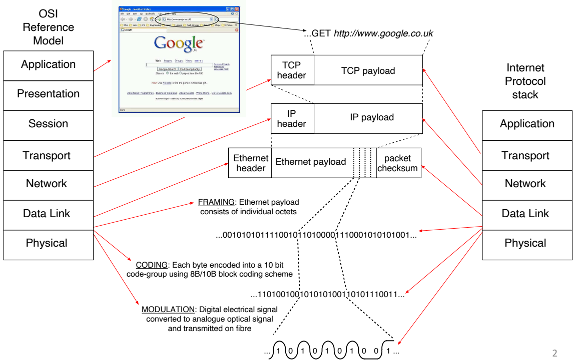
Computer Networking

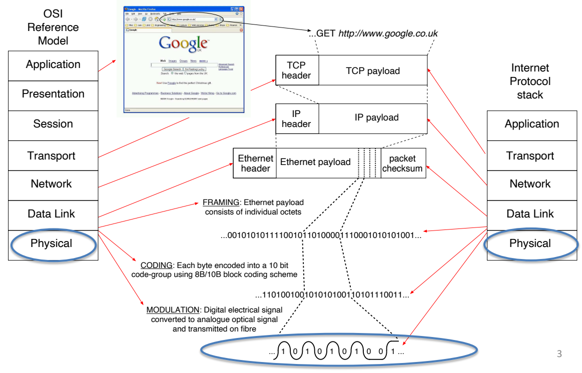
Slide Set 3

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

Internet protocol stack *versus* OSI Reference Model



Internet protocol stack *versus* OSI Reference Model



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. Aynchronous

Physical Channels / The Physical Layer

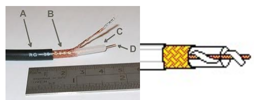
these example physical channels are also known as *Physical Media*

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:

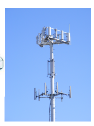
- high-speed operation
- point-to-point transmission
- (10's-100's Gbps)
- low error rate
- immune to electromagnetic noise

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.

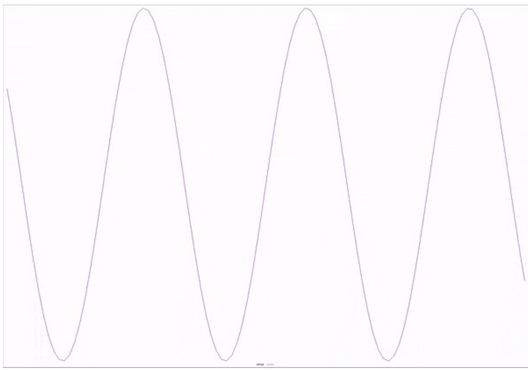
capacity: bandwidth

delay: speed of light in medium and distance travelled

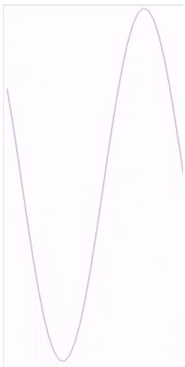
fidelity: signal to noise ratio

- 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

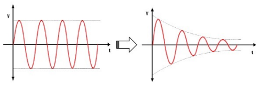
Noise may be systematic or random

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

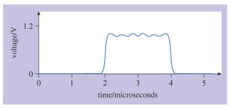
Random noise caused by thermal vibration (thermal noise)

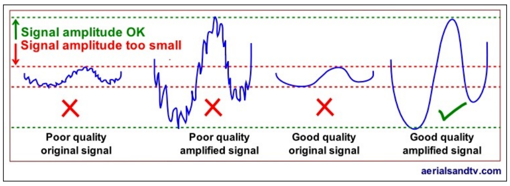
"White" noise is evenly distributed across frequencies
signal to noise ratio *S/N*
more distance more noise

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?

- for channels with 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 infinite information capacity
- 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

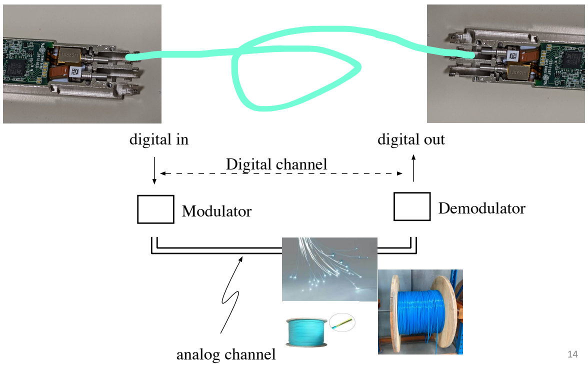
- Physical layer provides a channel
- Fixed rate for now
- Symbols are discrete values sent on the channel at fixed rate
- Symbols need not be binary
- Fidelity of the channel usually measured as a bit error rate — the probability that a bit sent as a 1 was interpreted as a 0 by the receiver or vice 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

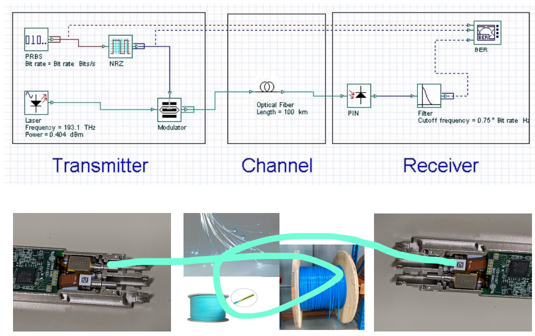
Two definitions:

- 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).

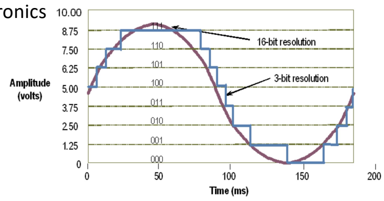


Communications



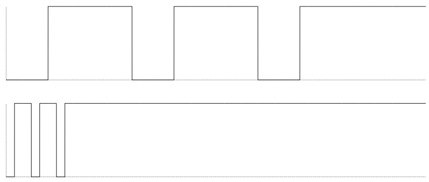
Analog/Digital Digital/Analog

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

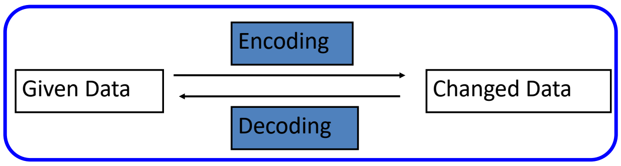
- Transmission is sporadic, divided into frames
- Receiver and transmitter have oscillators which are close in frequency producing tx clocks and rx clock
- Receiver synchronises the phase of the rx clock with the tx clock by looking at one or more bit transitions
- RX clock drifts with respect to the 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

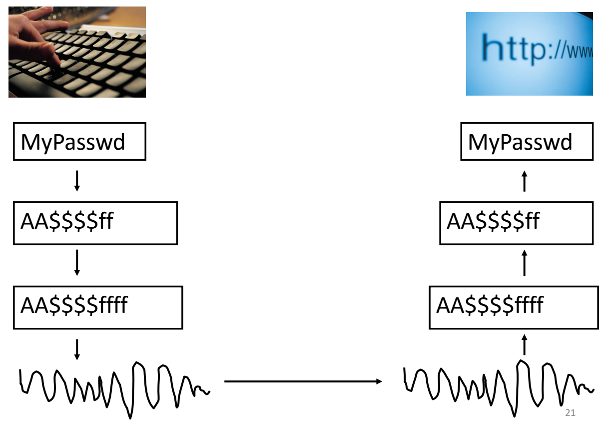
- Transmission is continuous
- Receiver continually adjusts its frequency to track clock from incoming signal
- Requires bit transitions to inform clock
- Phase locked loop: rx clock predicts when incoming clock will change and corrects slightly when wrong.

Bit transitions are critical

Coding – a channel function

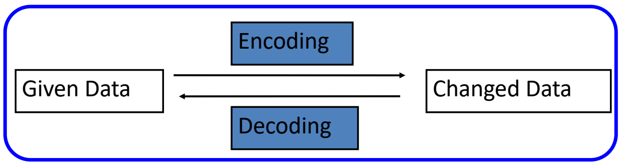
Change the representation of data.

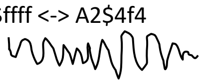




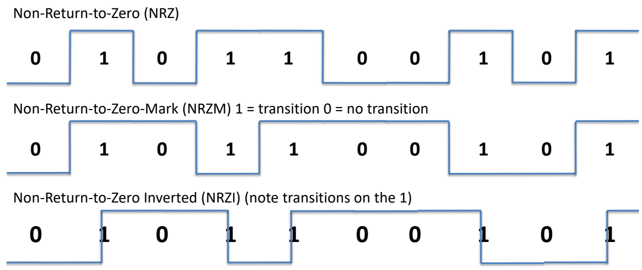
Coding

Change the representation of data.

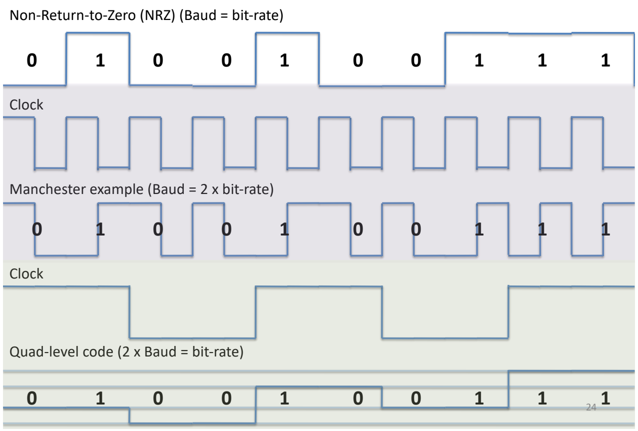


1. Encryption: MyPasswd <-> AA$$$$ff
2. Error Detection: AA$$$$ff <-> AA$$$$ffff
3. Compression: AA$$$$ffff <-> A2$4f4
4. Analog: A2$4f4 <-> 

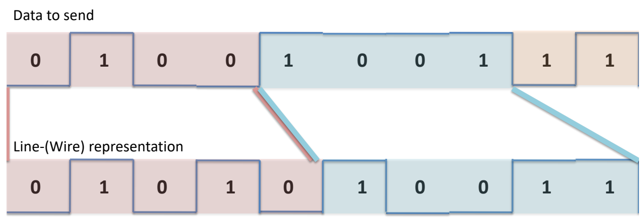
Line Coding Examples
where Baud=bit-rate



Line Coding Examples



Line Coding – Block Code example



Name	4b	5b	Description
0	0000	11110	hex data 0
1	0001	01001	hex data 1
2	0010	10100	hex data 2
3	0011	10101	hex data 3
4	0100	01010	hex data 4
5	0101	01011	hex data 5
6	0110	01110	hex data 6
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

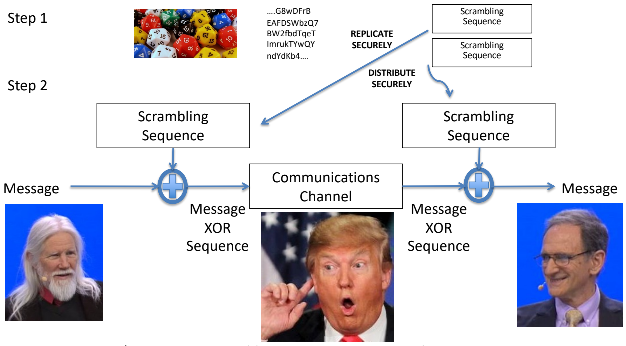
Name	4b	5b	Description
Q	-NONE-	00000	Quiet
I	-NONE-	11111	Idle
J	-NONE-	11000	SSD #1
K	-NONE-	10001	SSD #2
T	-NONE-	01101	ESD #1
R	-NONE-	00111	ESD #2
H	-NONE-	00100	Halt

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

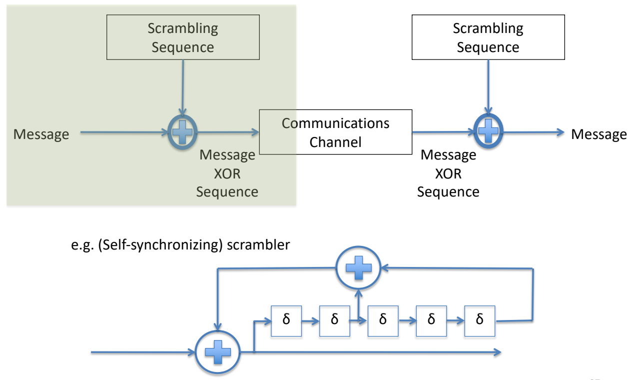


Step 3 Don't ever reuse Scrambling sequence, ever. <<< **this is quite important**

Whitfield Diffie

Martin Hellman

Line Coding Scrambling– no secrecy



Line Coding Examples (Hybrid)

...100111101101010001000101100111010001010010110101001001110101110100...

...10011110110101000101000101100111010001010010110101001001110101110100...

Inserted bits marking "start of frame/block/sequence"

Scramble / Transmit / Unscramble

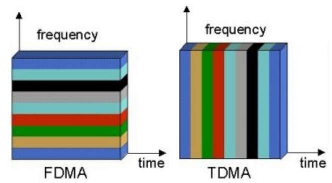
...0100010110011101000101001011010100100111010111010010010111011101111000...

Identify (and remove) "start of frame/block/sequence"

This gives you the Byte-delineations for *free*

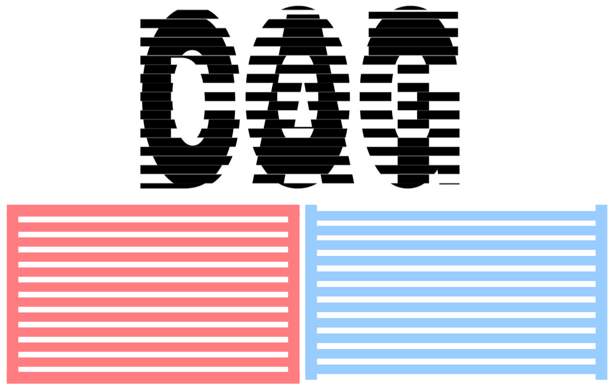
64b/66b combines a scrambler and a framer. The start of frame is a pair of bits 01 or 10: 01 means "this frame is data" 10 means "this frame contains data and control" – control could be configuration information, length of encoded data or simply "this line is idle" (no data at all)

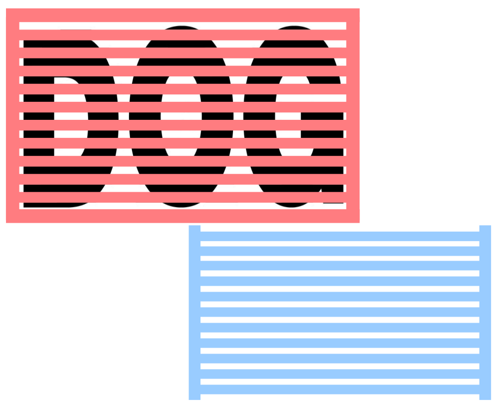
Multiple Access Mechanisms

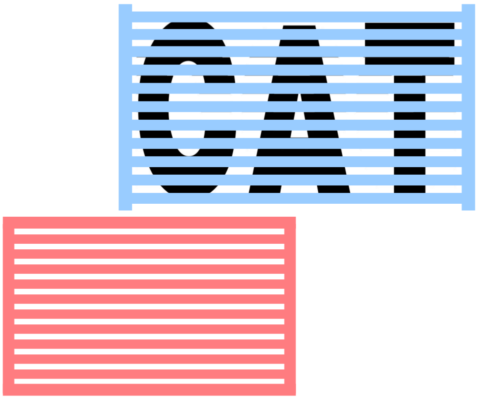


Each dimension is orthogonal (so may be trivially combined)

Other dimensions may also be available...



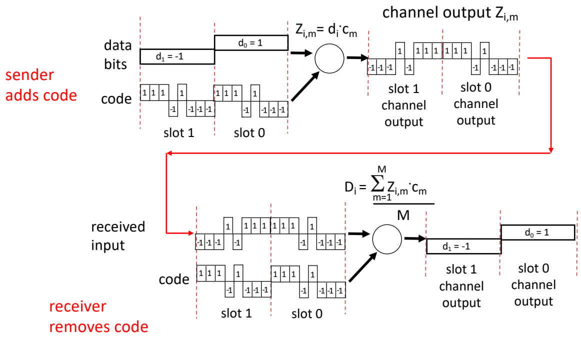




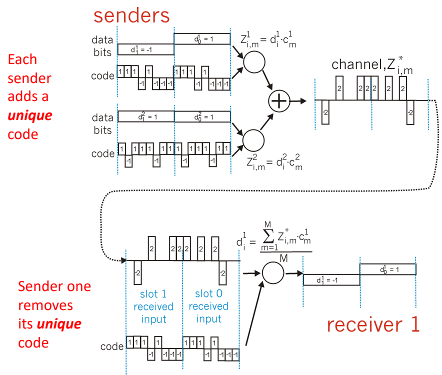
Code Division Multiple Access (CDMA)
(not to be confused with CSMA!)

- 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



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

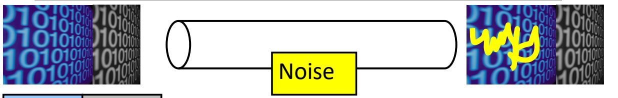
Transmission media are not perfect and cause signal impairments:

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?



0000 | 0000 → 0001 | 0000

Basic Idea :

1. Add additional information (redundancy) to a message.
2. Detect an error and discard

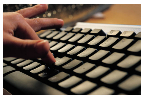
   Or, fix an error in the received message.

Coding – a channel function

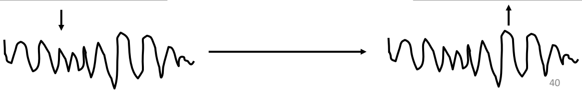
Change the representation of data.

Given Data → Encoding → Changed Data

Given Data ← Decoding ← Changed Data



MyPasswd → AA$$$$ff → AA$$$$ffff →



→ AA$$$$ffff → AA$$$$ff → MyPasswd

Coding Examples

Changig the representation of data.

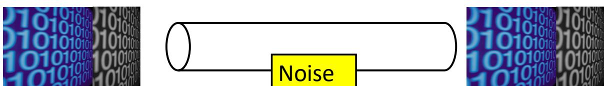
Given Data → Encoding → Changed Data

Given Data ← Decoding ← Changed 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.



Sent		Result	Received	
0000	0	X	0001	0
0001	1	✓	0001	1
1001	0	✓	1111	0

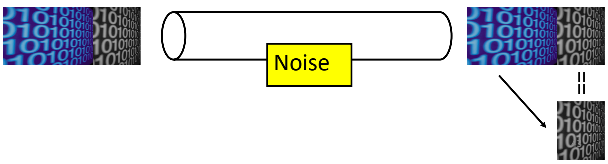
Problem: This simple parity cannot detect two-bit errors.

Error Detection Code

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

Receiver:

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



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

1111

1001

00

11

CRC

0

0

Parity

111

100

same check bits from Parity,
but different ones from CRC

Multiplication by 2^3
D2 = D * 2^3

Binary Division by 101
CheckBit = (D2) rem (101)

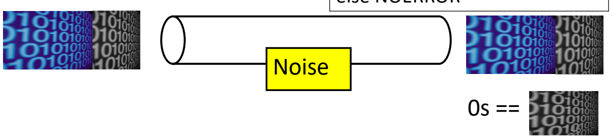
Add three 0's at the end

Kurose p478 §5.2.3
Peterson URL §2.4

Error Detection Code

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

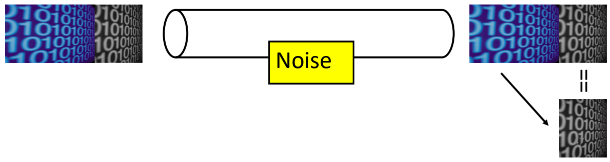
Receiver:
receive(X1);
receive(Y1);
Y2=generateCRC(X1Y1 div P);
if (Y2 != 0s) ERROR;
else NOERROR



Transforming Error Detection to...

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

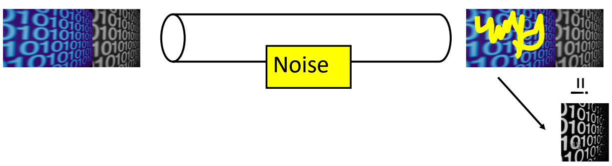
Receiver:
receive(X1Y1);
Y2=generateCheckBit(X1);
if (Y1 != Y2) ERROR;
else NOERROR



Forward Error Correction (FEC)

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

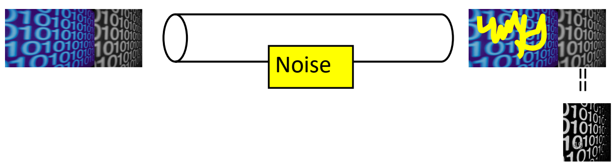
Receiver:
receive(X1Y1);
Y2=generateCheckBit(X1);
if (Y1 != Y2) FIXERROR(X1Y1);
else NOERROR



Forward Error Correction (FEC)

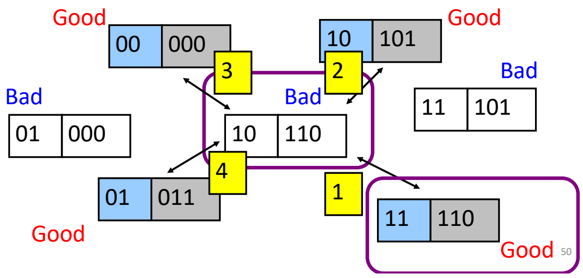
Sender:
Y = generateCheckBit(X);
send(XY);

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.

FEC: Kurose&Ross P618 §7.3.3
No Peterson&Davie reference

Topic 3: The Data Link Layer

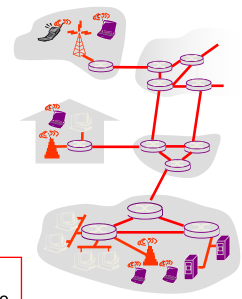
Our goals:

- 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

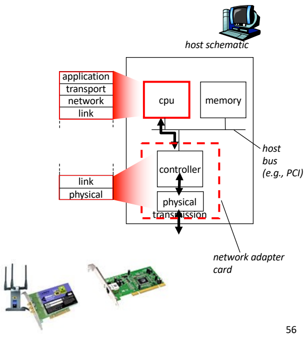
- *framing, physical addressing:*
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - "MAC" addresses used in frame headers to identify source, destination
    - This is **not** an IP address!
- *reliable delivery between adjacent nodes*
  - we revisit 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

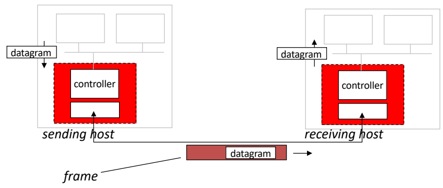
- *flow control:*
  - pacing between adjacent sending and receiving nodes
- *error control:*
  - *error detection*:
  - errors caused by signal attenuation, noise.
  - receiver detects presence of errors:
    - signals sender for retransmission or drops frame
  - error correction:
  - receiver identifies *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

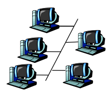


- sending side:
  - encapsulates datagram in frame
  - encodes data for the physical layer
  - adds error checking bits, provide reliability, flow control, etc.
- receiving side
  - decodes data from the physical layer
  - looks for errors, provide reliability, flow control, etc
  - extracts datagram,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

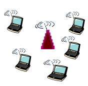
Multiple Access Links and Protocols

Two types of "links":

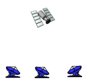
- point-to-point
  - point-to-point link between Ethernet switch and host
- broadcast (shared wire or medium)
  - old-fashioned wired Ethernet (*here be dinosaurs* – extinct)
  - upstream HFC (Hybrid Fiber-Coax – the Coax may be broadcast)
  - Home plug / Powerline networking
  - 802.11 wireless LAN



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

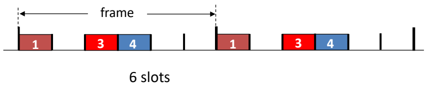
MAC Protocols: a taxonomy

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

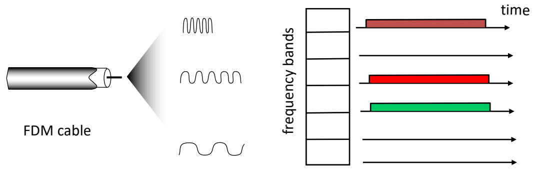
Channel Partitioning MAC protocols: TDMA
(we discussed this earlier)

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



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!

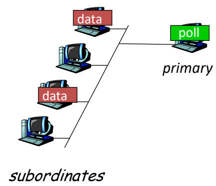
random access MAC protocols:
- efficient at low load: single node can fully utilize channel
- high load: collision overhead

"taking turns" protocols:

look for best of both worlds!

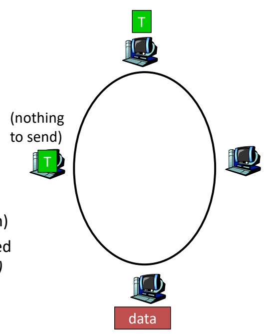
"Taking Turns" MAC protocols

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)

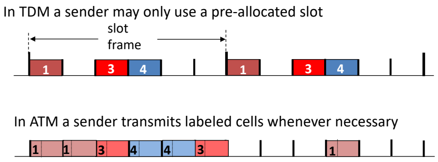


"Taking Turns" MAC protocols

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



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....

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

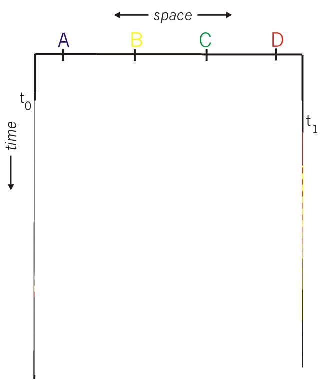
CSMA (Carrier Sense Multiple Access)

- 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?

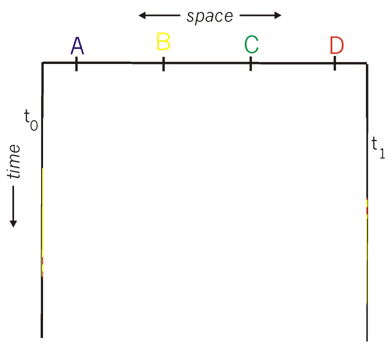
CSMA/CD (Collision Detection)

- 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)

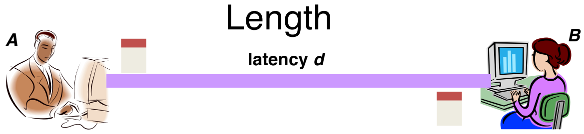
CSMA/CD Collision Detection

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***

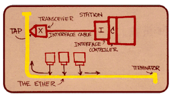
Performance of CSMA/CD

- 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: 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)**

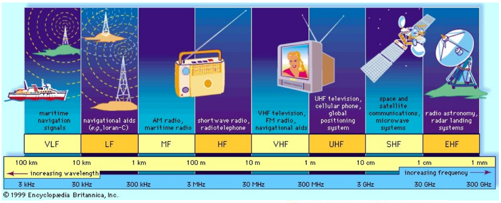
Benefits of Ethernet

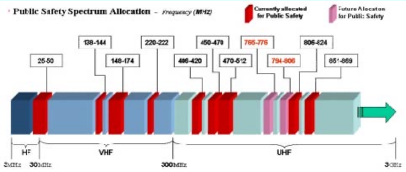
- Easy to administer and maintain
- Inexpensive
- Increasingly higher speed
- Evolvable!

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

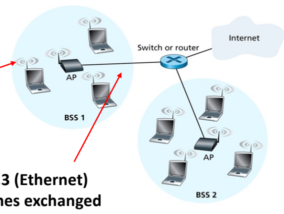
aka - WiFi …
What makes it special?

Deregulation > Innovation > Adoption > Lower cost = Ubiquitous technology

JUST LIKE ETHERNET – not lovely but sufficient

802.11 Architecture

802.11 frames exchanges



802.3 (Ethernet) frames exchanged

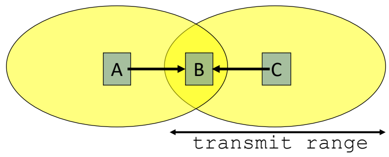
Figure 6.7 ♦ IEEE 802.11 LAN architecture

- 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?

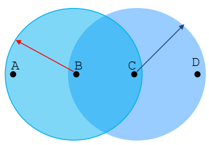
- Carrier Sense?
  - Sender can listen before sending
  - What does that tell the sender?
- Collision Detection?
  - Where do collisions occur?
  - How can you detect them?

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.

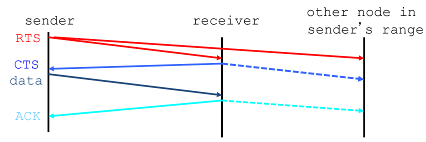
Key Points

- 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

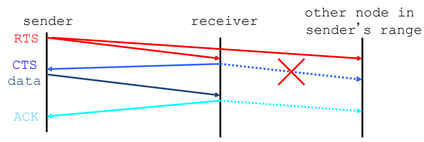
- Since can't detect collisions, we try to *avoid* them
- Carrier sense:
  - When medium busy, choose random interval
  - Wait that many **idle** timeslots to pass before sending
- When a collision is inferred, retransmit with binary exponential backoff (like Ethernet)
  - Use ACK from receiver to infer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

CSMA/CA -MA with Collision Avoidance



- 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

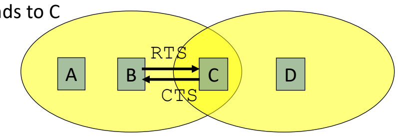
CSMA/CA, con't



- 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

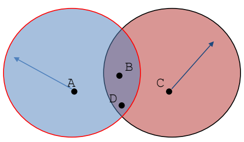


Overcome hidden terminal problems with contention-free protocol

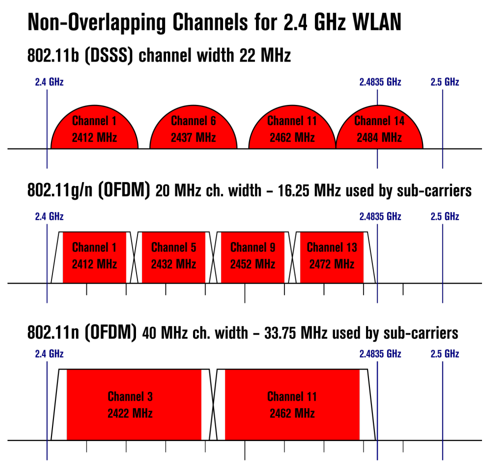
1. B sends to C Request To Send (RTS)
2. A hears RTS and defers (to allow C to answer)
3. C replies to B with Clear To Send (CTS)
4. D hears CTS and defers to allow the data
5. B sends to C

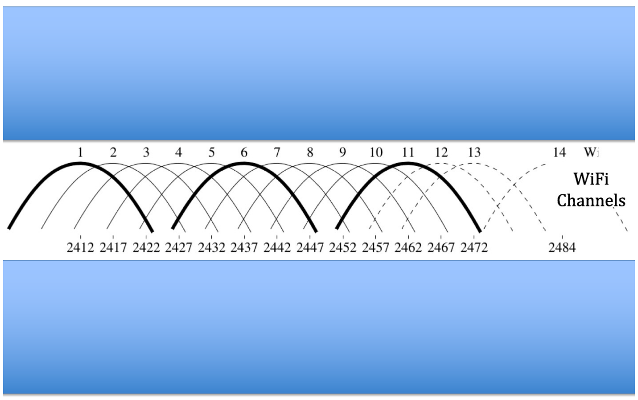
Preventing Collisions Altogether

- 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



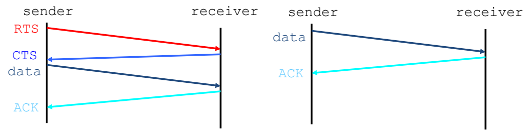


Wifi has been evolving!

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.

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
```

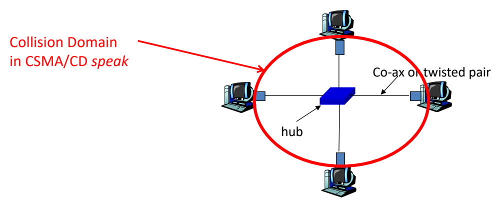

LAN Address (more)

- 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



CSMA in our home

Home Plug Powerline Networking….

Home Plug and similar Powerline Networking….

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

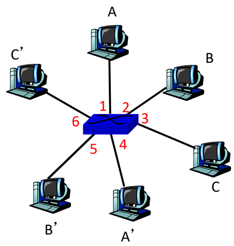
Switch (example: Ethernet Switch)

- 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

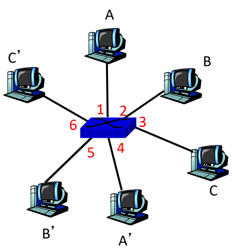
- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, but no collisions; full duplex
  - each link is its own collision domain
- *switching:* A-to-A' and B-to-B' simultaneously, without collisions
  - not possible with dumb hub



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

Switch Table

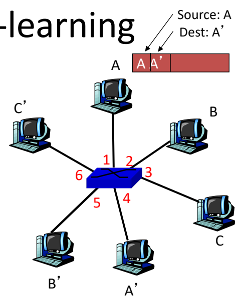
- Q: how does switch know that A' reachable via interface 4, B' reachable via interface 5?
- A: each switch has a switch table, each entry:
  - (MAC address of host, interface to reach host, time stamp)
- looks like a routing table!
- Q: how are entries created, maintained in switch table?
  - something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

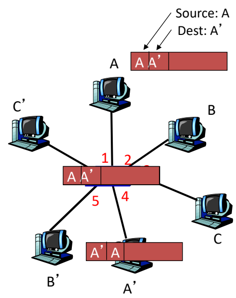
When frame received:

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

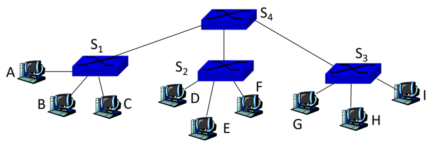


MAC addr	interface	TTL
A	*1*	*60*
A'	*4*	*60*

Switch table (initially empty)

Interconnecting switches

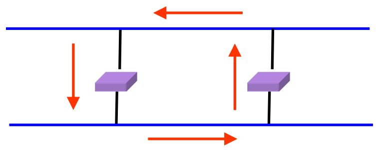
- switches can be connected together



- 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)

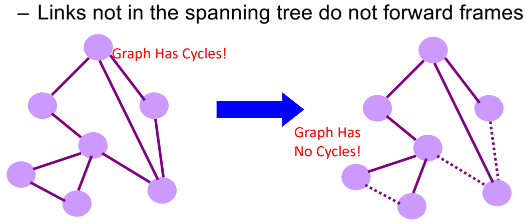
Flooding Can Lead to Loops

- 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

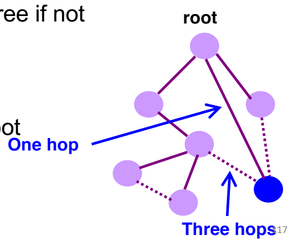


What Do We Know?

- *"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

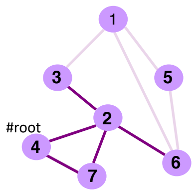


Steps in Spanning Tree Algorithm

- 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)

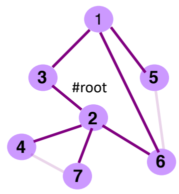
Example From Switch #4's Viewpoint

- 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?

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*

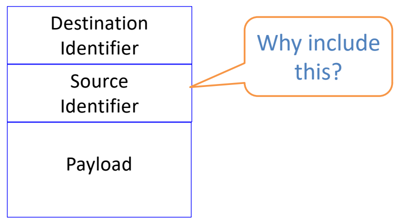
Forwarding: What path do I take next to get to the *something*

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?

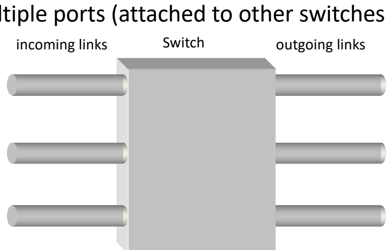
Packets (at a conceptual level)

- 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)

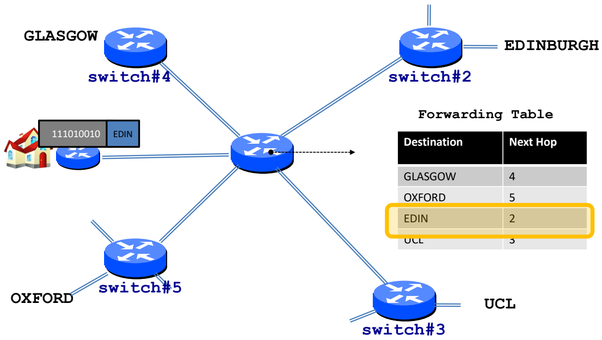
A Variety of *(Internet Protocol-based)* Routers

- 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)




7

Switches forward packets



8

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….

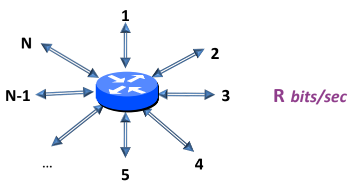
9

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….

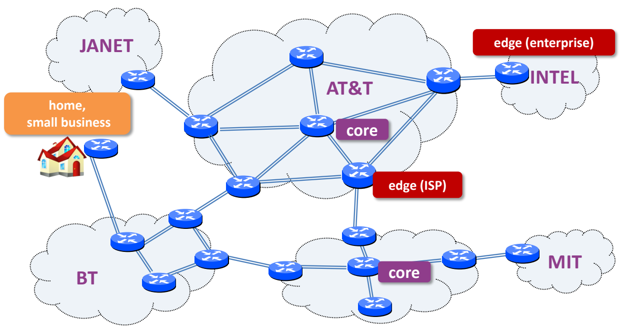
10

Router definitions

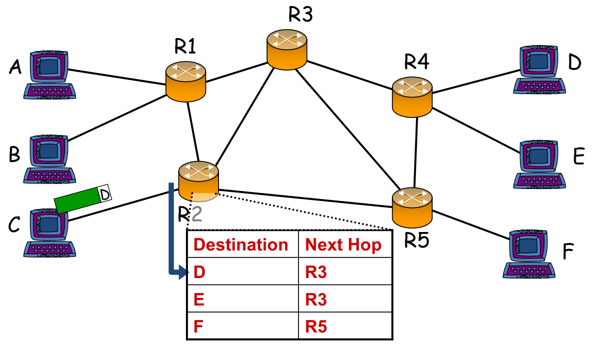


- **N = number of external router "ports"**
- **R = speed ("line rate") of a port**
- **Router capacity = N x R**

Networks and routers

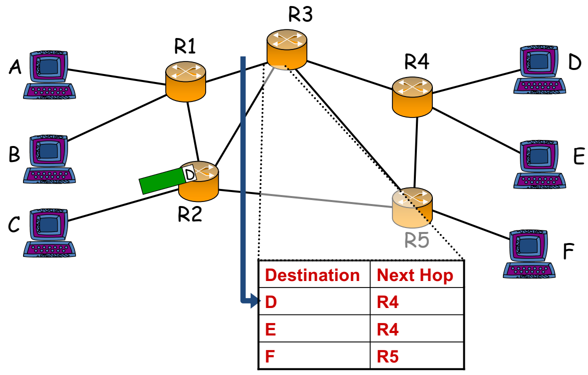


Basic Operation of Router



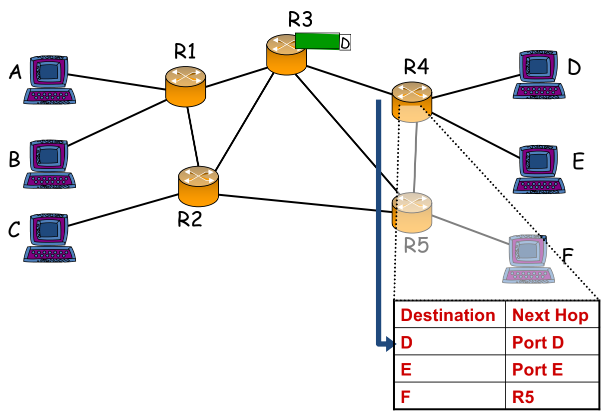
Destination	Next Hop
D	R3
E	R3
F	R5

Basic Operation of Router



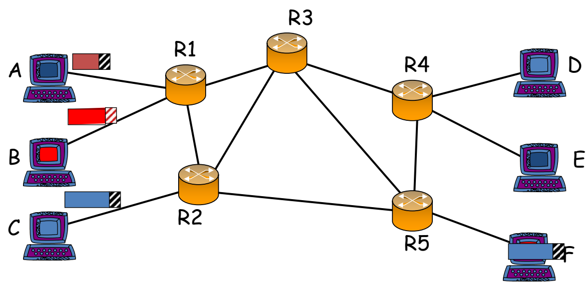
Destination	Next Hop
D	R4
E	R4
F	R5

Basic Operation of Router



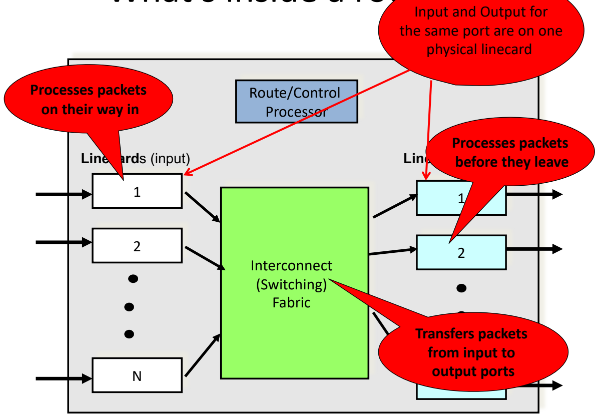
Destination	Next Hop
D	Port D
E	Port E
F	R5

What does a router do?

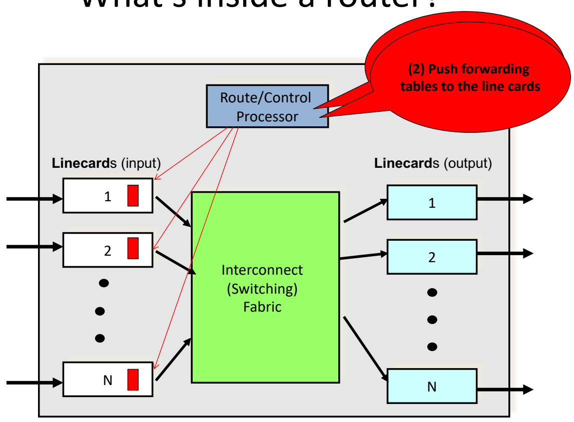


1. Every router performs a per-packet lookup for every packet
2. Each router performs a lookup in it's local lookup table
3. Each router performs lookups (ENTIRELY) independently of every other router

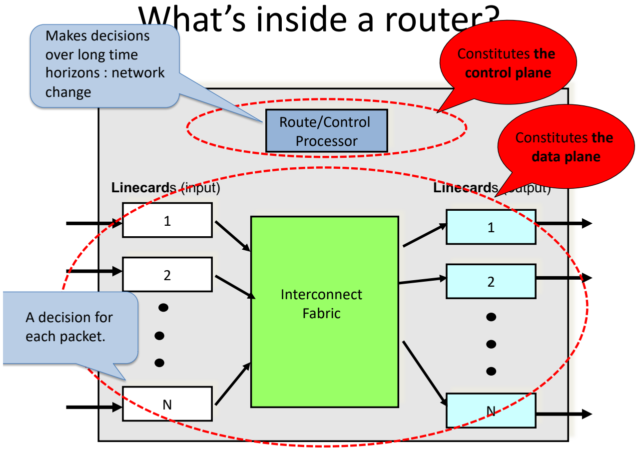
What's inside a router?

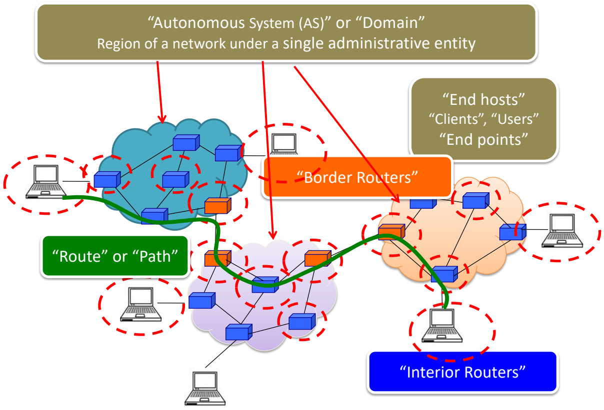


What's inside a router?

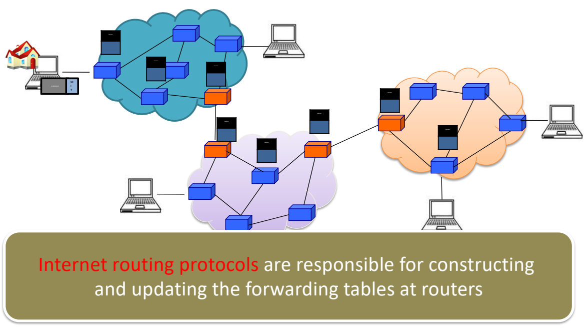


What's inside a router?



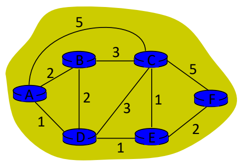


Context and Terminology



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

Outline

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

Link-State Routing

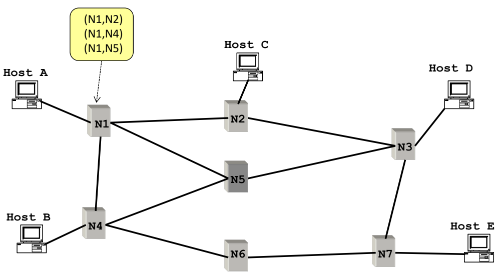
Examples:

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)

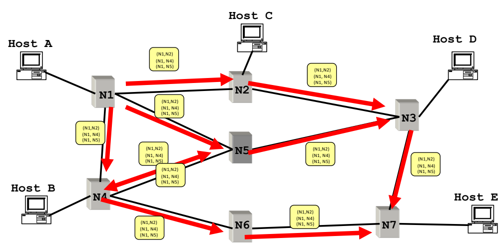
Link State Routing

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



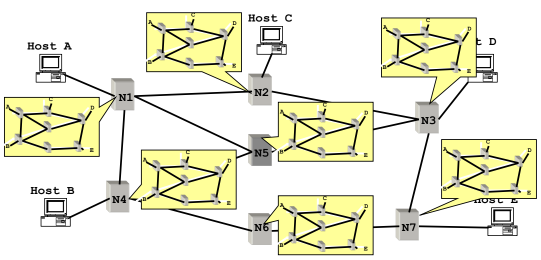
Link State Routing

- 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

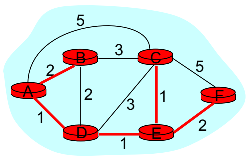


Dijkstra's Shortest Path Algorithm

- 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*



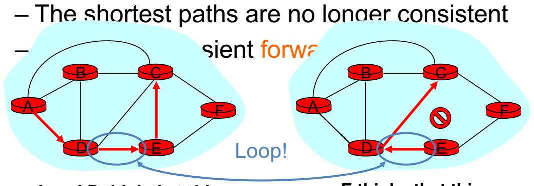
Destination	Link
B	(A,B)
C	(A,D)
D	(A,D)
E	(A,D)
F	(A,D)

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



A and D think that this is the path to C

E thinks that this is the path to C

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)

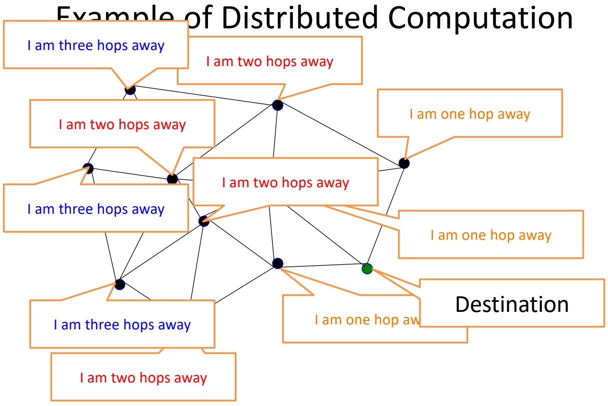
Go!

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

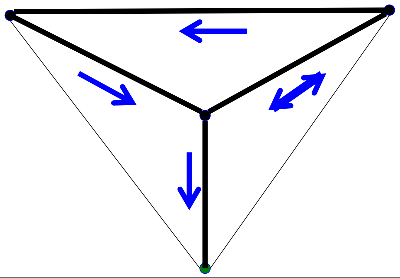
- What if we use a non-additive metric?
  - E.g., maximal capacity
- What if routers don't use the same metric?
  - I want low delay, you want low loss rate?
- What happens if nodes lie?

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

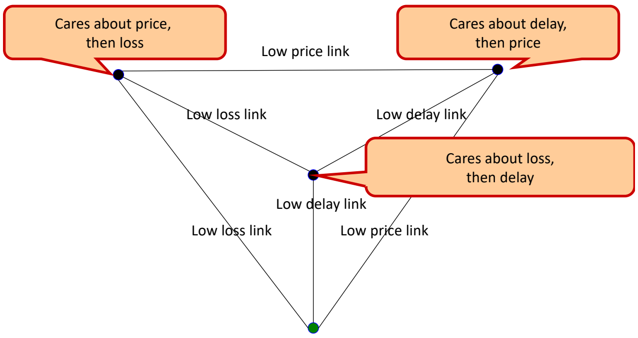


Additive measures avoid this problem!

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

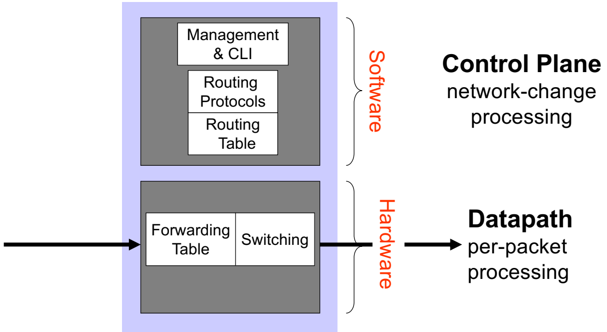
- Propagation delay
- Congestion
- Load balance
- Bandwidth (available, capacity, maximal, bbw)
- Price
- Reliability
- Loss rate
- Combinations of the above

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

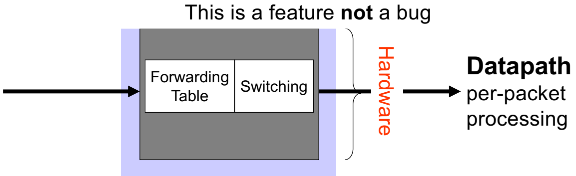


Independent operation!

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

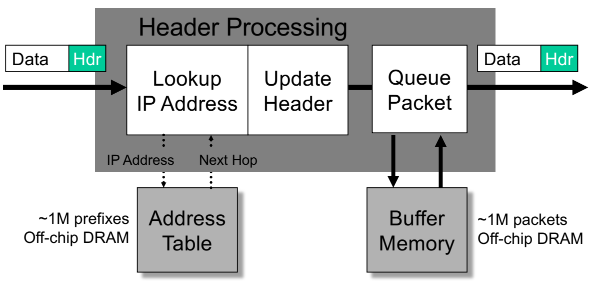
This is a feature **not** a bug



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

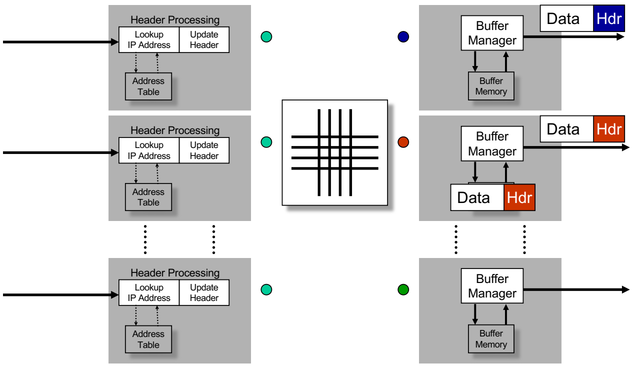
Entry	Destination	Port
1	0.0.0.0	1
2	0.0.0.1	2
⋮	⋮	⋮
2^{32}	255.255.255.255	12

} **~ 4 billion entries**

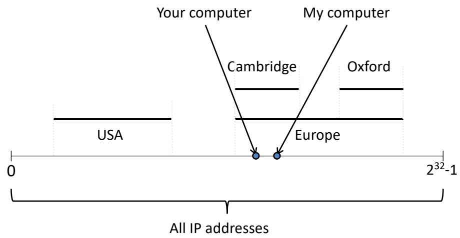
Improved approach:
Group entries to reduce table size

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

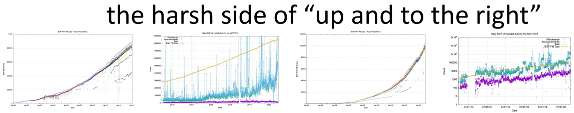
↓

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"

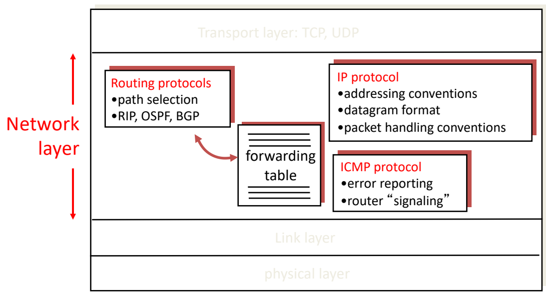


Open problems : continual growth is continual demand for innovation opportunities in control, algorithms, & network hardware

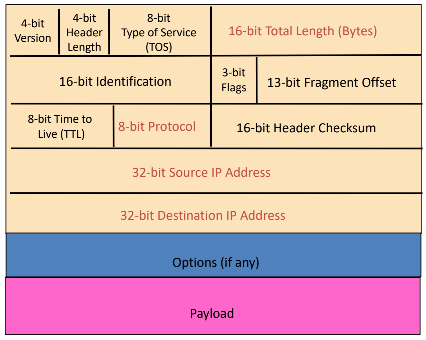
Hudson 2020 report https://blog.apnic.net/2021/01/05/bgp-in-2020-the-bgp-table/

The Internet version of a Network layer

Host, router network layer functions:



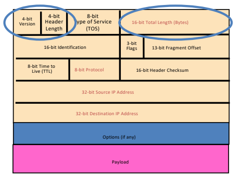
IPv4 Packet Structure
20 Bytes of Standard Header, then Options



(Packet) Network Tasks One-by-One

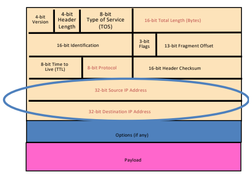
- Read packet correctly
- Get packet to the destination
- Get responses to the packet back to source
- Carry data
- Tell host what to do with packet once arrived
- Specify any special network handling of the packet
- Deal with problems that arise along the path

Reading Packet Correctly



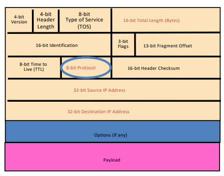
- Version number (4 bits)
  - Indicates the version of the IP protocol
  - Necessary to know what other fields to expect
  - Typically "4" (for IPv4), and sometimes "6" (for IPv6)
- Header length (4 bits)
  - Number of 32-bit words in the header
  - Typically "5" (for a 20-byte IPv4 header)
  - Can be 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

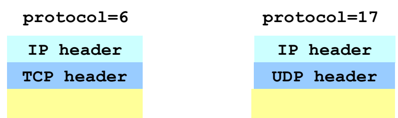


- Two IP addresses
  - Source IP address (32 bits)
  - Destination IP address (32 bits)
- Destination address
  - Unique identifier/locator for the receiving host
  - Allows each node to make forwarding decisions
- Source address
  - Unique identifier/locator for the sending host
  - Recipient can decide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

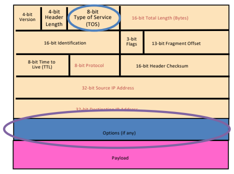
Telling Host How to Handle Packet



- 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)



Special Handling

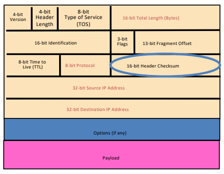


- Type-of-Service (8 bits)
  – Allow packets to be treated differently based on needs
  – E.g., low delay for audio, high bandwidth for bulk transfer
  – Has been redefined several times
- Options

Potential Problems

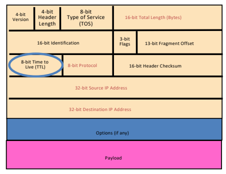
- Header Corrupted: **Checksum**
- Loop: **TTL**
- Packet too large: **Fragmentation**

Header Corruption

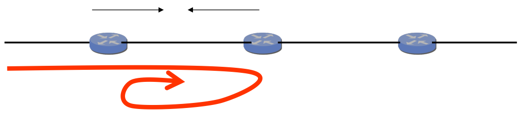


- Checksum (16 bits)
  – Particular form of checksum over packet header
- If not correct, router discards packets
  – So it doesn't act on bogus information
- Checksum recalculated at every router
  – **Why?**
  – **Why include TTL?**
  – **Why only header?**

Preventing Loops
(aka Internet Zombie plan)

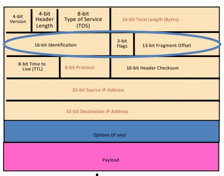


- Forwarding loops cause packets to cycle forever
  – As these accumulate, eventually consume **all** capacity



- 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**

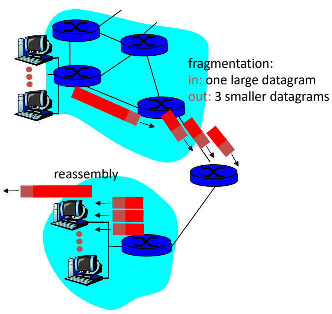
Fragmentation
(some assembly required)



- 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

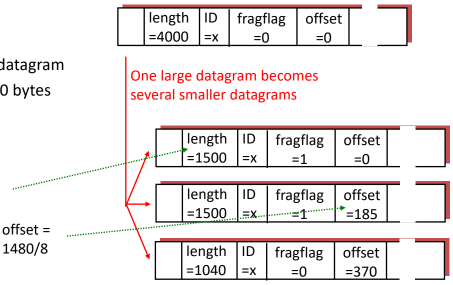
IP Fragmentation & Reassembly

- 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

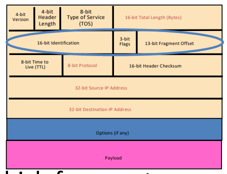
Example
- 4000 byte datagram
- MTU = 1500 bytes



1480 bytes in data field

Question: What happens when a fragment is lost?

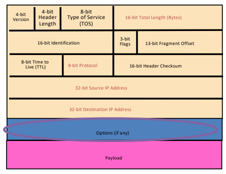
Fragmentation Details



- 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?

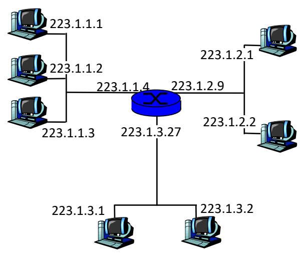
Options



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

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

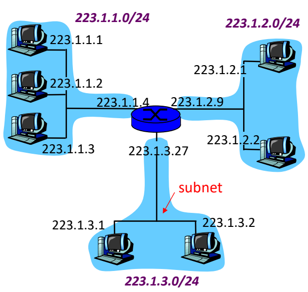


223.1.1.1 = 11011111 00000001 00000001 00000001

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



subnet part | host part

11011111 00000001 00000011 00000000

223.1.3.0/24

Subnet mask: /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

network consisting of 3 subnets

IP addresses: how to get one?

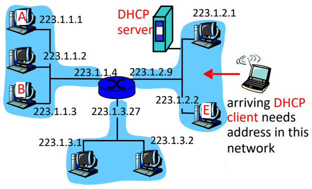
Q: How does a *host* get IP address?

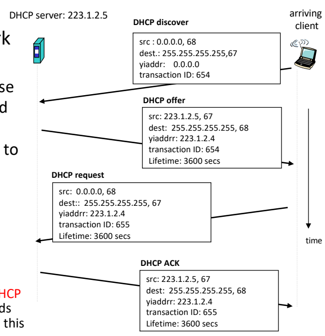
- hard-coded by system admin in a file
  - Windows: control-panel->network->configuration->tcp/ip->properties
  - UNIX: /etc/rc.config (circa 1980's your mileage will vary)
- DHCP: Dynamic Host Configuration Protocol: dynamically get address from as server
  - "plug-and-play"

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)





IP addresses: how to get one?

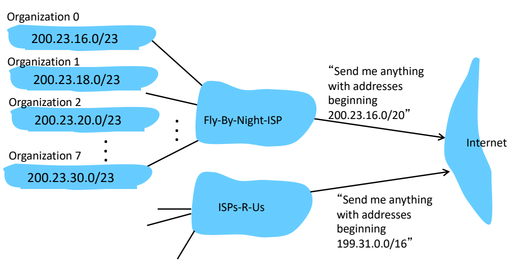
Q: How does *network* get subnet part of IP addr?

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

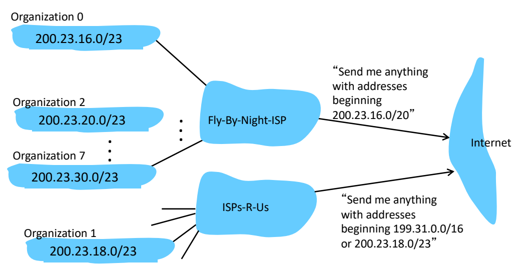
Hierarchical addressing: route aggregation

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...

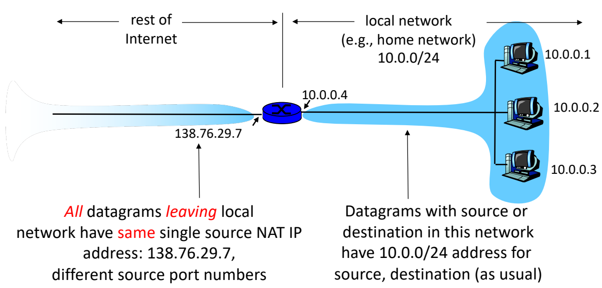
Q: How does an ISP get a block of addresses?

A: ICANN: Internet Corporation for Assigned Names and Numbers

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

Cant get more IP addresses? well there is always.....

NAT: Network Address Translation



NAT: Network Address Translation

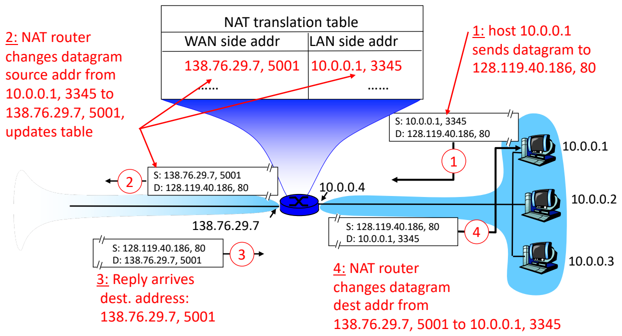
- Motivation: local network uses just one IP address as far as outside world is concerned:
  - range of addresses not needed from ISP: just one IP address for all devices
  - can change addresses of devices in local network without notifying outside world
  - can change ISP without changing addresses 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

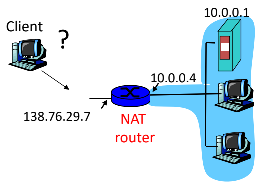


NAT: Network Address Translation

- 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

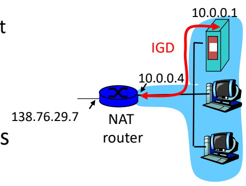
- client wants to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can't use it as destination addr)
  - only one externally visible NATted address: 138.76.29.7
- solution 1: statically configure NAT to forward incoming connection requests at given port to server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



NAT traversal problem

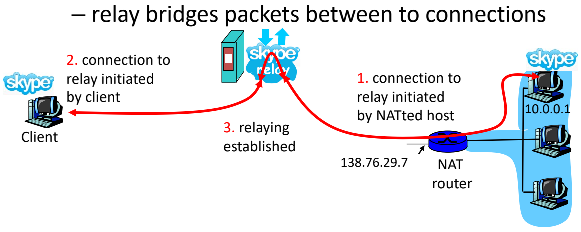
- solution 2: Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATted host to:
  - ❖learn public IP address (138.76.29.7)
  - ❖add/remove port mappings (with lease times)

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

```
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
```

trans-continent link

* 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

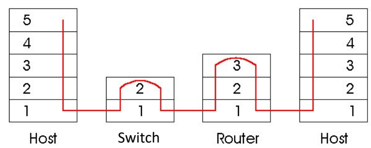
- used by hosts & routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer "above" IP:
  - ICMP msgs carried in IP datagrams
- ICMP message: type, code plus first 8 bytes of IP datagram causing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

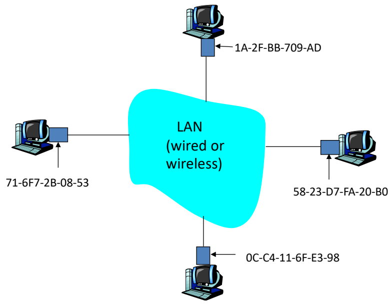


MAC Addresses (and IPv4 ARP)
or How do I glue my network to my data-link?

- 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 address



Ethernet
Broadcast address =
FF-FF-FF-FF-FF-FF

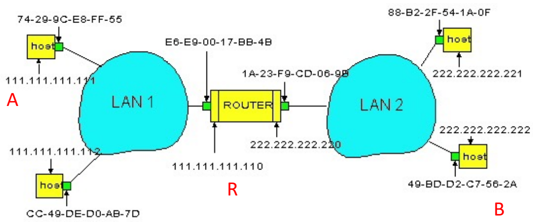
= adapter

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

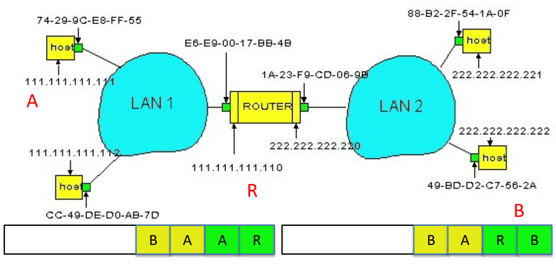
Example: A Sending a Packet to B

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



Example: A Sending a Packet to B

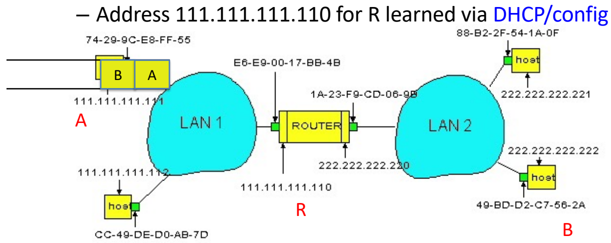
How does host A send an IP packet to host B?



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

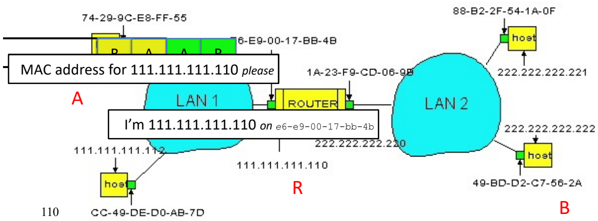
Host A Decides to Send Through R

- 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



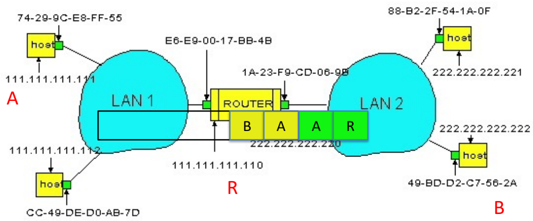
Host A Sends Packet Through R

- 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



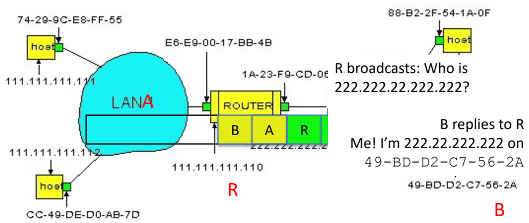
R Decides how to Forward Packet

- 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

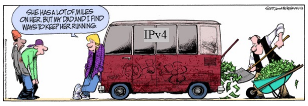
Key Ideas in Both ARP and DHCP

- 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

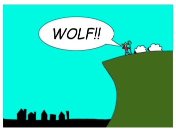


prematurely

- 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

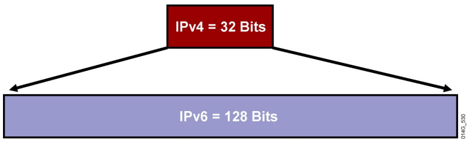
- Result is an elegant, if unambitious, protocol



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).

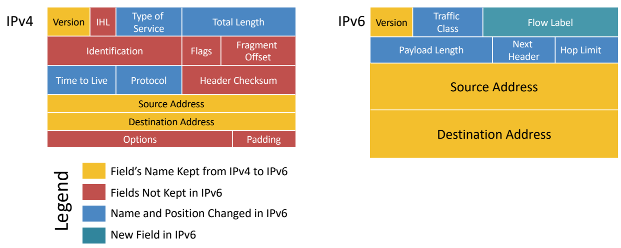
Larger Address Space

- 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



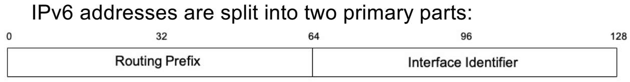
Other Significant Protocol Changes - 2

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 an transport-layer header (e.g., TCP)

IPv6 Basic Address Structure

IPv6 addresses are split into two primary parts:



- 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

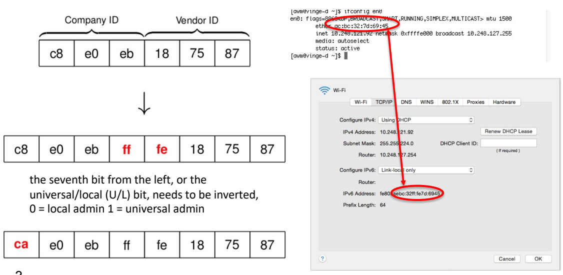
The algorithm (assuming one interface):

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



[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, router advertisements are generated intermittently 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

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

Incentive issues

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, ….

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

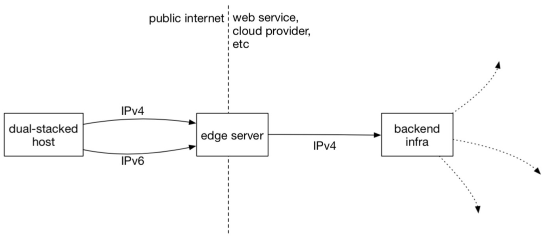
IPv6 Tunnel Broker

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

Happy Eyeballs 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]

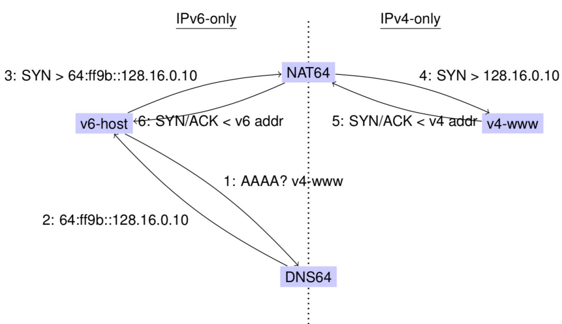
[9] https://tools.ietf.org/html/rfc5461
[10] https://tools.ietf.org/html/rfc3484

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

DNS64 & NAT64



464XLAT

- Problem: IPv6-only to the host, but an IPv4-only app trying to access an IPv4-only service
  - Some *applications* do not understand IPv6, so having an IPv6 address doesn't help
  - 464XLAT[12] solves this problem
  - In essence, DNS64 + NAT64 + a shim layer on the host itself to offer IPv4 addresses to apps

[12] https://tools.ietf.org/html/rfc6877

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

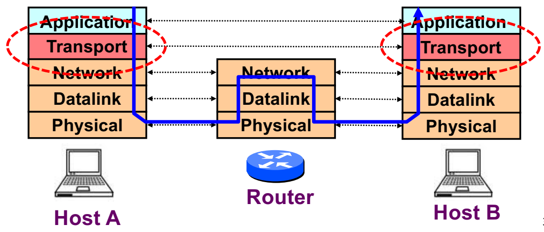
Our goals:

- 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

2

Transport Layer

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



3

Why a transport layer?

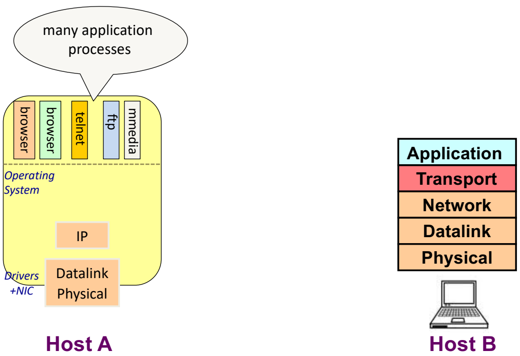
- IP packets are addressed to a host but end-to-end communication is between application/ processes/tasks at hosts
  - Need a way to decide which packets go to which applications (*more multiplexing*)

4

Why a transport layer?

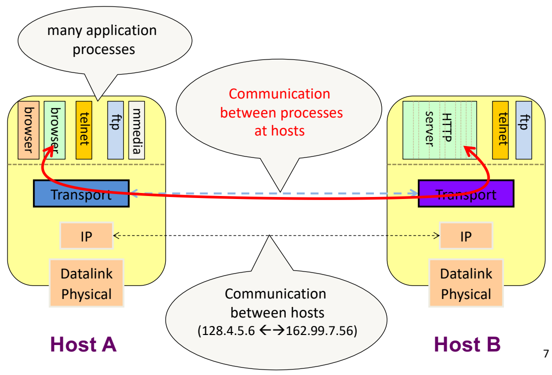
5

Why a transport layer?



6

Why a transport layer?



7

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

Context: Applications and Sockets

- Socket: software abstraction by which an application process exchanges network messages with the (transport layer in the) operating system
  - socketID = socket(…, socket.TYPE)
  - socketID.sendto(message, …)
  - socketID.recvfrom(…)

- Two important types of sockets
  - UDP socket: TYPE is SOCK_DGRAM
  - TCP socket: TYPE is SOCK_STREAM

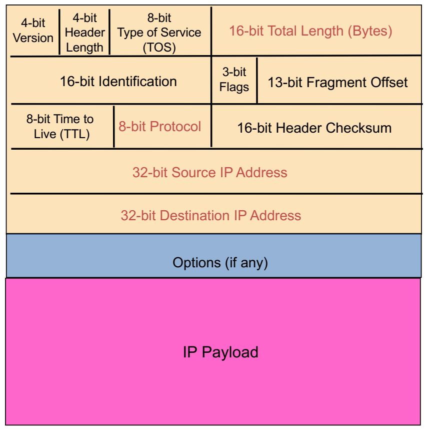
Ports

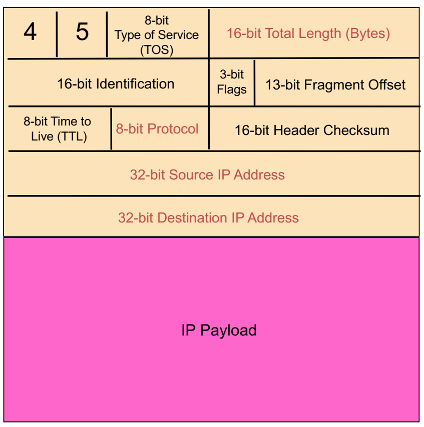
- Problem: deciding which app (socket) gets which packets

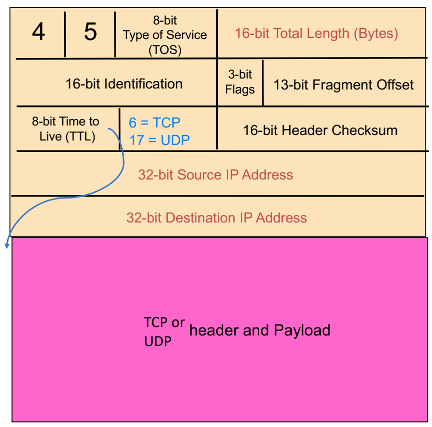
– Solution: ***port*** as a transport layer identifier
  - 16 bit identifier
  - OS stores mapping between sockets and ***ports***
  - a packet carries a source and destination port number in its transport layer header

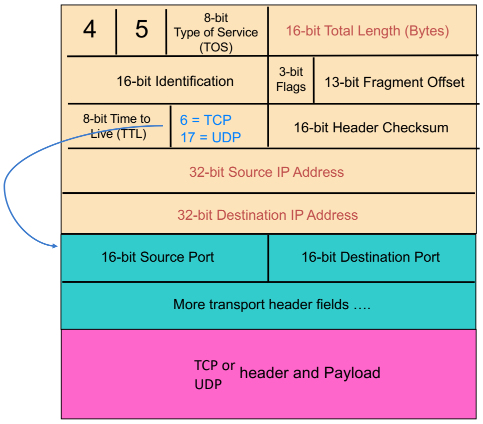
- For UDP ports (SOCK_DGRAM)
  - OS stores (local port, local IP address) ←→ socket

- For TCP ports (SOCK_STREAM)
  - OS stores (local port, local IP, remote port, remote IP) ←→ socket









Recap: Multiplexing and Demultiplexing

- Host receives IP packets
  - Each IP header has source and destination IP address
  - Each Transport Layer header has source and destination port number
- Host uses IP addresses and port numbers to direct the message to appropriate socket

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))

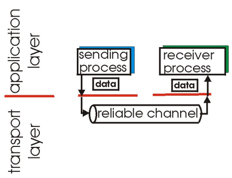
SRC port	DST port
checksum	length
DATA	

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

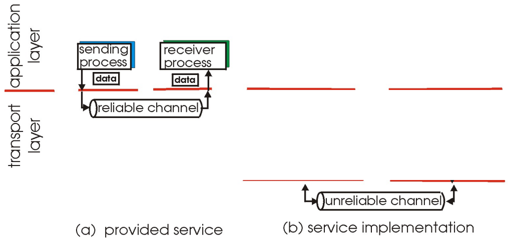
- In a perfect world, reliable transport is easy

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

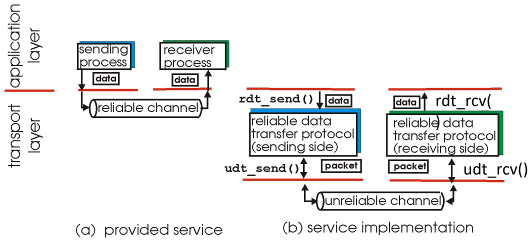
- important in app., transport, link layers
- top-10 list of important networking topics!



- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

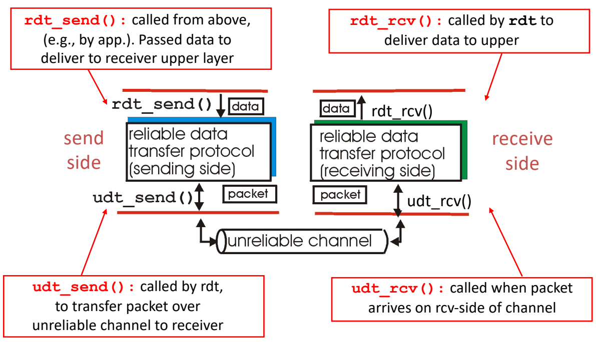
Principles of Reliable data transfer

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



- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

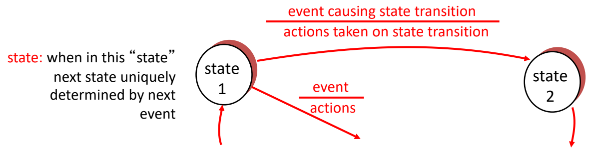
Reliable data transfer: getting started



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.

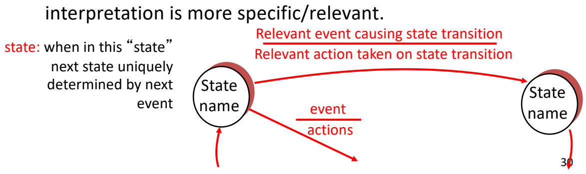
Beware

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

I've attempted to normalise the representation.

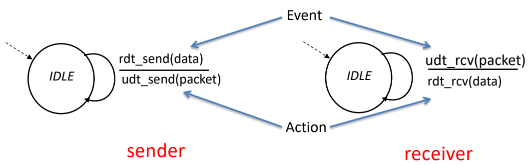
UPSHOT: these slides have differing information to the KR book (from which the RDT example is taken.)

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



Rdt1.0: reliable transfer over a reliable channel

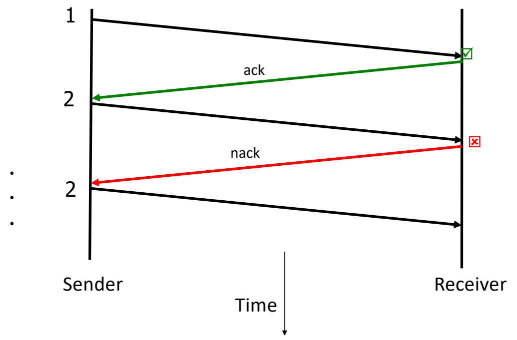
- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- separate FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver read data from underlying channel



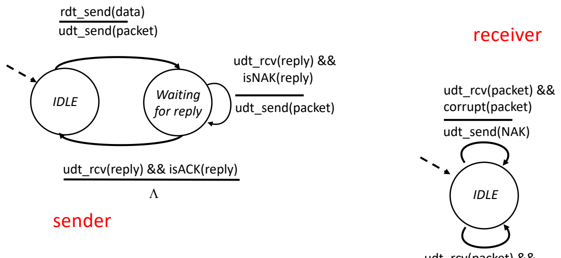
Rdt2.0: channel with bit errors

- 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

Dealing with Packet Corruption

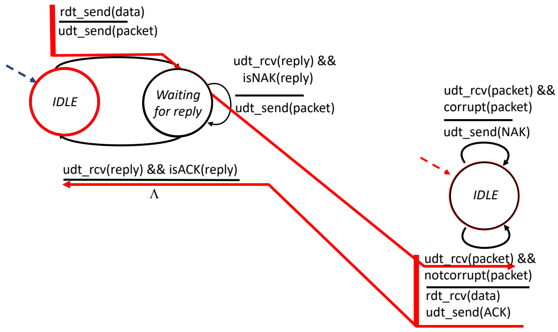


rdt2.0: FSM specification

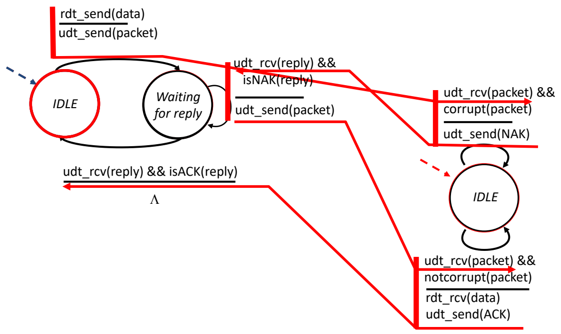


Note: the sender holds a copy of the packet being sent until the delivery is acknowledged.

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

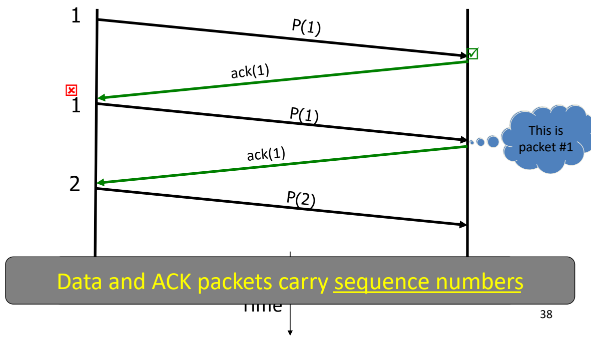
Handling duplicates:

- 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

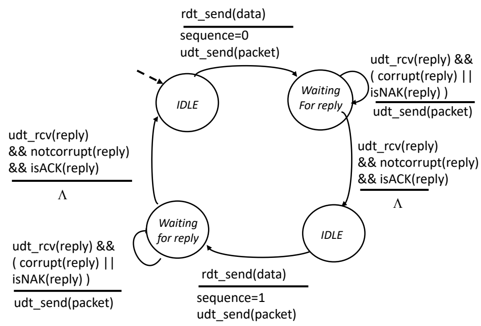
Sender sends one packet, then waits for receiver response

Dealing with Packet Corruption

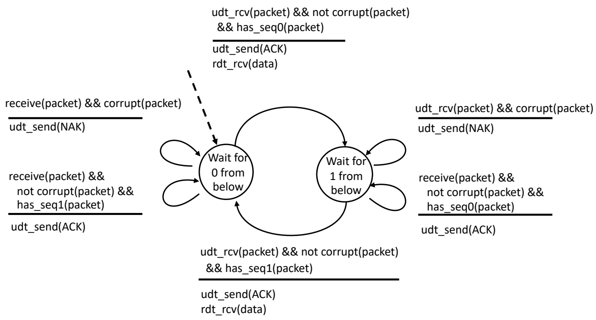


Data and ACK packets carry sequence numbers

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

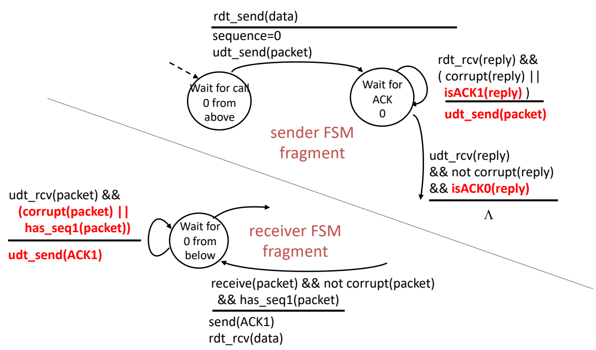
Receiver:

- 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*

rdt2.2: sender, receiver fragments



rdt3.0: channels with errors *and* loss

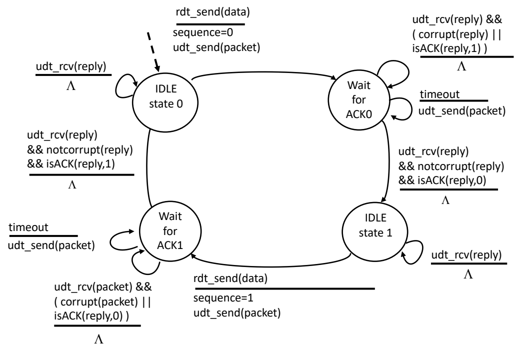
New assumption: underlying channel can also lose packets (data or ACKs)

– checksum, seq. #, ACKs, retransmissions will be of help, but not enough

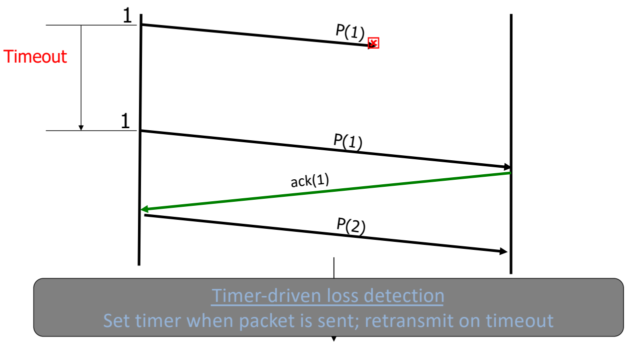
Approach: sender waits "reasonable" amount of time for ACK

- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  – retransmission will be duplicate, but use of seq. #'s already handles this
  – receiver must specify seq # of pkt being ACKed
- requires countdown timer

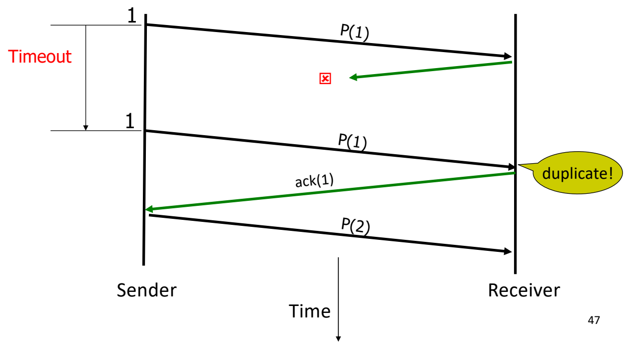
rdt3.0 sender



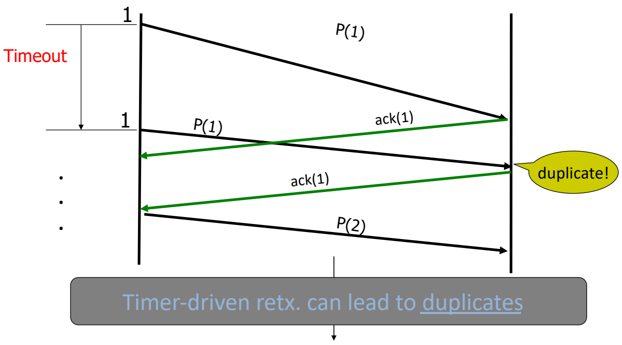
Dealing with Packet Loss



Dealing with Packet Loss



Dealing with Packet Loss



Performance of rdt3.0

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

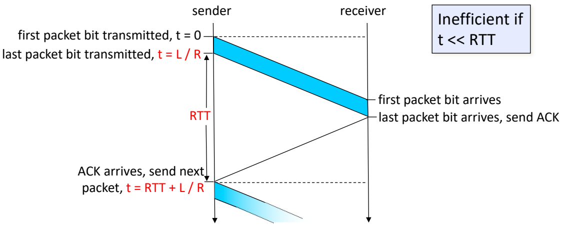
$$d_{trans} = \frac{L}{R} = \frac{8000\text{bits}}{10^9\text{bps}} = 8\,\text{microseconds}$$

- 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
- network protocol limits use of physical resources!

rdt3.0: stop-and-wait operation



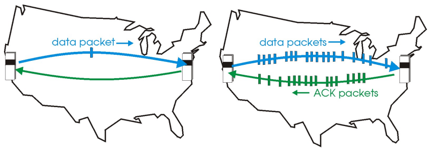
Inefficient if t << RTT

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

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



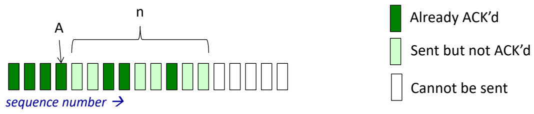
(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

A Sliding Packet Window

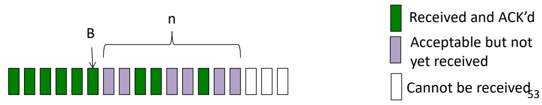
- window = set of adjacent sequence numbers
  - The size of the set is the window size; assume window size is *n*
- General idea: send up to *n* packets at a time
  - Sender can send packets in its window
  - Receiver can accept packets in its window
  - Window of acceptable packets "slides" on successful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

A Sliding Packet Window

- Let A be the last ack'd packet of sender without gap; then window of sender = {A+1, A+2, …, A+n}



- Let B be the last received packet without gap by receiver, then window of receiver = {B+1,…, B+n}

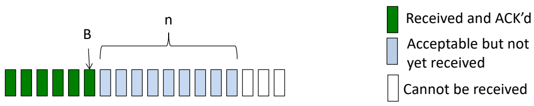


Acknowledgements w/ Sliding Window

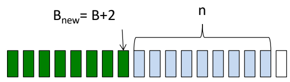
- Two common options
  - cumulative ACKs: ACK carries next in-order sequence number that the receiver expects

Cumulative Acknowledgements (1)

- At receiver



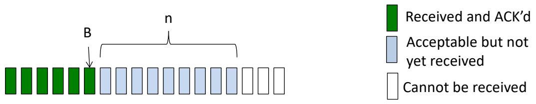
- After receiving B+1, B+2



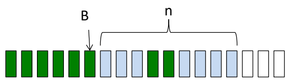
- Receiver sends ACK(B_{new}+1)

Cumulative Acknowledgements (2)

- At receiver



- After receiving B+4, B+5



How do we recover?

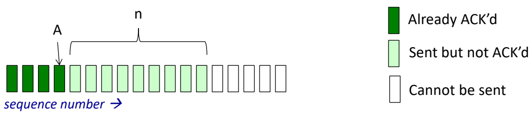
- Receiver sends ACK(B+1)

Go-Back-N (GBN)

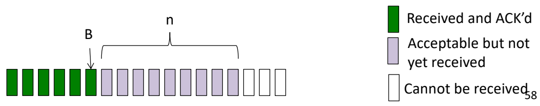
- Sender transmits up to *n* unacknowledged packets
- Receiver only accepts packets in order
  - discards out-of-order packets (i.e., packets other than *B+1*)
- Receiver uses cumulative acknowledgements
  - i.e., sequence# in ACK = next expected in-order sequence#
- Sender sets timer for 1st outstanding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

Sliding Window with GBN

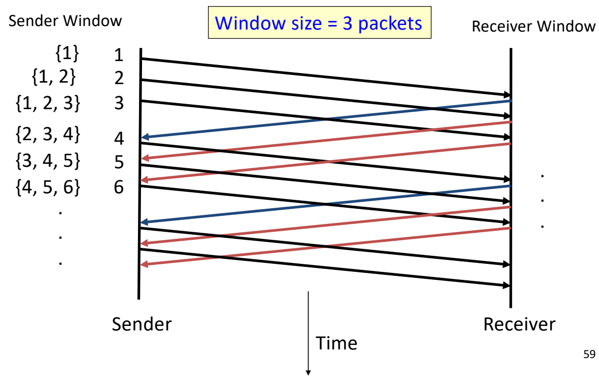
- Let A be the last ack'd packet of sender without gap; then window of sender = {A+1, A+2, …, A+n}



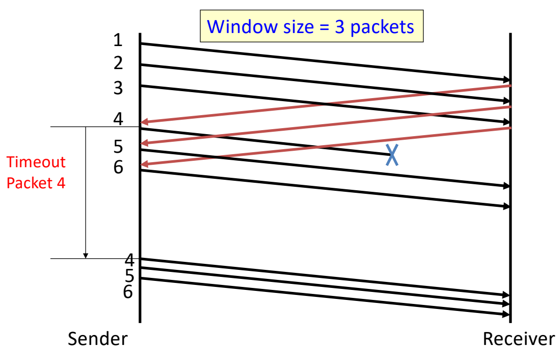
- Let B be the last received packet without gap by receiver, then window of receiver = {B+1,…, B+n}



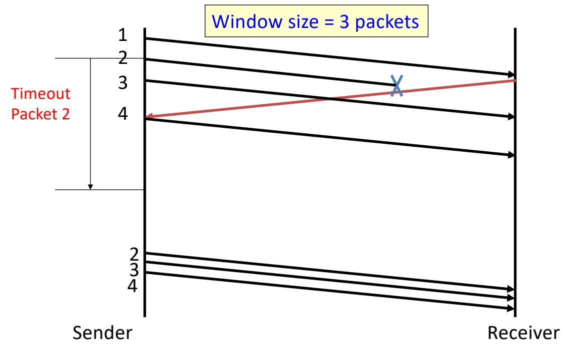
GBN Example w/o Errors



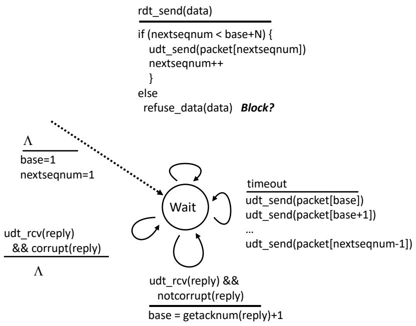
GBN Example with Errors



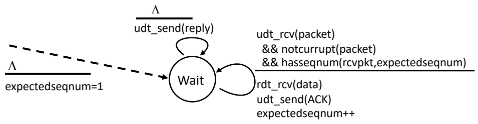
GBN Example with Errors - ALTERNATIVE



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 #

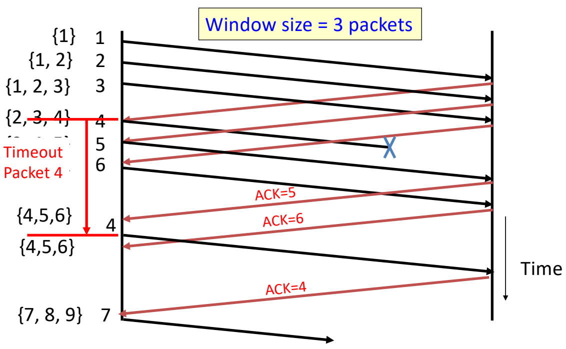
Acknowledgements w/ Sliding Window

- Two common options
  - cumulative ACKs: ACK carries next in-order sequence number the receiver expects
  - selective ACKs: ACK individually acknowledges correctly received packets
- Selective ACKs offer more precise information but require more complicated book-keeping
- Many variants that differ in implementation details

Selective Repeat (SR)

- Sender: transmit up to *n* unacknowledged packets
- Assume packet *k* is lost, *k+1* is not
- Receiver: indicates packet *k+1* correctly received
- Sender: retransmit only packet *k* on timeout
- Efficient in retransmissions but complex book-keeping
  - need a timer per packet

SR Example with Errors



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)

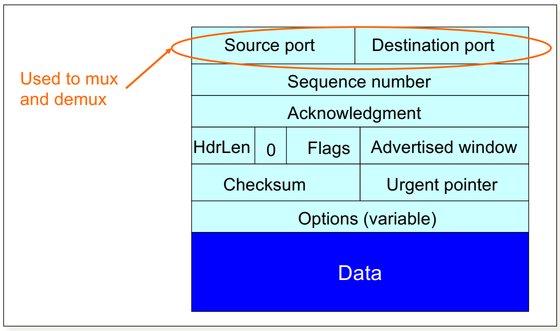
- Reliability protocols use the above to decide when and what to retransmit or acknowledge

What does TCP do?

Most of our previous tricks + a few differences

- 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 Header

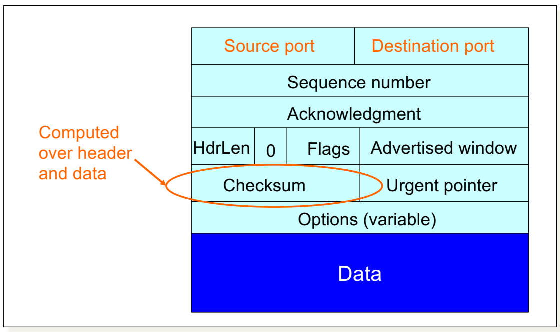


What does TCP do?

Many of our previous ideas, but some key differences

- Checksum

TCP Header



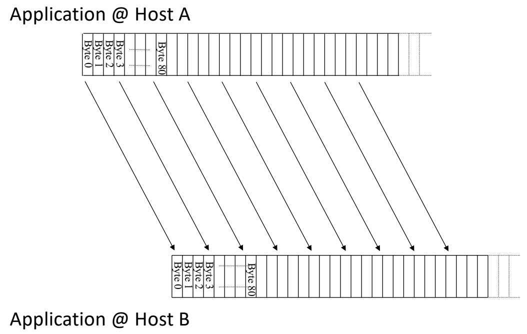
What does TCP do?

Many of our previous ideas, but some key differences

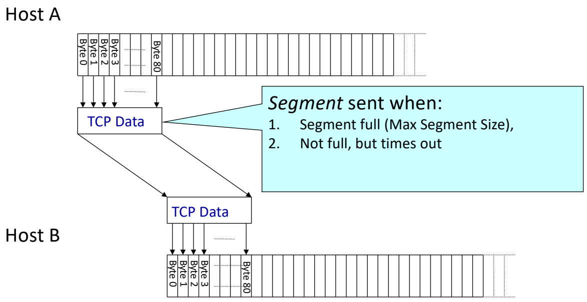
- Checksum
- **Sequence numbers are byte offsets**

TCP: Segments and Sequence Numbers

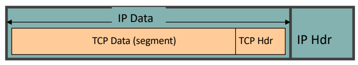
TCP “Stream of Bytes” Service…



… Provided Using TCP “Segments”

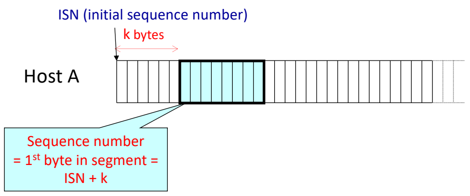


TCP Segment

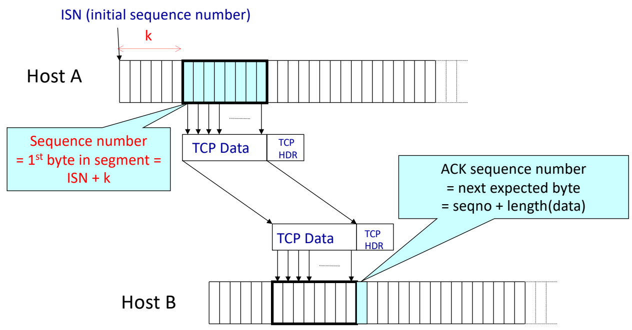


- IP packet
  - No bigger than Maximum Transmission Unit (MTU)
  - E.g., up to 1500 bytes with Ethernet
- TCP packet
  - IP packet with a TCP header and data inside
  - TCP header ≥ 20 bytes long
- TCP **segment**
  - No more than Maximum Segment Size (MSS) bytes
  - E.g., up to 1460 consecutive bytes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

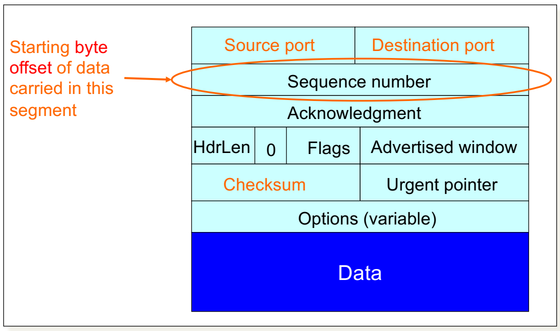
Sequence Numbers



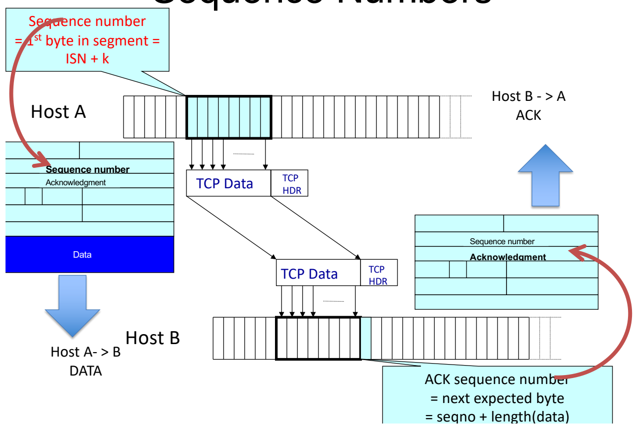
Sequence Numbers



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

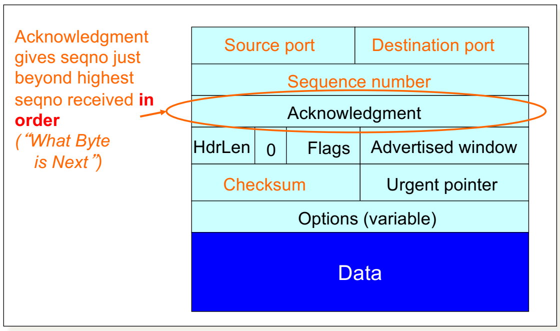
- Sender sends packet
  - Data starts with sequence number X
  - Packet contains B bytes [X, X+1, X+2, ….X+B-1]
- Upon receipt of packet, receiver sends an ACK
  - If all data prior to X already received:
    - ACK acknowledges X+B (because that is next expected byte)
  - If highest in-order byte received 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?

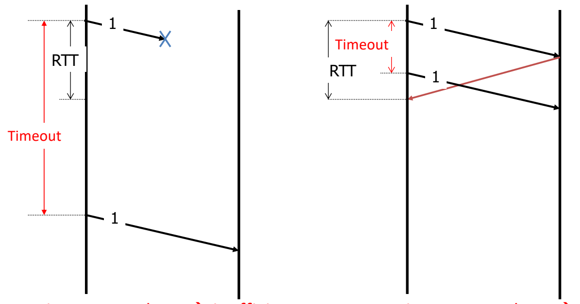
Most of our previous tricks, but a few differences

- 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

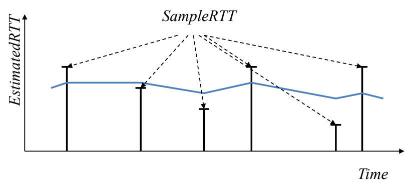
Retransmission Timeout

- 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

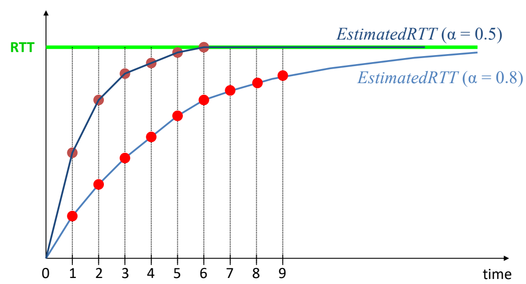
$$SampleRTT = AckRcvdTime - SendPacketTime$$
$$EstimatedRTT = \alpha \times EstimatedRTT + (1-\alpha) \times SampleRTT$$
$$0 < \alpha \leq 1$$



Exponential Averaging Example

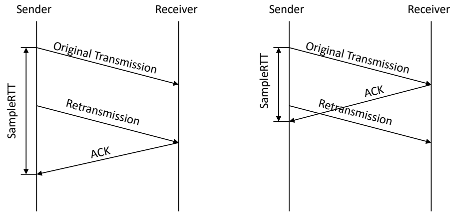
EstimatedRTT = α**EstimatedRTT* + (1 – α)**SampleRTT*

Assume RTT is constant → *SampleRTT* = RTT



Problem: Ambiguous Measurements

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

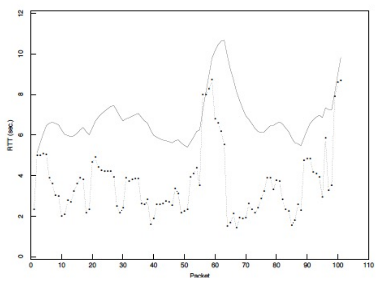


Karn/Partridge Algorithm

- Measure *SampleRTT* only for original transmissions
  - Once a segment has been retransmitted, do not use it for any further measurements
- Computes EstimatedRTT using $\alpha = 0.875$
- 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

Karn/Partridge in action

Figure 5: Performance of an RFC793 retransmit timer



from Jacobson and Karels, SIGCOMM 1988

Jacobson/Karels Algorithm

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

With Jacobson/Karels

Figure 5: Performance of an RFC793 retransmit timer

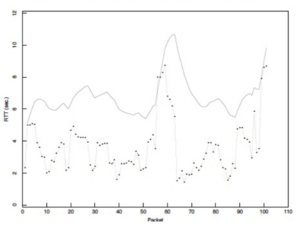
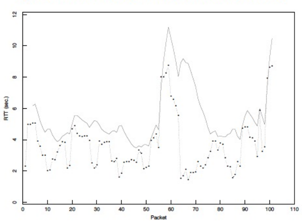


Figure 6: Performance of a Mean+Variance retransmit timer

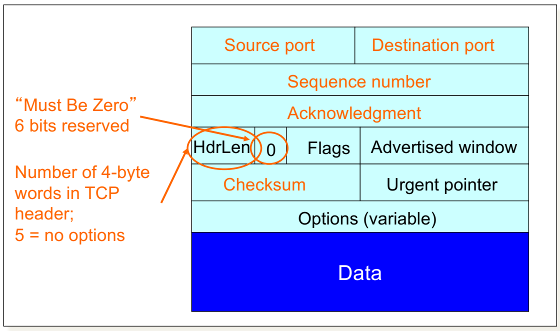


What does TCP do?

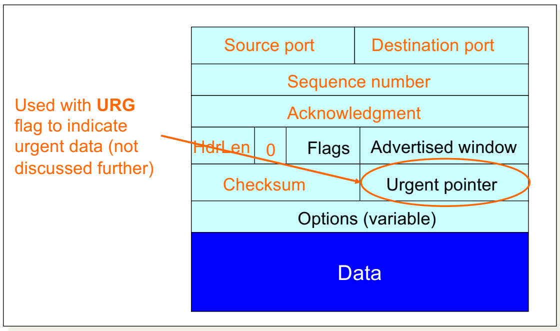
Most of our previous ideas, but some key differences

- 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

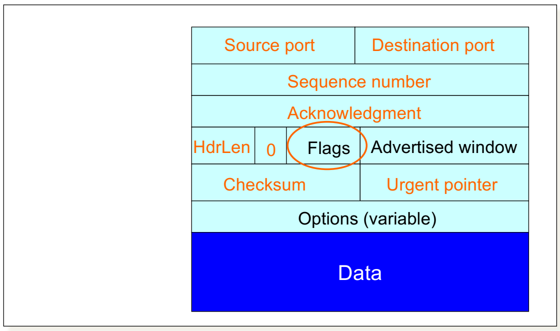
TCP Header: What's left?



TCP Header: What's left?



TCP Header: What's left?

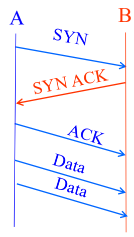


TCP Connection Establishment and Initial Sequence Numbers

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

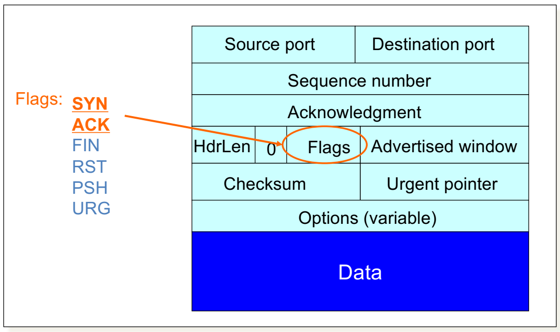
Establishing a TCP Connection



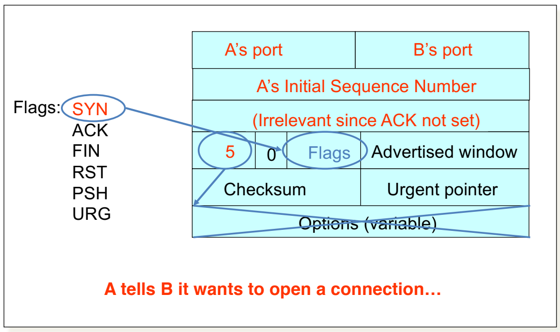
Each host tells its ISN to the other host.

- 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

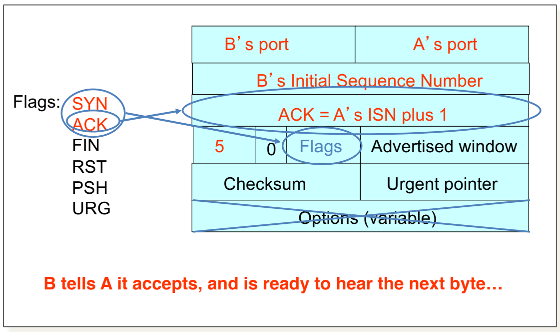
TCP Header



Step 1: A' s Initial SYN Packet

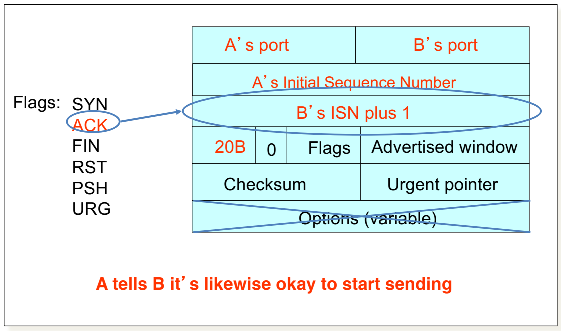


Step 2: B' s SYN-ACK Packet



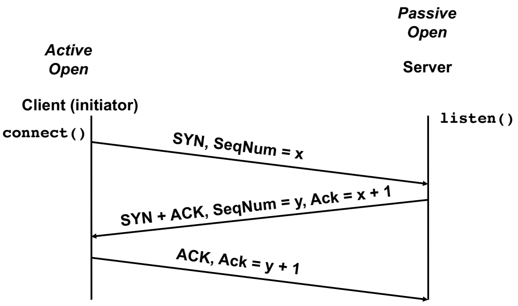
… upon receiving this packet, A can start sending data

Step 3: A' s ACK of the SYN-ACK



… upon receiving this packet, B can start sending data

Timing Diagram: 3-Way Handshaking

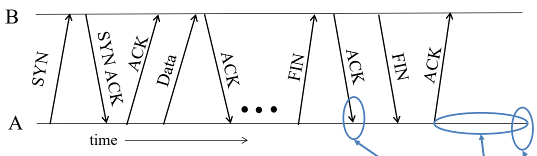


What if the SYN Packet Gets Lost?

- 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

Connection now **closed**

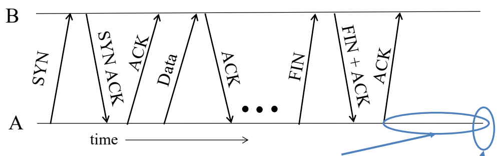
Connection now **half-closed**

TIME_WAIT:

Avoid reincarnation

B will retransmit FIN if ACK is lost

Normal Termination, Both Together



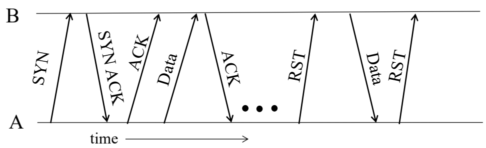
TIME_WAIT:
Avoid reincarnation

Can retransmit
FIN ACK if ACK lost

Connection
now closed

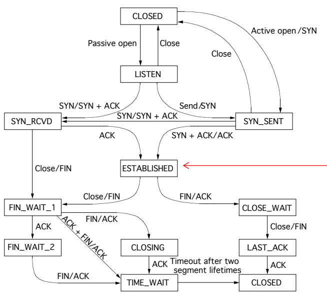
- Same as before, but B sets **FIN** with their ack of A's **FIN**

Abrupt Termination



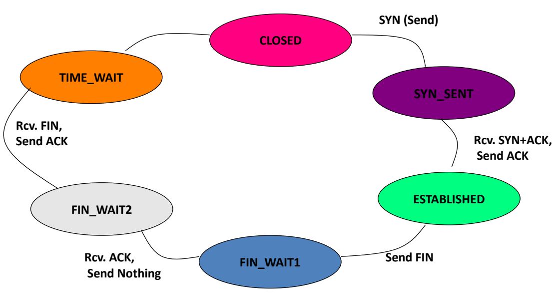
- A sends a RESET (**RST**) to B
  - E.g., because application process on A crashed
- That's it
  - B does not ack the **RST**
  - Thus, **RST** is not delivered reliably
  - And: any data in flight is lost
  - But: if B sends anything more, will elicit another **RST**

TCP State Transitions

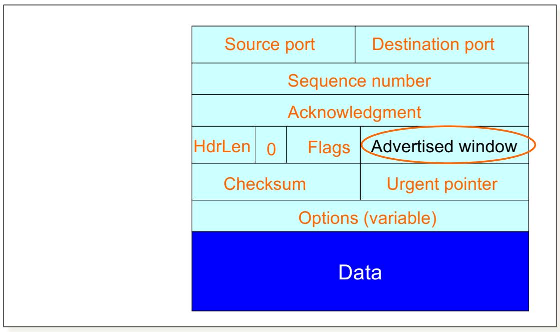


Data, ACK exchanges are in here

An Simpler View of the Client Side



TCP Header

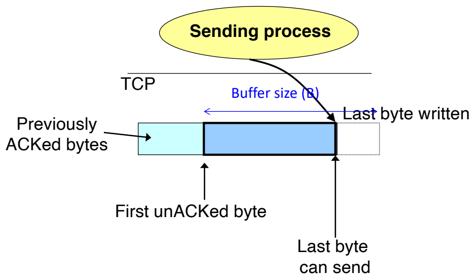


- What does TCP do?
  – ARQ windowing, set-up, tear-down
- Flow Control in TCP

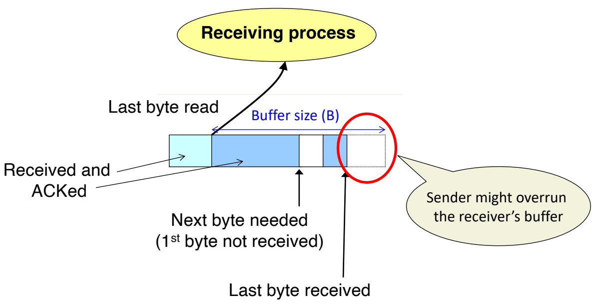
Recap: Sliding Window (so far)

- Both sender & receiver maintain a **window**
- Left edge of window:
  – Sender: beginning of unacknowledged data
  – Receiver: beginning of undelivered data
- Right edge: Left edge + *constant*
  – constant only limited by buffer size in the transport layer

Sliding Window at Sender (so far)



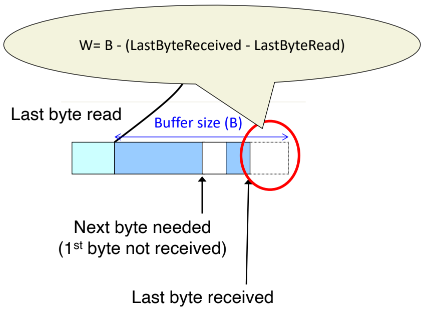
Sliding Window at Receiver (so far)



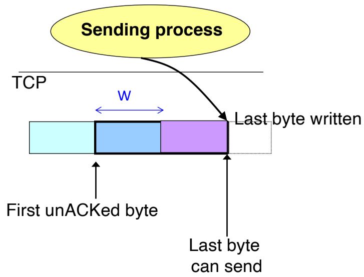
Solution: Advertised Window (Flow Control)

- Receiver uses an "Advertised Window" (W) to prevent sender from overflowing its window
  – Receiver indicates value of W in ACKs
  – Sender limits number of bytes it can have in flight <= W

Sliding Window at Receiver



Sliding Window at Sender (so far)



Sliding Window w/ Flow Control

- Sender: window advances when new data ack'd
- Receiver: window advances as receiving process consumes data
- Receiver advertises to the sender where the receiver window currently ends ("righthand edge")
  - Sender agrees not to exceed this amount

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?

TCP

- The concepts underlying TCP are simple
  - acknowledgments (feedback)
  - timers
  - sliding windows
  - buffer management
  - sequence numbers

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

We have seen:

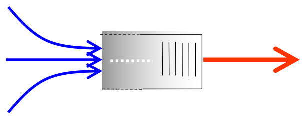
– Flow control: adjusting the sending rate to keep from overwhelming a slow *receiver*

Now lets attend…

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

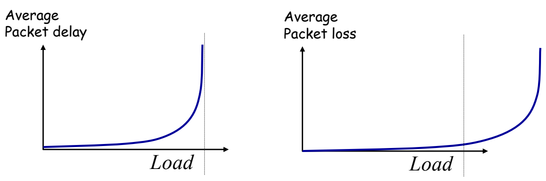
Statistical Multiplexing → Congestion

- If two packets arrive at the same time
  - A router can only transmit one
  - … and either buffers or drops the other
- If many packets arrive in a short period of time
  - The router cannot keep up with the arriving traffic
  - … delays traffic, and the buffer may eventually overflow
- Internet traffic is bursty



Congestion is undesirable

Typical queuing system with bursty arrivals



Must balance utilization versus delay and loss

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

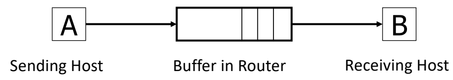
- A pragmatic and effective solution
  - but many other approaches exist

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

Three Issues to Consider

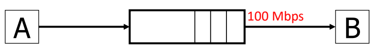
- Discovering the available (bottleneck) bandwidth
- Adjusting to variations in bandwidth
- Sharing bandwidth between flows

Abstract View



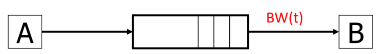
- Ignore internal structure of router and model it as having a single queue for a particular input-output pair

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

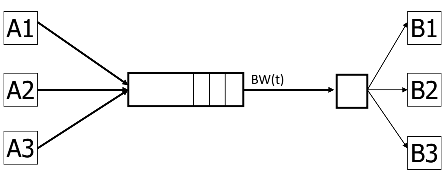


- Adjust rate to match instantaneous bandwidth
  - Assuming you have rough idea of bandwidth

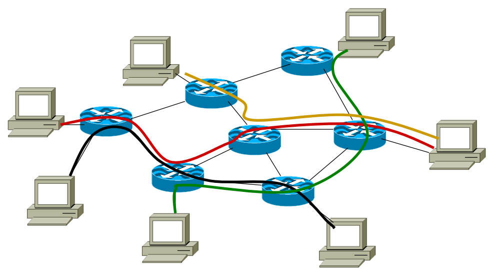
Multiple flows and sharing bandwidth

Two Issues:

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



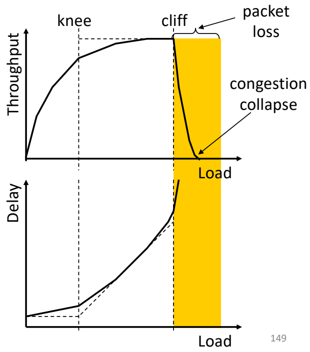
Reality



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

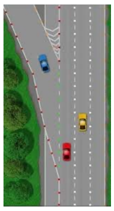
All three techniques have their place

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

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

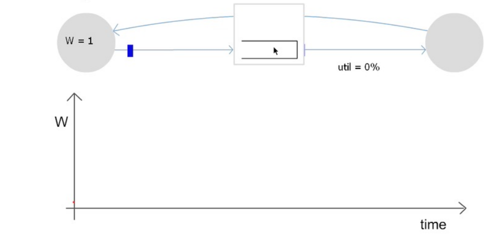
Buffer Sizing Story

	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

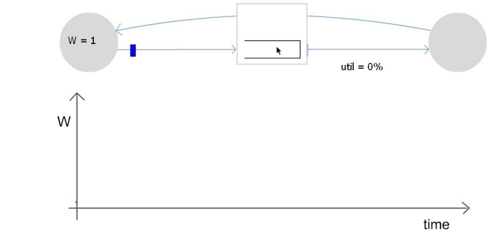
Rule for adjusting W
- If an ACK is received: W ← W+1/W
- If a packet is lost: W ← W/2



Continuous ARQ (TCP) adapting to congestion

Only W packets may be outstanding

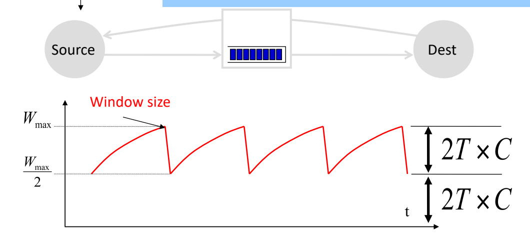
Rule for adjusting W
- If an ACK is received: W ← W+1/W
- If a packet is lost: W ← W/2



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	Small Buffers
	$2T \times C$	$\frac{2T \times C}{\sqrt{n}}$
# 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

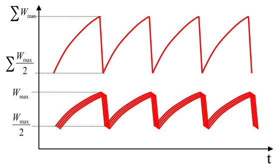
So how large should the buffers be?

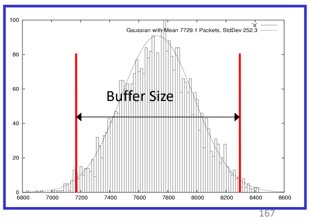
Buffer size matters
- Packet loss
  - Queue overload, and subsequent packet loss
- End-to-end delay
  - Transmission, propagation, and queueing delay
  - The only variable part is queueing delay
- Router architecture
  - Board space, power consumption, and cost
  - On chip buffers: higher density, higher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

Small Buffers – Intuition

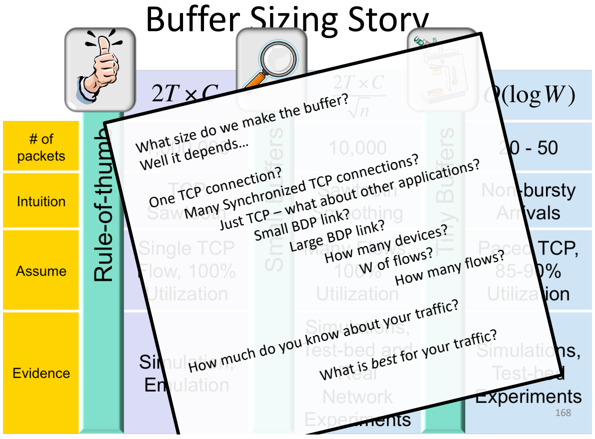
Synchronized Flows
- Aggregate window has same dynamics
- Therefore buffer occupancy has same dynamics
- Rule-of-thumb still holds.

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 LIE:** 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

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

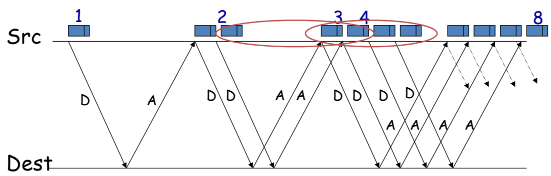
- Goal: estimate available bandwidth
  - start slow (for safety)
  - but ramp up quickly (for efficiency)
- Consider
  - RTT = 100ms, MSS=1000bytes
  - Window size to fill 1Mbps of BW = 12.5 packets
  - Window size to fill 1Gbps = 12,500 packets
  - Either is possible!

"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



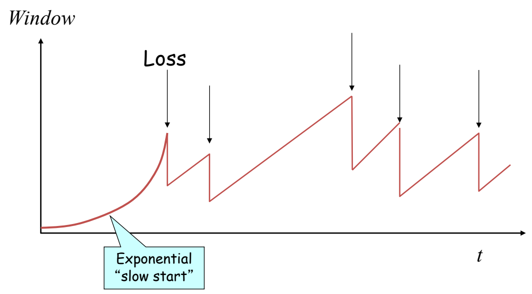
Adjusting to Varying Bandwidth

- 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

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

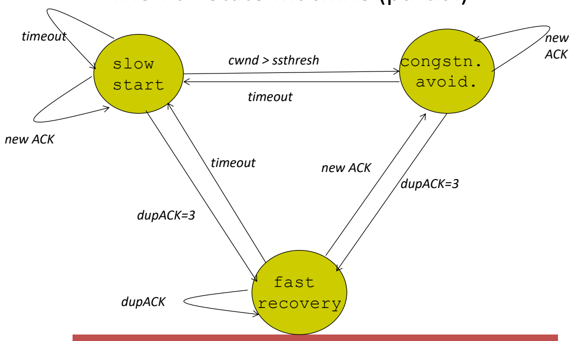
Example

- 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

Putting it all together: The TCP State Machine (partial)



- How are ssthresh, CWND and dupACKcount updated for each event that causes a state transition?

TCP Flavors

- TCP-Tahoe
  - cwnd =1 on triple dupACK
- TCP-Reno
  - cwnd =1 on timeout
  - cwnd = cwnd/2 on triple dupack
- TCP-newReno
  - TCP-Reno + improved fast recovery
- TCP-SACK
  - incorporates selective acknowledgements

- What does TCP do?
  - ARQ windowing, set-up, tear-down
- Flow Control in TCP
- Congestion Control in TCP
  - AIMD, Fast-Recovery, Throughput

TCP Flavors

- TCP-Tahoe
  - CWND =1 on triple dupACK
- TCP-Reno
  - CWND =1 on timeout
  - CWND = CWND/2 on triple dupack
- TCP-newReno
  - TCP-Reno + improved fast recovery
- TCP-SACK
  - incorporates selective acknowledgements

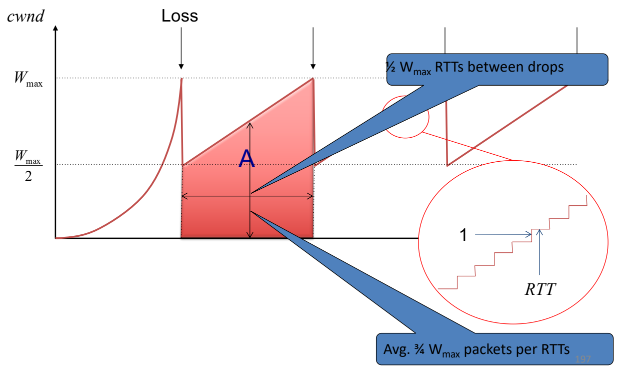
Our default assumption

Interoperability

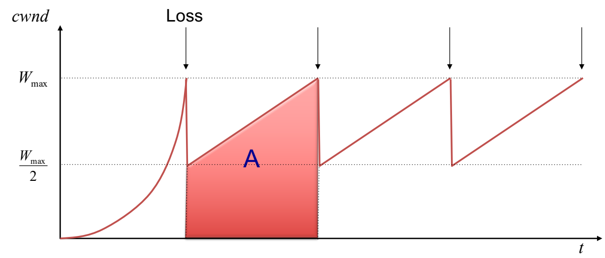
- How can all these algorithms coexist? Don't we need a single, uniform standard?
- What happens if I'm using Reno and you are using Tahoe, and we try to communicate?

TCP Throughput Equation

A Simple Model for TCP Throughput



A Simple Model for TCP Throughput



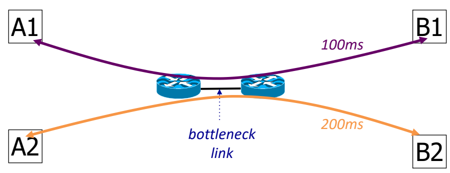
Packet drop rate, $p = 1/A$, where $A = \frac{3}{8} W_{max}^2$

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

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)

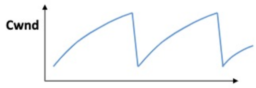
Implications (3): *Rate*-based CC

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

- 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 V TCP Reno

CUBIC Probing

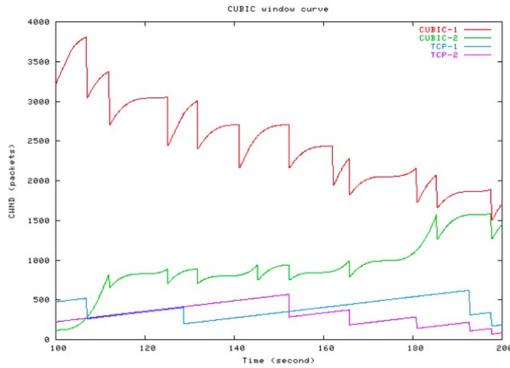


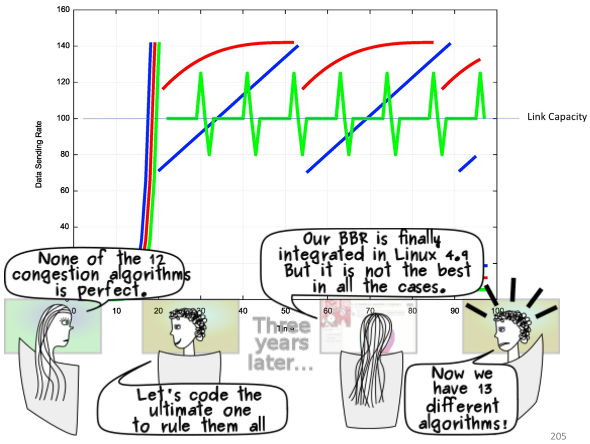
RENO Probing

Faster Increase
Sharp decrease

Slow Increase
Sharp decrease

New world of fairness….





Recap: TCP problems

- 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 if they're congested

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

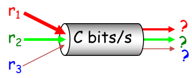
- Routers classify packets into "flows"
  - (For now) flows are packets between same source/destination
- Each flow has its own FIFO queue in router
- Router services flows in a fair fashion
  - When line becomes free, take packet from next flow in a fair order
- What does "fair" mean exactly?

Max-Min Fairness

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

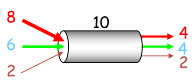
$$a_i = \min(f, r_i)$$

where f is the unique value such that $\mathrm{Sum}(a_i) = C$



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: $f = 4$



$f = 4$:
min(8, 4) = 4
min(6, 4) = 4
min(2, 4) = 2

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 $\mathrm{Sum}(a_i) = C$
- 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

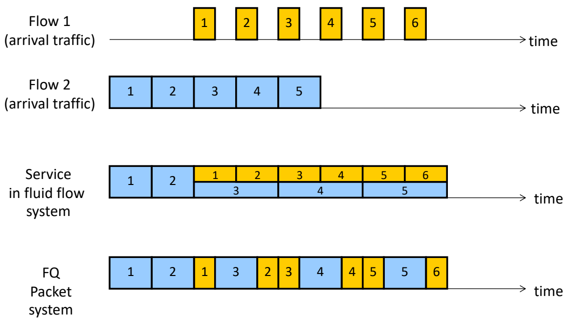
How do we deal with packets of different sizes?

- 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)

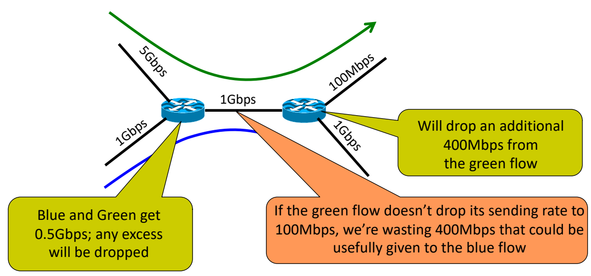
- Think of it as an implementation of round-robin generalized to the case where not all packets are equal sized
- Weighted fair queuing (WFQ): assign different flows different shares
- Today, some form of WFQ implemented in almost all routers
  - Not the case in the 1980-90s, when CC was being developed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

TCP in detail

- What does TCP do?
  - ARQ windowing, set-up, tear-down
- Flow Control in TCP
- Congestion Control in TCP
  - AIMD, Fast-Recovery, Throughput
- Limitations of TCP Congestion Control
- Router-assisted Congestion Control (eg ECN)

Transport Recap

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

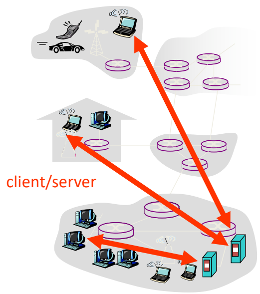
- UDP:
  - Minimalist - multiplexing and error detection

- TCP:
  - somewhat hacky
  - but practical/deployable
  - good enough to have raised the bar for the deployment of new, more optimal, approaches
  - though the needs of datacenters might 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
- Multimedia Applications (SIP)
  - One day (more…)…
- P2P Networks
  - Every device serves

Client-server paradigm reminder



server:
- always-on host
- permanent IP address
- server farms for scaling

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

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

Mapping from Names to Addresses

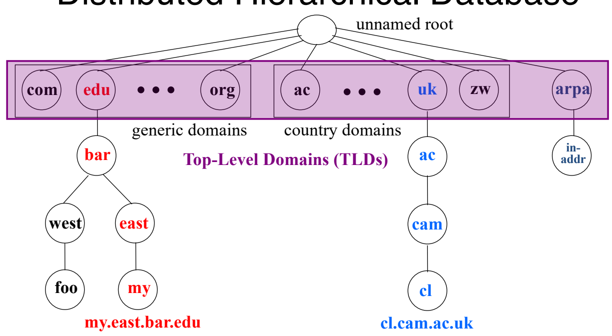
- Originally: per-host file /etc/hosts*
  - SRI (Menlo Park) kept master copy
  - Downloaded regularly
  - Flat namespace
- Single server not resilient, doesn't scale
  - Adopted a distributed hierarchical system
- Two intertwined hierarchies:
  - Infrastructure: hierarchy of DNS servers
  - Naming structure: www.bbc.co.uk

**C:\Windows\System32\drivers\etc\hosts for recent windows*

Domain Name System (DNS)

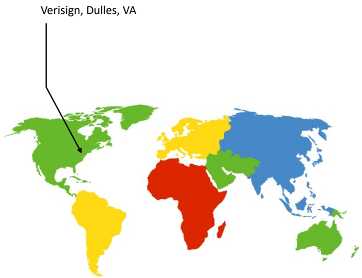
- Top of hierarchy: Root
  - Location hardwired into other servers
- Next Level: Top-level domain (TLD) servers
  - .com, .edu, etc.
  - .uk, .au, .to, etc.
  - Managed professionally
- Bottom Level: Authoritative DNS servers
  - Actually do the mapping
  - Can be maintained locally or by a service provider

Distributed Hierarchical Database



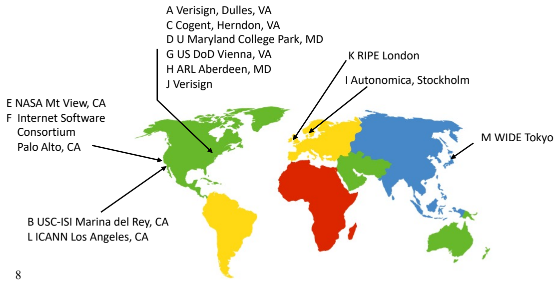
DNS Root

- Located in Virginia, USA
- How do we make the root scale?



DNS Root Servers

- 13 root servers (see http://www.root-servers.org/)
  - Labeled A through M
- Does this scale?



DNS Root Servers

- 13 root servers (see http://www.root-servers.org/)
  - Labeled A through M
- Replication via any-casting (localized routing for addresses)

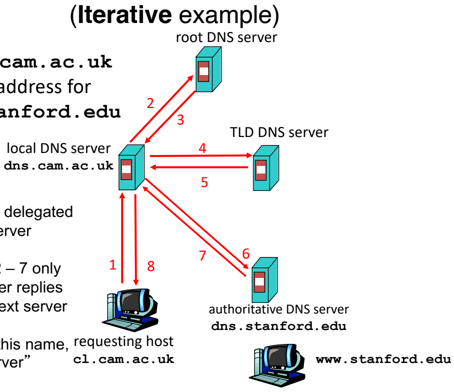
Using DNS

- Two components
  - Local DNS servers
  - Resolver software on hosts
- Local DNS server ("default name server")
  - Usually near the endhosts that use it
  - Local hosts configured with local server (e.g., /etc/resolv.conf) or learn server via DHCP
- Client application
  - Extract server name (e.g., from the URL)
  - Do gethostbyname() to trigger resolver code

How Does Resolution Happen?

(**Iterative** example)

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



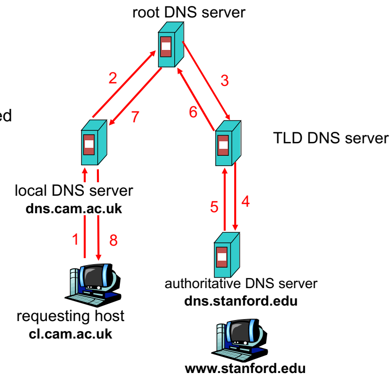
iterated query:

- 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

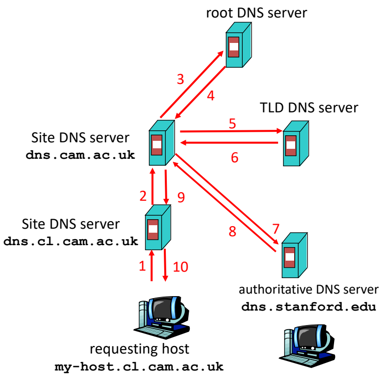
recursive query:

- 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
- Iterative 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 can greatly reduce overhead
  - The top-level servers very rarely change
  - Popular sites (e.g., www.bbc.co.uk) visited often
  - Local DNS server often has the information cached
- How DNS caching works
  - DNS servers cache responses to queries
  - Responses include a "time to live" (TTL) field
  - Server deletes cached entry after TTL expires

Negative Caching

- Remember things that don't work
  - Misspellings like *bbcc.co.uk* and *www.bbc.com.uk*
  - These can take a long time to fail the first time
  - Good to remember that they don't work
  - … so the failure takes less time the next time around
- But: negative caching is optional
  - And not widely implemented

Reliability

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

Invalid queries categories

From https://www.caida.org/publications/presentations/2008/wide_castro_root_servers/wide_castro_root_servers.pdf

- Unused query class:
  - Any class not in IN, CHAOS, HESIOD, NONE or ANY
- A-for-A: A-type query for a name is already a IPv4 Address
  - <IN, A, 192.16.3.0>
- Invalid TLD: a query for a name with an invalid TLD
  - <IN, MX, localhost.lan>
- Non-printable characters:
  - <IN, A, www.ra^B.us.>
- Queries with '_':
  - <IN, SRV, _ldap._tcp.dc._msdcs.SK0530-K32-1.>
- RFC 1918 PTR:
  - <IN, PTR, 171.144.144.10.in-addr.arpa.>
- Identical queries:
  - a query with the same class, type, name and id (during the whole period)
- Repeated queries:
  - a query with the same class, type and name
- Referral-not-cached:
  - a query seen with a referral previously given.

Invalid TLD

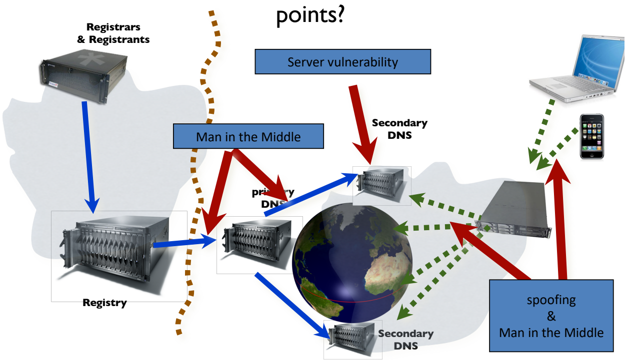
From https://www.caida.org/publications/presentations/2008/wide_castro_root_servers/wide_castro_root_servers.pdf

- Queries for invalid TLD represent 22% of the total traffic at the roots
  - 20.6% during DITL 2007
- Top 10 invalid TLD represent 10.5% of the total traffic
- RFC 2606 reserves some TLD to avoid future conflicts
- We propose:
  - Include some of these TLD (local, lan, home, localdomain) to RFC 2606
  - Encourage cache implementations to answer queries for RFC 2606 TLDs locally (with data or error)

TLD	Percentage of total queries	
	2007	2008
local	5.018	5.098
belkin	0.436	0.781
localhost	2.205	0.710
lan	0.509	0.679
home	0.321	0.651
invalid	0.602	0.623
domain	0.778	0.550
localdomain	0.318	0.332
wpad	0.183	0.232
corp	0.150	0.231

awm22: at least WORKGROUP is no longer here!
It was the top in valid TLD for years…

Data flow through the DNS
Where are the vulnerable points?



DNS and 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!

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

- Scaling the key signing and key distribution

Solution: Using the DNS to Distribute Keys

- Distributing keys through 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

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 endeavor
  - 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 Components

- Infrastructure:
  - Clients
  - Servers
  - Proxies
- Content:
  - Individual objects (files, etc.)
  - Web sites (coherent collection of objects)
- Implementation
  - HTML: formatting content
  - URL: naming content
  - HTTP: protocol for exchanging content

    Any content not just HTML!

HTML: HyperText Markup Language

- A *Web page* has:
  - Base HTML file
  - Referenced objects (*e.g.*, images)
- HTML has several functions:
  - Format text
  - Reference images
  - Embed *hyperlinks* (HREF)

URL Syntax

protocol : //hostname[:port] /directorypath /resource

protocol	http, ftp, https, smtp, rtsp, *etc.*
hostname	DNS name, IP address
port	Defaults to protocol's standard port *e.g.* http: 80 https: 443
directory path	Hierarchical, reflecting file system
resource	Identifies the desired resource Can also extend to program executions: `http://us.f413.mail.yahoo.com/ym/ShowLetter?box=%40B%40Bulk&MsgId=2604_1744106_29699_1123_1261_0_28917_3552_1289957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&head=b`

HyperText Transfer Protocol (HTTP)

- Request-response protocol
- Reliance on a global namespace
- Resource *metadata*
- *Stateless*
- ASCII format (ok this changed….)

$ telnet www.cl.cam.ac.uk 80
GET /win HTTP/1.0
<blank line, i.e., CRLF>

Steps in HTTP Request

- HTTP Client initiates TCP connection to server
  - SYN
  - SYNACK
  - ACK
- Client sends HTTP request to server
  - Can be piggybacked on TCP's ACK
- HTTP Server responds to request
- Client receives the request, terminates connection
- TCP connection termination exchange

How many RTTs for a single request?

Client-Server Communication

- two types of HTTP messages: *request, response*
- HTTP request message: (GET POST HEAD ….)

request line (GET, POST, HEAD commands)

header lines

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

(extra carriage return, line feed)

Carriage return, line feed indicates end of message

HTTP response message

status line (protocol status code status phrase)

header lines

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

data, e.g., requested HTML file

Different Forms of Server Response

- Return a file
  - URL matches a file (*e.g.*, `/www/index.html`)
  - Server returns file as the response
  - Server generates appropriate response header
- Generate response dynamically
  - URL triggers a program on the server
  - Server runs program and sends output to client
- Return meta-data with no body

HTTP Resource Meta-Data

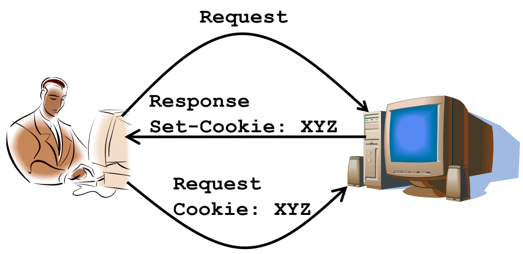
- Meta-data
  - Info *about* a resource, stored as a separate entity
- Examples:
  - Size of resource, last modification time, type of content
- Usage example: Conditional GET Request
  - Client requests object “**`If-modified-since`**”
  - If unchanged, “**`HTTP/1.1 304 Not Modified`**”
  - No body in the server’s response, only a header

HTTP is *Stateless*

- Each request-response treated independently
  - Servers *not* required to retain state
- **Good**: Improves scalability on the server-side
  - Failure handling is easier
  - Can handle higher rate of requests
  - Order of requests doesn’t matter
- **Bad**: Some applications need persistent state
  - Need to uniquely identify user or store temporary info
  - *e.g.,* Shopping cart, user profiles, usage tracking, …

State in a Stateless Protocol:
Cookies

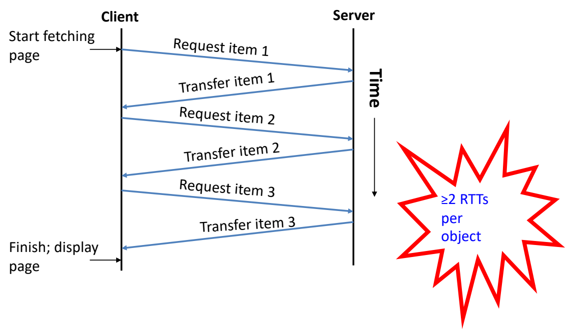
- *Client-side* state maintenance
  - Client stores small(?) state on behalf of server
  - Client sends state in future requests to the server
- Can provide authentication



HTTP Performance

- Most Web pages have multiple objects
  - *e.g.,* HTML file and a bunch of embedded images
- How do you retrieve those objects (naively)?
  - *One item at a time*
- Put stuff in the optimal place?
  - *Where is that precisely?*
    - ***Enter the Web cache and the CDN***

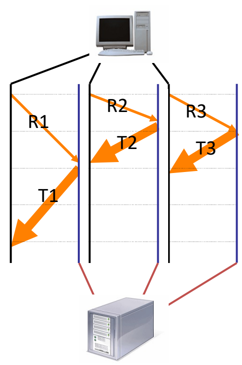
Fetch HTTP Items: Stop & Wait



Improving HTTP Performance:
Concurrent Requests & Responses

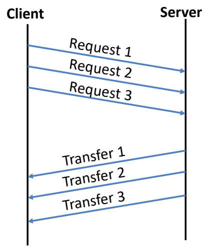
- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses

- Client = ☺
- Server = ☺
- Network = ☹ Why?



Improving HTTP Performance:
Pipelined Requests & Responses

- *Batch* requests and responses
  - Reduce connection overhead
  - Multiple requests sent in a single batch
  - Maintains order of responses
  - Item 1 always arrives before item 2
- How is this different from concurrent requests/responses?
  - Single TCP connection



Improving HTTP Performance:
Persistent Connections

- Enables multiple transfers per connection
  - Maintain TCP connection across multiple requests
  - Including transfers subsequent to current page
  - Client or server can tear down connection
- Performance advantages:
  - Avoid overhead of connection set-up and tear-down
  - Allow TCP to learn more accurate RTT estimate
  - Allow TCP congestion window to increase
  - i.e., leverage previously discovered bandwidth
- Default in HTTP/1.1

HTTP *evolution*

- 1.0 – one object per TCP: simple but slow
- Parallel connections - multiple TCP, one object each: wastes b/w, may be svr limited, out of order
- 1.1 pipelining – aggregate retrieval time: ordered, multiple objects sharing single TCP
- 1.1 persistent – aggregate TCP overhead: lower overhead in time, increase overhead at ends (e.g., when should/do you close the connection?)

Scorecard: Getting n Small Objects

Time dominated by latency

- One-at-a-time: ~2n RTT
- Persistent: ~ (n+1)RTT
- M concurrent: ~2[n/m] RTT
- Pipelined: ~2 RTT
- Pipelined/Persistent: ~2 RTT first time, RTT later

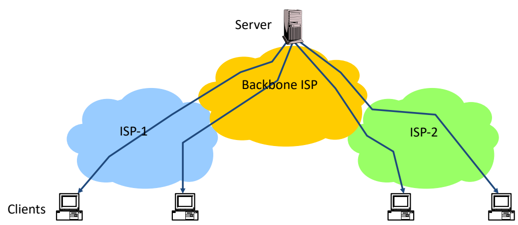
Scorecard: Getting n Large Objects

Time dominated by bandwidth

- One-at-a-time: ~ nF/B
- M concurrent: ~ [n/m] F/B
  - assuming shared with large population of users
- Pipelined and/or persistent: ~ nF/B
  - The only thing that helps is getting more bandwidth..

Improving HTTP Performance:
Caching

- Many clients transfer the same information
  - Generates redundant server and network load
  - Clients experience unnecessary latency



Improving HTTP Performance:
Caching: How

- Modifier to GET requests:
  - `If-modified-since` – returns “not modified” if resource not modified since specified time
- Response header:
  - `Expires` – how long it’s safe to cache the resource
  - `No-cache` – ignore all caches; always get resource directly from server

Improving HTTP Performance:
Caching: Why

- Motive for placing content closer to client:
  - User gets better response time
  - Content providers get happier users
    - Time is money, really!
  - Network gets reduced load
- Why does caching work?
  - Exploits *locality of reference*
- How well does caching work?
  - Very well, up to a limit
  - Large overlap in content
  - But many unique requests

Improving HTTP Performance:
Caching on the Client

Example: Conditional GET Request

- Return resource only if it has changed at the server
  - Save server resources!

Request from client to server:

```
GET /~awm22/win HTTP/1.1
Host: www.cl.cam.ac.uk
User-Agent: Mozilla/4.03
If-Modified-Since: Sun, 27 Aug 2006 22:25:50 GMT
<CRLF>
```

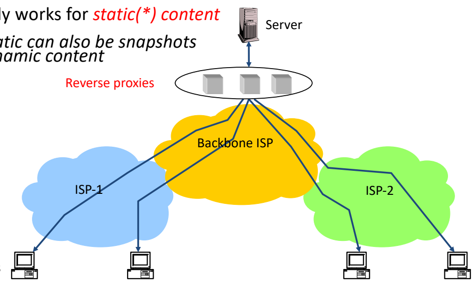
- How?
  - Client specifies “if-modified-since” time in request
  - Server compares this against “last modified” time of desired resource
  - Server returns “304 Not Modified” if resource has not changed
  - …. or a “200 OK” with the latest version otherwise

Improving HTTP Performance:
Caching with Reverse Proxies

Cache documents close to **server**
→ decrease server load

- Typically done by content providers
- Only works for *static(*) content*

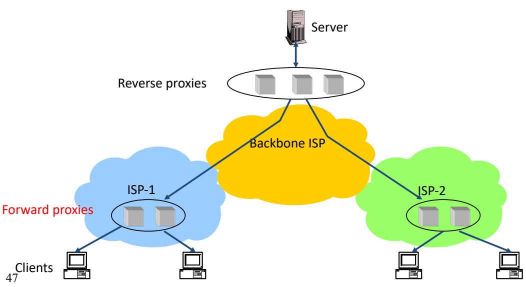
(*) static can also be snapshots of dynamic content



Improving HTTP Performance:
Caching with Forward Proxies

Cache documents close to **clients**
→ reduce network traffic and decrease latency

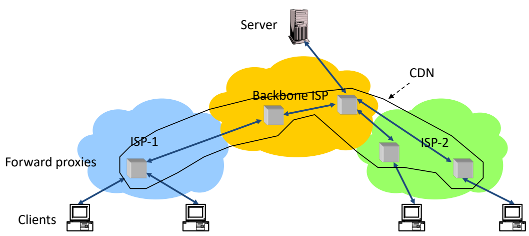
- Typically done by ISPs or corporate LANs



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

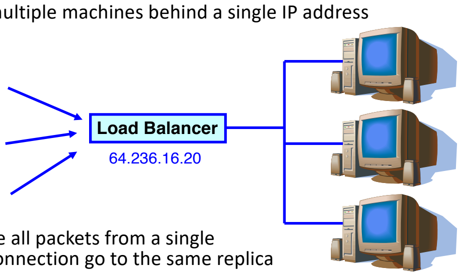
- Multiple Web sites on a single machine
  - Hosting company runs the Web server on behalf of multiple sites (*e.g.*, www.foo.com and www.bar.com)
- Problem: `GET /index.html`
  - `www.foo.com/index.html` or `www.bar.com/index.html`?
- Solutions:
  - Multiple server processes on the same machine
    - 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

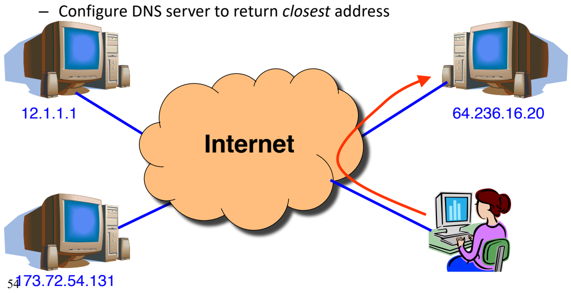
- Single IP address, multiple machines
  - Run multiple machines behind a single IP address



  - 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
- CDN using anycast
  same address from DNS name but local routes
- CDN based on rewriting HTML URLs
  (akami example just covered – akami uses DNS too)

After HTTP/1.1

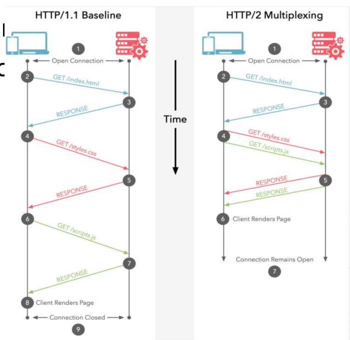
SPDY (speedy) and its moral successor HTTP/2

- Binary protocol
  – More efficient to parse
  – More compact on the wire
  – Much less error prone as compared
  – to textual protocols

After HTTP/1.1

SPDY (speedy) a

- Binary protoco
- Multiplexing
  – Interleaved

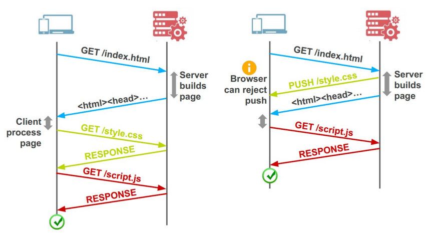


After HTTP/1.1

SPDY (speedy) and its moral successor HTTP/2

- Binary protocol
- Multiplexing
- Priority control over Frames
- Header Compression
- Server Push
  – Proactively push stuff to client that it will need

After HTTP/1.1



- Server Push
  – Proactively push stuff to client that it will need

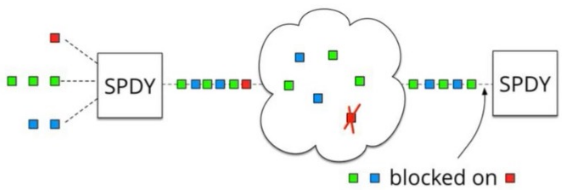
After HTTP/1.1

SPDY (speedy) and its moral successor HTTP/2

- Binary protocol
- Multiplexing
- Priority control over Frames
- Header Compression
- Server Push

SPDY

- SPDY + HTTP/2: One single TCP connection instead of multiple
- Downside: Head of line blocking
- In TCP, packets need to be processed in



Add QUIC and stir…
Quick UDP Internet Connections

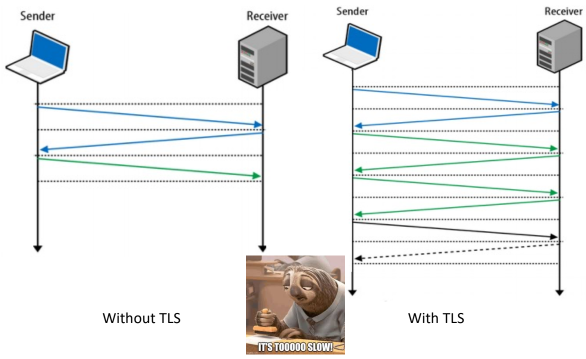
Objective: Combine speed of UDP protocol with TCP's reliability

- 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:

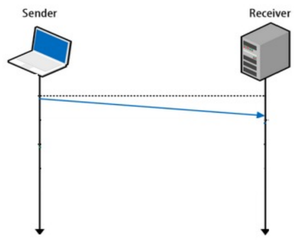
- Reliable transport over UDP (seriously)
- Uses FEC
- Default crypto
- Restartable connections

3-Way Handshake



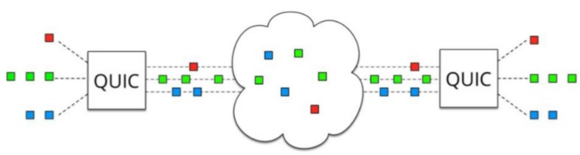
UDP

- Fire and forget
  - Less time spent to validate packets
  - Downside - no reliability, this has to be built 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 is combining best parts of HTTP/2 over UDP:
  - Multiplexing on top of non-blocking transport protocol



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 when watching videos over 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

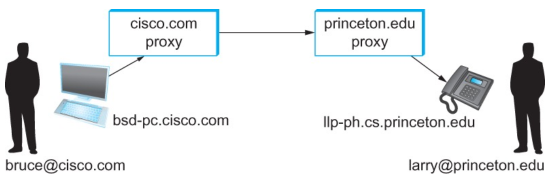
Honda M. et al. "**Is it still possible to extend TCP?**", IMC'11
https://dl.acm.org/doi/abs/10.1145/2068816.2068834

SIP – Session Initiation Protocol

Session?

Anyone smell an OSI / ISO standards document burning?

SIP - VoIP



Establishing communication through SIP proxies.

SIP?

- SIP – bringing the fun/complexity of telephony to the Internet
  - User location
  - User availability
  - User capabilities
  - Session setup
  - Session management
    - (e.g. "call forwarding")

H.323 – ITU

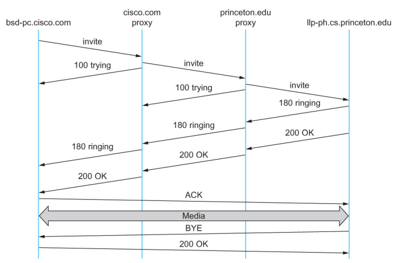
- Why have one standard when there are at least two….

- The full H.323 is hundreds of pages
  - The protocol is known for its complexity – an ITU hallmark

- SIP is not much better

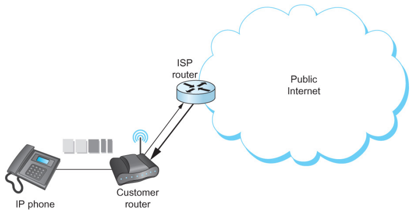
  - IETF grew up and became the ITU….

Multimedia Applications



Message flow for a basic SIP session

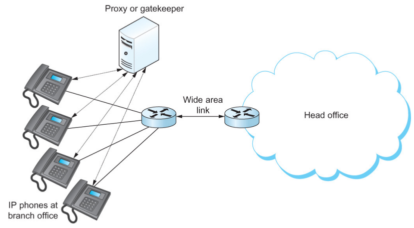
The (still?) missing piece:
Resource Allocation for Multimedia Applications



I can 'differentiate' VoIP from data but…
I can only control data going into the Internet

Multimedia Applications

- Resource Allocation for Multimedia Applications

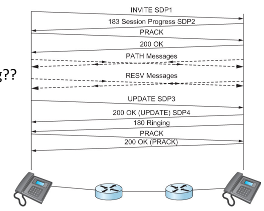


Admission control using session control protocol.

Resource Allocation for Multimedia Applications

Coming soon… ~~1995~~ ~~2000~~ ~~2010~~ ~~2020~~ who are we kidding??

Co-ordination of SIP signaling and resource reservation.



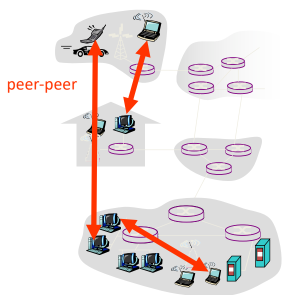
So where does it happen?
Inside single institutions or *domains of control…..*
(Universities, Hospitals, big corp…)

What about my aDSL/CABLE/etc it combines voice and data?
Phone company **controls** the multiplexing on the line
and throughout their own network too…… everywhere else is *best-effort*

Every host is a server:
Peer-2-Peer

Pure P2P architecture

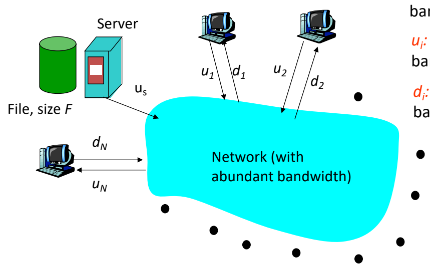
- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses



- Three topics:
  - File distribution
  - Searching for information
  - Case Study: Skype

File Distribution: Server-Client vs P2P

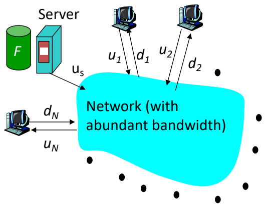
Question : How much time to distribute file from one server to N *peers*?



u_s: server upload bandwidth
u_i: peer i upload bandwidth
d_i: peer i download bandwidth

File distribution time: server-client

- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download

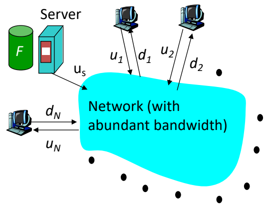


Time to distribute F to N clients using client/server approach $= d_{cs} = \max \{ NF/u_s, \; F/\min_i(d_i) \}$

increases linearly in N (for large N)

File distribution time: P2P

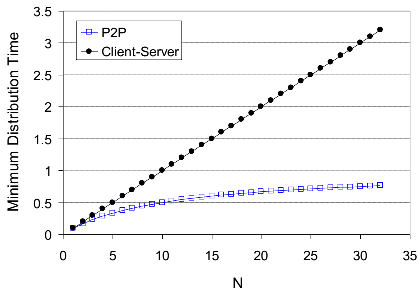
- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \sum u_i$



$d_{P2P} = \max \{ F/u_s, \; F/\min_i(d_i), \; NF/(u_s + \sum u_i) \}$

Server-client vs. P2P: example

Client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$



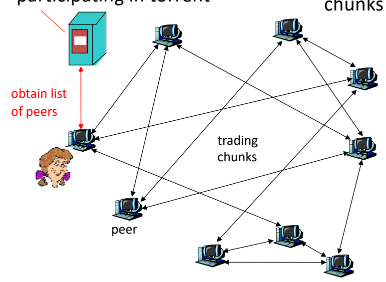
File distribution: BitTorrent*

*rather old BitTorrent

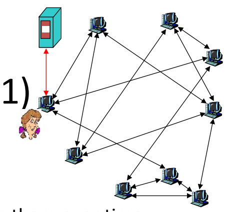
- P2P file distribution

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file



BitTorrent (1)



- file divided into 256KB *chunks*.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers.
- peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain

BitTorrent (2)

Pulling Chunks

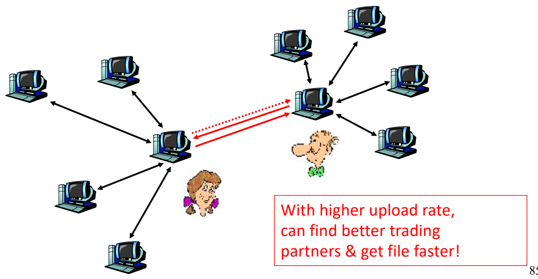
- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - rarest first

Sending Chunks: tit-for-tat

- Alice sends chunks to four neighbors currently sending her chunks *at the highest rate*
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - newly chosen peer may join top 4
  - "optimistically unchoke"

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



Distributed Hash Table (DHT)

- DHT = distributed P2P database
- Database has (key, value) pairs;
  - key: ss number; value: human name
  - key: content type; value: IP address
- Peers query DB with key
  - DB returns values that match the key
- Peers can also insert (key, value) peers

Distributed Hash Table (DHT)

- DHT = distributed P2P database
- Database has (key, value) pairs;
  - key: ss number; value: human name
  - key: content type; value: IP address
- Peers query DB with key
  - DB returns values that match the key
- Peers can also insert (key, value) peers

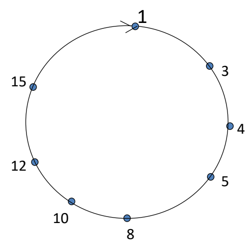
DHT Identifiers

- Assign integer identifier to each peer in range $[0,2^n-1]$.
  - Each identifier can be represented by n bits.
- Require each key to be an integer in same range.
- To get integer keys, hash original key.
  - eg, key = h("Game of Thrones season 29")
  - This is why they call it a distributed "hash" table

How to assign keys to peers?

- Central issue:
  - Assigning (key, value) pairs to peers.
- Rule: assign key to the peer that has the closest ID.
- Convention in lecture: closest is the immediate successor of the key.
- Ex: n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1

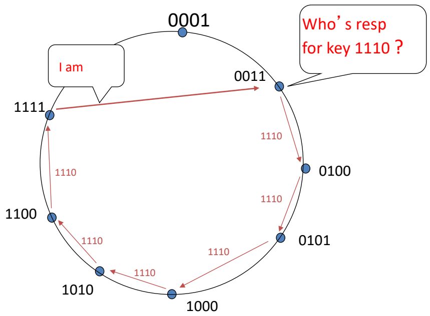
Circular DHT (1)



- Each peer *only* aware of immediate successor and predecessor.
- "Overlay network" – logical structure

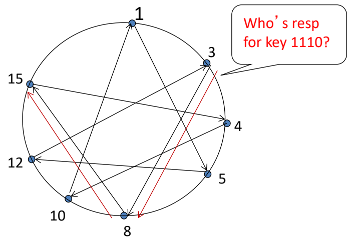
Circle DHT (2)

O(N) messages on avg to resolve query, when there are N peers



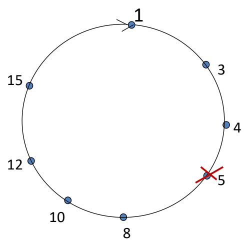
Define closest as closest successor

Circular DHT with Shortcuts



- Each peer keeps track of IP addresses of predecessor, successor, short cuts.
- Reduced from 6 to 2 messages.
- Possible to design shortcuts so O(log N) neighbors, O(log N) messages in query

Peer Churn

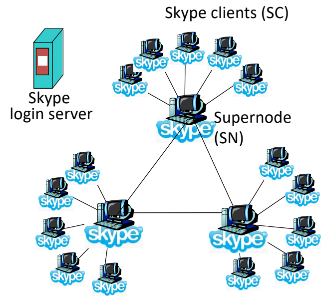


•To handle peer churn, require each peer to know the IP address of its two successors.
• Each peer periodically pings its two successors to see if they are still alive.

- Peer 5 abruptly leaves
- Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8' s immediate successor its second successor.
- What if peer 13 wants to join?

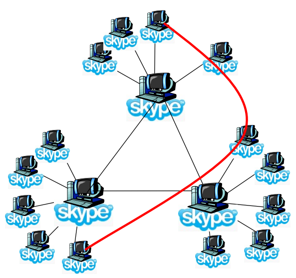
P2P Case study: Skype (pre-Microsoft)

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SNs
- Index maps usernames to IP addresses; distributed over SNs



Peers as relays

- Problem when both Alice and Bob are behind "NATs".
  - NAT prevents an outside peer from initiating a call to insider peer
- Solution:
  - Using Alice' s and Bob' s SNs, Relay is chosen
  - Each peer initiates session with relay.
  - Peers can now communicate through NATs via relay



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
- Multimedia Applications (SIP)
  - Extremely hard to do better than worst-effort
- P2P Network examples