss	Sender	High bandwidth	
C2TCP[11][12]	Loss/Delay	Sender	Ultra-low latency and high bandwidth	
NATCP[13]	Multi-bit signal	Sender	Near Optimal Performance	
Elastic-TCP	Loss/Delay	Sender	High bandwidth/short & long-distance	
Agile-TCP	Loss	Sender	High bandwidth/short-distance	
H-TCP	Loss	Sender	High bandwidth	
FAST	Delay	Sender	High bandwidth	Proportional
Compound TCP	Loss/Delay	Sender	High bandwidth	Proportional
Westwood	Loss/Delay	Sender	L	
Jersey	Loss/Delay	Sender	L	
BBR[14]	Delay	Sender	BLVC, Bufferbloat	
CLAMP	Multi-bit signal	Receiver, Router	V	Max-min
TFRC	Loss	Sender, Receiver	No Retransmission	Minimum delay
XCP	Multi-bit signal	Sender, Receiver, Router	BLFC	Max-min
VCP	2-bit signal	Sender, Receiver, Router	BLF	Proportional
MaxNet	Multi-bit signal	Sender, Receiver, Router	BLFSC	Max-min
JetMax	Multi-bit signal	Sender, Receiver, Router	High bandwidth	Max-min
RED	Loss	Router	Reduced delay	
ECN	Single-bit signal	Sender, Receiver, Router	Reduced loss	

TCP Throughput Equation

A Simple Model for TCP Throughput



A Simple Model for TCP Throughput



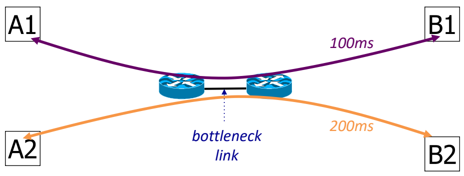
Packet drop rate, $p = 1/A$, where $A = \frac{3}{8}W_{max}^2$

Throughput, $B = \dfrac{A}{\left(\frac{W_{max}}{2}\right)RTT} = \sqrt{\frac{3}{2}}\dfrac{1}{RTT\sqrt{p}}$

Implications (1): Different RTTs

$$\text{Throughput} = \sqrt{\frac{3}{2}} \frac{1}{RTT\sqrt{p}}$$

- Flows get throughput inversely proportional to RTT
- TCP unfair in the face of heterogeneous RTTs!



Implications (2): High Speed TCP

$$\text{Throughput} = \sqrt{\frac{3}{2}} \frac{1}{RTT\sqrt{p}}$$

- Assume RTT = 100ms, MSS=1500bytes
- What value of *p* is required to reach 100Gbps throughput
  - ~ 2×10^{-12}
- How long between drops?
  - ~ 16.6 hours
- How much data has been sent in this time?
  - ~ 6 petabits
- These are not practical numbers!

Adapting TCP to High Speed

- Once past a threshold speed, increase CWND faster
  - A proposed standard [Floyd'03]: once speed is past some threshold, change equation to $p^{-.8}$ rather than $p^{-.5}$
  - Let the additive constant in AIMD depend on CWND
- Other approaches?
  - Multiple simultaneous connections (*hacky* but works today)
  - Router-assisted approaches (will see shortly)

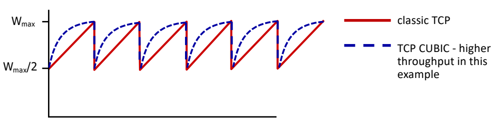
Implications (3): *Rate*-based CC

$$\text{Throughput} = \sqrt{\frac{3}{2}} \frac{1}{RTT\sqrt{p}}$$

- TCP throughput is "choppy"
  - repeated swings between W/2 to W
- Some apps would prefer sending at a steady rate
  - e.g., streaming apps
- A solution: "Equation-Based Congestion Control"
  - ditch TCP's increase/decrease rules and just follow the equation
  - measure drop percentage *p*, and set rate accordingly
- Following the TCP equation ensures we're "TCP friendly"
  - i.e., use no more than TCP does in similar setting

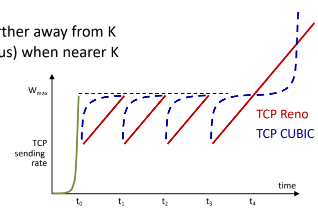
TCP CUBIC

- Is there a better way than AIMD to "probe" for usable bandwidth?
- Insight/intuition:
  - W_{max}: sending rate at which congestion loss was detected
  - congestion state of bottleneck link probably (?) hasn't changed much
  - after cutting rate/window in half on loss, initially ramp to to W_{max} *faster*, but then approach W_{max} more *slowly*



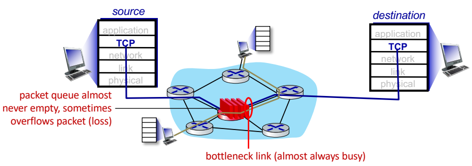
TCP CUBIC

- K: point in time when TCP window size will reach W_{max}
  - K itself is tuneable
- increase W as a function of the *cube* of the distance between current time and K
  - larger increases when further away from K
  - smaller increases (cautious) when nearer K
- TCP CUBIC default in Linux, most popular TCP for popular Web servers



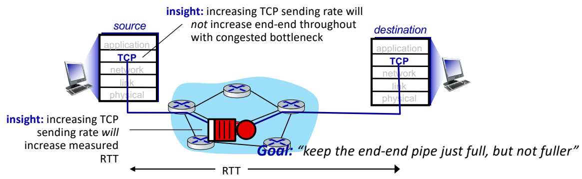
TCP and the congested "bottleneck link"

- TCP (classic, CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: the *bottleneck link*



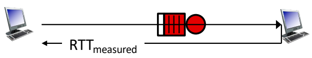
TCP and the congested "bottleneck link"

- TCP (classic, CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: the *bottleneck link*
- understanding congestion: useful to focus on congested bottleneck link



Delay-based TCP Congestion Control

Keeping sender-to-receiver pipe "just full enough, but no fuller": keep bottleneck link busy transmitting, but avoid high delays/buffering



$$\text{measured throughput} = \frac{\text{\# bytes sent in last RTT interval}}{RTT_{measured}}$$

Delay-based approach:

- RTT_{min} - minimum observed RTT (uncongested path)
- uncongested throughput with congestion window `cwnd` is $cwnd/RTT_{min}$

if measured throughput "very close" to uncongested throughput
increase `cwnd` linearly /* since path not congested */
else if measured throughput "far below" uncongested throughput
decrease `cwnd` linearly /* since path is congested */

Delay-based TCP Congestion Control

- congestion control without inducing/forcing loss
- maximizing throughput ("keeping the just pipe full… ") while keeping delay low ("…but not fuller")
- a number of deployed TCPs take a delay-based approach
  - BBR deployed on Google's (internal) backbone network

Recap: TCP problems

Routers tell endpoints if they're congested

- Misled by non-congestion losses
- Fills up queues leading to high delays
- Short flows complete before discovering available capacity
- AIMD impractical for high speed links
- Sawtooth discovery too choppy for some apps
- Unfair under heterogeneous RTTs
- Tight coupling with reliability mechanisms
- Endhosts can cheat

Routers tell endpoints what rate to send at

Routers enforce fair sharing

Could fix many of these with some help from routers!

Router-Assisted Congestion Control

- Three tasks for CC:
  - Isolation/fairness
  - Adjustment*
  - Detecting congestion

* This may be *automatic* eg loss-response of TCP

How can routers ensure each flow gets its “fair share”?

Fairness: General Approach

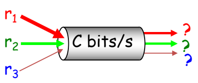
- Routers classify packets into “flows”
  - (For now) flows are packets between same source/destination
- Each flow has its own FIFO queue in router
- Router services flows in a fair fashion
  - When line becomes free, take packet from next flow in a fair order
- What does “fair” mean exactly?

Max-Min Fairness

- Given set of bandwidth demands r_i and total bandwidth C, max-min bandwidth allocations are:

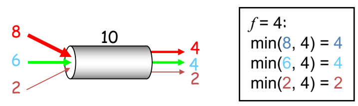
  $a_i = \min(f, r_i)$

  where f is the unique value such that Sum(a_i) = C



Example

- $C = 10$; $r_1 = 8$, $r_2 = 6$, $r_3 = 2$; $N = 3$
- $C/3 = 3.33 \rightarrow$
  - Can service all of r_3
  - Remove r_3 from the accounting: $C = C - r_3 = 8$; $N = 2$
- $C/2 = 4 \rightarrow$
  - Can’t service all of r_1 or r_2
  - So hold them to the remaining fair share: $f = 4$



Max-Min Fairness

- Given set of bandwidth demands r_i and total bandwidth C, max-min bandwidth allocations are:

  $a_i = \min(f, r_i)$
- where f is the unique value such that Sum(a_i) = C
- Property:
  - If you don’t get full demand, no one gets more than you
- This is what round-robin service gives if all packets are the same size

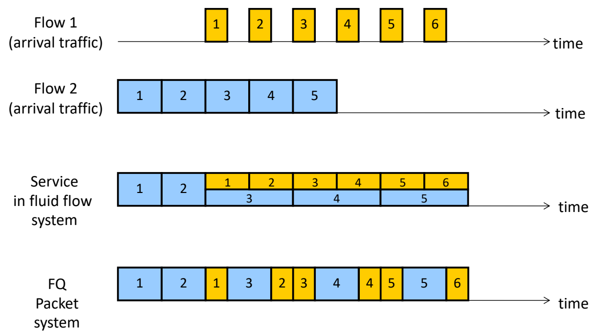
How do we deal with packets of different sizes?

- Mental model: Bit-by-bit round robin (“fluid flow”)
- Can you do this in practice?
- No, packets cannot be preempted
- But we can approximate it
  - This is what “fair queuing” routers do

Fair Queuing (FQ)

- For each packet, compute the time at which the last bit of a packet would have left the router *if* flows are served bit-by-bit
- Then serve packets in the increasing order of their deadlines

Example



Fair Queuing (FQ)

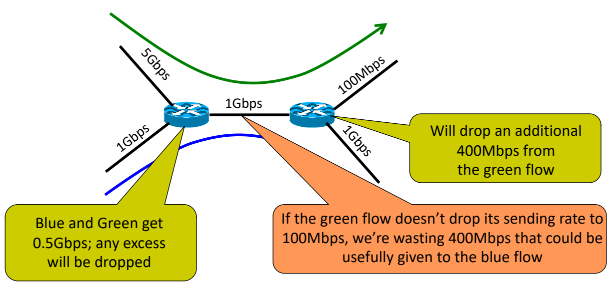
- Think of it as an implementation of round-robin generalized to the case where not all packets are equal sized
- Weighted fair queuing (WFQ): assign different flows different shares
- Today, some form of WFQ implemented in almost all routers
  - Not the case in the 1980-90s, when CC was being developed
  - Mostly used to isolate traffic at larger granularities (e.g., per-prefix)

FQ vs. FIFO

- FQ advantages:
  - Isolation: cheating flows don't benefit
  - Bandwidth share does not depend on RTT
  - Flows can pick any rate adjustment scheme they want
- Disadvantages:
  - More complex than FIFO: per flow queue/state, additional per-packet book-keeping

FQ in the big picture

- FQ does not eliminate congestion → it just manages the congestion



FQ in the big picture

- FQ does not eliminate congestion → it just manages the congestion
  - robust to cheating, variations in RTT, details of delay, reordering, retransmission, *etc.*
- But congestion (and packet drops) still occurs
- And we still want end-hosts to discover/adapt to their fair share!
- What would the end-to-end argument say w.r.t. congestion control?

Fairness is a controversial goal

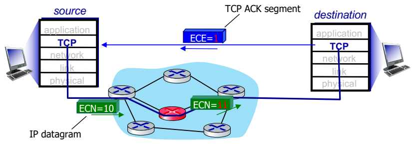
- What if you have 8 flows, and I have 4?
  - Why should you get twice the bandwidth
- What if your flow goes over 4 congested hops, and mine only goes over 1?
  - Why shouldn't you be penalized for using more scarce bandwidth?
- And what is a flow anyway?
  - TCP connection
  - Source-Destination pair?
  - Source?

Explicit Congestion Notification (ECN)

- Single bit in packet header; set by congested routers
  - If data packet has bit set, then ACK has ECN bit set
- Many options for when routers set the bit
  - tradeoff between (link) utilization and (packet) delay
- Congestion semantics can be exactly like that of drop
  - I.e., endhost reacts as though it saw a drop
- Advantages:
  - Don't confuse corruption with congestion; recovery w/ rate adjustment
  - Can serve as an early indicator of congestion to avoid delays
  - Easy (easier) to incrementally deploy
    - defined as extension to TCP/IP in RFC 3168 (uses diffserv bits in the IP header)

Explicit congestion notification (ECN)

TCP deployments often implement *network-assisted* congestion control:
- two bits in IP header (ToS field) marked *by network router* to indicate congestion
  - *policy* to determine marking chosen by network operator
- congestion indication carried to destination
- destination sets ECE bit on ACK segment to notify sender of congestion
- involves both IP (IP header ECN bit marking) and TCP (TCP header C,E bit marking)



Securing TCP

Vanilla TCP & UDP sockets:
- no encryption
- cleartext passwords sent into socket traverse Internet in cleartext (!)

Transport Layer Security (TLS)
- provides encrypted TCP connections
- data integrity
- end-point authentication

TLS implemented in application layer
- apps use TLS libraries, that use TCP in turn
- cleartext sent into "socket" traverse Internet *encrypted*

SSL vs. TLS
Simple: SSL is deprecated

TLS refers to secure socket layers in actual use.

Transport Recap

A *"big bag"*:
Multiplexing, reliability, error-detection, error-recovery, flow and congestion control, ….

- UDP:
  - Minimalist - multiplexing and error detection
- TCP:
  - somewhat hacky
  - but practical/deployable
  - good enough to have raised the bar for the deployment of new approaches
  - though the needs of datacenters change the status quos
- Beyond TCP (discussed in Topic 6):
  - QUIC / application-aware transport layers

Topic 6 – Applications

- Infrastructure Services (DNS)
  - Now with added security…
- Traditional Applications (web)
  - Now with added QUIC
- P2P Networks
  - Every device serves

Some network apps

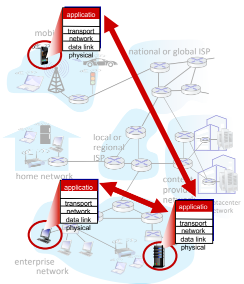
- social networking
- Web
- text messaging
- e-mail
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- P2P file sharing
- voice over IP (e.g., Skype)
- real-time video conferencing (e.g., Zoom)
- Internet search
- remote login
- …

Q: your favorites?

Creating a network app

write programs that:
- run on (different) end systems
- communicate over network
- e.g., web server software communicates with browser software

no need to write software for network-core devices
- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation

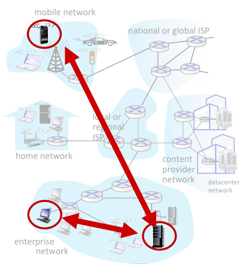


Client-server paradigm

server:
- always-on host
- permanent IP address
- often in data centers, for scaling

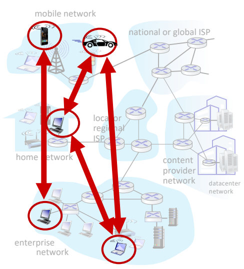
clients:
- contact, communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do *not* communicate directly with each other

- examples: HTTP, IMAP, FTP



Peer-peer architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management
- example: P2P file sharing



An application-layer protocol defines:

- types of messages exchanged,
  - e.g., request, response
- message syntax:
  - what fields in messages & how fields are delineated
- message semantics
  - meaning of information in fields
- rules for when and how processes send & respond to messages

open protocols:
- defined in RFCs, everyone has access to protocol definition
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:
- e.g., Skype, Zoom

Relationship Between Names&Addresses

- Addresses can change underneath
  - Move www.bbc.co.uk to 212.58.246.92
  - Humans/Apps should be unaffected
- Name could map to multiple IP addresses
  - www.bbc.co.uk to multiple replicas of the Web site
  - Enables
    - Load-balancing
    - Reducing latency by picking nearby servers
- Multiple names for the same address
  - E.g., aliases like www.bbc.co.uk and bbc.co.uk
  - Mnemonic stable name, and dynamic canonical name
    - Canonical name = actual name of host

DNS: Domain Name System

people: many identifiers:
- NI #, name, passport #

Internet hosts, routers:
- IP address (32 bit or 128bit) - used for addressing datagrams
- "name", e.g., cam.ac.uk- used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System (DNS):
- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, DNS servers communicate to *resolve* names (address/name translation)
  - *note:* core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

DNS: services, structure

DNS services:
- hostname-to-IP-address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - 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: 770B DNS queries/day
- Akamai DNS servers alone: 2.6T DNS queries/day

Thinking about the DNS

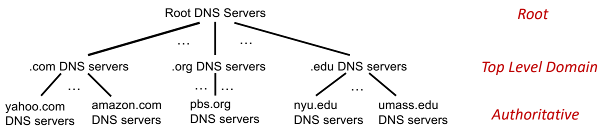
humongous distributed database:
- ~ billion records, each simple

handles many *trillions* of queries/day:
- *many* more reads than writes
- *performance matters:* almost every Internet transaction interacts with DNS - msecs count!

organizationally, physically decentralized:
- millions of different organizations responsible for their records

"bulletproof": reliability, security

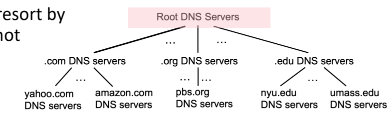
DNS: a distributed, hierarchical database



Client wants IP address for www.amazon.com; 1st approximation:
- client queries root server to find .com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: root name servers

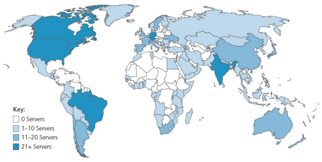
- official, contact-of-last-resort by name servers that can not resolve name



DNS: root name servers

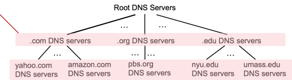
- official, contact-of-last-resort by name servers that can not resolve name
- *incredibly important* Internet function
  - Internet couldn't function without it!
  - DNSSEC – provides security (authentication, message integrity)
- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain

13 logical root name "servers" worldwide each "server" replicated many times (~200 servers in US)



Top-Level Domain, and authoritative 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 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

Using DNS

- Two components
  - DNS servers
  - Resolver software on each hosts
- Local DNS server ("default name server")
  - Usually near the endhosts that use it
  - each ISP has local DNS name server; to find yours:
    - MacOS: % scutil --dns
    - Windows: >ipconfig /all
- Client application
  - Extract server name (e.g., from the URL)
  - Do gethostbyname() to trigger resolver code

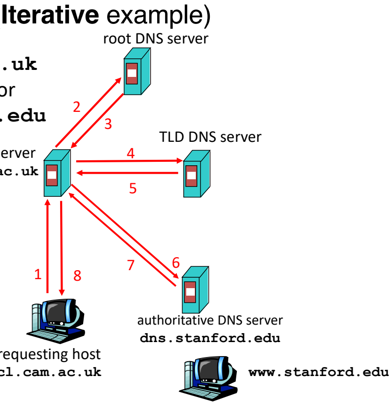
Local DNS name Servers

- when host makes DNS query, it is sent to its *local* DNS server
  - Local DNS server returns reply, answering:
    - from its local cache of recent name-to-address translation pairs (possibly out of date!)
    - forwarding request into DNS hierarchy for resolution
  - each ISP has local DNS name server; to find yours:
    - MacOS: `% scutil --dns`
    - Windows: `>ipconfig /all`
- local DNS server doesn't strictly belong to hierarchy, acting as they do on behalf of other hosts.

How Does Resolution Happen?

(**Iterative** example)

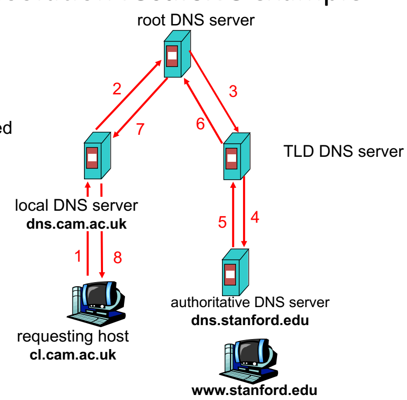
Host at **cl.cam.ac.uk** wants IP address for **www.stanford.edu**



iterated query:
- Host enquiry is delegated to local DNS server
- Consider transactions 2 – 7 only
- contacted server replies with name of next server to contact
- "I don't know this name, but ask this server"

DNS name resolution **recursive** example

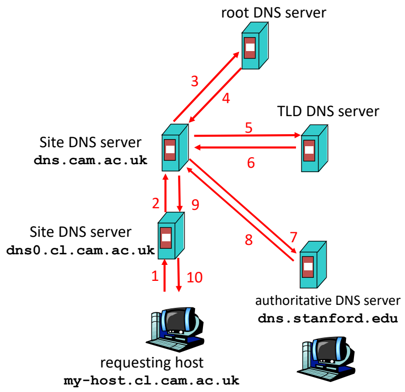
recursive query:
- puts burden of name resolution on contacted name server
- heavy load?



Recursive and Iterative Queries - **Hybrid** case

recursive query:
- Ask server to get answer for you
- E.g., requests 1,2 and responses 9,10

Interative query:
- Ask server who to ask next
- E.g., all other request-response pairs



DNS Caching

- Performing all these queries takes time
  - And all this before actual communication takes place
  - E.g., 1-second latency before starting Web download
- Caching greatly reduces overhead
  - The top-level servers very rarely change
  - Popular sites (e.g., www.bbc.co.uk) visited often
  - Local DNS servers have regularly used information cached
- How DNS caching works
  - DNS servers will cache responses to queries
  - Responses include a "time to live" (TTL) field
  - Server deletes cached entry after TTL expires
  - Cached entries may be *out-of-date*
    - if named host changes IP address, may not be known Internet-wide until all TTLs expire!
    - *best-effort name-to-address translation!*

Reliability

- DNS servers are replicated
  - Name service available if at least one replica is up
  - Queries can be load-balanced between replicas
- Anycast provides reliability for ROOT servers
- Usually, UDP is used for queries
  - Need reliability: must implement this on top of UDP
  - DNS spec. supports TCP too, but not always available
- Try alternate servers on timeout
  - Exponential backoff when retrying same server
- Same identifier for all queries
  - Don't care which server responds

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

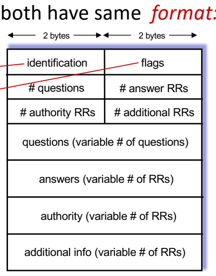
type=CNAME
- `name` is alias name for some "canonical" (the real) name
- www.ibm.com is really servereast.backup2.ibm.com
- `value` is canonical name

type=MX
- `value` is name of SMTP mail server associated with `name`

DNS protocol messages

DNS *query* and *reply* messages, both have same *format:*

message header:
- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



DNS protocol messages

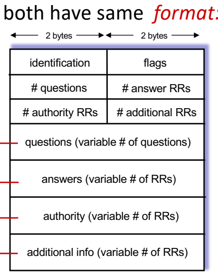
DNS *query* and *reply* messages, both have same *format:*

name, type fields for a query

RRs in response to query

records for authoritative servers

additional " helpful" info that may be used



Getting your info into the DNS

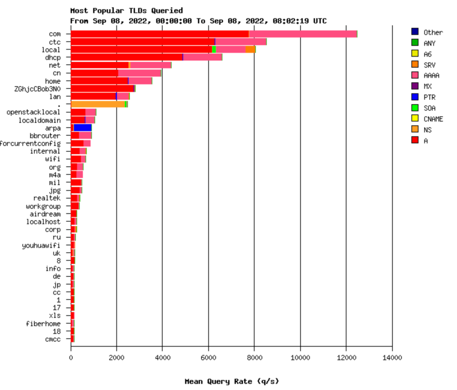
example: new startup "Network Utopia"

- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts NS, A RRs into .com TLD server:
    `(networkutopia.com, dns1.networkutopia.com, NS)`
    `(dns1.networkutopia.com, 212.212.212.1, A)`
- create authoritative server locally with IP address `212.212.212.1`
  - type A record for www.networkuptopia.com
  - type MX record for networkutopia.com

Most popular TLD

At least WORKGROUP is no longer here!
It was the top invalid TLD for years…

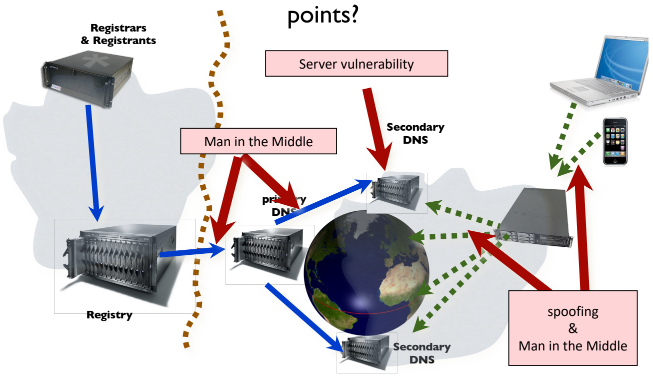
7 of top 10 invalid!



33

Data flow through the DNS

Where are the vulnerable points?



DNS attack surface

DDoS attacks

- bombard root servers with traffic
  - not successful to date
  - traffic filtering
  - local DNS servers cache IPs of TLD servers, allowing root server bypass
- bombard TLD servers
  - potentially more dangerous

Spoofing attacks

- intercept DNS queries, returning bogus replies
  - DNS cache poisoning
  - RFC 4033: DNSSEC authentication services

DNS Security

- No way to verify answers
  - Opens up DNS to many potential attacks
  - DNSSEC fixes this
- Most obvious vulnerability: recursive resolution
  - Using recursive resolution, host must trust DNS server
  - When at Starbucks, server is under their control
    And can return whatever values it wants
- More subtle attack: Cache poisoning
  - Those "additional" records can be anything!

36

DNSSEC protects all these end-to-end

- provides message authentication and integrity verification through cryptographic signatures
  - You know who provided the signature
  - No modifications between signing and validation
- It does **not** provide authorization
- It does **not** provide confidentiality
- It does **not** provide protection against DDOS

DNSSEC in practice

Problem: Scaling the key signing and key distribution
Solution: Using the DNS to Distribute Keys

- Distribute keys through the DNS hierarchy
  - Use one trusted key to establish authenticity of other keys
  - Building chains of trust from the root down
  - Parents need to sign the keys of their children
- Only the root key needed in ideal world
  - Parents always delegate security to child

On osx "*host –av www.cl.cam.ac.uk*

```
 % host -va www.cl.cam.ac.uk
Trying "www.cl.cam.ac.uk"
Trying "www.cl.cam.ac.uk"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25214
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 23, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.cl.cam.ac.uk. IN ANY

;; ANSWER SECTION:
www.cl.cam.ac.uk. 1200 IN RRSIG NSEC 5 5 1200 20230317214336 20230215204336 31575 cl.cam.ac.uk.
h2JCZHfF9m+jqvHQR6z67LDC3g1xKoCRDmrss+LVrAXJqWsi+d8+/Gio /U07C2SXKGa3NXj5ByNvqH2HJs6Loc1emEEeiPlSYqkOUSGTLRQhlEqy
bHEfeCPV4hJy/NSuRvcxZCqpgEDSbF5K0JzqI6dnPCoOoMFdSrA4n9OT kDQ=
www.cl.cam.ac.uk. 1200 IN NSEC www-443-120.cl.cam.ac.uk. A PTR TXT AAAA SSHFP RRSIG NSEC CAA
www.cl.cam.ac.uk. 21600 IN RRSIG SSHFP 5 5 21600 20230306104604 20230204102237 31575 cl.cam.ac.uk.
kfCKxAD9cyLJDj/UEJl7Sr8b55yH8dxfYc+BF9tgcqbReo2GNLQelOZN rB5JAhoewZ9HxlASO5rzCX1BR9AP0H+Rk7BNbDp8rvnO9G8PWGQcpKHm
BJWb9nj2a/zi360WCUGH/u8GlPxw0L7b2P460DYxE4wDmL3jNjvw61Ca Y3g=
www.cl.cam.ac.uk. 21600 IN SSHFP 3 2 B7E1DF5B943C481A263307EDEE23F0719858CDF516F2482B54A4B248 0118CAE9
www.cl.cam.ac.uk. 21600 IN SSHFP 2 1 6FB539DBE0E273B56327E619BB1814DB4CE810D8
www.cl.cam.ac.uk. 21600 IN SSHFP 4 2 293F122F4D4970C42B767898C5505C3EB838E0C5BB432EF36AF33C03 1FA792FA
www.cl.cam.ac.uk. 21600 IN SSHFP 4 1 5E0CBE0730922925C446DF1B2DCE336AA7565122
www.cl.cam.ac.uk. 21600 IN SSHFP 1 1 FF45237DF493102CF7478AE0A96DE773FB4877C4
www.cl.cam.ac.uk. 21600 IN SSHFP 1 2 2953D9172EC850D2A46FA0245DFFAE978EE31B3BED233DED77BC937B 115952D7
www.cl.cam.ac.uk. 21600 IN SSHFP 3 1 FD276CF12A0B909533ABFA5931622950308AF099
www.cl.cam.ac.uk. 21600 IN SSHFP 2 2 403A5EE7B8ADD3E16B5973874E54CDFAC82268CC63B4CFD90E74DDC6 4E2EDF6F
www.cl.cam.ac.uk. 21600 IN RRSIG AAAA 5 5 21600 20230316213444 20230214203939 31575 cl.cam.ac.uk.
eVJM0NwnGPVC9y+96IJq48feYCDxTlEZ66fcH83aO2VFXoCblJkLUCoK e0TeobR+mnLad0XJFUocfjKorIV6s1CNzG90nmV1+dxQD1VBxQzBrV9A
k+JqokUQbkvb0UsV4UIWUvRav0M1GccXS5Nxzl/HDITMyVXMZx/Citlr lqU=
www.cl.cam.ac.uk. 21600 IN AAAA 2a05:b400:110::80:14
www.cl.cam.ac.uk. 21600 IN RRSIG TXT 5 5 21600 20230321232700 20230219230148 31575 cl.cam.ac.uk.
Tjlztn2dsdjr5wGAkuPVTy/0V/BBTDEC3K8x7nNnml9dRoy/ncRLWEyA 9XsxENQ2Oei7evt6pelFstpVwny6F9nMs+xAFYDiX0PpcJ4pZMNADOgs
BNXhR2XS0IknnuqUuWPlH1WTFZqBd27gIsQ0F+79Atj3MHQ5hBZlCS0n EoE=
www.cl.cam.ac.uk. 21600 IN TXT "pseudo IP address for cl.cam.ac.uk departmental WWW server (IPv4 and IPv6)"
www.cl.cam.ac.uk. 21600 IN RRSIG PTR 5 5 21600 20230314163841 20230212160205 31575 cl.cam.ac.uk.
gceJom14zzcCsdYuU2ymxuMABhlZO6VDIp8J4seDqDt09924Avcpl2P9 5wlDds02a0JGTqbnnDiydKFgOA6fJHNMCEbAQr6GVjl/Fg+YLWH8YwLx
Uu9q7FqmZyXLk67wC9ji17VZ3V9Co5Kd2kWuDOvp7Xb5OevPKAll//tL BDk=
www.cl.cam.ac.uk. 21600 IN PTR svr-www-00.cl.cam.ac.uk.
www.cl.cam.ac.uk. 21600 IN RRSIG A 5 5 21600 20230314163841 20230212160205 31575 cl.cam.ac.uk.
j8k+Q8L2C7zzHSvpWpg+1t5WPk9IeTZ9GOvw/0v1pbYVXJfeHuNm9ERM Ff/hEkZm21ooFIBtrbTe/m5b+kBBm2OETDbbGP+na3/DEQyWFp0sHe6j
S2ZS9KeOXENJk92eA+/dBAWe0vFvTpXrZ/thp61ctqm9b3mR8AbnuNgu uHs=
www.cl.cam.ac.uk. 21600 IN A 128.232.0.20
www.cl.cam.ac.uk. 600 IN CAA 0 issuewild ";"
www.cl.cam.ac.uk. 600 IN CAA 0 issue "quovadisglobal.com"
www.cl.cam.ac.uk. 600 IN CAA 0 issue "letsencrypt.org"
www.cl.cam.ac.uk. 600 IN RRSIG CAA 5 5 600 20230324004125 20230222002906 31575 cl.cam.ac.uk.
A930aBg5uKP2l2aYxJ1gbCSnbR/o8n8oOs54fBOSUOkE55YmQWRNkNEW AGuuJltIz0I/lJ9eH4Jf+VL7KO1AimzS2ae6GXnXogP3shaz16jh+psX
rRQhKa2S0LcfRJM2j3ltct88AewpLk4nrv5rlvCS2yumGQlvKaMuEaga R14=

Received 1858 bytes from 2a05:b400:110::d:0#53 in 2 ms
```


Why is the web so successful?

- What do the web, youtube, facebook, twitter, instagram, ….. have in common?
  - The ability to self-publish
- Self-publishing that is easy, independent, *free*
- No interest in collaborative and idealistic endeavors
  - People aren't looking for Nirvana (or even Xanadu)
  - People also aren't looking for technical perfection
- Want to make their mark, and find something neat
  - Two sides of the same coin, creates synergy
  - "Performance" more important than dialogue….

Web and HTTP

First, a quick review…

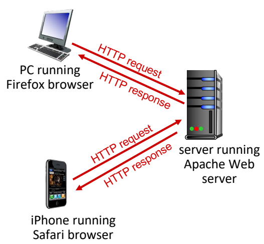
- web page consists of *objects,* each of which can be stored on different Web servers
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects, each* addressable by a *URL,* e.g.,

$$\underbrace{\texttt{www.university.ac.uk}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

HTTP overview

HTTP: hypertext transfer protocol

- Web's application-layer protocol
- client/server model:
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests



HTTP overview (continued)

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

Reminder: Distributed Systems are Hard!

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

HTTP connections: two types

Non-persistent HTTP

1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection closed

downloading multiple objects required multiple connections

Persistent HTTP

- TCP connection opened to a server
- multiple objects can be sent over *single* TCP connection between client, and that server
- TCP connection closed

Non-persistent HTTP: example

User enters URL: **`www.university.ac.uk/someDepartment/home.index`**
(containing text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.university.ac.uk on port 80

1b. HTTP server at host www.university.ac.uk waiting for TCP connection at port 80 "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Non-persistent HTTP: example (cont.)

User enters URL: **`www.university.ac.uk/someDepartment/home.index`**
(containing text, references to 10 jpeg images)

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

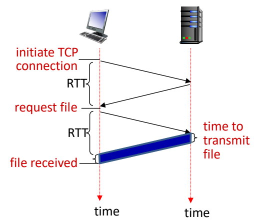
time

Non-persistent HTTP: response time

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

HTTP response time (per object):

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- object/file transmission time



Non-persistent HTTP response time = 2RTT+ file transmission time

Persistent HTTP (HTTP 1.1)

Non-persistent HTTP issues:

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

Persistent HTTP (HTTP1.1):

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

HTTP request message

- two types of HTTP messages: *request*, *response*
- HTTP request message:
  - ASCII (human-readable format)

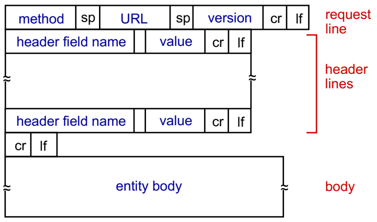
carriage return character
line-feed character

request line (GET, POST, HEAD commands)

carriage return, line feed at start of line indicates end of header lines

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

HTTP request message: general format



Other HTTP request messages

POST method:
- web page often includes form input
- user input sent from client to server in entity body of HTTP POST request message

GET method (for sending data to server):
- include user data in URL field of HTTP GET request message (following a '?'):

`www.somesite.com/animalsearch?monkeys&banana`

HEAD method:
- requests headers (only) that would be returned *if* specified URL were requested with an HTTP GET method.

PUT method:
- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body of POST HTTP request message

HTTP response message

status line (protocol status code status phrase) → `HTTP/1.1 200 OK`

HTTP response Status Codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

200 OK
- request succeeded, requested object later in this message

301 Moved Permanently
- requested object moved, new location specified later in this message (in Location: field)

400 Bad Request
- request msg not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported

Trying out HTTP (client side) for yourself

1. Netcat (telnet will also work) to your favorite Web server:

% nc -c -v `www.cl.cam.ac.uk` 80

- opens TCP connection to port 80 (default HTTP server port) at `www.cl.cam.ac.uk` anything typed in will be sent to port 80 at `www.cl.cam.ac.uk`

2. type in a GET HTTP request:

```
GET /~awm22/index.php HTTP/1.1
Host: www.cl.cam.ac.uk
```

- by typing this in (hit carriage return twice), you send a minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

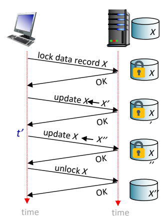
Although in readable ascii – you will notice this is not the webpage but a redirect Automatically moving to an https secure connection

Maintaining user/server state: cookies

Recall: HTTP GET/response interaction is *stateless*

- no notion of multi-step exchanges of HTTP messages to complete a Web "transaction"
  - no need for client/server to track "state" of multi-step exchange
  - all HTTP requests are independent of each other
  - no need for client/server to "recover" from a partial-but-never-entirely-completed transaction

a stateful protocol: client makes two changes to X, or none at all



Q: what happens if network connection or client crashes at *t'* ?

Maintaining user/server state: cookies

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

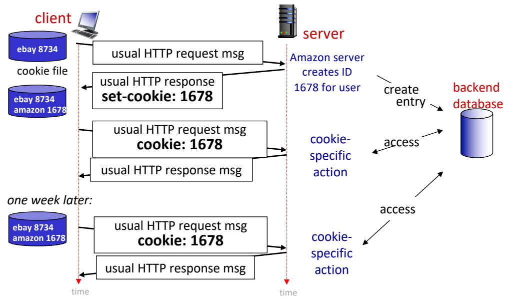
four components:

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

Maintaining user/server state: cookies



HTTP cookies: comments

What cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

Challenge: How to keep state?

- *at protocol endpoints:* maintain state at sender/receiver over multiple transactions
- *in messages:* cookies in HTTP messages carry state

aside

cookies and privacy:

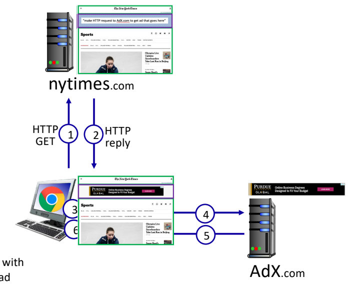
- cookies permit sites to *learn* a lot about you on their site.
- third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites

Example: displaying a NY Times web page

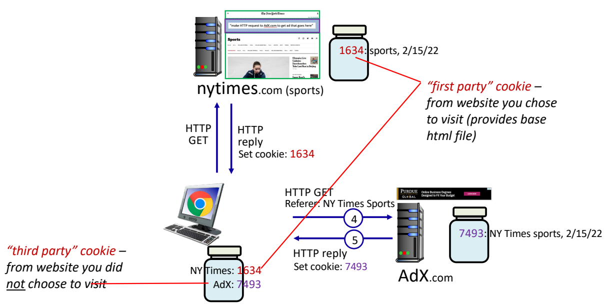
1 2 GET base html file from nytimes.com

4 5 fetch ad from AdX.com

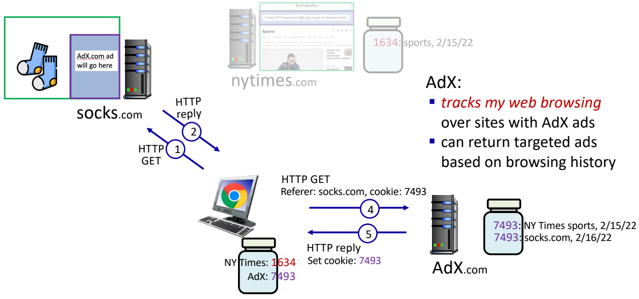
7 display composed page



Cookies: tracking a user's browsing behavior



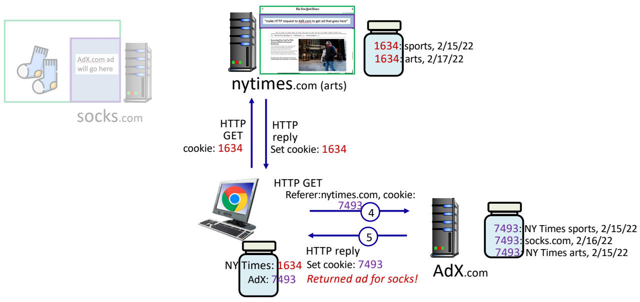
Cookies: tracking a user's browsing behavior



AdX:

- *tracks my web browsing* over sites with AdX ads
- can return targeted ads based on browsing history

Cookies: tracking a user's browsing behavior (one day later)



Cookies: tracking a user's browsing behavior

Cookies can be used to:

- track user behavior on a given website (first party cookies)
- track user behavior across multiple websites (third party cookies) without user ever choosing to visit tracker site (!)
- tracking may be *invisible* to user:
  - –rather than displayed ad triggering HTTP GET to tracker, could be an invisible link

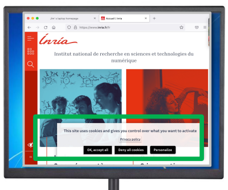
third party tracking via cookies:

- disabled by default in Firefox, Safari browsers
- to be disabled in Chrome browser in 2023

GDPR (EU General Data Protection Regulation) and cookies

> "Natural persons may be associated with online identifiers […] such as internet protocol addresses, cookie identifiers or other identifiers […].
> This may leave traces which, in particular when combined with unique identifiers and other information received by the servers, may be used to create profiles of the natural persons and identify them."
>
> GDPR, recital 30 (May 2018)

when cookies can identify an individual, cookies are considered personal data, subject to GDPR personal data regulations

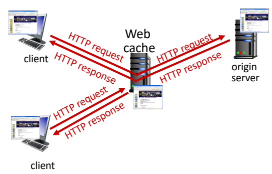


User has explicit control over whether or not cookies are allowed

Web caches

Goal: satisfy client requests without involving origin server

- user configures browser to point to a (local) *Web cache*
- browser sends all HTTP requests to cache
  - *if* object in cache: cache returns object to client
  - *else* cache requests object from origin server, caches received object, then returns object to client



Web caches (aka proxy servers)

- Web cache acts as both client and server
  - server for original requesting client
  - client to origin server
- server tells cache about object's allowable caching in response header:

```
Cache-Control: max-age=<seconds>
```

```
Cache-Control: no-cache
```

Why Web caching?

- reduce response time for client request
  - cache is closer to client
- reduce traffic on an institution's access link
- Internet is dense with caches
  - enables "poor" content providers to more effectively deliver content

Caching example

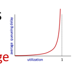
Scenario:

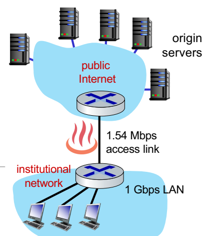
- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Performance:

- access link utilization = .97
- LAN utilization: .0015
- end-end delay = Internet delay + access link delay + LAN delay = 2 sec + minutes + usecs

problem: large queueing delays at high utilization!



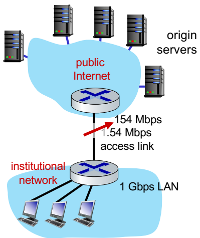


Option 1: buy a faster access link

Scenario:
- access link rate: ~~1.54~~ 154 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Performance:
- access link utilization = ~~.97~~ .0097
- LAN utilization: .0015
- end-end delay = Internet delay + access link delay + LAN delay
  = 2 sec + ~~minutes~~ msecs + usecs

Cost: faster access link (expensive!)



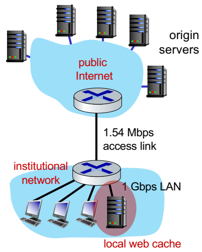
Option 2: install a web cache

Scenario:
- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Cost: web cache (cheap!)

Performance:
- LAN utilization: .?
- access link utilization = ?
- average end-end delay = ?

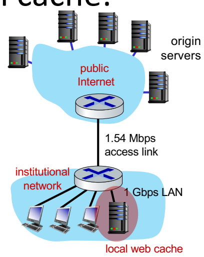
How to compute link utilization, delay?



Calculating access link utilization, end-end delay with cache:

suppose cache hit rate is 0.4:
- 40% requests served by cache, with low (msec) delay
- 60% requests satisfied at origin
  - rate to browsers over access link = 0.6 * 1.50 Mbps = .9 Mbps
  - access link utilization = 0.9/1.54 = .58 means low (msec) queueing delay at access link
- average end-end delay:
  = 0.6 * (delay from origin servers)
  + 0.4 * (delay when satisfied at cache)
  = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs

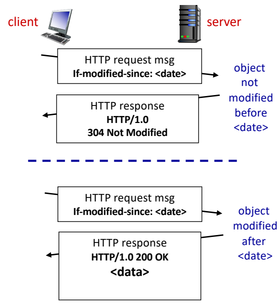
lower average end-end delay than with 154 Mbps link (and cheaper too!)



Browser caching: Conditional GET

Goal: don't send object if browser has up-to-date cached version
- no object transmission delay (or use of network resources)

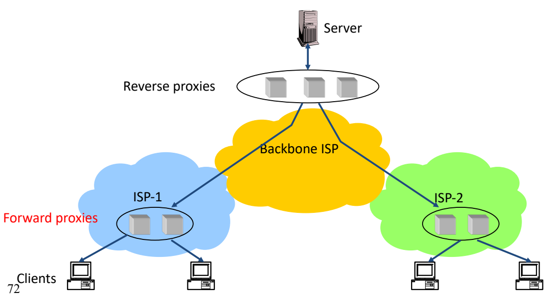
- *client:* specify date of browser-cached copy in HTTP request
  If-modified-since: <date>
- *server:* response contains no object if browser-cached copy is up-to-date:
  HTTP/1.0 304 Not Modified



Improving HTTP Performance:
Caching with Forward Proxies

Cache documents close to **clients**
→ reduce network traffic and decrease latency

- Typically done by ISPs or corporate LANs to reduce link usage

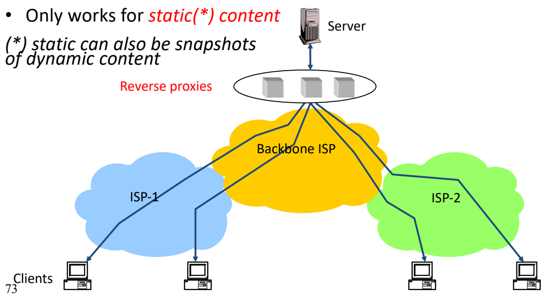


Improving HTTP Performance:
Caching with Reverse Proxies

Cache documents close to **server**
→ decrease server load

- Typically done by content providers (e.g. scaling capacity for news site)
- Only works for *static(*) content*

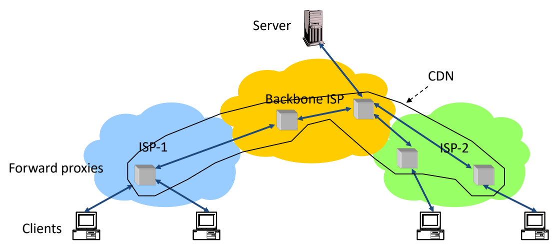
(*) static can also be snapshots of dynamic content



Improving HTTP Performance:
Caching w/ Content Distribution Networks

- Integrate forward and reverse caching functionality
  - One overlay network (usually) administered by one entity
  - *e.g.,* Akamai
- Provide document caching
  - **Pull:** Direct result of clients' requests
  - **Push:** Expectation of high access rate
- Also do some processing
  - Handle *dynamic* web pages
  - *Transcoding*
  - *Maybe do some security function – watermark IP*

Improving HTTP Performance:
Caching with CDNs (cont.)



Improving HTTP Performance:
CDN Example – Akamai

- Akamai creates new domain names for each client content provider.
  - e.g., *a128.g.akamai.net*
- The CDN's DNS servers are authoritative for the new domains
- The client content provider modifies its content so that embedded URLs reference the new domains.
  - "Akamaize" content
  - e.g.: *http://www.bbc.co.uk/popular-image.jpg* becomes *http://a128.g.akamai.net/popular-image.jpg*
- *Requests now sent to CDN's infrastructure…*

Hosting: Multiple Sites Per Machine

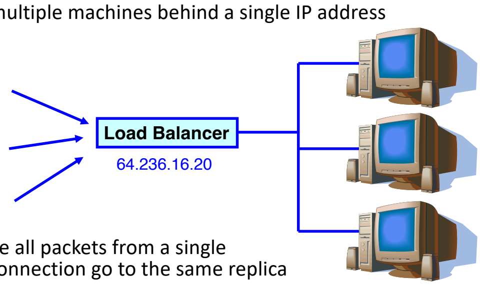
- Multiple Web sites on a single machine
  - Hosting company runs the Web server on behalf of multiple sites (*e.g.*, www.foo.com and www.bar.com)
- Problem: `GET /index.html`
  - `www.foo.com/index.html` or `www.bar.com/index.html`?
- Solutions:
  - Multiple server processes on the same machine
    - Have a separate IP address (or port) for each server
  - Include site name in HTTP request
    - Single Web server process with a single IP address
    - Client includes "Host" header (*e.g.*, `Host: www.foo.com`)
    - *Required header* with HTTP/1.1

Hosting: Multiple Machines Per Site

- Replicate popular Web site across many machines
  - Helps to handle the load
  - Places content closer to clients
- Helps when content isn't cacheable
- Problem: Want to direct client to particular replica
  - Balance load across server replicas
  - Pair clients with nearby servers

Multi-Hosting at Single Location

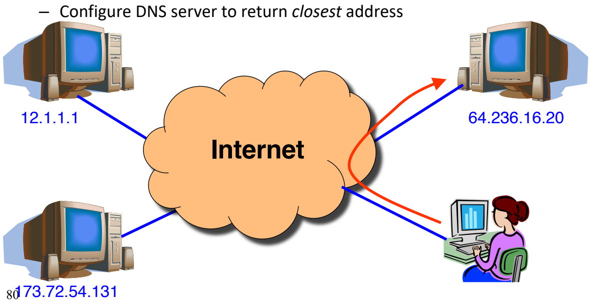
- Single IP address, multiple machines
  - Run multiple machines behind a single IP address



  - Ensure all packets from a single TCP connection go to the same replica

Multi-Hosting at Several Locations

- Multiple addresses, multiple machines
  - Same name but different addresses for all of the replicas
  - Configure DNS server to return *closest* address



CDN examples round-up

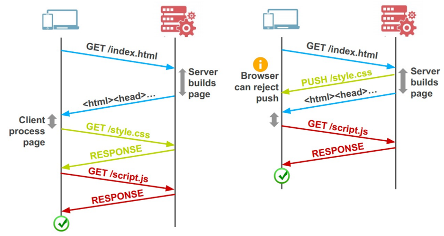
- CDN using DNS
  DNS has information on loading/distribution/location
  (akami uses this one)
- CDN using anycast
  same address from DNS name but local routes
  (ROOT DNS servers and 8.8.8.8 use this one)
- CDN based on rewriting HTML URLs
  (akami example in previous slides)

After HTTP/1.1

SPDY (speedy) and its moral successor HTTP/2

- Binary protocol
- Multiplexing
- Priority control over Frames
- Header Compression
- Server Push

After HTTP/1.1



- Server Push
  - Proactively push stuff to client that it will need

After HTTP/1.1

SPDY (speedy) and its moral successor HTTP/2

- Binary protocol
  - More efficient to parse
  - More compact on the wire
  - Much less error prone as compared
  - to textual protocols

Wireshark decoders for the win

HTTP/2

Key goal: decreased delay in multi-object HTTP requests

HTTP1.1: introduced multiple, pipelined GETs over single TCP connection

- server responds *in-order* (FCFS: first-come-first-served scheduling) to GET requests
- with FCFS, small object may have to wait for transmission (head-of-line (HOL) blocking) behind large object(s)
- loss recovery (retransmitting lost TCP segments) stalls object transmission

HTTP/2

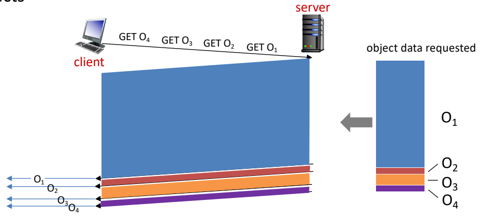
Key goal: decreased delay in multi-object HTTP requests

HTTP/2: [RFC 7540, 2015] increased flexibility at *server* in sending objects to client:

- methods, status codes, most header fields unchanged from HTTP 1.1
- transmission order of requested objects based on client-specified object priority (not necessarily FCFS)
- *push* unrequested objects to client
- divide objects into frames, schedule frames to mitigate HOL blocking

HTTP/2: mitigating HOL blocking

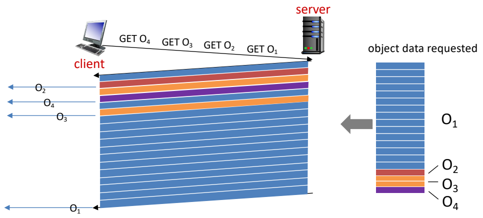
HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



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

HTTP/2: mitigating HOL blocking

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



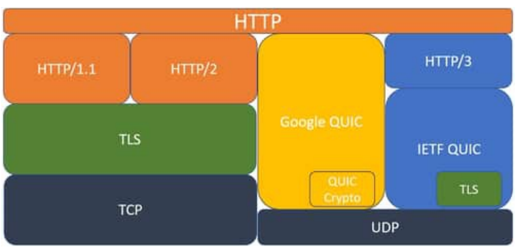
O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

HTTP/2 to HTTP/3

HTTP/2 over single TCP connection means:

- recovery from packet loss still stalls all object transmissions
  - as in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling, increase overall throughput
- no security over vanilla TCP connection
- HTTP/3: adds security, per object error- and congestion-control (more pipelining) over UDP

As at ~~2021~~ when I last looked



Other ongoing work includes QUIC for datagrams
Seriously! It adds QUIC crypto to "UDP" so isn't totally silly.

Add QUIC and stir…
Quick UDP Internet Connections

Objective: Combine speed of UDP protocol with TCP's reliability

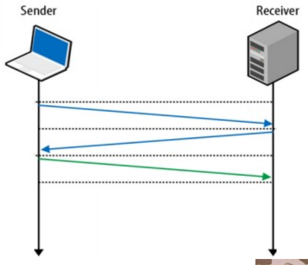
Problem: Very hard to make changes to TCP

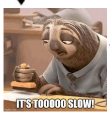
- *Faster to implement new protocol on top of UDP*
- (Roll out features in TCP if they prove theory)

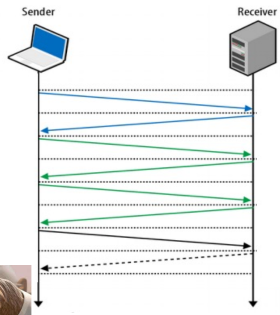
QUIC (First presented to IETF in ~2013):

- Reliable transport over UDP
- Uses FEC
- Default crypto
- Restartable connections

3-Way Handshake





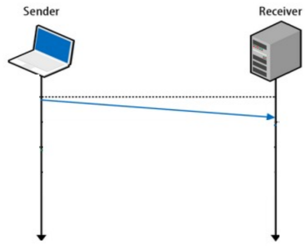


Without TLS

With TLS

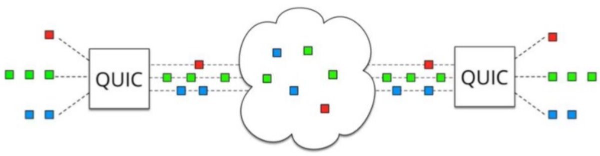
UDP

- Fire and forget
  - Less time spent to validate packets
  - Downside - no reliability, this has to be added on top of UDP



QUIC

- UDP does NOT depend on order of arriving packets
- Lost packets will only impact an individual resource, e.g., CSS or JS file.
- QUIC combined the best parts of HTTP/2 over UDP:
  - Multiplexing on top of non-blocking transport protocol



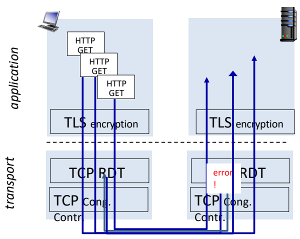
QUIC: Quick UDP Internet Connections

adopts approaches we've studied in this topic for connection establishment, error control, congestion control

- **error and congestion control:** "Readers familiar with TCP's loss detection and congestion control will find algorithms here that parallel well-known TCP ones." [from QUIC specification]
- **connection establishment:** reliability, congestion control, authentication, encryption, state established in one RTT

- multiple application-level "streams" multiplexed over single QUIC connection
  - separate reliable data transfer, security
  - common congestion control

QUIC: streams – parallelism no HOL blocking in transport or application



(a) HTTP 1.1

QUIC – more than just UDP

- QUIC outshines TCP under poor network conditions, shaving a full second off the Google Search page load time for the slowest 1% of connections.

- These benefits are even more apparent for video services like YouTube
  - Users report 30% fewer rebuffers with QUIC.

Why QUIC over UDP and not a new proto

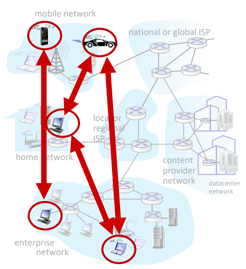
- IP proto value for new transport layer
- Change the protocol – risk the wraith of
  - Legacy code
  - Firewalls
  - Load-balancer
  - NATs (the high-priest of middlebox)
- Same problem faces any significant TCP change

Honda M. et al. **"Is it still possible to extend TCP?",** IMC'11
https://dl.acm.org/doi/abs/10.1145/2068816.2068834

Every host is a server: Peer-2-Peer

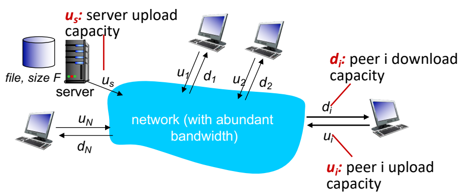
Peer-to-peer (P2P) architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, and new service demands
- peers are intermittently connected and change IP addresses
  - complex management
- examples: P2P file sharing (BitTorrent), streaming (KanKan), VoIP (Skype)



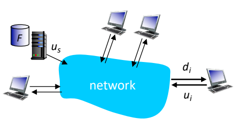
File distribution: client-server vs P2P

Q: how much time to distribute file (size *F*) from one server to *N peers*?
- peer upload/download capacity is limited resource



File distribution time: client-server

- *server transmission:* must sequentially send (upload) *N* file copies:
  - time to send one copy: F/u_s
  - time to send *N* copies: NF/u_s
- *client:* each client must download file copy
  - d_{min} = min client download rate
  - min client download time: F/d_{min}



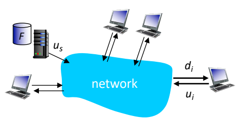
time to distribute F to N clients using client-server approach

$$D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$$

increases linearly in N

File distribution time: P2P

- *server transmission:* must upload at least one copy:
  - time to send one copy: F/u_s
- *client:* each client must download file copy
  - min client download time: F/d_{min}
- *clients:* as aggregate must download *NF* bits
  - max upload rate (limiting max download rate) is $u_s + \Sigma u_i$



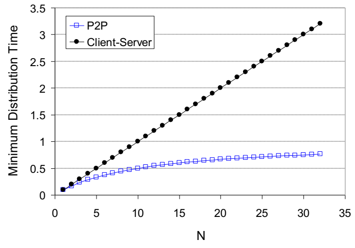
time to distribute F to N clients using P2P approach

$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

increases linearly in *N* …
… but so does this, as each peer brings service capacity

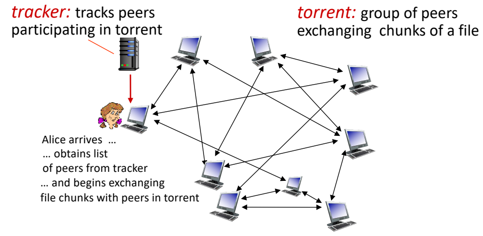
Client-server vs. P2P: example

client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$

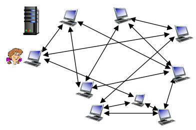


P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks



P2P file distribution: BitTorrent



- 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
- peer exchanges prioritize rarer blocks
- *churn:* peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

BitTorrent: requesting, sending file chunks

Requesting chunks:

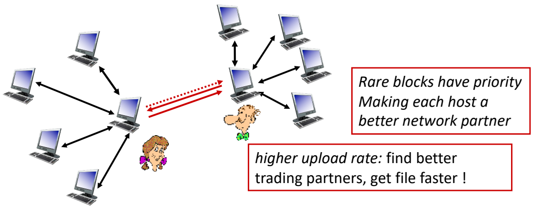
- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

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

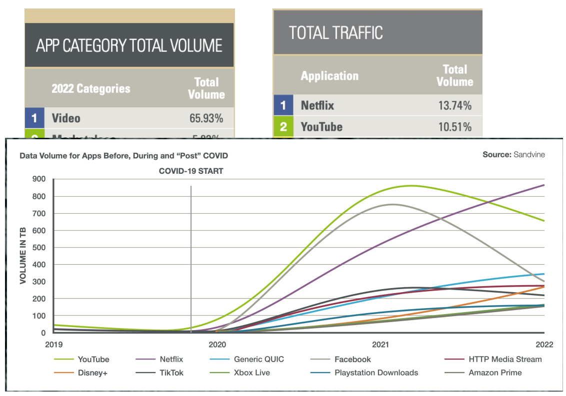
(1) Alice "optimistically unchokes" Bob
(2) Alice becomes one of Bob's top-four providers; Bob reciprocates
(3) Bob becomes one of Alice's top-four providers



Internet

(current data is $$$ or hard to get)

This info taken from an annual Sandvine report for 2022 https://www.sandvine.com

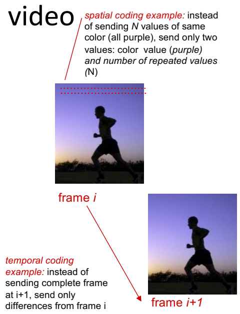


Video Streaming and CDNs: context

- stream video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube, Amazon Prime: 80% of residential ISP traffic (2020)
- *challenge:* scale - how to reach ~1B users?
- *challenge:* heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- *solution:* distributed, application-level infrastructure

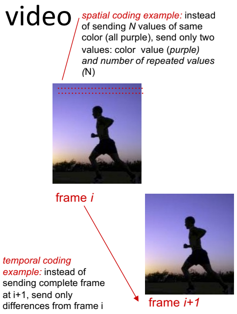
Multimedia: video

- video: sequence of images displayed at constant rate
  - e.g., 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits
- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)



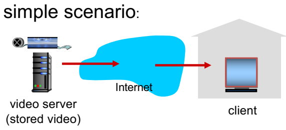
Multimedia: video

- CBR: (constant bit rate): video encoding rate fixed
- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
- examples:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, 64Kbps – 12 Mbps)



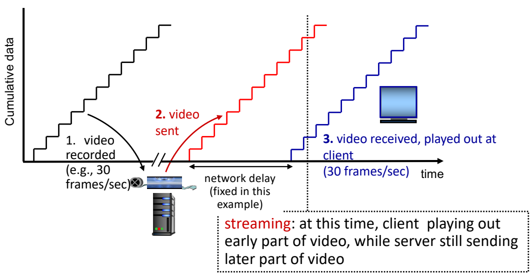
Streaming stored video

simple scenario:



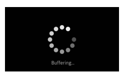
Main challenges:
- server-to-client bandwidth will *vary* over time, with changing network congestion levels (in house, access network, network core, video server)
- packet loss, delay due to congestion will delay playout, or result in poor video quality

Streaming stored video

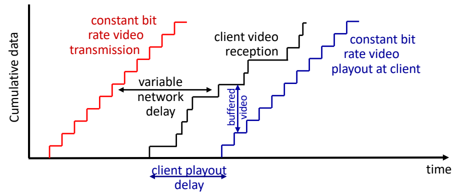


Streaming stored video: challenges

- continuous playout constraint: during client video playout, playout timing must match original timing
  - … but network delays are variable (jitter), so will need client-side buffer to match continuous playout constraint
- other challenges:
  - client interactivity: pause, fast-forward, rewind, jump through video
  - video packets may be lost, retransmitted



Streaming stored video: playout buffering

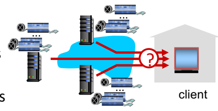


- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Streaming multimedia: DASH

*D*ynamic, *A*daptive
*S*treaming over *H*TTP

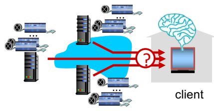
server:
- divides video file into multiple chunks
- each chunk encoded at multiple different rates
- different rate encodings stored in different files
- files replicated in various CDN nodes
- *manifest file:* provides URLs for different chunks



client:
- periodically estimates server-to-client bandwidth
- consulting manifest, requests one chunk at a time
  - chooses maximum coding rate sustainable given current bandwidth
  - can choose different coding rates at different points in time (depending on available bandwidth at time), and from different servers

Streaming multimedia: DASH

- *"intelligence"* at client: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is "close" to client or has high available bandwidth)



Streaming video = encoding + DASH + playout buffering

Content distribution networks (CDNs)

challenge: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?

- *option 1:* single, large "mega-server"
  - single point of failure
  - point of network congestion
  - long (and possibly congested) path to distant clients

....quite simply: this solution *doesn't scale*

Content distribution networks (CDNs)

challenge: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?

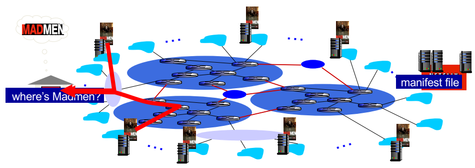
- *option 2:* store/serve multiple copies of videos at multiple geographically distributed sites *(CDN)*
  - *enter deep:* push CDN servers deep into many access networks
    - close to users
    - Akamai: 240,000 servers deployed in > 120 countries (2015)
  - *bring home:* smaller number (10's) of larger clusters in POPs near access nets
    - used by Limelight

Akamai

Limelight NETWORKS

Content distribution networks (CDNs)

- CDN: stores copies of content (e.g. MADMEN) at CDN nodes
- subscriber requests content, service provider returns manifest
  - using manifest, client retrieves content at highest supportable rate
  - may choose different rate or copy if network path congested



Content distribution networks (CDNs)

OTT challenges: coping with a congested Internet from the “edge”
- what content to place in which CDN node?
- from which CDN node to retrieve content? At which rate?

Summary

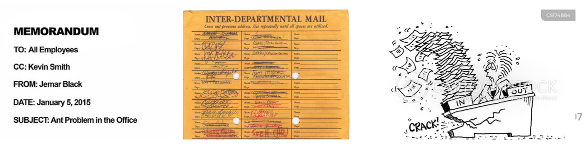
- Applications have protocols too
- We covered examples from
  - Traditional Applications (web)
  - Scaling and Speeding the web (CDN/Cache tricks)
- Infrastructure Services (DNS)
  - Cache and Hierarchy
- P2P Network examples
- Evolving standards (Email)
- Video CDN Stream challenges

Email

Still the best/worst most useful/useless service

Email was the exemplar of the *Electronic Office*

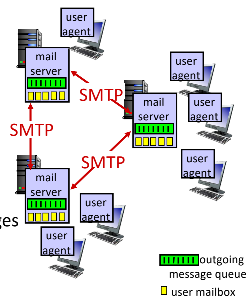
Because every business thought in *memo*



E-mail

Three major components:
- user agents
- mail servers
- simple mail transfer protocol: SMTP

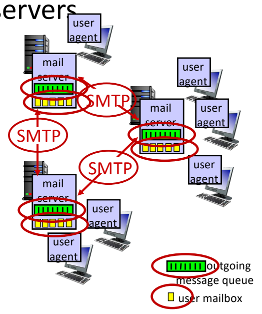
User Agent
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Outlook, iPhone mail client
- outgoing, incoming messages stored on server



E-mail: mail servers

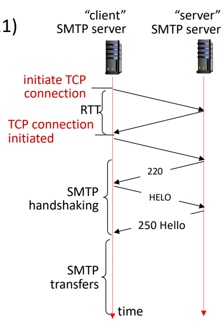
mail servers:
- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages

SMTP protocol between mail servers to send email messages
- client: sending mail server
- “server”: receiving mail server



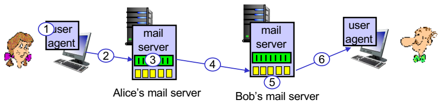
SMTP RFC (5321)

- uses TCP to reliably transfer email message from client (mail server initiating connection) to server, port 25
  - direct transfer: sending server (acting like client) to receiving server
- three phases of transfer
  - SMTP handshaking (greeting)
  - SMTP transfer of messages
  - SMTP closure
- command/response interaction (like HTTP)
  - commands: ASCII text
  - response: status code and phrase



Scenario: Alice sends e-mail to Bob

1) Alice uses UA to compose e-mail message "to" bob@someschool.edu
2) Alice's UA sends message to her mail server using SMTP; message placed in message queue
3) client side of SMTP at mail server opens TCP connection with Bob's mail server
4) SMTP client sends Alice's message over the TCP connection
5) Bob's mail server places the message in Bob's mailbox
6) Bob invokes his user agent to read message



Sample SMTP interaction

```
S: 220 hamburger.edu
```

SMTP: observations

comparison with HTTP:

- HTTP: client pull
- SMTP: client push
- 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

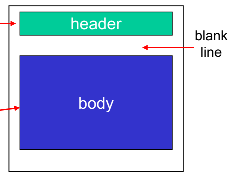
Mail message format

SMTP: protocol for exchanging e-mail messages, defined in RFC 5321 (like RFC 7231 defines HTTP)

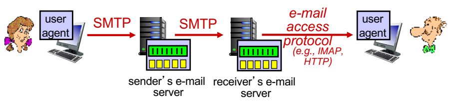
RFC 2822 defines *syntax* for e-mail message itself (like HTML defines syntax for web documents)

- header lines, e.g.,
  - To:
  - From:
  - Subject:

  these lines, within the body of the email message area different from ~~SMTP MAIL FROM:, RCPT TO:~~ commands!
- Body: the "message", ASCII characters only



Retrieving email: mail access protocols



- SMTP: delivery/storage of e-mail messages to receiver's server
- mail access protocol: retrieval from server
  - IMAP: Internet Mail Access Protocol [RFC 3501]: messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server
- HTTP: gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of STMP (to send), IMAP (or POP) to retrieve e-mail messages