Computer Networking

Lent Term M/W/F 11-midday
LT1 in Gates Building

Slide Set 2

Andrew W. Moore
andrew.moore@cl.cam.ac.uk
January 2013

1

Topic 2 – Internet and Architecture

- Protocol Standardization
- Internet Philosophy and Tensions
- The architects process
  - How to break system into modules
  - Where modules are implemented
  - Where is state stored

2

Recall What is a protocol?

human protocols:

- "what's the time?"
- "I have a question"
- introductions

... specific msgs sent

... specific actions taken when msgs received, or other events

network protocols:

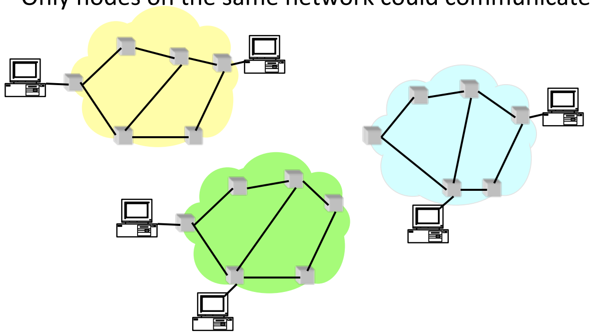
- machines rather than humans
- all communication activity in Internet governed by protocols

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

3

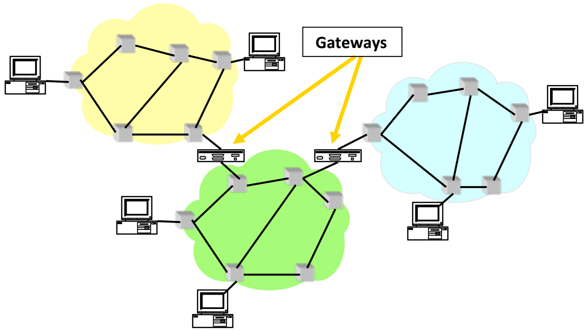
So many Standards Problem

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



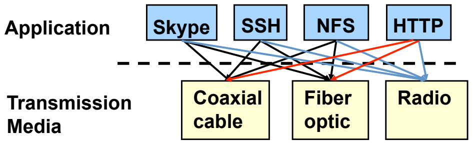
4

INTERnet Solution



5

A Multitude of Apps Problem

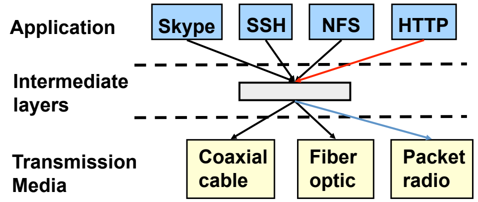


- Re-implement every application for every technology?
- No! But how does the Internet design avoid this?

6

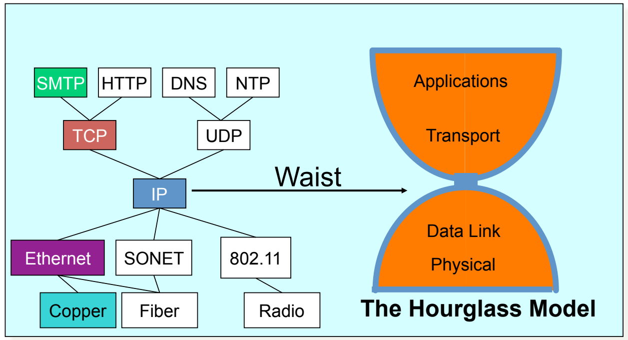
Solution: Intermediate Layers

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



7

The Internet *Hourglass*



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

8

Protocol Standardization

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

9

Internet Motto

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

David Clark

D. Clark, "The Design Philosophy of the DARPA Internet Protocols", Sigcomm'88, 106-114, Palo Alto, CA, Sept 1988.

10

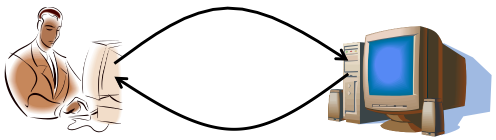
Alternative to Standardization?

- Have one implementation used by everyone
- Open-source projects
  - Which has had more impact, Linux or POSIX?
- Or just sole-sourced implementation
  - Skype, many P2P implementations, etc.

11

Client-Server Communication

- Client "sometimes on"
  - Initiates a request to the server when interested
  - E.g., Web browser on your laptop or cell phone
  - Doesn't communicate directly with other clients
  - Needs to know the server's address
- Server is "always on"
  - Services requests from many client hosts
  - E.g., Web server for the *www.cnn.com* Web site
  - Doesn't initiate contact with the clients
  - Needs a fixed, well-known address



12

Peer-to-Peer Designs

- No always-on server at the center of it all
  - Hosts can come and go, and change addresses
  - Hosts may have a different address each time
- Example: peer-to-peer file sharing
  - All hosts are both servers and clients!
  - Scalability by harnessing millions of peers
  - "self-scaling"
- Not just for file sharing!
  - This is how many datacenter applications are built
  - Better reliability, scalability, less management…
    - Sound familiar?

13

Internet Design Goals (Clark '88)

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

14

Connect Existing Networks

- Internet (e.g., IP) should be designed such that all current networks could support IP.

15

Robust

- As long as the network is not partitioned, two endpoints should be able to communicate
- Failures (excepting network partition) should not interfere with endpoint semantics

- *Very successful, not clear how relevant now*
- *Second notion of robustness is underappreciated*

16

Types of Delivery Services

- Use of the term "communication services" already implied an application-neutral network
- Built lowest common denominator service
  - Allow end-based protocols to provide better service
- Example: recognition that TCP wasn't needed (or wanted) by some applications
  - Separated TCP from IP, and introduced UDP

17

Variety of Networks

- Incredibly successful!
  - Minimal requirements on networks
  - No need for reliability, in-order, fixed size packets, etc.
  - A result of aiming for lowest common denominator
- IP over everything
  - Then: ARPANET, X.25, DARPA satellite network..
  - Now: ATM, SONET, WDM…

18

Decentralized Management

- Both a curse and a blessing
  - Important for easy deployment
  - Makes management hard today

19

Host Attachment

- Clark observes that cost of host attachment may be higher because hosts have to be smart
- But the administrative cost of adding hosts is very low, which is probably more important

20

Cost Effective

- Cheaper than telephone network
- But much more expensive than circuit switching
- Perhaps it is cheap where it counts (low-end) and more expensive for those who can pay….

21

Resource Accountability

- Failure!
  - No coordinated resource accounting
  - No coordinated resource management
  - No coordinated resource control
  - No coordinated resource ….

BUT Failure is information too

22

Real Goals

Internet Motto

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

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

23

Questions to think about….

- What priorities would a commercial design have?
- What would the resulting design look like?
- What goals are missing from this list?
- Which goals led to the success of the Internet?

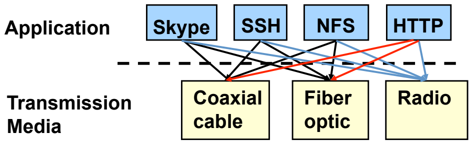
24

The Networking Dilemma

- Many different networking technologies
- Many different network applications
- How do you prevent incompatibilities?

25

The Problem

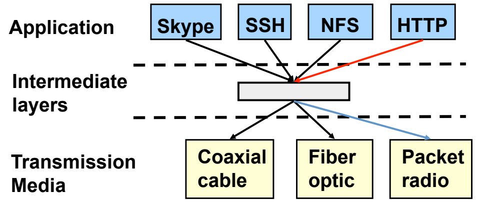


- Re-implement every application for every technology?
- No! But how does the Internet design avoid this?

26

Solution: Intermediate Layers

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



27

Network Architecture

- Architecture is <u>not</u> the implementation itself
- Architecture is how to organize/structure the elements of the system and their implementation
- What *interfaces* are supported?
  - Using what sort of abstractions
- *Where* functionality is implemented?
  - The modular design of the network

28

Computer System Modularity

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

29

Computer System Modularity (cnt' d)

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

30

Network System Modularity

Like software modularity, but:

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

31

Remember that slide!

- The relationship between architectural principles and architectural decisions is crucial to understand

32

Topic 3: The Data Link Layer

Our goals:

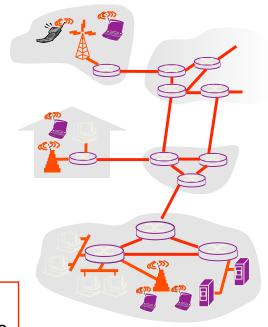
- understand principles behind data link layer services:
  (these are methods & mechanisms in your networking toolbox)
  - error detection, correction
  - sharing a broadcast channel: multiple access
  - link layer addressing
  - reliable data transfer, flow control: \
  - instantiation and implementation of various link layer technologies
  - Wired Ethernet (aka 802.3)
  - Wireless Ethernet (aka 802.11 WiFi)

2

Link Layer: Introduction

Some 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

3

Link Layer (Channel) Services

- *framing, link access:*
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - "MAC" addresses used in frame headers to identify source, dest
    - different from IP address!
- *reliable delivery between adjacent nodes*
  - we learned how to do this already (chapter 3)!
  - seldom used on low bit-error link (fiber, some twisted pair)
  - wireless links: high error rates
    - Q: why both link-level and end-end reliability?

4

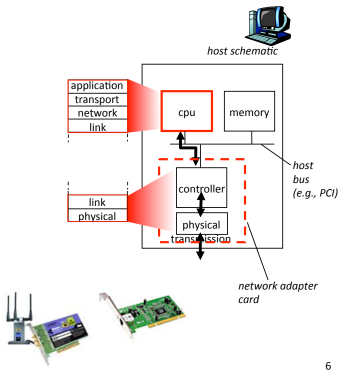
Link Layer (Channel) Services - 2

- *flow control:*
  - pacing between adjacent sending and receiving nodes
- *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
- *half-duplex and full-duplex*
  - with half duplex, nodes at both ends of link can transmit, but not at same time

5

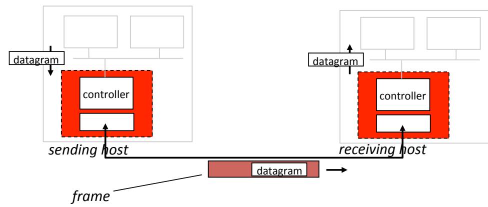
Where is the link layer implemented?

- 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



6

Adaptors Communicating

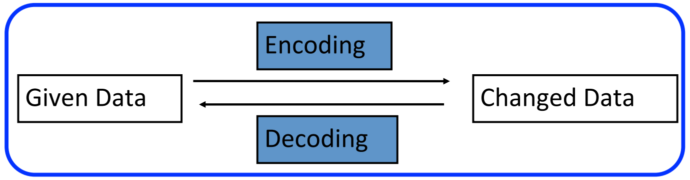


- sending side:
  - encapsulates datagram in frame
  - encodes data for the physical layer
  - adds error checking bits, provide reliability, flow control, etc.
- receiving side
  - decodes data from the physical layer
  - looks for errors, provide reliability, flow control, etc
  - extracts datagram,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

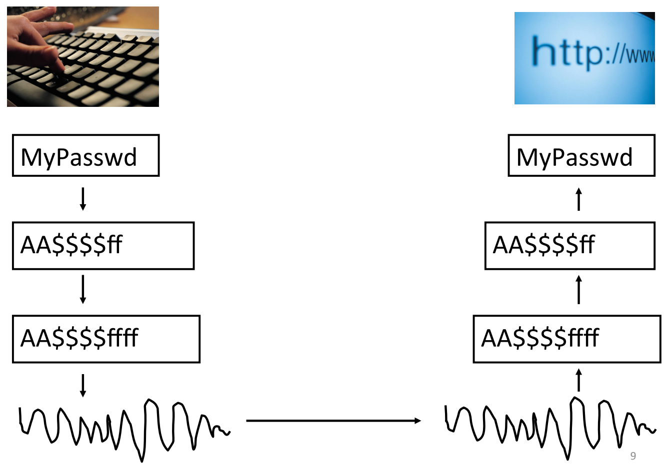
7

Coding – a channel function

Change the representation of data.



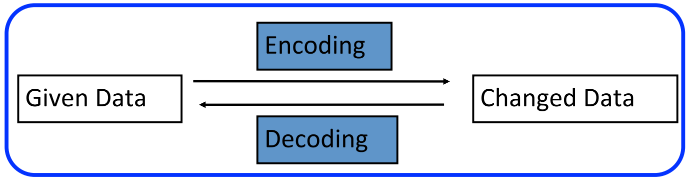
8



9

Coding

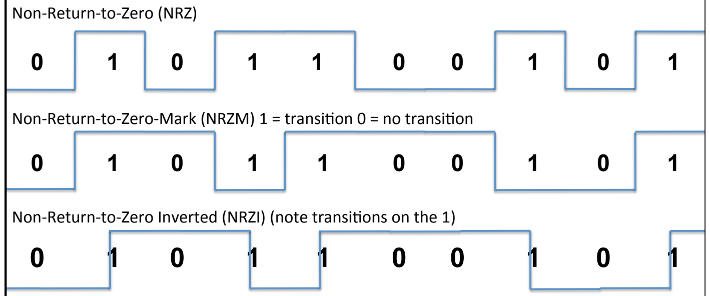
Change the representation of data.



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

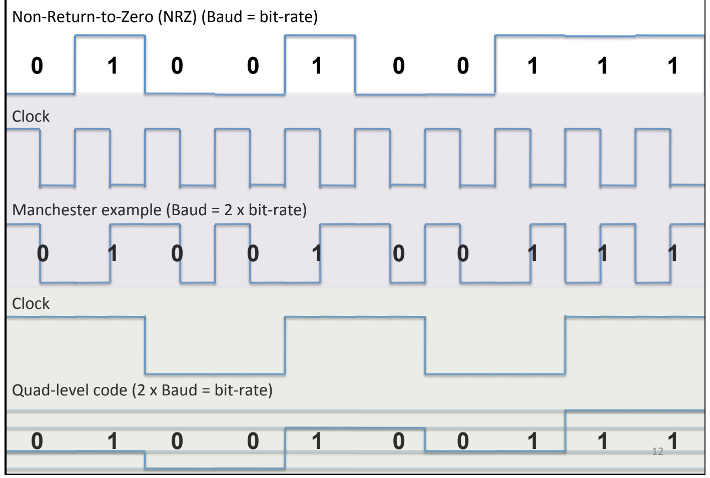
10

Line Coding Examples where Baud=bit-rate



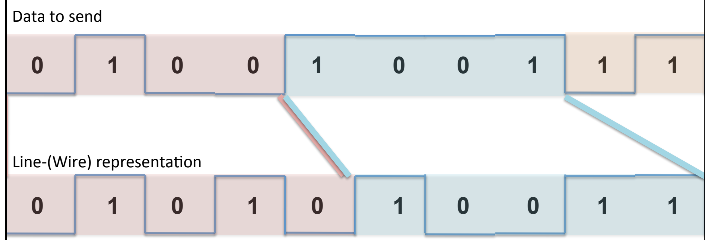
11

Line Coding Examples - II



12

Line Coding Examples - III



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

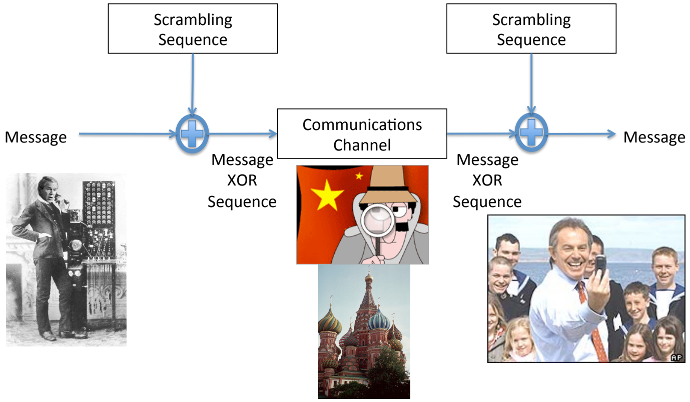
Name	4b	5b	Description
Q	-NONE-	00000	Quiet
I	-NONE-	11111	Idle
J	-NONE-	11000	SSD #1
K	-NONE-	10001	SSD #2
T	-NONE-	01101	ESD #1
R	-NONE-	00111	ESD #2
H	-NONE-	00100	Halt

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.

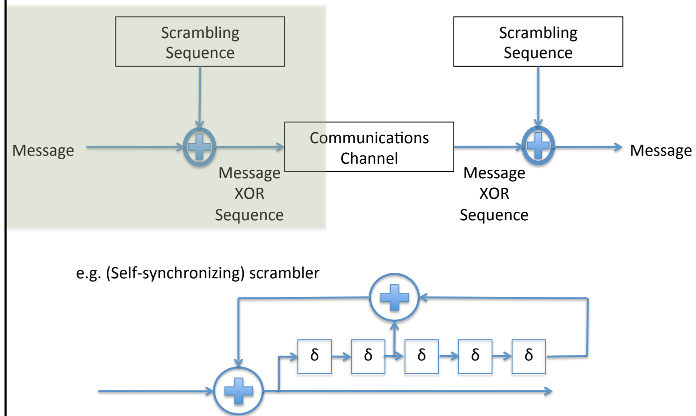
1Gbps uses an 8b/10b codec; encoding entire bytes at a time but with 25% overhead

Line Coding Examples - IV



14

Line Coding Examples - V



15

Line Coding Examples – VI (Hybrid)

…1001111011010100010001011001110100010100101101010010011101011110100…

…100111101101010001010001011001110100010100101101010010011101011110100…

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

Scramble / Transmit / Unscramble

…0100010110011101000101001011010100100111010111010010010111011101111000…

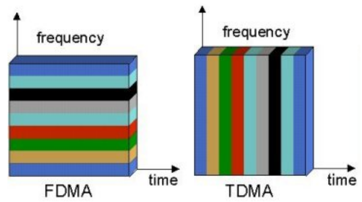
Identify (and remove) "start of frame/block/sequence"
This gives you the Byte-delineations for *free*

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)

16

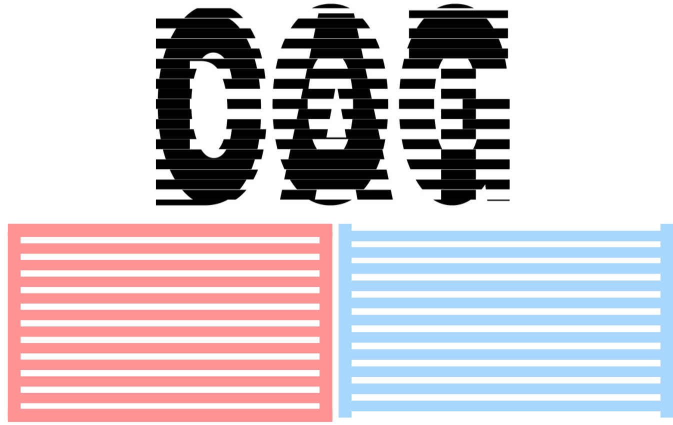
17

Multiple Access Mechanisms

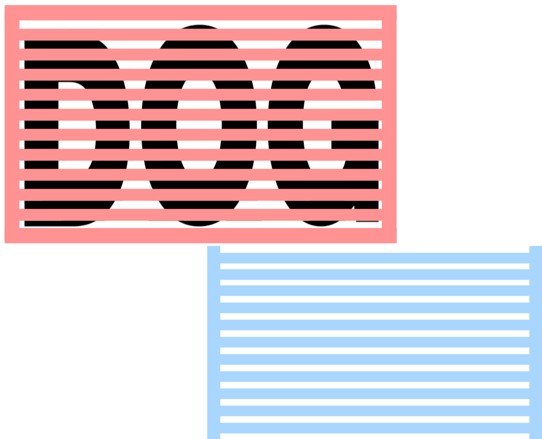


Each dimension is orthogonal (so may be trivially combined)
There are other dimensions too; can you think of them?

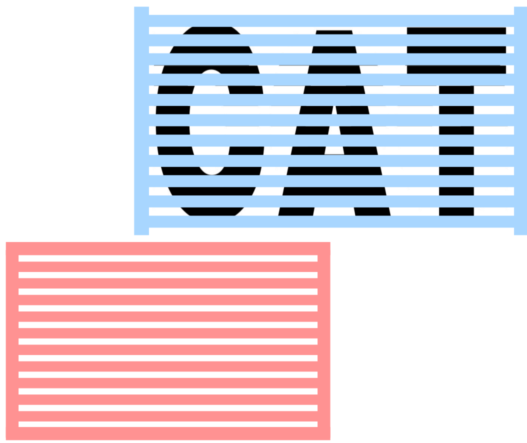
18



19



20



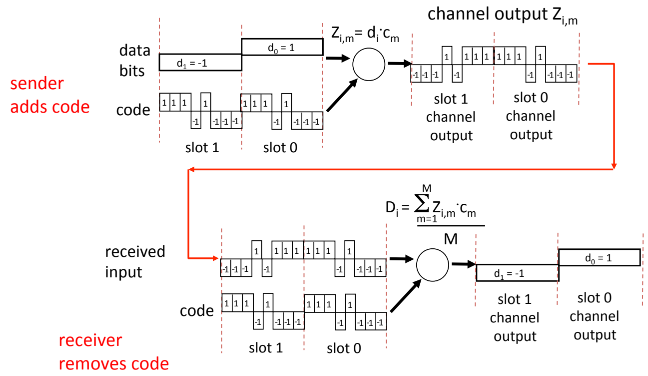
21

Code Division Multiple Access (CDMA)

- 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) X (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")

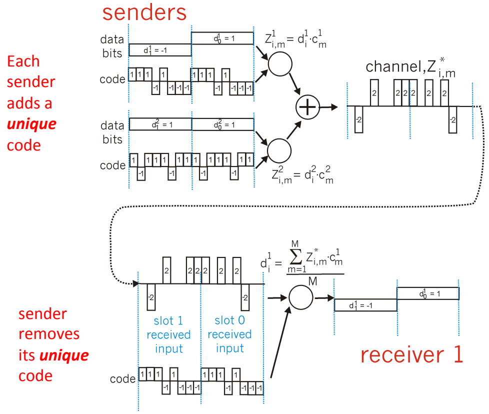
22

CDMA Encode/Decode



23

CDMA: two-sender interference



24

Coding Examples summary

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

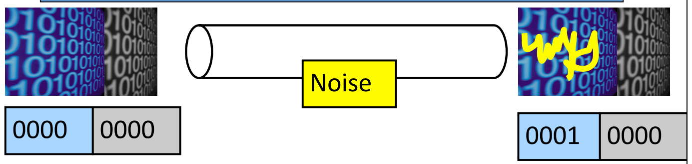
Like earlier coding schemes and error correction/detection; you can combine these
- e.g, 10Gb/s Ethernet may use a hybrid

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

25

Error Detection and Correction

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

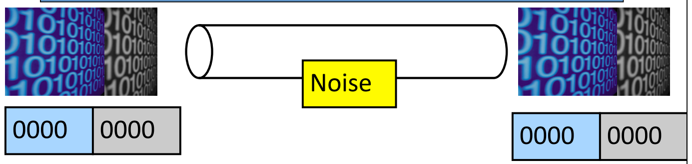


Basic Idea :
1. Add additional information to a message.
2. Detect an error and re-send a message.
   Or, fix an error in the received message.

26

Error Detection and Correction

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



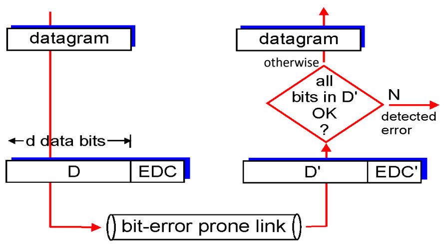
Basic Idea :
1. Add additional information to a message.
2. Detect an error and re-send a message.
   Or, fix an error in the received message.

27

Error Detection

EDC= Error Detection and Correction bits (redundancy = overhead)
D = Data protected by error checking, may include header fields

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

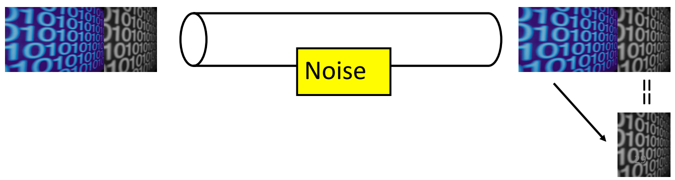


28

Error Detection Code

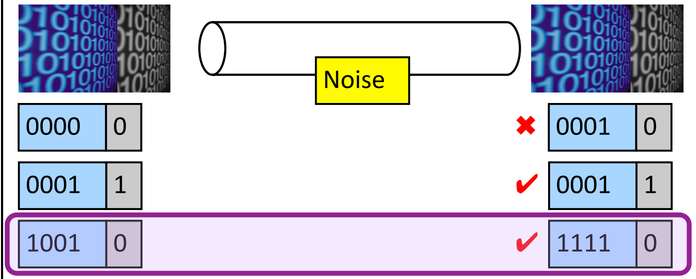
Sender:
Y = generateCheckBit(X);
send(XY);

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



Error Detection Code: Parity

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

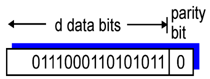


Problem: This simple parity cannot detect two-bit errors.

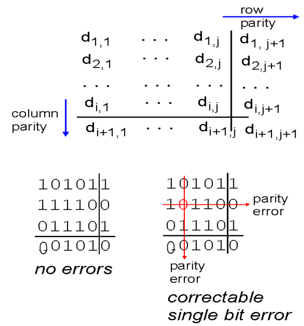
30

Parity Checking

Single Bit Parity:
Detect single bit errors



Two Dimensional Bit Parity:
Detect *and correct* single bit errors



31

Internet checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted packet (note: used at transport layer only)

Sender:
- treat segment contents as sequence of 1bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:
- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?*

32

Error Detection Code: CRC

- CRC means "Cyclic Redundancy Check".
- 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.

33

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 p97 §2.4.3

34

The divisor (G) – Secret sauce of CRC

- If the divisor were 100, instead of 101, data 1111 and 1001 would give the same check bit 00.
- Mathematical analysis about the divisor:
  - Last bit should be 1.
  - Should contain at least two 1's.
  - Should be divisible by 11.
- ATM, HDLC, Ethernet each use a CRC with well-chosen fixed divisors

Divisor analysis keeps mathematicians in jobs
(a branch of *pure* math: combinatorial mathematics)

35

Checksumming: Cyclic Redundancy Check recap

- view data bits, D, as a binary number
- choose r+1 bit pattern (generator), G
- goal: choose r CRC bits, R, such that
  - <D,R> exactly divisible by G (modulo 2)
  - receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
  - can detect all burst errors less than r+1 bits
- widely used in practice (Ethernet, 802.11 WiFi, ATM)

← d bits → ← r bits →

D: data bits to be sent	*R:* CRC bits

bit pattern

$D * 2^r$ XOR R — *mathematical formula*

36

CRC Another Example – this time with long division

Want:

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

equivalently:

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

equivalently:

if we divide $D \cdot 2^r$ by P, want remainder R

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

```
           101011
     1001 )101110000  → D
P ←        1001
            101
            000
            1010
            1001
             110
             000
             1100
             1001
              1010
              1001
               011
R ←
```

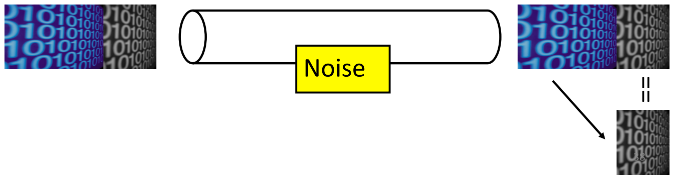
FYI: *in K&R P is called the Generator: G*

37

Error Detection Code becomes….

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

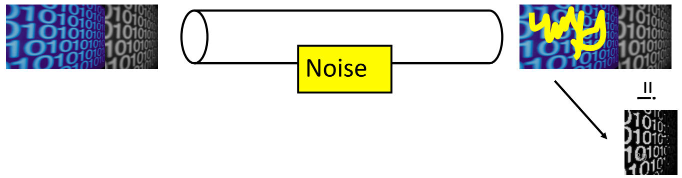
Receiver:
receive(X1Y1);
Y2=generateCheckBit(X1);
if (Y1 != Y2) ERROR;
else NOERROR



Forward Error Correction (FEC)

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

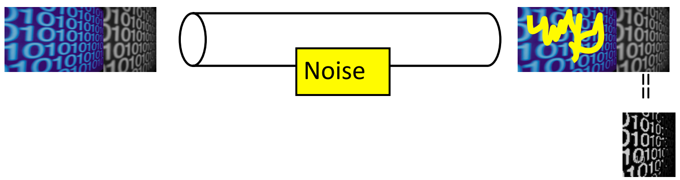
Receiver:
receive(X1Y1);
Y2=generateCheckBit(X1);
if (Y1 != Y2) FIXERROR(X1Y1);
else NOERROR



Forward Error Correction (FEC)

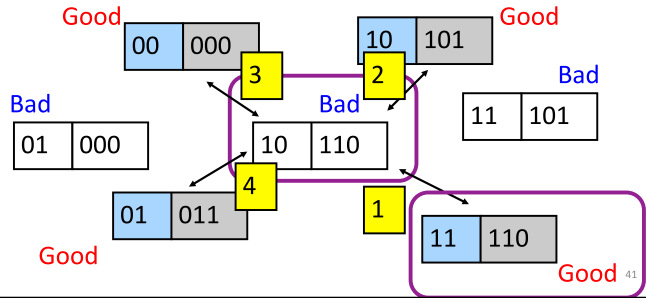
Sender:
Y = generateCheckBit(X);
send(XY);

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.

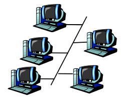
Usage:
- Correction: A lot of noise. Expensive to re-send.
- Detection: Less noise. Easy to re-send.
- Can be used together.

42

Multiple Access Links and Protocols

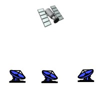
Two types of "links":

- 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)
  - 802.11 wireless LAN



shared wire (e.g., cabled Ethernet)

shared RF (e.g., 802.11 WiFi)



shared RF (satellite)

humans at a cocktail party (shared air, acoustical)

43

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

44

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

45

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

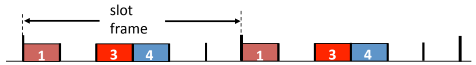
46

Channel Partitioning MAC protocols: TDMA

(time travel warning – we mentioned 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



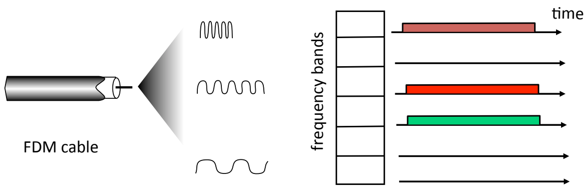
47

Channel Partitioning MAC protocols: FDMA

(time travel warning – we mentioned 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



48

"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

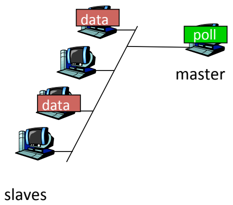
look for best of both worlds!

49

"Taking Turns" MAC protocols

Polling:

- master node "invites" slave nodes to transmit in turn
- typically used with "dumb" slave devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (master)

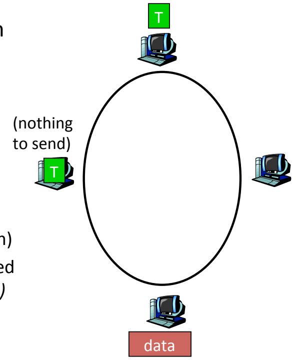


50

"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)*

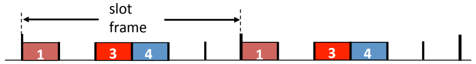


Cambridge students – this is YOUR heritage
Cambridge RING, Cambridge Fast RING,
Cambridge Backbone RING, these things gave us ATDM (and ATM)

51

ATM

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



In ATM a sender transmits labeled cells whenever necessary

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

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

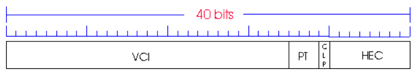
Use the media when you need it, but
ATM had virtual circuits and these needed setup....
Worse ATM had an utterly irrational size

52

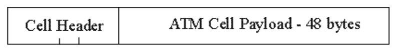
ATM Layer: ATM cell (size = best known stupid feature)

- 48-byte payload
  - Why?: small payload -> short cell-creation delay for digitized voice
  - halfway between 32 and 64 (compromise!)
- 5-byte ATM cell header (10% of payload)

Cell header

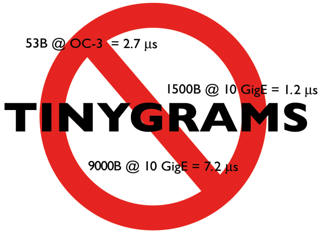


Cell format



53

ATM – redux, the irony (a 60 second sidetrack)



Size issues once plagued ATM
- too little time to do useful work

now plague the common Internet MTU

Even jumbo grams (9kB) are argued as *not big enough*

Consider issues
- default Ethernet CRC not robust for 9k packets
  - IPv6 checksum implications
- MTU discovery ugliness
  - (discovering MTU is hard anyway)

Make it big!
625kB @ 10 GigE = 500 µs
http://www.psc.edu/~mathis/MTU

- Is time-per-packet a sensible justification?

54

None of these are the "Internet way"…

(Bezerkely, 60's, free *stuff*, no G-man)

- Seriously; why not?
- What's wrong with
  - TDMA
  - FDMA
  - Polling
  - Token passing
  - ATM

Management. Suites. Rules. Schedules.
Signs, signs, everywhere a sign….

- Turn to random access
  - Optimize for the common case (no collision)
  - Don't avoid collisions, just recover from them….
    - Sound familiar?

What could possibly go wrong….

55

Random Access Protocols

- When node has packet to send
  - transmit at full channel data rate R.
  - no *a priori* coordination among nodes
- two or more transmitting nodes ➜ "collision",
- random access MAC protocol specifies:
  - how to detect collisions
  - how to recover from collisions (e.g., via delayed retransmissions)
- Examples of random access MAC protocols:
  - ALOHA and slotted ALOHA
  - CSMA, CSMA/CD, CSMA/CA

56

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)

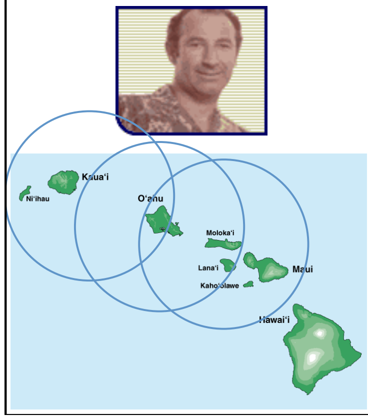
57

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

58

Where it all Started: AlohaNet



- Norm Abramson left Stanford to surf
- Set up first data communication system for Hawaiian islands
- Hub at U. Hawaii, Oahu
- Had two radio channels:
  - Random access:
    - Sites sending data
  - Broadcast:
    - Hub rebroadcasting data

59

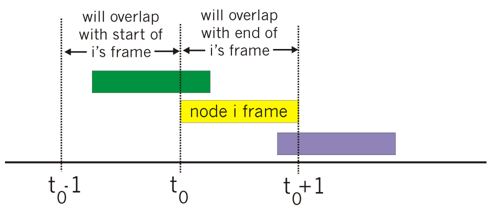
Aloha Signaling

- Two channels: random access, broadcast
- Sites send packets to hub (random)
  - If received, hub sends ACK (random)
  - If not received (collision), site resends
- Hub sends packets to all sites (broadcast)
  - Sites can receive even if they are also sending
- Questions:
  - When do you resend? Resend with probability p
  - How does this perform? Need a clean model….

60

Pure (unslotted) ALOHA

- unslotted Aloha: simple, no synchronization
- when frame first arrives
  - transmit immediately
- collision probability increases:
  - frame sent at t_0 collides with other frames sent in $[t_0-1, t_0+1]$



61

Pure Aloha efficiency

P(success by given node) = P(node transmits) ·

P(no other node transmits in $[p_0-1, p_0]$ ·

P(no other node transmits in $[p_0-1, p_0]$

$= p \cdot (1-p)^{N-1} \cdot (1-p)^{N-1}$

$= p \cdot (1-p)^{2(N-1)}$

… choosing optimum p and then letting n -> ∞ …

= 1/(2e) = .18

Best described as unspectacular; *but* better than what went before.

62

Slotted ALOHA

Assumptions

- All frames same size
- Time divided into equal slots (time to transmit a frame)
- Nodes are synchronized
- Nodes begin to transmit frames only at start of slots
- If multiple nodes transmit, nodes detect collision

Operation

- When node gets fresh data, transmits in next slot
- No collision: success!
- Collision: node retransmits with probability **p** until success

63

Slot-by-Slot Example

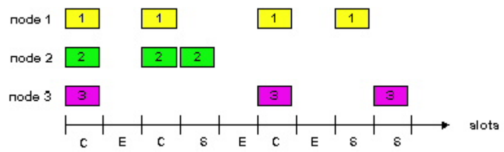
64

Efficiency of Slotted Aloha

- Suppose N stations have packets to send
  - Each transmits in slot with probability *p*
- Probability of successful transmission:
  by a particular node i: $S_i = p\,(1-p)^{(N-1)}$
  by any of N nodes: $S = N\,p\,(1-p)^{(N-1)}$
- What value of p maximizes prob. of success:
  - For fixed p, S ➔ 0 as N increases
  - But if p = 1/N, then S ➔ 1/e = 0.37 as N increases
- Max efficiency is only slightly greater than 1/3!

65

Pros and Cons of Slotted Aloha



Pros

- Single active node can continuously transmit at full rate of channel
- Highly decentralized: only need slot synchronization
- Simple

Cons

- Wasted slots:
  - Idle
  - Collisions
- Collisions consume entire slot
- Clock synchronization

66

Improving on Slotted Aloha

- Fewer wasted slots
  - *Need to decrease collisions and empty slots*
- Don't waste full slots on collisions
  - *Need to decrease time to detect collisions*
- Avoid need for synchronization
  - *Synchronization is hard to achieve*

67

CSMA (Carrier Sense Multiple Access)

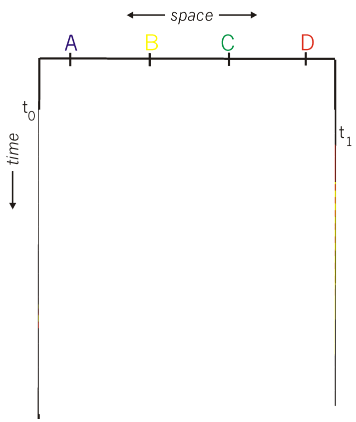
- CSMA: listen before transmit
  - If channel sensed idle: transmit entire frame
  - If channel sensed busy, defer transmission
- Human analogy: don't interrupt others!
- Does this eliminate all collisions?
  - No, because of nonzero propagation delay

68

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?

69

CSMA/CD (Collision Detection)

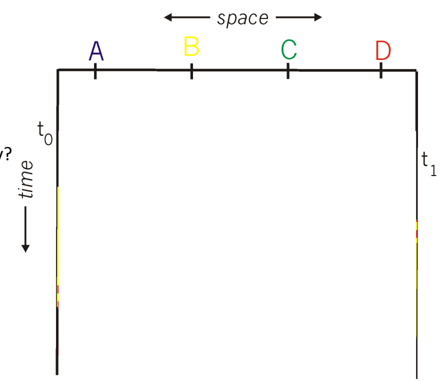
- CSMA/CD: carrier sensing, deferral as in CSMA
  - **Collisions detected within short time**
  - Colliding transmissions aborted, reducing wastage
- Collision detection easy in wired LANs:
  - Compare transmitted, received signals
- Collision detection difficult in wireless LANs:
  - Reception shut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

70

CSMA/CD Collision Detection

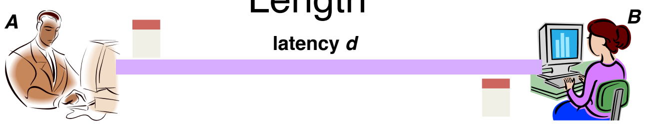
B and D can tell that collision occurred.

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



71

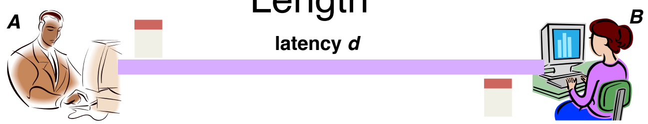
Limits on CSMA/CD Network Length



- 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***

72

Limits on CSMA/CD Network Length



- *A* needs to wait for time ***2d*** to detect collision
  - So, *A* should keep transmitting during this period
  - … and keep an eye out for a possible collision
- Imposes restrictions. E.g., for 10 Mbps Ethernet:
  - Maximum length of the wire: 2,500 meters
  - Minimum length of a frame: 512 bits (64 bytes)
    - 512 bits = 51.2 μsec (at 10 Mbit/sec)
    - For light in vacuum, 51.2 μsec ≈ 15,000 meters vs. 5,000 meters "round trip" to wait for collision
  - What about 10Gbps Ethernet?

73

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

74

Benefits of Ethernet

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

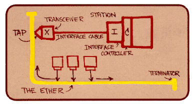
75

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

76

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)**

77

Binary Exponential Backoff (BEB)

- Think of time as divided in slots
- After each collision, pick a slot randomly within next 2^m slots
  - Where m is the number of collisions since last successful transmission
- Questions:
  - Why backoff?
  - Why random?
  - Why 2^m?
  - Why not listen while waiting?

78

Behavior of BEB Under Light Load

Look at collisions between two nodes

- First collision: pick one of the next two slots
  - Chance of success after first collision: 50%
  - Average delay 1.5 slots
- Second collision: pick one of the next four slots
  - Chance of success after second collision: 75%
  - Average delay 2.5 slots
- In general: after m^{th} collision
  - Chance of success: $1-2^{-m}$
  - Average delay (in slots): $½ + 2^{(m-1)}$

79

BEB: Theory vs Reality

In theory, there is no difference between theory and practice. But, in practice, there is.

80

BEB Reality

- Performs well (far from optimal, but no one cares)
  - *Large packets are ~23 times as large as minimal slot*
- Is now mostly irrelevant
  - *Almost all current ethernets are **switched***

81

BEB Theory

- A very interesting algorithm
- Stability for finite N only proved in 1985
  - Ethernet can handle nonzero traffic load without collapse
- All backoff algorithms unstable for infinite N (1985)
  - Poisson model: infinite user pool, total demand is finite
- Not of practical interest, but gives important insight
  - Multiple access should be in your "bag of tricks"

82

Question

- Two hosts, each with infinite packets to send
- What happens under BEB?
- Throughput high or low?
- Bandwidth shared equally or not?

83

MAC "Channel Capture" in BEB

- Finite chance that first one to have a successful transmission will never relinquish the channel
  - The other host will *never* send a packet
- Therefore, asymptotically channel is fully utilized and completely allocated to one host

84

Example

- Two hosts, each with infinite packets to send
  - Slot 1: collision
  - Slot 2: each resends with prob ½
    - Assume host A sends, host B does not
  - Slot 3: A and B both send (collision)
  - Slot 4: A sends with probability ½, B with prob. ¼
    - Assume A sends, B does not
  - Slot 5: A definitely sends, B sends with prob. ¼
    - Assume collision
  - Slot 6: A sends with probability ½, B with prob. 1/8
- Conclusion: if A gets through first, the prob. of B sending successfully halves with each collision

85

Another Question

- Hosts now have large but finite # packets to send
- What happens under BEB?
- Throughput high or low?

86

Answer

- Efficiency less than one, no matter how many packets
- Time you wait for loser to start is proportion to time winner was sending….

87

Different Backoff Functions

- Exponential: backoff ~ a^i
  - Channel capture?
  - Efficiency?
- Superlinear polynomial: backoff ~ i^p p>1
  - Channel capture?
  - Efficiency?
- Sublinear polynomial: backoff ~ i^p p≤1
  - Channel capture?
  - Efficiency?

88

Different Backoff Functions

- Exponential: backoff ~ a^i
  - Channel capture *(loser might not send until winner idle)*
  - Efficiency less than 1 *(time wasted waiting for loser to start)*
- Superlinear polynomial: backoff ~ i^p p>1
  - Channel capture
  - Efficiency is 1 (for any finite # of hosts N)
- Sublinear polynomial: backoff ~ i^p p≤1
  - No channel capture *(loser not shut out)*
  - Efficiency is less than 1 (and goes to zero for large N)
    - *Time wasted resolving collisions*

89

Summary of MAC protocols

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

90

MAC Addresses (and 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, also sometimes software settable

91

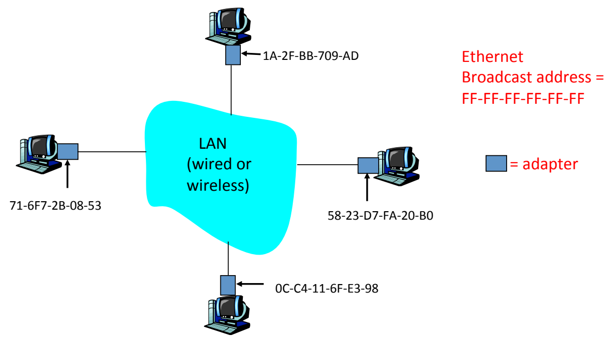
LAN Address (more)

- MAC address allocation administered by IEEE
- manufacturer buys portion of MAC address space (to assure uniqueness)
- analogy:
  - (a) MAC address: like Social Security Number
  - (b) IP address: like 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

92

LAN Addresses and ARP

Each adapter on LAN has unique LAN address



93

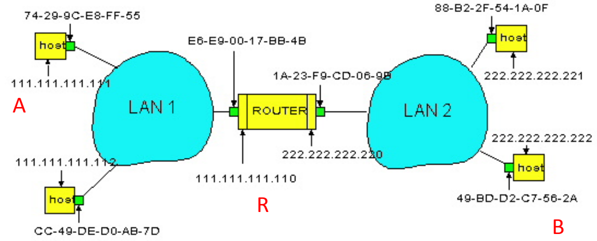
Address Resolution Protocol

- 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

94

Example: A Sending a Packet to B

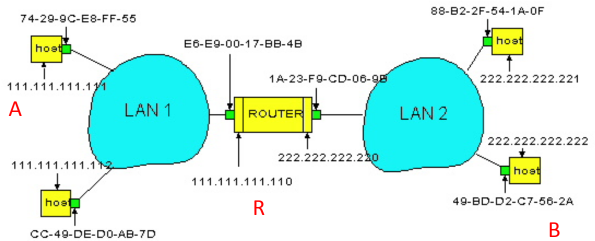
How does host A send an IP packet to host B?



95

Example: A Sending a Packet to B

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

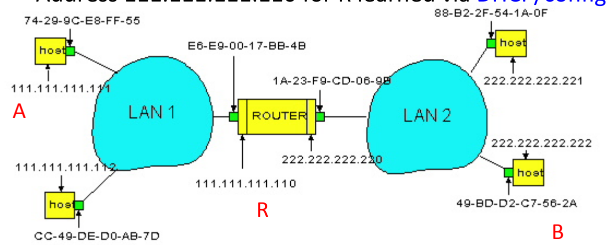


1. **A sends packet to R.**
2. **R sends packet to B.**

96

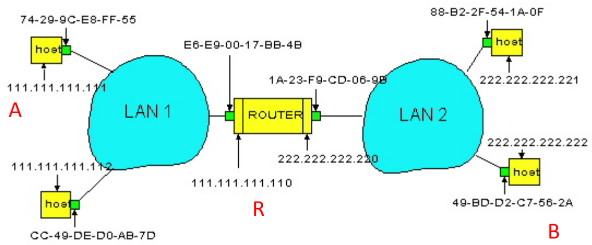
Host A Decides to Send Through R

- 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



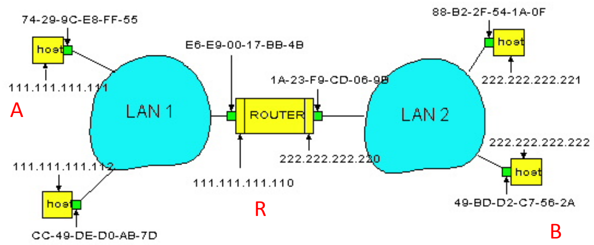
Host A Sends Packet Through R

- Host A learns the MAC address of R’s interface
  - ARP request: broadcast request for 111.111.111.110
  - ARP response: R responds with EE9-00-17-BB-4B
- Host A encapsulates the packet and sends to R



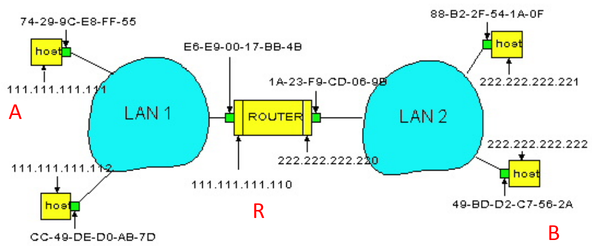
R Decides how to Forward Packet

- 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 not its MAC address

101

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

102

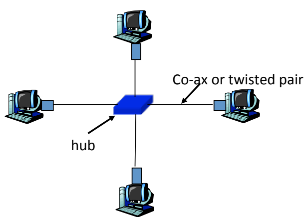
Why Not Use DNS-Like Tables?

- 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

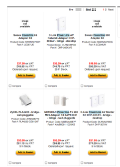
103

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

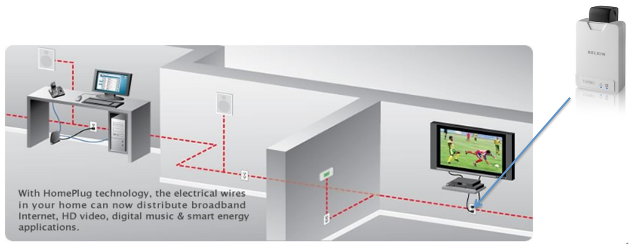


104



CSMA/CD Lives….

Home Plug and similar Powerline Networking….



105

Switch

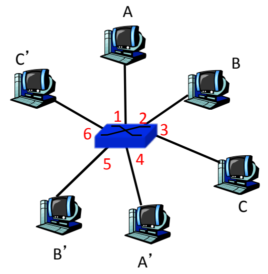
(like a Hub but smarter)

- 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

106

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

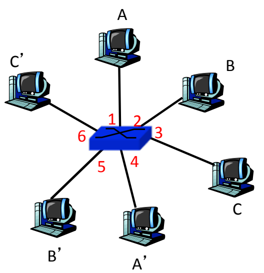


switch with six interfaces (1,2,3,4,5,6)

107

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?

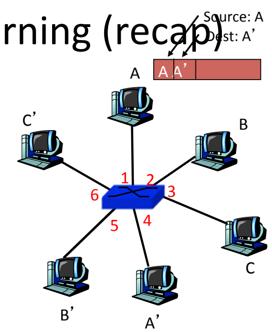


switch with six interfaces (1,2,3,4,5,6)

108

Switch: self-learning (recap)

- 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)

109

Switch: frame filtering/forwarding

When frame received:

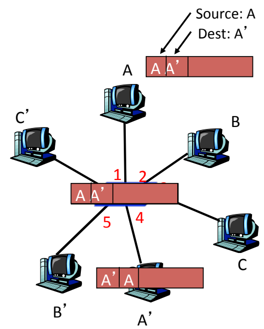
1. record link associated with sending host
2. index switch table using MAC dest address
3. **if** entry found for destination
   then {
   if dest on segment from which frame arrived
   then drop the frame
   else forward the frame on interface indicated
   }
   else flood

forward on all but the interface on which the frame arrived

110

Self-learning, forwarding: example

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



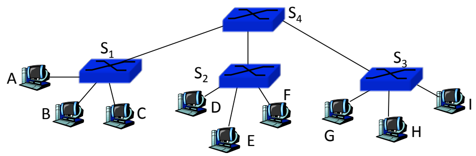
MAC addr	interface	TTL
A	*1*	*60*
A'	*4*	*60*

Switch table (initially empty)

111

Interconnecting switches

- switches can be connected together

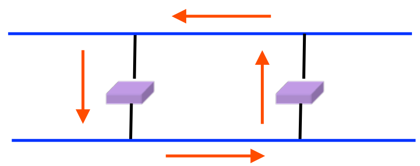


- Q: sending from A to G - how does S_1 know to forward frame destined to F via S_4 and S_3?
- A: self learning! (works exactly the same as in single-switch case – flood/forward/drop)

112

Flooding Can Lead to Loops

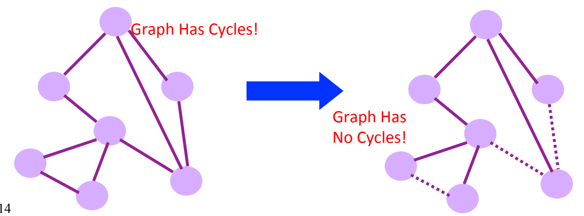
- Flooding can lead to forwarding loops
  - E.g., if the network contains a cycle of switches
  - "Broadcast storm"



113

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



114

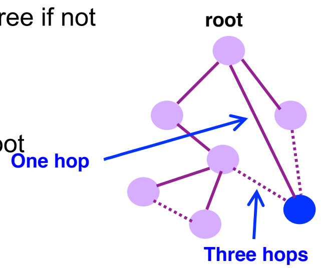
What Do We Know?

- 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

115

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



116

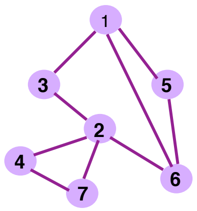
Steps in Spanning Tree Algorithm

- 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)

117

Example From Switch #4's Viewpoint

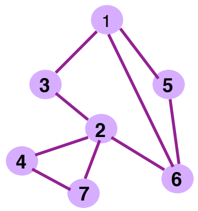
- Switch #4 thinks it is the root
  - Sends (4, 0, 4) message to 2 and 7
- Then, switch #4 hears from #2
  - Receives (2, 0, 2) message from 2
  - … and thinks that #2 is the root
  - And realizes it is just one hop away
- Then, switch #4 hears from #7
  - Receives (2, 1, 7) from 7
  - And realizes this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



118

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



119

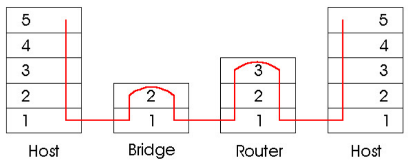
Robust Spanning Tree Algorithm

- 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…

120

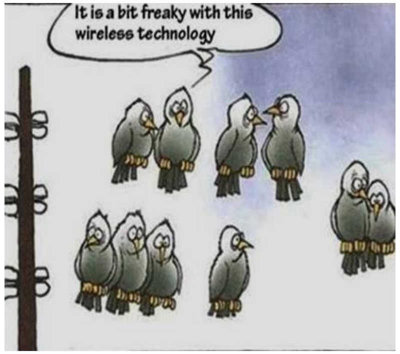
Switches vs. Routers Summary

- both store-and-forward devices
  - routers: network layer devices (examine network layer headers)
  - switches are link layer devices
- routers maintain routing tables, implement routing algorithms
- switches maintain switch tables, implement filtering, learning algorithms

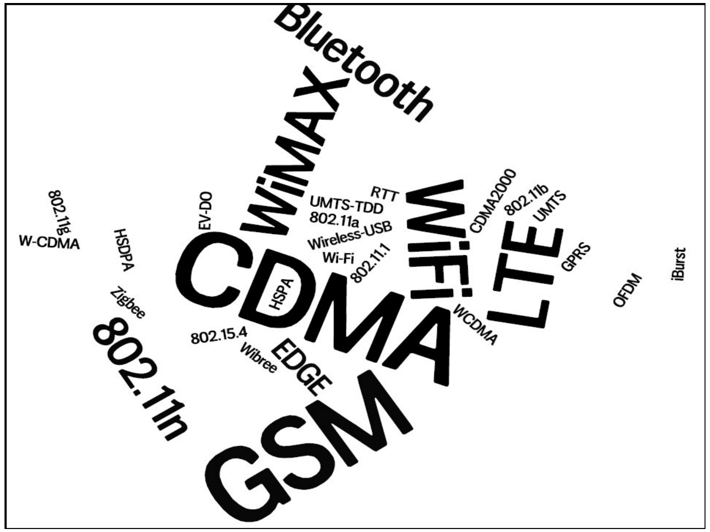


121

Wireless



122

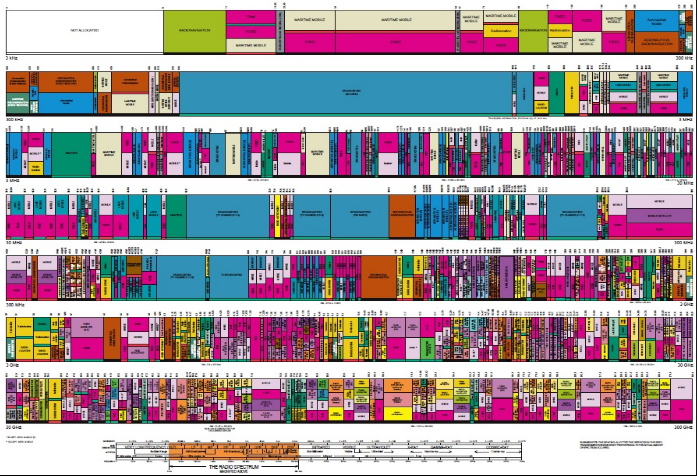


Metrics for evaluation / comparison of wireless technologies

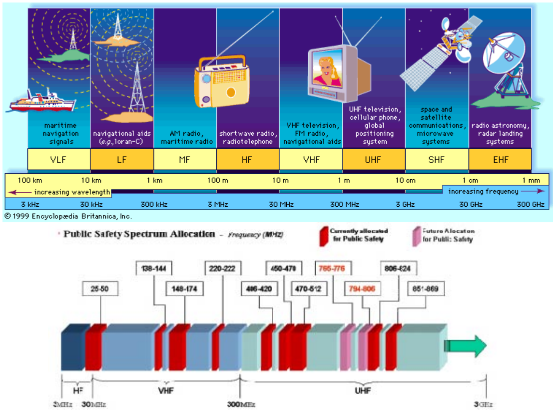
- Bitrate or Bandwidth
- Range - PAN, LAN, MAN, WAN
- Two-way / One-way
- Multi-Access / Point-to-Point
- Digital / Analog
- Applications and industries
- Frequency – Affects most physical properties:
  - Distance (free-space loss)
  - Penetration, Reflection, Absorption
  - Energy proportionality
 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

124

Modern art?



The Wireless Spectrum



126

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):
  - 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).
- IEEE 802.15 – lower power wireless:
  - 802.15.1: 2.4Ghz, 2.1 Mbps (Bluetooth)
  - 802.15.4: 2.4Ghz, 250 Kbps (Sensor Networks)

127 127

Wireless Link Characteristics

(Figure Courtesy of Kurose and Ross)

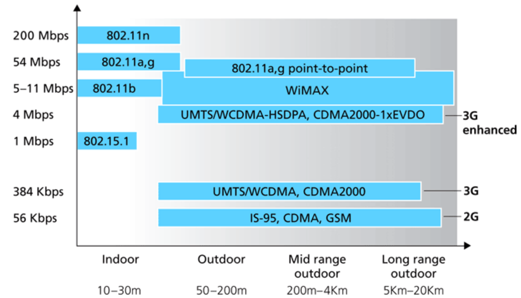


Figure 6.2 ♦ Link characteristics of selected wireless network standards

128 128

Antennas / Aerials

- An electrical device which converts electric currents into radio waves, and vice versa.

2-3dB 8-12dB 15-18dB 28-34dB

- Q: What does "higher-gain antenna" mean?
- A: Antennas are passive devices –
  more gain means focused and more directional.
- Directionality means more energy gets to where it needs to go and less interference everywhere.

- What are omni-directional antennas?

129

What has changed?

130

How many radios/antennas ?

- WiFi 802.11n (maybe MiMo?)
- 2G - GSM
- 3G – HSDPA+
- 4G – LTE
- Bluetooth (4.0)
- NFC
- GPS Receiver
- FM-Radio receiver
  (antenna is the headphones cable)

131

What Makes Wireless Different?

- Broadcast and multi-access medium…
  Just like AlohaNet – isn't this where we came in?
- 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?

132

Path Loss / Path Attenuation

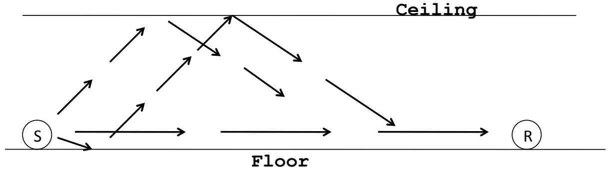
- Free Space Path Loss:
  - d = distance
  - λ = wave length
  - f = frequency
  - c = speed of light

$$\mathrm{FSPL} = \left(\frac{4\pi d}{\lambda}\right)^2 = \left(\frac{4\pi d f}{c}\right)^2$$

- Reflection, Diffraction, Absorption
- Terrain contours (Urban, Rural, Vegetation).
- Humidity

133

Multipath Effects

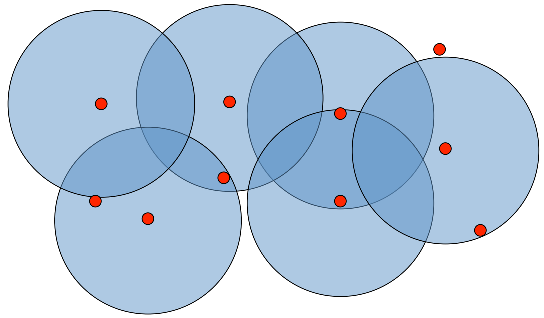


- Signals bounce off surface and interfere with one another
- Self-interference

134 134

Ideal Radios

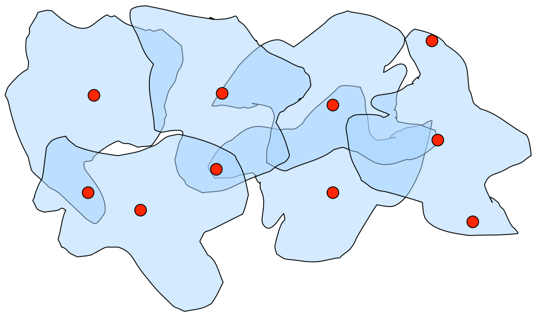
(courtesy of Gilman Tolle and Jonathan Hui, ArchRock)



135

Real Radios

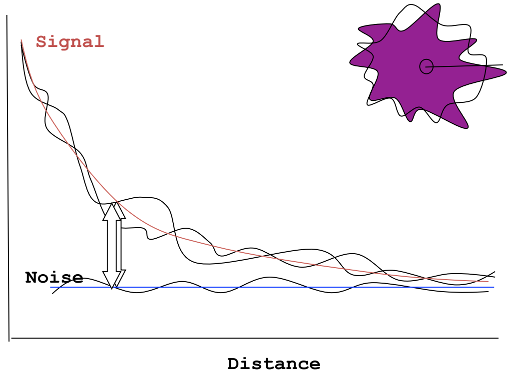
(courtesy of Gilman Tolle and Jonathan Hui, ArchRock)



136

The Amoebod "cell"

(courtesy of David Culler, UCB)



137 137

Interference from Other Sources

- External Interference
  - Microwave is turned on and blocks your signal
  - Would that affect the sender or the receiver?
- Internal Interference
  - Hosts within range of each other collide with one another's transmission
- We have to tolerate path loss, multipath, etc., but we can try to avoid internal interference

138

SNR – the key to communication:

Signal to Noise Ratio

Bitrate (aka data-rate)

➢The higher the SNR –
 the higher the (theoretical) bitrate.

➢Modern radios use adaptive /dynamic bitrates.

Q: In face of loss,
 should we decrease or increase the bitrate?

A: If caused by free-space loss or multi-path fading
 -lower the bitrate.
 If external interference - often higher bitrates
 (shorter bursts) are probabilistically better.

139

Wireless Bit Errors

- The lower the SNR (Signal/Noise) the higher the Bit Error Rate (BER)
- We could make the signal stronger…
- Why is this not always a good idea?
  - Increased signal strength requires more power
  - Increases the interference range of the sender, so you interfere with more nodes around you
    - And then they increase their power…….
- How would TCP behave in face of losses?
  - TCP conflates loss (congestion) with loss local errors
- Local link-layer Error Correction schemes can correct some problems (should be TCP aware).

140 140

802.11

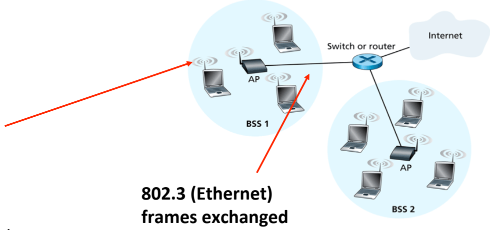
aka - WiFi …
What makes it special?

Deregulation > Innovation > Adoption > Lower cost = Ubiquitous technology

141 141

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

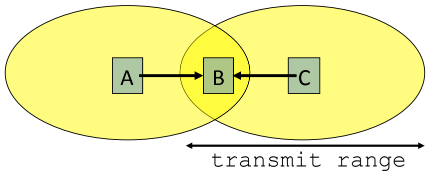
142 142

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?

143

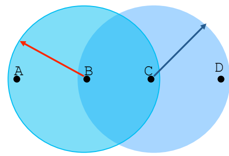
Hidden Terminals



- 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

144 144

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.

145 145

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

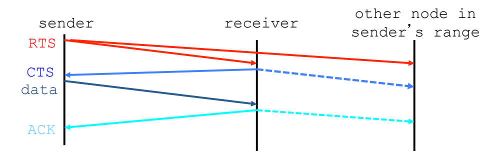
146

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

147

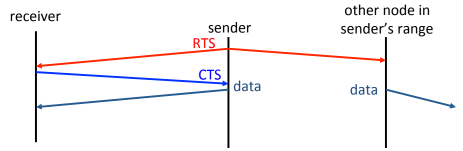
CSMA/CA -MA with Collision Avoidance



- 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

148 148

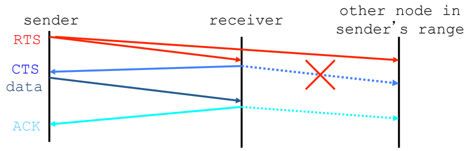
CSMA/CA, con't



- If other nodes hear RTS, but not CTS: send
  - Presumably, destination for first sender is out of node's range …

149 149

CSMA/CA, con't

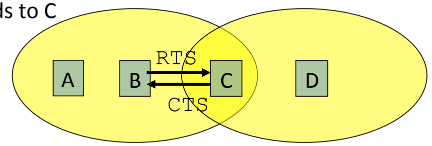


- If other nodes hear RTS, but not CTS: send
  - Presumably, destination for first sender is out of node's range …
  - … Can cause problems when a CTS is lost
- When you hear a CTS, you keep quiet until scheduled transmission is over (hear ACK)

150 150

RTS / CTS Protocols (CSMA/CA)

B sends to C



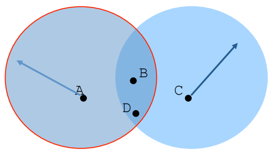
Overcome hidden terminal problems with contention-free protocol

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

151 151

Preventing Collisions Altogether

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



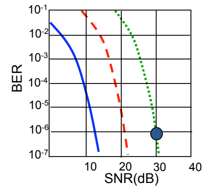
  - Now A and C can send without any interference!
- Most cards have only 1 transceiver
  - **Not Full Duplex: Cannot send and receive at the same time**
  - Aggregate Network throughput doubles

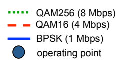
152 152

802.11: advanced capabilities

Rate Adaptation

- base station, mobile dynamically change transmission rate (physical layer modulation technique) as mobile moves, SNR varies





1. SNR decreases, BER increase as node moves away from base station
2. When BER becomes too high, switch to lower transmission rate but with lower BER

153

802.11: advanced capabilities

Power Management

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

154

Topic 4: Network Layer

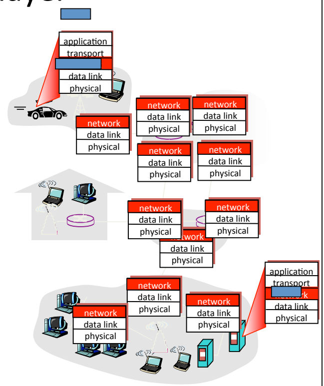
Our goals:

- 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

2

Network layer

- transport segment from sending to receiving host
- on sender side encapsulates segments into datagrams
- on receiver side, delivers segments to transport layer
- network layer protocols in *every* host, router
- router examines header fields in all IP datagrams passing through it



3

Name: a *something*

Address: Where a *something* is

Routing: How do I get to the *something*

4

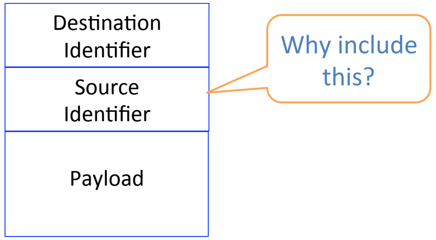
Addressing (at a conceptual level)

- 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?

5

Packets (at a conceptual level)

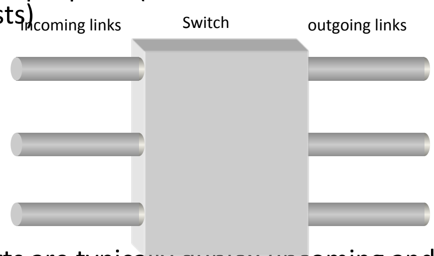
- Assume packet headers contain:
  - Source ID, Destination ID, and perhaps other information



6

Switches/Routers

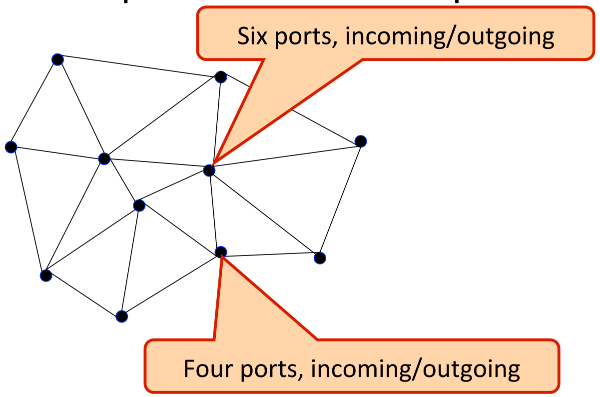
- Multiple ports (attached to other switches or hosts)



- Ports are typically duplex (incoming and outgoing)

7

Example of Network Graph



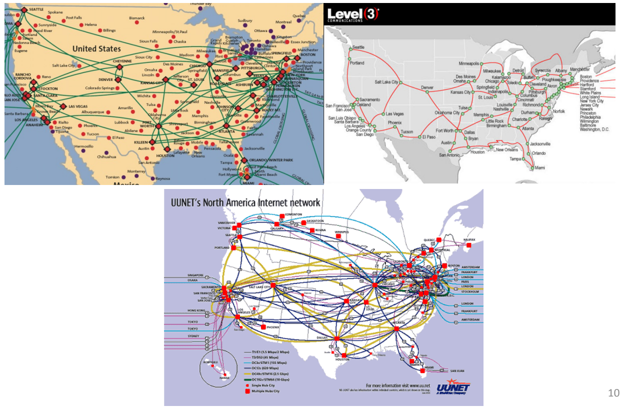
8

A Variety of 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)

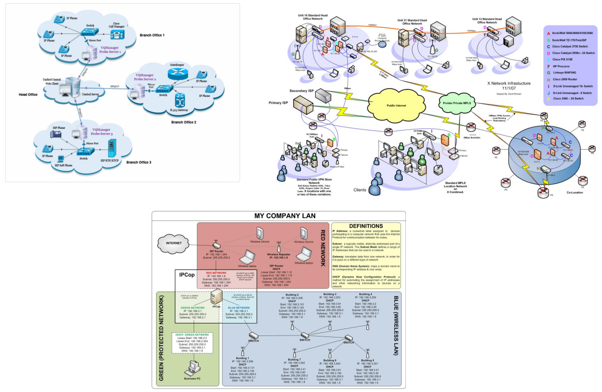
9

ISP networks



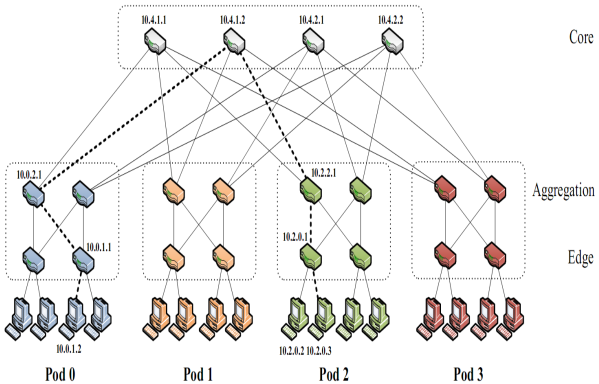
10

Enterprise Network



11

Partial Datacenter Network



12

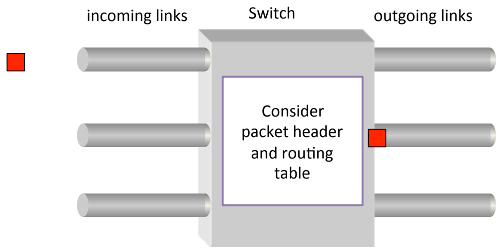
Switches

- Enterprise/Edge: typically 24 to 48 ports
- Aggregation switches: 192 ports or more
- Backbone: typically fewer ports
- Border: typically very few ports

13

Forwarding Decisions

- When packet arrives, must choose outgoing port



- Decision is based on routing state (table) in switch

14

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* means collection of routing state in each of the routers
  - Will focus on where this routing state comes from
  - But first, a few preliminaries….

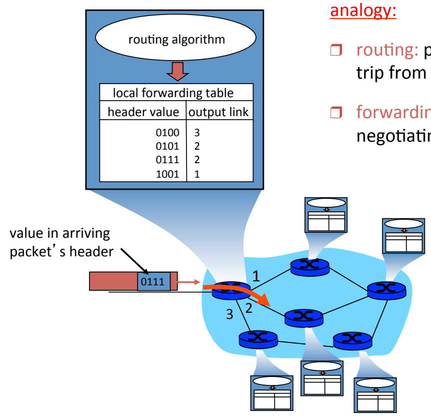
15

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….

16

Interplay between routing and forwarding



analogy:

- routing: process of planning trip from source to dest
- forwarding: process of negotiating each intersection

17

Connection setup

- 3rd important function in *some* network architectures:
  – ATM, frame relay, X.25, Software Defined Networks
- before datagrams flow, two end hosts *and* intervening routers establish virtual connection
  – routers get involved
- network vs transport layer connection service:
  – network: between two hosts (may also involve intervening routers in case of VCs)
  – transport: between two processes

  Remember: Ask youself "what is doing the multiplexing?"

18

Network service model

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

Example services for individual datagrams:

- guaranteed delivery
- guaranteed delivery with less than 40 msec delay

Example services for a flow of datagrams:

- in-order datagram delivery
- guaranteed minimum bandwidth to flow
- restrictions on changes in inter-packet spacing

19

Network layer service models:

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

20

Network layer connection and connection-less service

- datagram network provides network-layer connectionless service
- Virtual Circuit (VC) – a connection-orientated network – provides network-layer connection service
- analogous to the transport-layer services, but:
  – service: host-to-host
  – no choice: network provides one or the other
  – implementation: in network core

21

Virtual circuits

"source-to-dest path behaves much like telephone circuit"
- performance-wise
- network actions along source-to-dest path

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

22

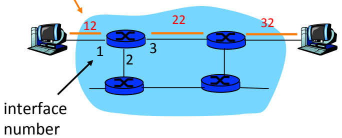
VC implementation

a VC consists of:
1. path from source to destination
2. VC numbers, one number for each link along path
3. entries in forwarding tables in routers along path

- packet belonging to VC carries VC number (rather than dest address)
- VC number can be changed on each link.
  - New VC number comes from forwarding table

23

Forwarding table



Forwarding table in northwest router:

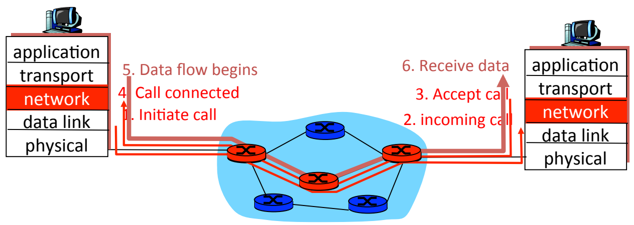
Incoming interface	Incoming VC #	Outgoing interface	Outgoing VC #
1	12	3	22
2	63	1	18
3	7	2	17
1	97	3	87
…	…	…	…

Routers maintain connection state information!

24

Virtual circuits: signaling protocols

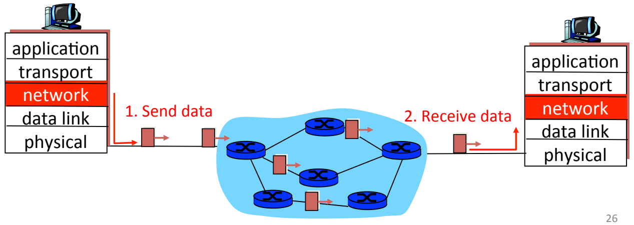
- used to setup, maintain teardown VC
- used in ATM, frame-relay, X.25
- not used in today's Internet



25

Datagram networks

- no call setup at network layer
- routers: no state about end-to-end connections
  - no network-level concept of "connection"
- packets forwarded using destination host address
  - packets between same source-dest pair may take different paths



26

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

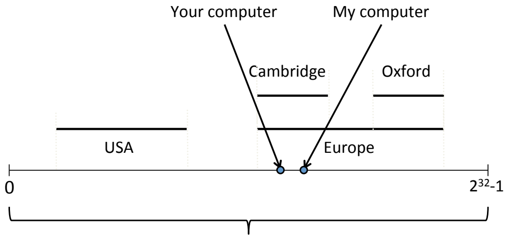
} **~ 4 billion entries**

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

27

IP addresses as a line



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

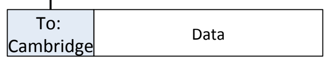
28

Longest Prefix Match (LPM)

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

Universities (entries 1–2)
Continents (entries 3–4)
Planet (entry 5)

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



29

Longest Prefix Match (LPM)

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

Datagram or VC network: why?

Internet (datagram)
- data exchange among computers
  - "elastic" service, no strict timing req.
- "smart" end systems (computers)
  - can adapt, perform control, error recovery
  - simple inside network, complexity at "edge"
- many link types
  - different characteristics
  - uniform service difficult

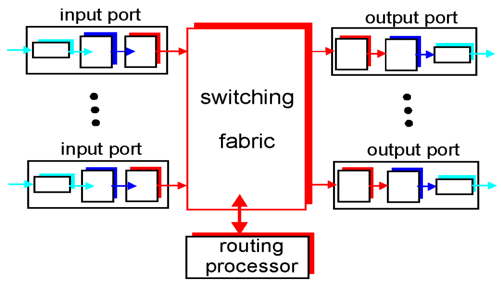
ETHERNET

ATM (VC)
- evolved from telephony
- human conversation:
  - strict timing, reliability requirements
  - need for guaranteed service
- "dumb" end systems
  - telephones
  - complexity inside network

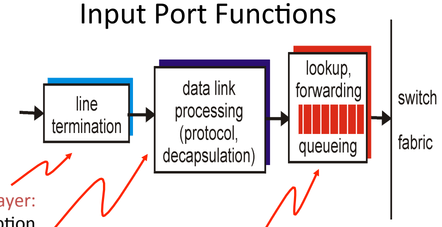
aDSL

Router Architecture Overview

Two key router functions:
- run routing algorithms/protocol (RIP, OSPF, BGP)
- *forwarding* datagrams from incoming to outgoing link



Input Port Functions



Physical layer:
bit-level reception

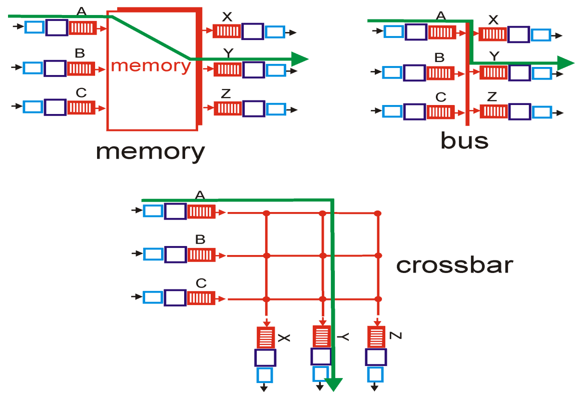
Data link layer:
e.g., Ethernet
see chapter 5

Decentralized switching:

- given datagram dest., lookup output port using forwarding table in input port memory
- goal: complete input port processing at 'line speed'
- queuing: if datagrams arrive faster than forwarding rate into switch fabric

34

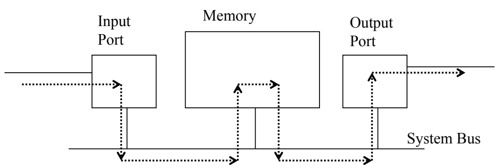
Three types of switching fabrics
(comparison criteria: speed, contention, complexity)



35

Switching Via Memory

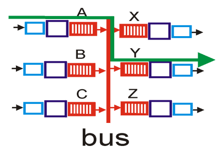
First generation routers:
- traditional computers with switching under direct control of CPU
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)



36

Switching Via a Bus

- datagram from input port memory to output port memory via a shared bus
- bus contention: switching speed limited by bus bandwidth
- Lots of ports?? speed up the bus
  no contention bus speed =
  2 x port speed x port count
- 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers

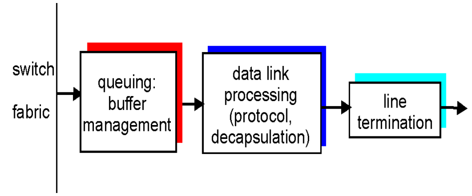


Switching Via An Interconnection Network

- overcome bus bandwidth limitations
- Banyan networks, other interconnection nets initially developed to connect processors in multiprocessor
- advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco 12000: switches 60 Gbps through the interconnection network

38

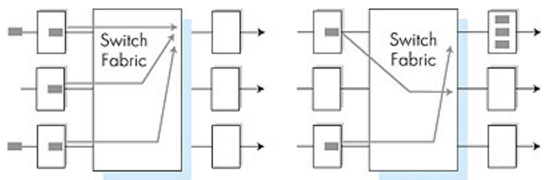
Output Ports



- *Buffering* required when datagrams arrive from fabric faster than the transmission rate
- *Scheduling discipline* chooses among queued datagrams for transmission
  - ➔ Who goes next?

39

Output port queueing



Output Port Contention at Time t

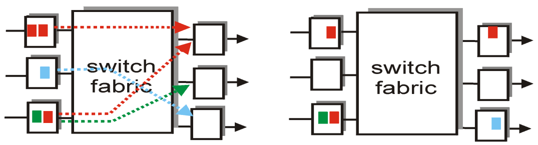
One Packet Time Later

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

40

Input Port Queuing

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



output port contention at time t - only one red packet can be transferred

green packet experiences HOL blocking

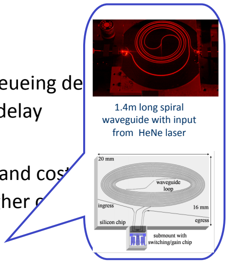
41

Buffers in Routers

- So how large should the buffers be?

Buffer size matters

- End-to-end delay
  - Transmission, propagation, and queueing de
  - The only variable part is queueing delay
- Router architecture
  - Board space, power consumption, and cos
  - On chip buffers: higher density, higher c
  - Optical buffers: all-optical routers



You are now touching the edge of the *research* zone……

42

Buffer Sizing Story

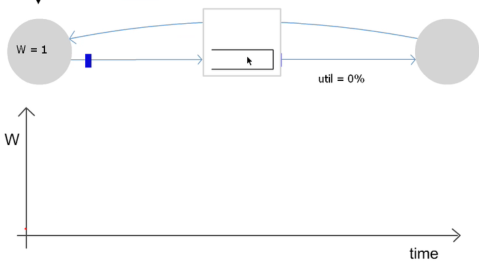
	Rule-of-thumb	Small Buffers	Tiny Buffers
	$2T \times C$	$\frac{2T \times C}{\sqrt{n}}$	$O(\log W)$
# of packets	1,000,000	10,000	20 - 50
Intuition	TCP Sawtooth	Sawtooth Smoothing	Non-bursty Arrivals
Assume	Single TCP Flow, 100% Utilization	Many Flows, 100% Utilization	Paced TCP, 85-90% Utilization
Evidence	Simulation, Emulation	Simulations, Test-bed and Real Network Experiments	Simulations, Test-bed Experiments

43

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

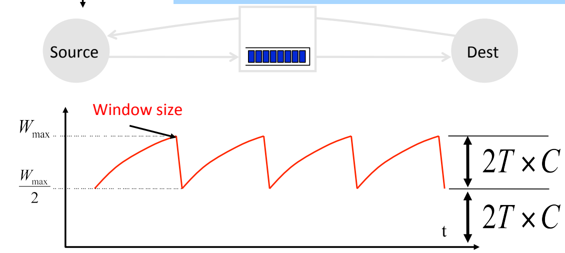


44

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

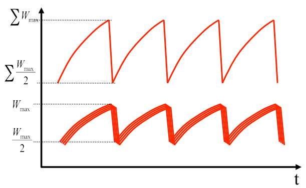


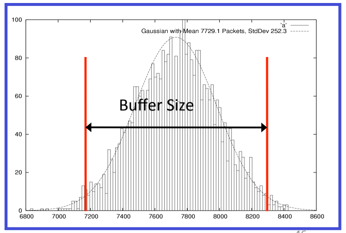
45

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

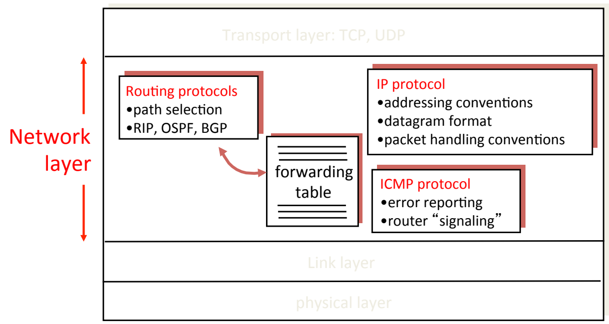




46

The Internet version of a Network layer

Host, router network layer functions:



47

IPv4 Packet Structure
20 Bytes of Standard Header, then Options

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

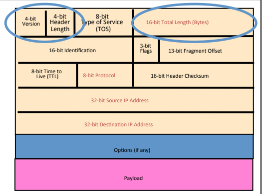
48

(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

49

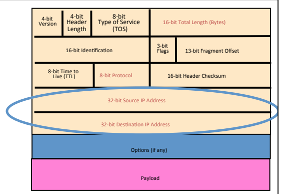
Reading Packet Correctly



- 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

50

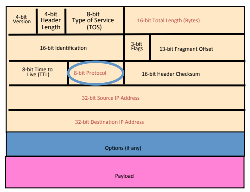
Getting Packet to Destination and Back



- 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

51

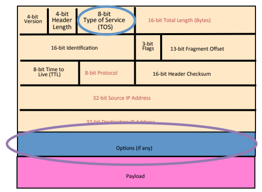
Telling Host How to Handle Packet



- 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)

52

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

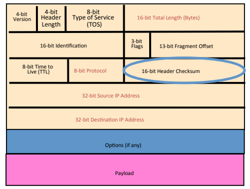
53

Potential Problems

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

54

Header Corruption

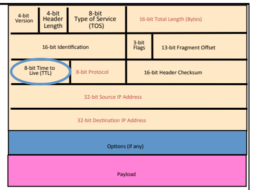


- Checksum (16 bits)
  - Particular form of checksum over packet header
- If not correct, router discards packets
  - So it doesn't act on bogus information
- Checksum recalculated at every router
  - **Why?**
  - **Why include TTL?**
  - **Why only header?**

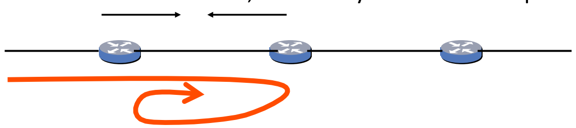
55

Preventing Loops

(aka Internet Zombie plan)



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

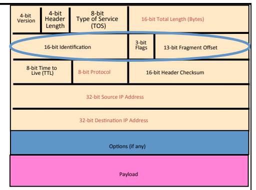


- Time-to-Live (TTL) Field (8 bits)
  - Decremented at each hop, packet discarded if reaches 0
  - ...and "time exceeded" message is sent to the source
    - Using "ICMP" control message; basis for **traceroute**

56

Fragmentation

(some assembly required)

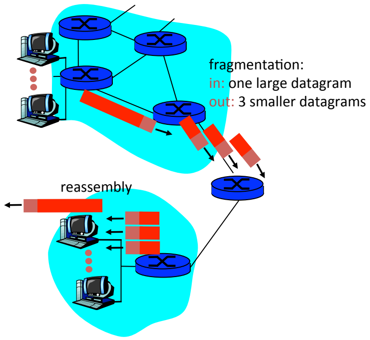


- Fragmentation: when forwarding a packet, an Internet router can split it into multiple pieces ("fragments") if too big for next hop link
- Must reassemble to recover original packet
  - Need fragmentation information (32 bits)
  - Packet identifier, flags, and fragment offset

57

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

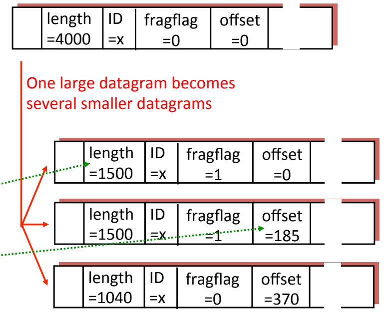


4-58

IP Fragmentation and Reassembly

Example

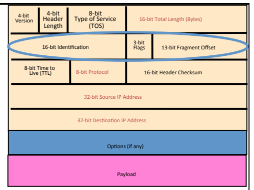
- 4000 byte datagram
- MTU = 1500 bytes



Pop quiz question: What happens when a fragment is lost?

4-59

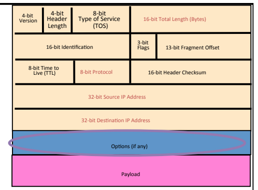
Fragmentation Details



- 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

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

Options

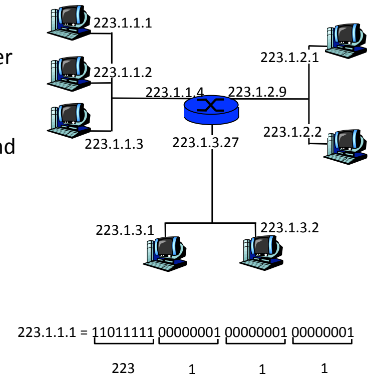


- End of Options List
- No Operation (padding between options)
- Record Route
- Strict Source Route
- Loose Source Route
- Timestamp
- Traceroute
- Router Alert
- .....

61

IP Addressing: introduction

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

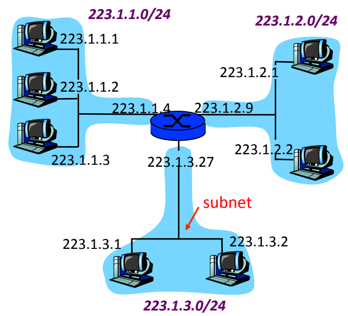


62

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

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



Subnet mask: /24

network consisting of 3 subnets

63

IP addresses: how to get one?

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

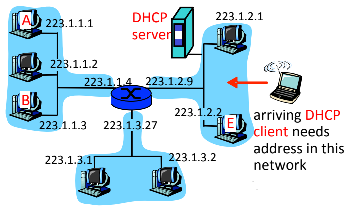
- hard-coded by system admin in a file
  - Windows: control-panel->network->configuration->tcp/ip->properties
  - UNIX: /etc/rc.config
- DHCP: Dynamic Host Configuration Protocol: dynamically get address from server
  - "plug-and-play"

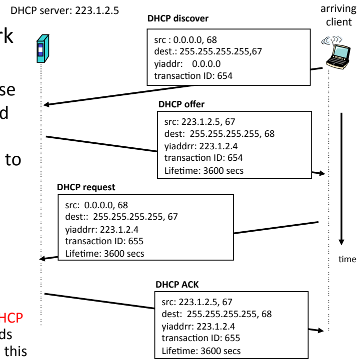
64

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)





65

IP addresses: how to get one?

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

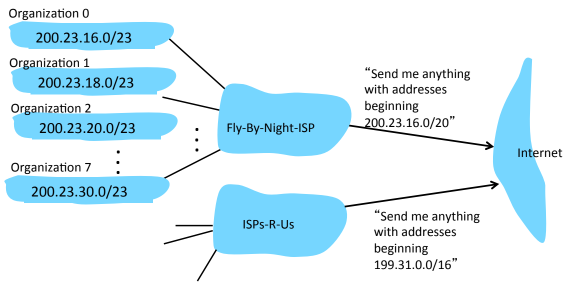
A: gets allocated portion of its provider ISP's address space

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

66

Hierarchical addressing: route aggregation

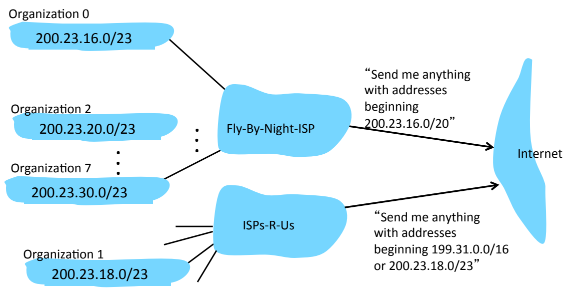
Hierarchical addressing allows efficient advertisement of routing information:



67

Hierarchical addressing: more specific routes

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



68

IP addressing: the last word...

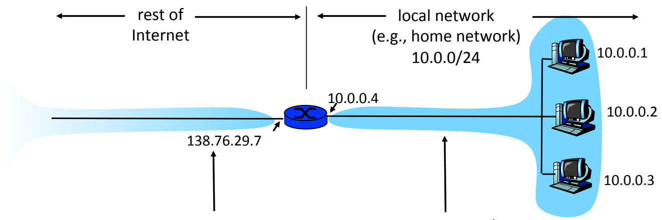
Q: How does an ISP get block of addresses?

A: ICANN: Internet Corporation for Assigned Names and Numbers

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

69

NAT: Network Address Translation



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

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

NAT: Network Address Translation

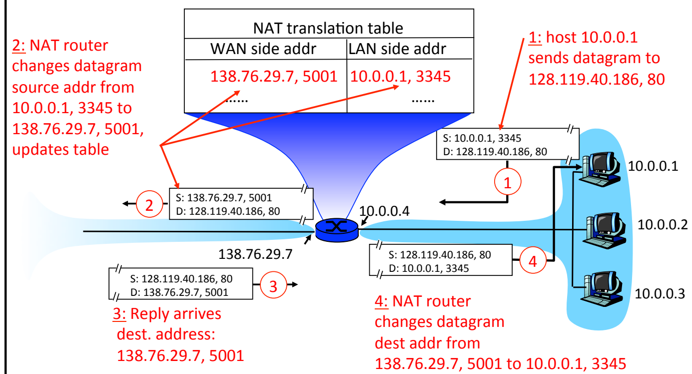
- Motivation: local network uses just one IP address as far as outside world is concerned:
  - range of addresses not needed from ISP: just one IP address for all devices
  - can change addresses of devices in local network without notifying outside world
  - can change ISP without changing addresses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



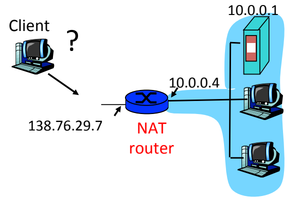
NAT: Network Address Translation

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

74

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., (123.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000

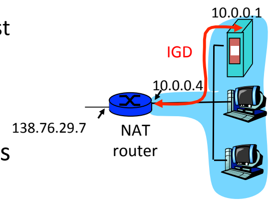


75

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)

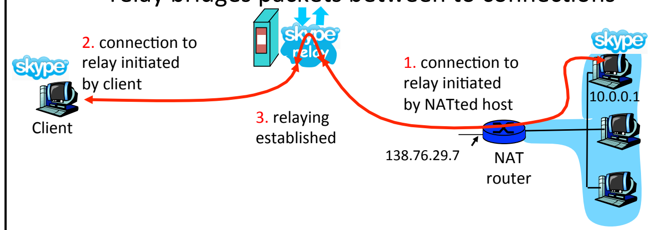
  i.e., automate static NAT port map configuration



76

NAT traversal problem

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



77

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

78

Traceroute and ICMP

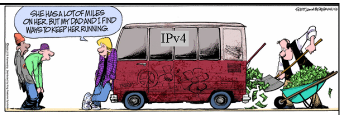
- Source sends series of UDP segments to dest
  - First has TTL =1
  - Second has TTL=2, etc.
  - Unlikely port number
- When nth datagram arrives to nth router:
  - Router discards datagram
  - And sends to source an ICMP message (type 11, code 0)
  - Message includes name of router& IP address
- When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

79

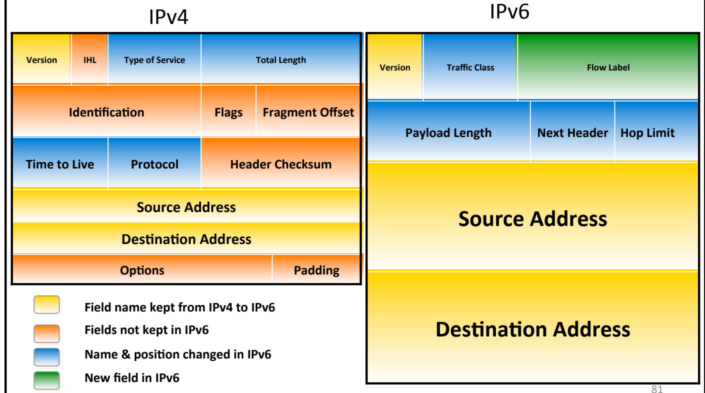
IPv6



- Motivated (prematurely) by address exhaustion
  - Addresses ***four*** times as big
- 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

80

IPv4 and IPv6 Header Comparison



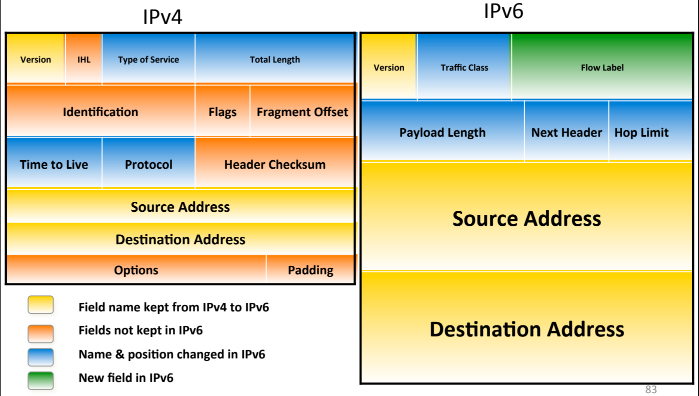
81

Summary of Changes

- Eliminated fragmentation ***(why?)***
- Eliminated header length ***(why?)***
- Eliminated checksum ***(why?)***
- New options mechanism (next header) ***(why?)***
- Expanded addresses ***(why?)***
- Added Flow Label ***(why?)***

82

IPv4 and IPv6 Header Comparison



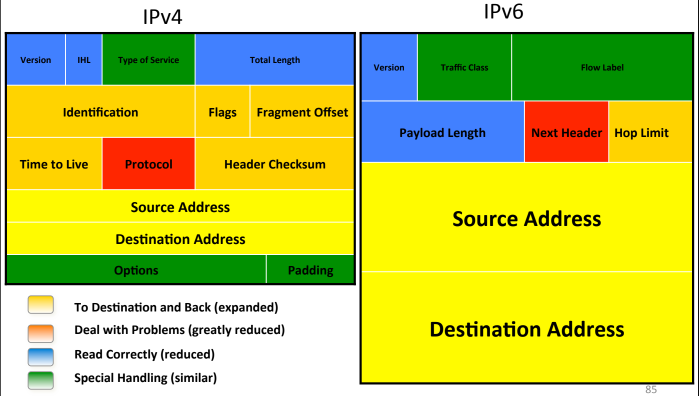
83

Philosophy of Changes

- Don't deal with problems: leave to ends
  - Eliminated fragmentation
  - Eliminated checksum
  - ***Why retain TTL?***
- Simplify handling:
  - New options mechanism (uses next header approach)
  - Eliminated header length
    - ***Why couldn't IPv4 do this?***
- Provide general flow label for packet
  - Not tied to semantics
  - Provides great flexibility

84

Comparison of Design Philosophy



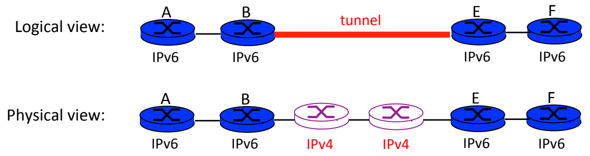
85

Transition From IPv4 To IPv6

- Not all routers can be upgraded simultaneous
  - no "flag days"
  - How will the network operate with mixed IPv4 and IPv6 routers?
- *Tunneling:* IPv6 carried as payload in IPv4 datagram among IPv4 routers

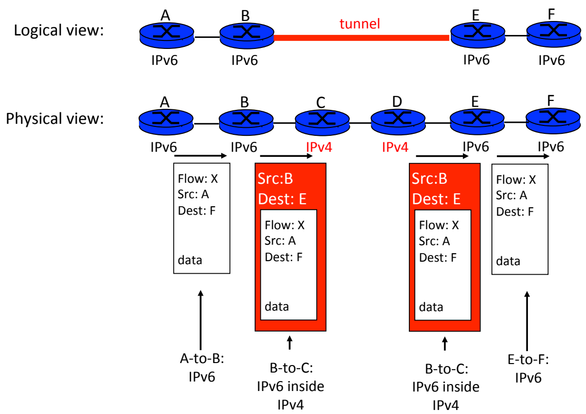
86

Tunneling



87

Tunneling



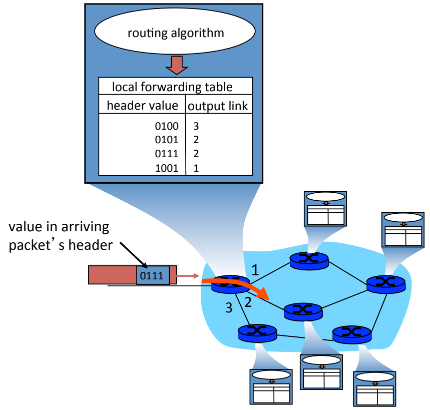
88

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?

89

Interplay between routing and forwarding



90

"Valid" Routing State

- Global routing state is "valid" if it produces forwarding decisions that always deliver packets to their destinations
  - ***Valid is not standard terminology***
- Goal of routing protocols: compute valid state
  - But how can you tell if routing state if valid?

91

Necessary and Sufficient Condition

- Global routing state is valid ***if and only if***:
  - There are no dead ends (other than destination)
  - There are no loops
- A dead end is when there is no outgoing port
  - A packet arrives, but the forwarding decision does not yield any outgoing port
- A loop is when a packet cycles around the same set of nodes forever

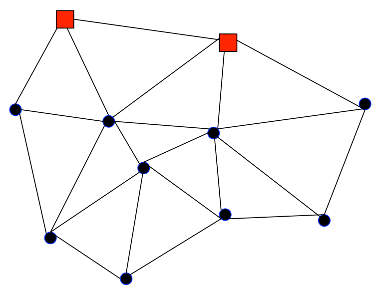
92

Necessary: Obvious

- If you run into a deadend before hitting destination, you'll never reach the destination
- If you run into a loop, you'll never reach destination
  - With deterministic forwarding, once you loop, you'll loop forever (assuming routing state is static)

93

Wandering Packets



Packet reaches deadend and stops

Packet falls into loop and never reaches destination

94

Sufficient: Easy

- Assume no deadends, no loops
- Packet must keep wandering, without repeating
  - If ever enter same switch from same port, will loop
  - Because forwarding decisions are deterministic
- Only a finite number of possible ports for it to visit
  - It cannot keep wandering forever without looping
  - Must eventually hit destination

95

The "Secret" of Routing

- Avoiding deadends is easy
- Avoiding loops is hard
- The key difference between routing protocols is how they avoid loops!
  - Don't focus on details of mechanisms
  - Just ask "how are loops avoided?"
- Will return to this later…. a little this term

  a lot more in Part II *Principles of Communications*

96

Making Forwarding Decisions

- Map PacketState+RoutingState into OutgoingPort
  - At line rates…..
- Packet State:
  - Destination ID
  - Source ID
  - Incoming Port *(from switch, not packet)*
  - Other packet header information?
- Routing State:
  - Stored in router

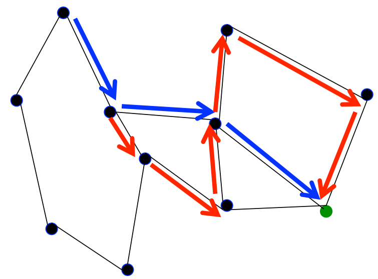
97

Forwarding Decision Dependencies

- *Must* depend on destination
- Could also depend on :
  - Source: requires n^2 state
  - Input port: not clear what this buys you
  - Other header information: let's ignore for now
- We will focus only on destination-based routing
  - But first consider the alternative

98

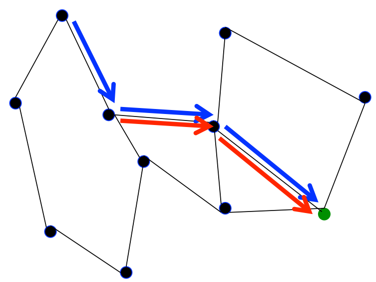
Source/Destination-Based Routing



Paths from two different sources (to same destination) can be very different

99

Destination-Based Routing



Paths from two different sources (to same destination) must coincide once they overlap

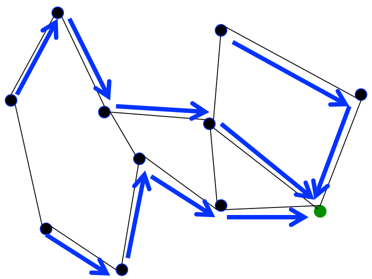
100

Destination-Based Routing

- Paths to same destination never cross
- Once paths to destination meet, they never split
- Set of paths to destination create a "delivery tree"
  - Must cover every node exactly once
  - **Spanning Tree rooted at destination**

101

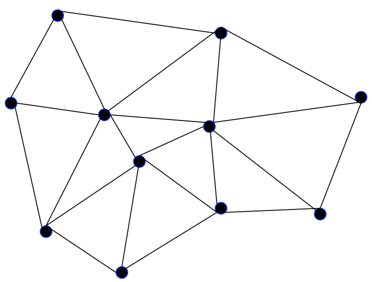
A "Delivery Tree" for a Destination



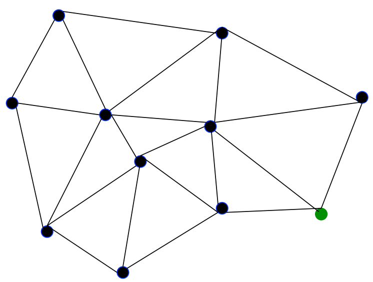
Checking Validity of Routing State

- Focus only on a single destination
  – Ignore all other routing state
- Mark outgoing port with arrow
  – There can only be one at each node
- Eliminate all links with no arrows
- Look at what's left….

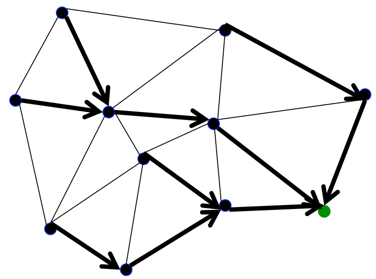
Example 1



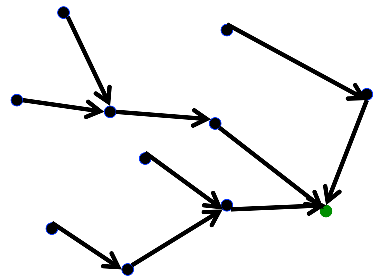
Pick Destination



Put Arrows on Outgoing Ports

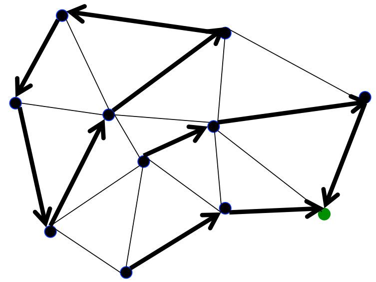


Remove Unused Links

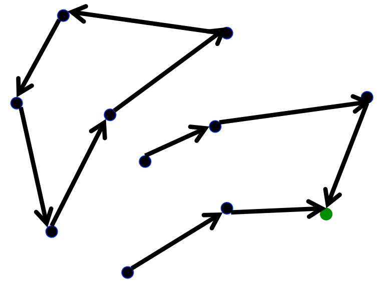


Leaves Spanning Tree: Valid

Second Example



Second Example



Is this valid?

Lesson….

- Very easy to check validity of routing state for a particular destination
- Deadends are obvious
  - Node without outgoing arrow
- Loops are obvious
  - Disconnected from rest of graph

110

Computing Routing State

111

Forms of Route Computation

- Learn from observing….
  - Not covered in your reading
- Centralized computation
  - One node has the entire network map
- Pseudo-centralized computation
  - All nodes have the entire network map
- Distributed computation
  - No one has the entire network map

112

How Can You Avoid Loops?

- Restrict topology to spanning tree
  - If the topology has no loops, packets can't loop!
- Central computation
  - Can make sure no loops
- Minimizing metric in distributed computation
  - Loops are never the solution to a minimization problem

113

Self-Learning on Spanning Tree

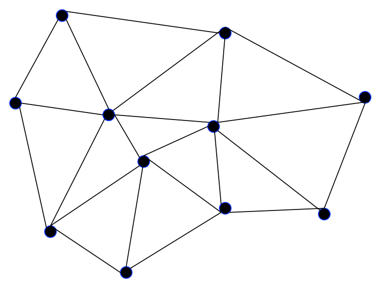
114

Easiest Way to Avoid Loops

- Use a topology where loops are impossible!
- Take arbitrary topology
- Build spanning tree (algorithm covered later)
  - Ignore all other links (as before)
- Only one path to destinations on spanning trees
- Use "learning switches" to discover these paths
  - No need to compute routes, just observe them

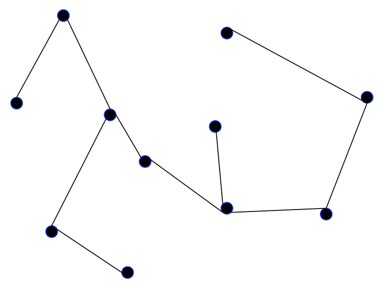
115

Consider previous graph



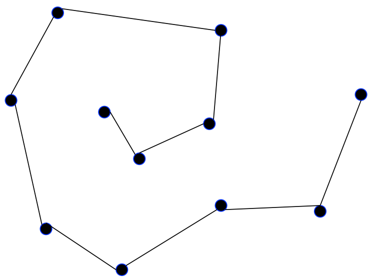
116

A Spanning Tree



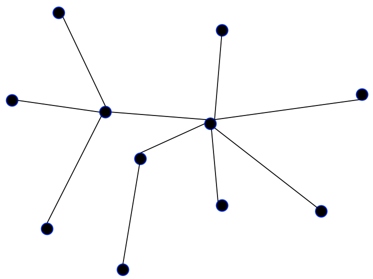
117

Another Spanning Tree



118

Yet Another Spanning Tree



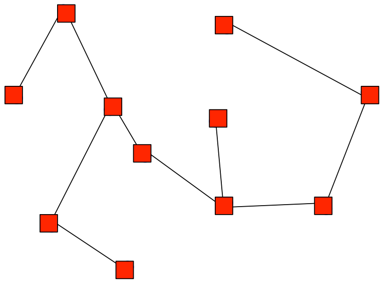
119

Flooding on a Spanning Tree

- If you want to send a packet that will reach all nodes, then switches can use the following rule:
  - Ignoring all ports not on spanning tree!
- Originating switch sends "flood" packet out all ports
- When a "flood" packet arrives on one incoming port, send it out all other ports

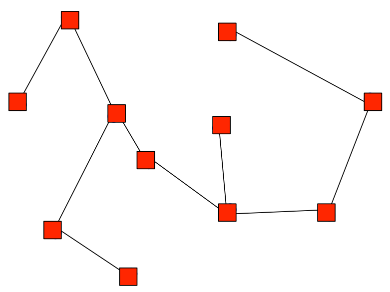
120

Flooding on Spanning Tree



121

Flooding on Spanning Tree (Again)



122

Flooding on a Spanning Tree

- This works because the lack of loops prevents the flooding from cycling back on itself
- Eventually all nodes will be covered, exactly once

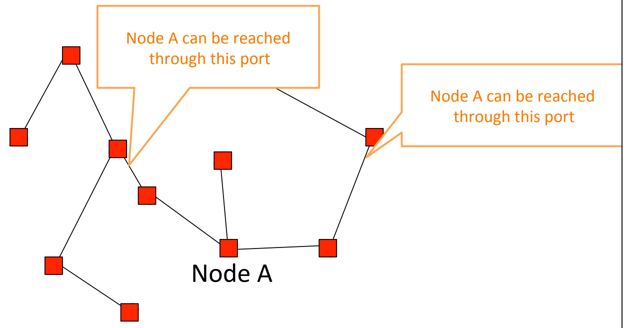
123

This Enables Learning!

- There is only one path from source to destination
- Each switch can learn how to reach a another node by remembering where its flooding packets came from!
- If flood packet from Node A entered switch from port 4, then to reach Node A, switch sends packets out port 4

124

Learning from Flood Packets



Once a node has sent a flood message, all other switches know how to reach it….

125

General Approach

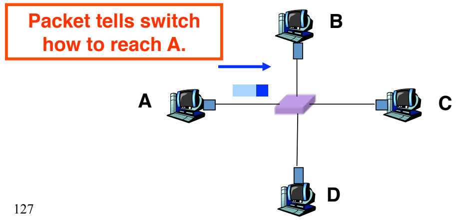
- Flood first packet
- All switches learn where you are
- When destination responds, all switches learn where it is…
- Done.

126

Self-Learning Switch

When a packet arrives

- Inspect *source* ID, associate with *incoming* port
- Store mapping in the switch table
- Use time-to-live field to eventually forget mapping

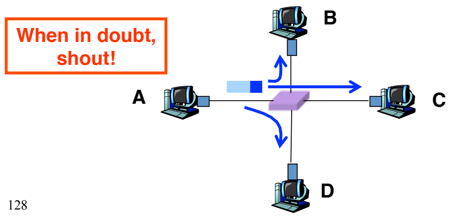


127

Self Learning: Handling Misses

When packet arrives with unfamiliar destination

- Forward packet out **all** other ports
- Response will teach switch about that destination



128

General Rule

When switch receives a packet:

index the switch table using destination ID

if entry found for destination {

    if dest on port from which packet arrived

        then drop packet

        else forward packet on port indicated

}

else flood

Why do this?

forward on all but the interface on which the frame arrived

129

Summary of Learning Approach

- Avoids loop by restricting to spanning tree
- This makes flooding possible
- Flooding allows packet to reach destination
- And in the process switches learn how to reach source of flood
- No route "computation"

130

Weaknesses of This Approach?

- Requires loop-free topology (Spanning Tree)
- Slow to react to failures (entries time out)
- Very little control over paths
- Spanning Trees suck.

- Other route protocols will be covered in *Principles of Communications (Part II)*

131

Topic 5 – Transport

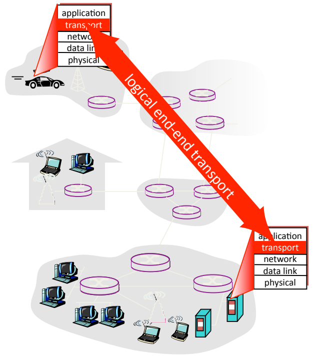
Our goals:

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

2

Transport services and protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



3

Transport vs. network layer

- *network layer:* logical communication between hosts
- *transport layer:* logical communication between processes
  - relies on, enhances, network layer services

Household analogy:

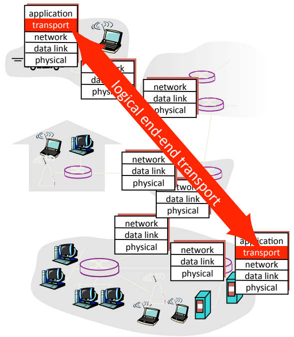
12 kids sending letters to 12 kids

- processes = kids
- app messages = letters in envelopes
- hosts = houses
- transport protocol = Ann and Bill
- network-layer protocol = postal service

4

Internet transport-layer protocols

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - no-frills extension of "best-effort" IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



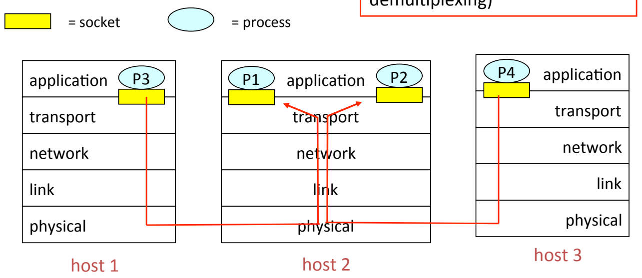
5

Multiplexing/demultiplexing

(Transport-layer style)

Demultiplexing at rcv host:
delivering received segments to correct socket

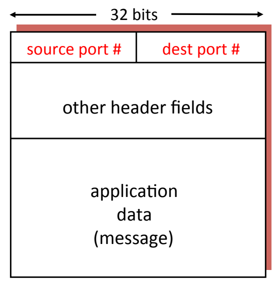
Multiplexing at send host:
gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)



6

How transport-layer demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries 1 transport-layer segment
  - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP segment format

7

Connectionless demultiplexing

- Create sockets with port numbers:

```
DatagramSocket mySocket1 = new
   DatagramSocket(12534);
DatagramSocket mySocket2 = new
   DatagramSocket(12535);
```

- UDP socket identified by two-tuple:

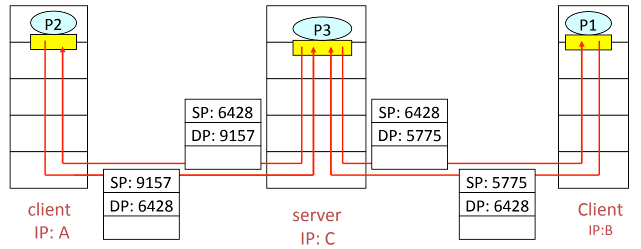
(dest IP address, dest port number)

- When host receives UDP segment:
  - checks destination port number in segment
  - directs UDP segment to socket with that port number
- IP datagrams with different source IP addresses and/or source port numbers directed to same socket

8

Connectionless demux (cont)

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```



SP provides "return address"

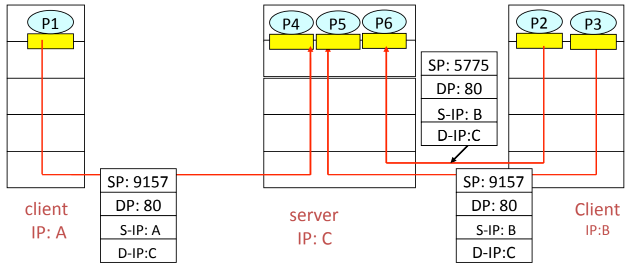
9

Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- recv host uses all four values to direct segment to appropriate socket
- Server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- Web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

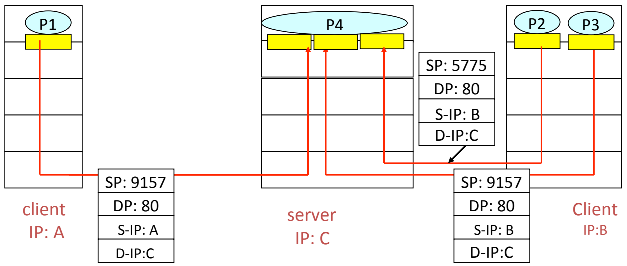
10

Connection-oriented demux (cont)



11

Connection-oriented demux: Threaded Web Server



12

UDP: User Datagram Protocol [RFC 768]

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

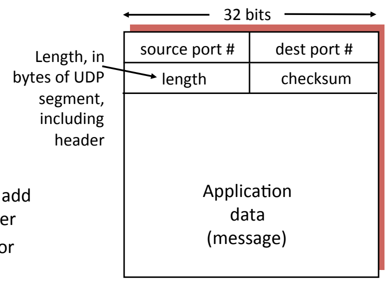
Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

13

UDP: more

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!



UDP segment format

14

UDP checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

Sender:

- treat segment contents as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?* More later ....

15

Internet Checksum

(time travel warning – we covered this earlier)

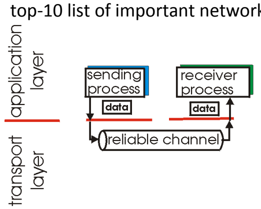
- Note
  - When adding numbers, a carryout from the most significant bit needs to be added to the result
- Example: add two 16-bit integers

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

16

Principles of Reliable data transfer

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



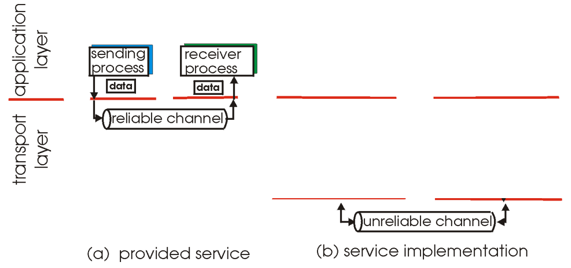
(a) provided service

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

17

Principles of Reliable data transfer

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

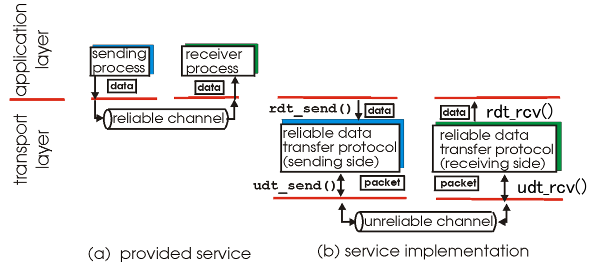


- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

18

Principles of Reliable data transfer

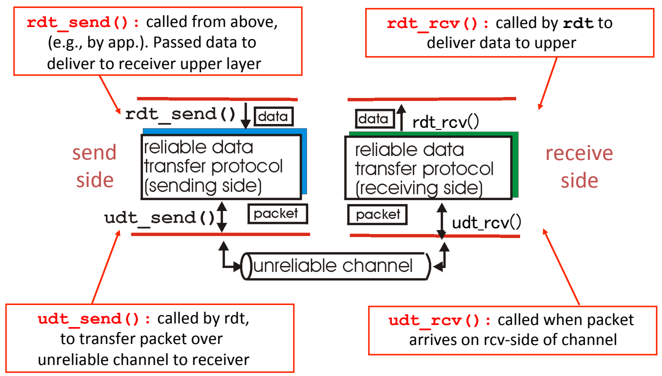
- important in app., transport, link layers
- top-10 list of important networking topics!



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

19

Reliable data transfer: getting started

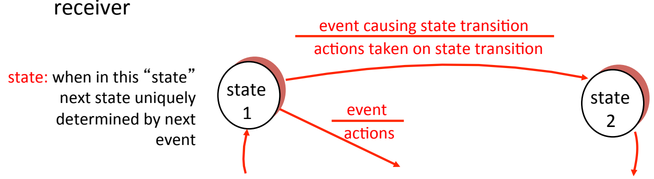


20

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



21

KR state machines – a note.

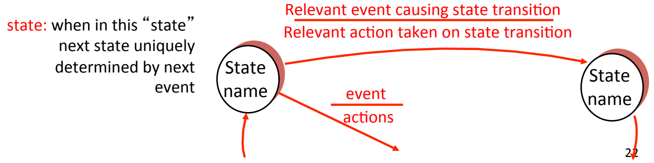
Beware

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

I've attempted to normalise the representation.

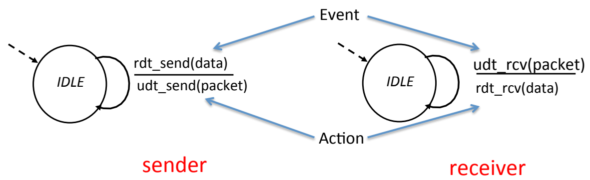
UPSHOT: these slides have differing information to the KR book (from which the RDT example is taken.)

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



Rdt1.0: reliable transfer over a reliable channel

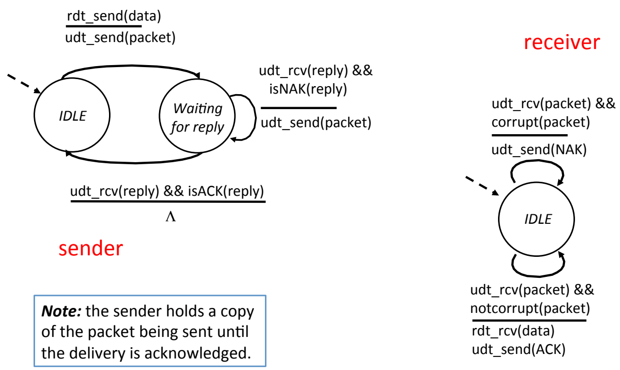
- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- separate FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver read data from underlying channel



Rdt2.0: channel with bit errors

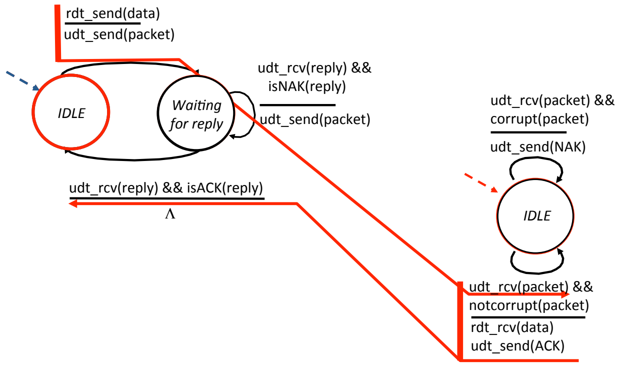
- underlying channel may flip bits in packet
  - checksum to detect bit errors
- *the* question: how to recover from errors:
  - *acknowledgements (ACKs):* receiver explicitly tells sender that packet received is OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that packet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

rdt2.0: FSM specification



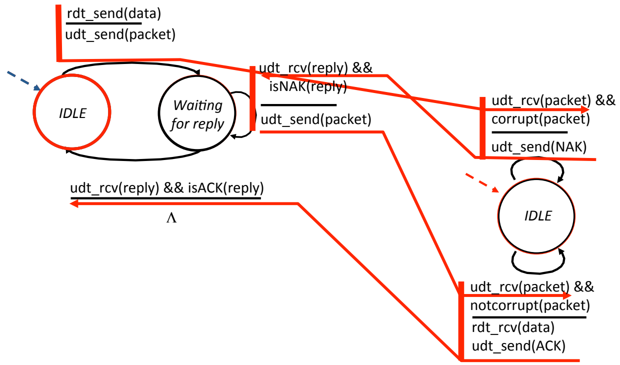
Note: the sender holds a copy of the packet being sent until the delivery is acknowledged.

rdt2.0: operation with no errors



26

rdt2.0: error scenario



27

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

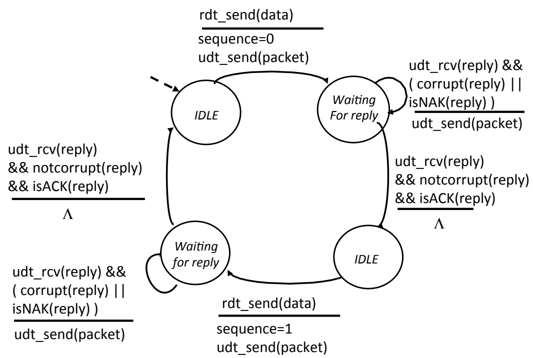
Handling duplicates:

- 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

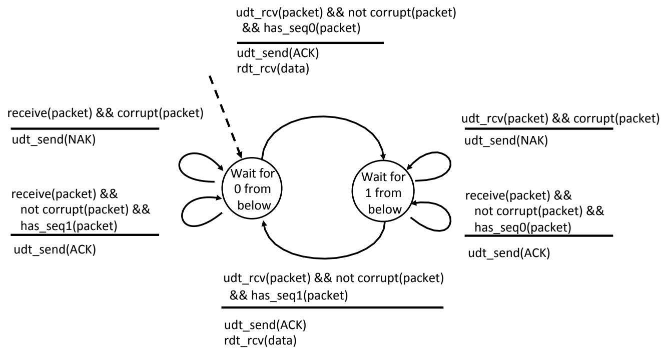
28

rdt2.1: sender, handles garbled ACK/NAKs



29

rdt2.1: receiver, handles garbled ACK/NAKs



30

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

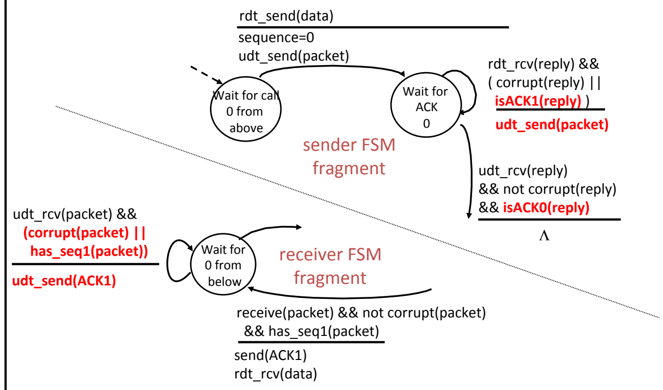
31

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*

32

rdt2.2: sender, receiver fragments



33

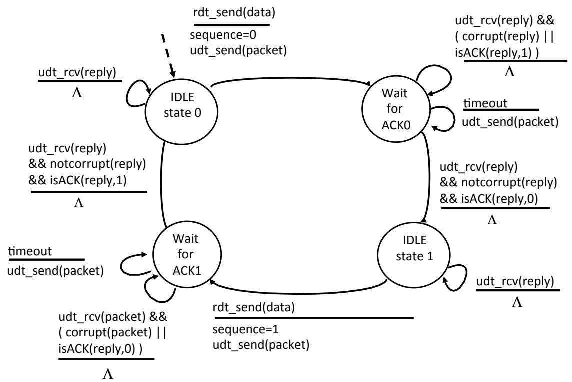
rdt3.0: channels with errors *and* loss

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

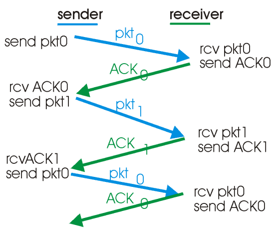
34

rdt3.0 sender

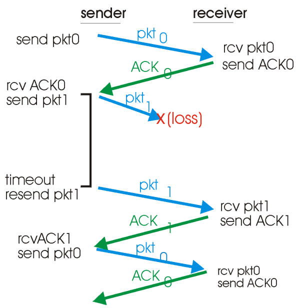


35

rdt3.0 in action



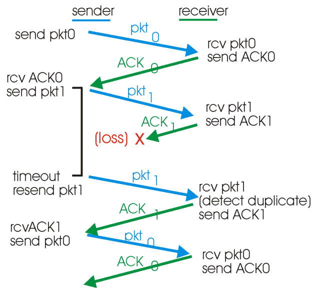
(a) operation with no loss



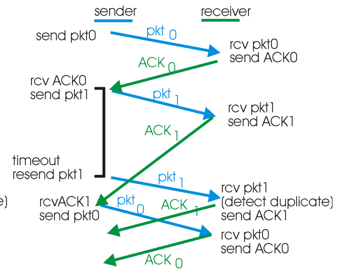
(b) lost packet

36

rdt3.0 in action



(c) lost ACK



(d) premature timeout

37

Performance of rdt3.0

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

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

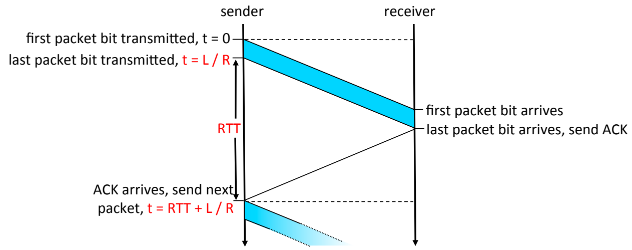
  - U sender: utilization – fraction of time sender busy sending

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

  - 1KB pkt every 30 msec -> 33kB/sec thruput over 1 Gbps link
  - network protocol limits use of physical resources!

38

rdt3.0: stop-and-wait operation



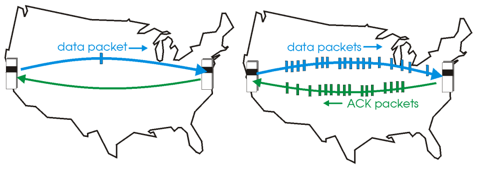
$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

39

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

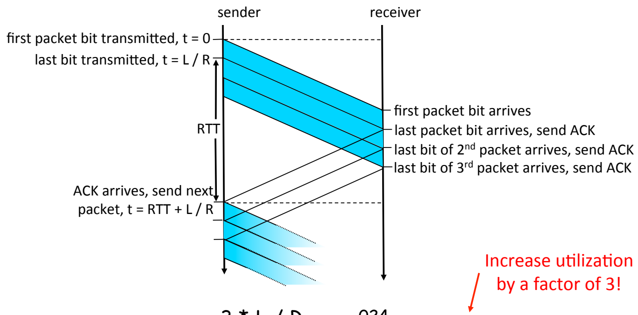


(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

- Two generic forms of pipelined protocols: *go-Back-N, selective repeat*

40

Pipelining: increased utilization



$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

41

Pipelining Protocols

Go-back-N: big picture:
- Sender can have up to N unacked packets in pipeline
- Rcvr only sends cumulative acks
  - Doesn't ack packet if there's a gap
- Sender has timer for oldest unacked packet
  - If timer expires, retransmit all unacked packets

Selective Repeat: big pic
- Sender can have up to N unacked packets in pipeline
- Rcvr acks individual packets
- Sender maintains timer for each unacked packet
  - When timer expires, retransmit only unack packet

42

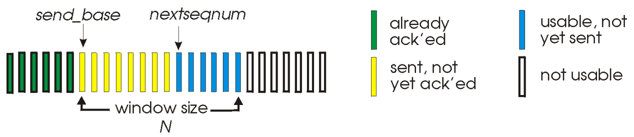
Selective repeat: big picture

- Sender can have up to N unacked packets in pipeline
- Rcvr acks individual packets
- Sender maintains timer for each unacked packet
  - When timer expires, retransmit only unack packet

43

Go-Back-N

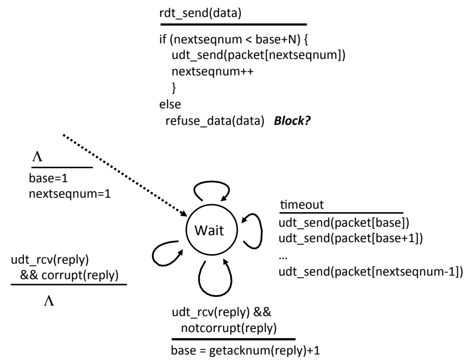
Sender:
- k-bit seq # in pkt header
- "window" of up to N, consecutive unack'ed pkts allowed



- ACK(n): ACKs all pkts up to, including seq # n - "cumulative ACK"
  - may receive duplicate ACKs (see receiver)
- timer for each in-flight pkt
- *timeout(n):* retransmit pkt n and all higher seq # pkts in window

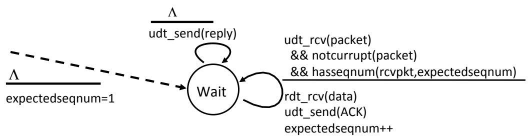
44

GBN: sender extended FSM



45

GBN: receiver extended FSM

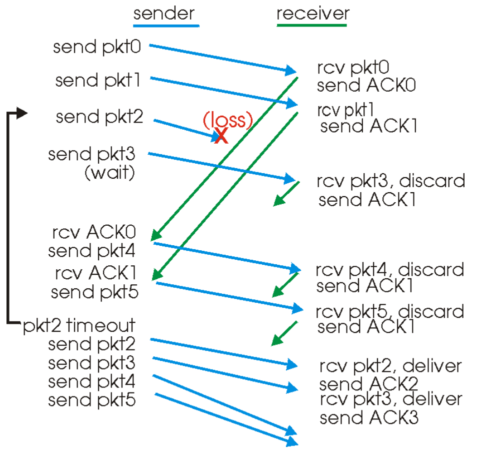


ACK-only: always send an ACK for correctly-received packet with the highest *in-order* seq #

- 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 #

46



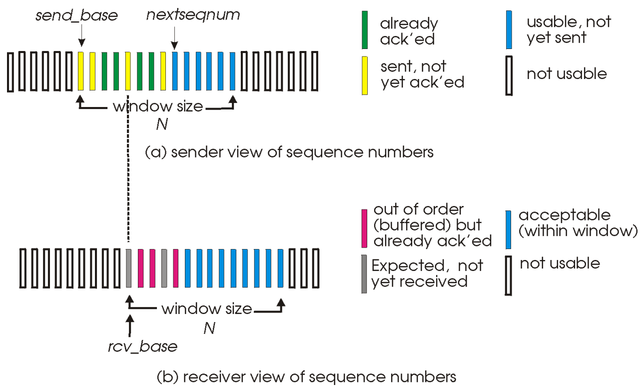
47

Selective Repeat

- receiver *individually* acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt
- sender window
  - N consecutive seq #'s
  - again limits seq #s of sent, unACKed pkts

48

Selective repeat: sender, receiver windows



49

Selective repeat

sender

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

timeout(n):
- resend pkt n, restart timer

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

receiver

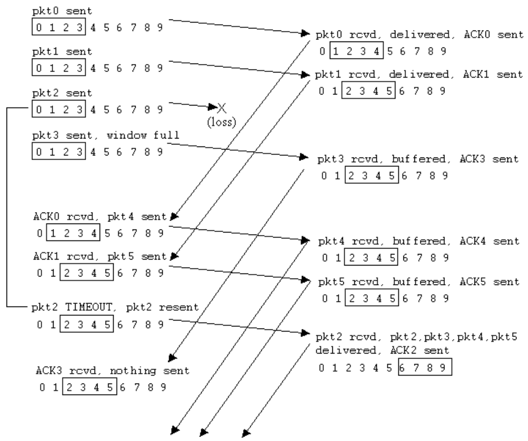
pkt n in [rcvbase, rcvbase+N-1]
- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

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

otherwise:
- ignore

50

Selective repeat in action



51

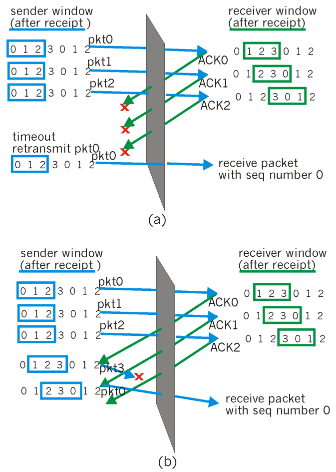
Selective repeat: dilemma

Example:
- seq #'s: 0, 1, 2, 3
- window size=3

- receiver sees no difference in two scenarios!
- incorrectly passes duplicate data as new in (a)

Q: what relationship between seq # size and window size?

window size ≤ (½ of seq # size)



52

Automatic Repeat Request (ARQ)

+ Self-clocking (Automatic)

+ Adaptive

+ Flexible

- Slow to start / adapt

consider high Bandwidth/Delay product

Now lets move from the generic to the specific….

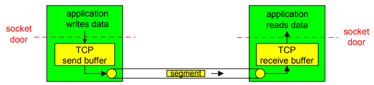
TCP arguably the most successful protocol in the Internet…..

its an ARQ protocol

53

TCP: Overview
RFCs: 793, 1122, 1323, 2018, 2581, …

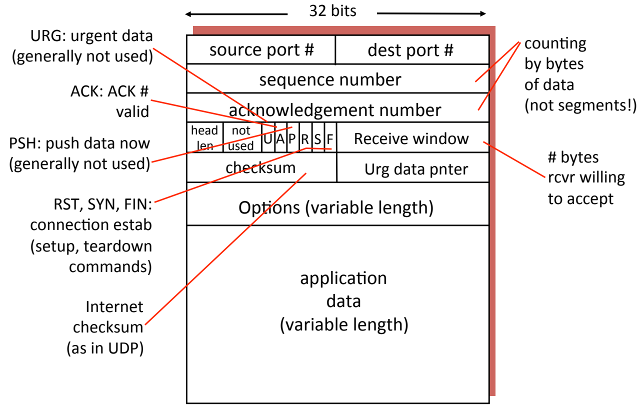
- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte stream:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- *send & receive buffers*



- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init' s sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

54

TCP segment structure



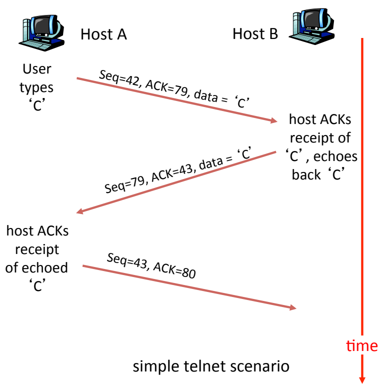
55

TCP seq. #' s and ACKs

Seq. #' s:
- byte stream "number" of first byte in segment' s data

ACKs:
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments
- A: TCP spec doesn' t say, - up to implementor



simple telnet scenario

This has led to a world of hurt….

56

TCP out of order attack

- ARQ with SACK means
  recipient needs copies of all packets
- Evil attack one:
  send a long stream of TCP data to a server but don't send the first byte
- Recipient keeps all the subsequent data and waits…..
  - Filling buffers.
    - Critical buffers…

- Send a legitimate request

GET index.html

this gets through an intrusion-detection system

then send a new segment replacing bytes 4-13 with "password-file"

A dumb example.

Neither of these attacks would work on a modern system.

57

TCP Round Trip Time and Timeout

Q: how to set TCP timeout value?

- longer than RTT
  - but RTT varies
- too short: premature timeout
  - unnecessary retransmissions
- too long: slow reaction to segment loss

Q: how to estimate RTT?

- **SampleRTT**: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **SampleRTT** will vary, want estimated RTT "smoother"
  - average several recent measurements, not just current **SampleRTT**

58

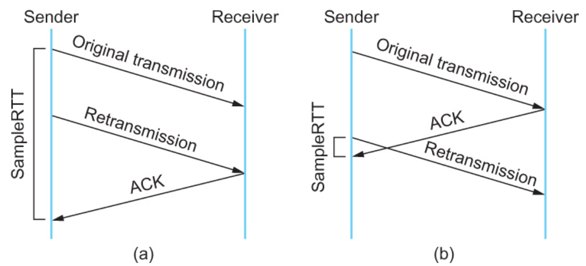
TCP Round Trip Time and Timeout

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

- Exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: $\alpha = 0.125$

59

Some RTT estimates are never good



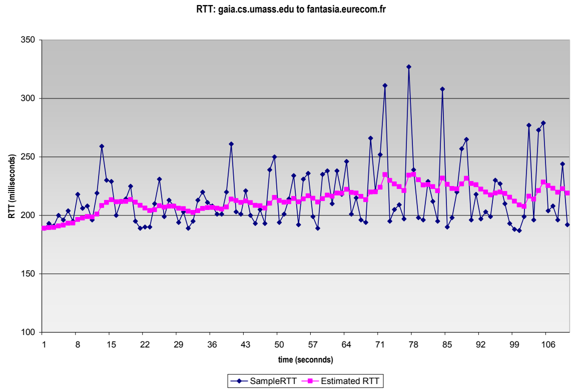
Associating the ACK with (a) original transmission versus (b) retransmission

Karn/Partridge Algorithm – Ignore retransmission in measurements

(and increase timeout; this makes retransmissions decreasingly aggressive)

60

Example RTT estimation:



61

TCP Round Trip Time and Timeout

Setting the timeout

- **EstimtedRTT** plus "safety margin"
  - large variation in **EstimatedRTT** -> larger safety margin
- first estimate of how much SampleRTT deviates from EstimatedRTT:

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

(typically, β = 0.25)
```

Then set timeout interval:

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

62

TCP reliable data transfer

- TCP creates rdt service on top of IP's unreliable service
- Pipelined segments
- Cumulative acks
- TCP uses single retransmission timer
- Retransmissions are triggered by:
  - timeout events
  - duplicate acks
- Initially consider simplified TCP sender:
  - ignore duplicate acks
  - ignore flow control, congestion control

63

TCP sender events:

data rcvd from app:

- Create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running (think of timer as for oldest unacked segment)
- expiration interval: `TimeOutInterval`

timeout:

- retransmit segment that caused timeout
- restart timer

Ack rcvd:

- If acknowledges previously unacked segments
  - update what is known to be acked
  - start timer if there are outstanding segments

64

TCP sender (simplified)

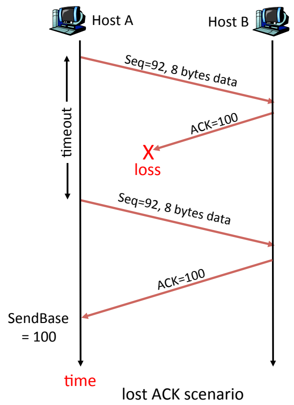
```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

loop (forever) {
  switch(event)

  event: data received from application above
      create TCP segment with sequence number NextSeqNum
      if (timer currently not running)
          start timer
      pass segment to IP
      NextSeqNum = NextSeqNum + length(data)

  event: timer timeout
      retransmit not-yet-acknowledged segment with
           smallest sequence number
      start timer

  event: ACK received, with ACK field value of y
      if (y > SendBase) {
          SendBase = y
          if (there are currently not-yet-acknowledged segments)
               start timer
          }

} /* end of loop forever */
```

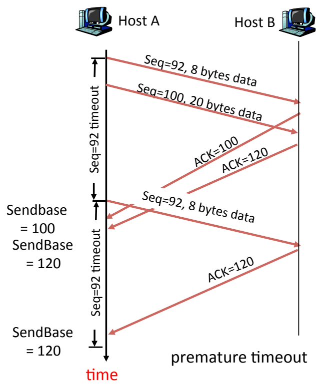
Comment:
- SendBase-1: last cumulatively ack'ed byte

Example:
- SendBase-1 = 71; y= 73, so the rcvr wants 73+ ; y > SendBase, so that new data is acked

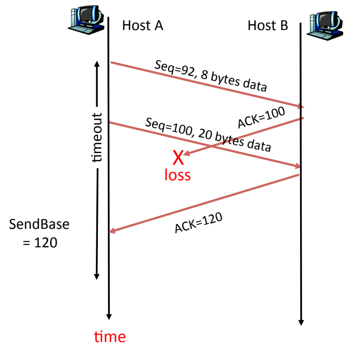
65

TCP: retransmission scenarios





TCP retransmission scenarios (more)



Implicit ACK
(e.g. **not** Go-Back-N)

ACK=120 implicitly ACK's 100 too

TCP ACK generation [RFC 1122, RFC 2581]

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

Fast Retransmit

- Time-out period often relatively long:
  - long delay before resending lost packet
- Detect lost segments via duplicate ACKs.
  - Sender often sends many segments back-to-back
  - If segment is lost, there will likely be many duplicate ACKs.
- If sender receives 3 duplicate ACKs, it supposes that segment after ACKed data was lost:
  - fast retransmit: resend segment before timer expires

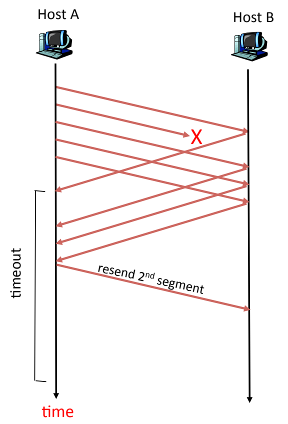


Figure 3.37 Resending a segment after triple duplicate ACK

70

Fast retransmit algorithm:

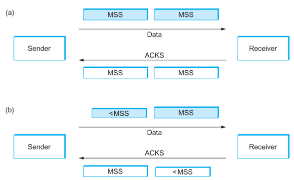
```
event: ACK received, with ACK field value of y
          if (y > SendBase) {
              SendBase = y
              if (there are currently not-yet-acknowledged segments)
                  start timer
             }
          else {
               increment count of dup ACKs received for y
               if (count of dup ACKs received for y = 3) {
                   resend segment with sequence number y
               }
```

a duplicate ACK for already ACKed segment

fast retransmit

71

Silly Window Syndrome



MSS advertises the amount a receiver can accept

If a transmitter has something to send – it will.

This means small MSS values may persist
- indefinitely.

Solution

Wait to fill each segment, but don't wait indefinitely.

NAGLE's Algorithm

If we wait too long interactive traffic is difficult

If we don't want we get *silly window* syndrome

Solution: Use a timer, when the timer expires – send the (unfilled) segment.

72

Flow Control ≠ Congestion Control

- ***Flow control*** involves preventing senders from overrunning the capacity of the receivers
- ***Congestion control*** involves preventing too much data from being injected into the network, thereby causing switches or links to become overloaded

73

Flow Control – (bad old days?)

In-Line flow control

- **XON/XOFF** (^s/^q)
- **data-link** dedicated symbols aka Ethernet
  (more in the Advanced Topic on Datacenters)

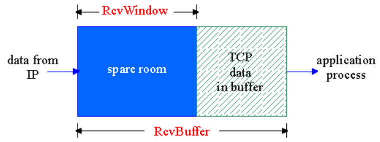
Dedicated wires

- **RTS/CTS** handshaking
- **Read (or Write) Ready**
  signals from memory interface saying slow-down/stop…

74

TCP Flow Control

- receive side of TCP connection has a receive buffer:



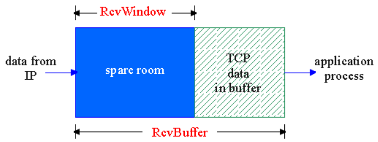
- app process may be slow at reading from buffer

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

- speed-matching service: matching the send rate to the receiving app' s drain rate

75

TCP Flow control: how it works



(Suppose TCP receiver discards out-of-order segments)

- spare room in buffer

= `RcvWindow`

= `RcvBuffer-[LastByteRcvd - LastByteRead]`

- Rcvr advertises spare room by including value of `RcvWindow` in segments
- Sender limits unACKed data to `RcvWindow`
  - guarantees receive buffer doesn' t overflow

76

TCP Connection Management

Recall: TCP sender, receiver establish "connection" before exchanging data segments

- initialize TCP variables:
  - seq. #s
  - buffers, flow control info (e.g. `RcvWindow`)
- *client:* connection initiator

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

- *server:* contacted by client

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

Three way handshake:

Step 1: client host sends TCP SYN segment to server
- specifies initial seq #
- no data

Step 2: server host receives SYN, replies with SYNACK segment
- server allocates buffers
- specifies server initial seq. #

Step 3: client receives SYNACK, replies with ACK segment, which may contain data

77

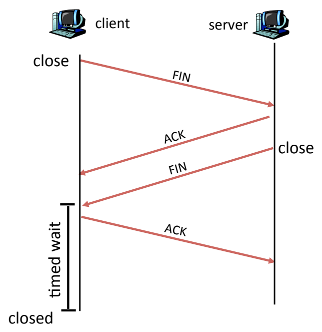
TCP Connection Management (cont.)

Closing a connection:

client closes socket: `clientSocket.close();`

Step 1: client end system sends TCP FIN control segment to server

Step 2: server receives FIN, replies with ACK. Closes connection, sends FIN.



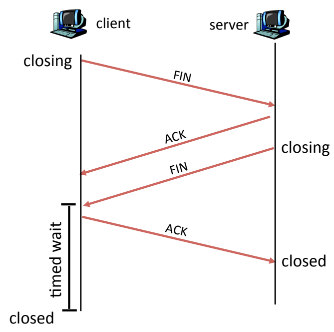
TCP Connection Management (cont.)

Step 3: client receives FIN, replies with ACK.

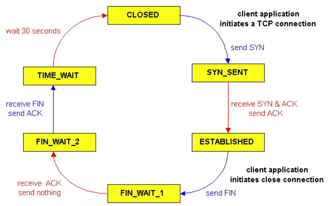
- Enters "timed wait" - will respond with ACK to received FINs

Step 4: server, receives ACK. Connection closed.

Note: with small modification, can handle simultaneous FINs.

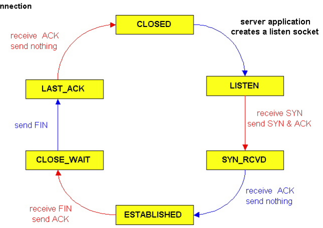


TCP Connection Management (cont)



TCP client lifecycle

TCP server lifecycle



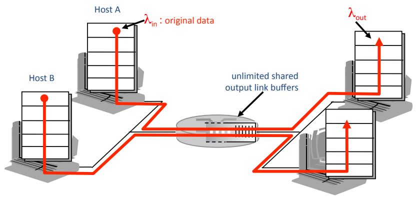
Principles of Congestion Control

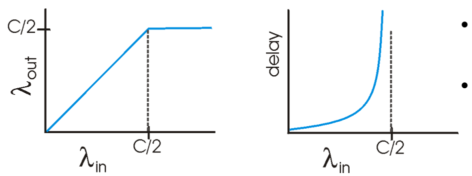
Congestion:

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

Causes/costs of congestion: scenario 1

- two senders, two receivers
- one router, infinite buffers
- no retransmission



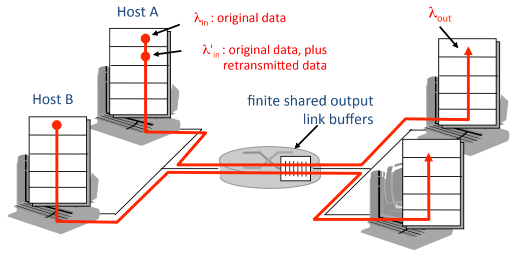


- large delays when congested
- maximum achievable throughput

82

Causes/costs of congestion: scenario 2

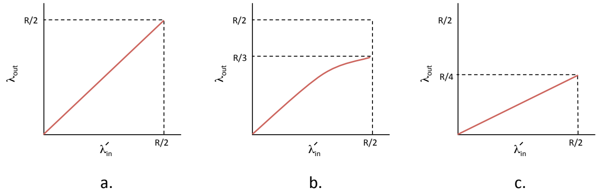
- one router, *finite* buffers
- sender retransmission of lost packet



83

Causes/costs of congestion: scenario 2

- always: $\lambda_{in} = \lambda_{out}$ (goodput)
- "perfect" retransmission only when loss: $\lambda'_{in} > \lambda_{out}$
- retransmission of delayed (not lost) packet makes λ'_{in} larger (than perfect case) for same λ_{out}



"costs" of congestion:

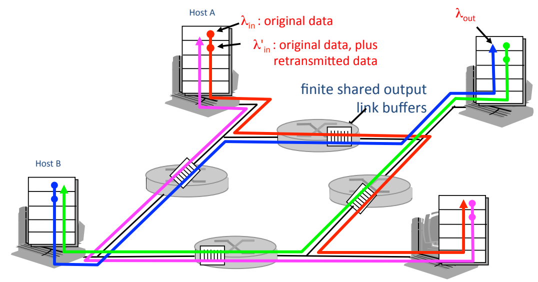
- more work (retrans) for given "goodput"
- unneeded retransmissions: link carries multiple copies of pkt

84

Causes/costs of congestion: scenario 3

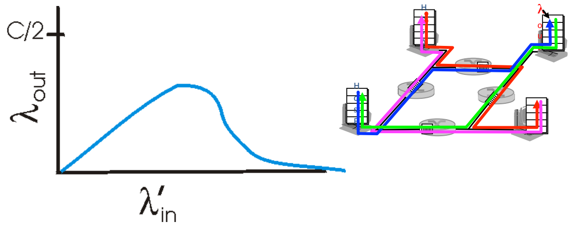
- four senders
- multihop paths
- timeout/retransmit

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



85

Causes/costs of congestion: scenario 3



Another "cost" of congestion:

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

Congestion Collapse example: Cocktail party effect

86

Approaches towards congestion control

Two broad approaches towards congestion control:

End-end congestion control:

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

Network-assisted congestion control:

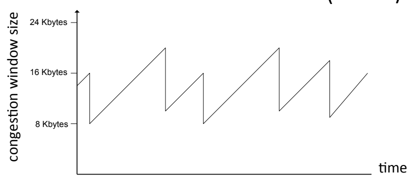
- routers provide feedback to end systems
  - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - explicit rate sender should send at

87

TCP congestion control: additive increase, multiplicative decrease

- *Approach:* increase transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **CongWin** by 1 MSS every RTT for each received ACK until loss detected (W ← W + 1/W)
  - *multiplicative decrease*: cut **CongWin** in half after loss (W←W/2)

Saw tooth behavior: probing for bandwidth



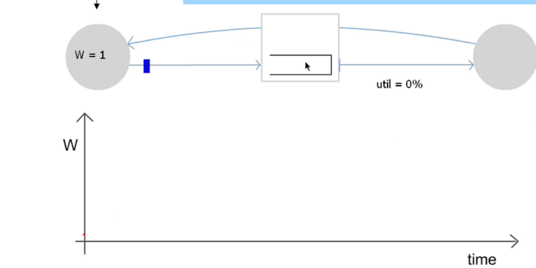
88

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



SLOW START IS NOT SHOWN!

89

TCP Congestion Control: details

- sender limits transmission:
  `LastByteSent-LastByteAcked ≤ CongWin`
- Roughly,

  rate = CongWin / RTT Bytes/sec

- **CongWin** is dynamic, function of perceived network congestion

How does sender perceive congestion?

- loss event = timeout *or* 3 duplicate acks
- TCP sender reduces rate (**CongWin**) after loss event

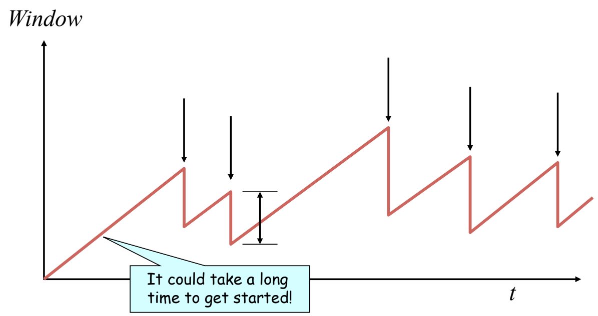
three mechanisms:

- AIMD
- slow start
- conservative after timeout events

90

AIMD Starts Too Slowly!

Need to start with a small CWND to avoid overloading the network.



91

TCP Slow Start

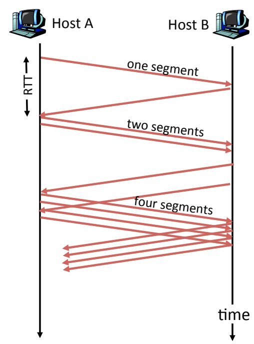
- When connection begins, **CongWin** = 1 MSS
  - Example: MSS = 500 bytes & RTT = 200 msec
  - initial rate = 20 kbps
- available bandwidth may be >> MSS/RTT
  - desirable to quickly ramp up to respectable rate

- When connection begins, increase rate exponentially fast until first loss event

92

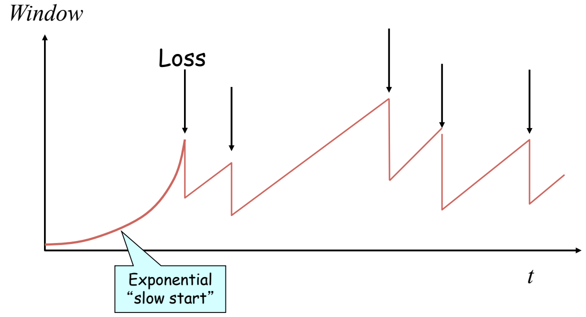
TCP Slow Start (more)

- When connection begins, increase rate exponentially until first loss event:
  - double **CongWin** every RTT
  - done by incrementing **CongWin** for every ACK received
- Summary: initial rate is slow but ramps up exponentially fast



93

Slow Start and the TCP Sawtooth



Why is it called slow-start? Because TCP originally had no congestion control mechanism. The source would just start by sending a whole window's worth of data.

94

Refinement: inferring loss

- After 3 dup ACKs:
  - **CongWin** is cut in half
  - window then grows linearly
- But after timeout event:
  - **CongWin** instead set to 1 MSS;
  - window then grows exponentially
  - to a threshold, then grows linearly

Philosophy:

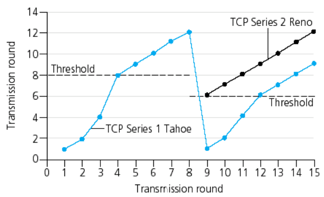
- 3 dup ACKs indicates network capable of delivering some segments
- timeout indicates a "more alarming" congestion scenario

95

Refinement

Q: When should the exponential increase switch to linear?

A: When **CongWin** gets to 1/2 of its value before timeout.



Implementation:

- Variable Threshold
- At loss event, Threshold is set to 1/2 of CongWin just before loss event

96

Summary: TCP Congestion Control

- When **CongWin** is below **Threshold**, sender in slow-start phase, window grows exponentially.
- When **CongWin** is above **Threshold**, sender is in congestion-avoidance phase, window grows linearly.
- When a triple duplicate ACK occurs, **Threshold** set to **CongWin/2** and **CongWin** set to **Threshold**.
- When timeout occurs, **Threshold** set to **CongWin/2** and **CongWin** is set to 1 MSS.

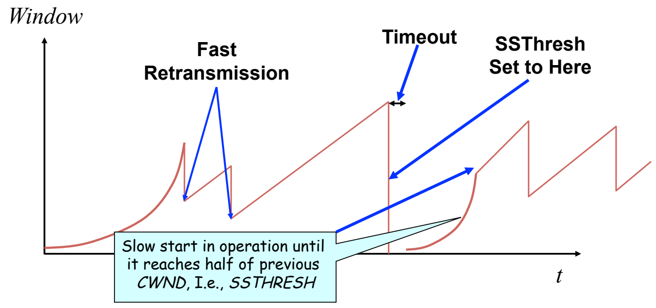
97

TCP sender congestion control

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2, CongWin = Threshold, Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
SS or CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, Set state to "Slow Start"	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

98

Repeating Slow Start After Timeout



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

99

TCP throughput

- What's the average throughout of TCP as a function of window size and RTT?
  - Ignore slow start
- Let W be the window size when loss occurs.
- When window is W, throughput is W/RTT
- Just after loss, window drops to W/2, throughput to W/2RTT.
- Average throughout: .75 W/RTT

100

TCP Futures: TCP over "long, fat pipes"

- Example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- Requires window size W = 83,333 in-flight segments
- Throughput in terms of loss rate *p*:

$$\frac{1.22 \cdot MSS}{RTT\sqrt{p}}$$

- ➜ $L = 2 \cdot 10^{-10}$ *Ouch!*
- New versions of TCP for high-speed

101

Calculation on Simple Model
(cwnd in units of MSS)

- Assume loss occurs whenever cwnd reaches W
  - Recovery by fast retransmit
- Window: W/2, W/2+1, W/2+2, …W, W/2, …
  - W/2 RTTs, then drop, then repeat
- Average throughput: .75W(MSS/RTT)
  - One packet dropped out of (W/2)*(3W/4)
  - Packet drop rate p = (8/3) W^{-2}
- Throughput = (MSS/RTT) sqrt(3/2p)

HINT: KNOW THIS SLIDE

102

Three Congestion Control Challenges – or Why AIMD?

- Single flow adjusting to bottleneck bandwidth
  - Without any *a priori* knowledge
  - Could be a Gbps link; could be a modem
- Single flow adjusting to variations in bandwidth
  - When bandwidth decreases, must lower sending rate
  - When bandwidth increases, must increase sending rate
- Multiple flows sharing the bandwidth
  - Must avoid overloading network
  - And share bandwidth "fairly" among the flows

103

Problem #1: Single Flow, Fixed BW

- Want to get a first-order estimate of the available bandwidth
  - Assume bandwidth is fixed
  - Ignore presence of other flows
- Want to start slow, but rapidly increase rate until packet drop occurs ("slow-start")
- Adjustment:
  - cwnd initially set to 1 (MSS)
  - cwnd++ upon receipt of ACK

104

Problems with Slow-Start

- Slow-start can result in many losses
  - Roughly the size of cwnd ~ BW*RTT
- Example:
  - At some point, cwnd is enough to fill "pipe"
  - After another RTT, cwnd is double its previous value
  - All the excess packets are dropped!
- Need a more gentle adjustment algorithm once have rough estimate of bandwidth
  - Rest of design discussion focuses on this

105

Problem #2: Single Flow, Varying BW

Want to track available bandwidth

- Oscillate around its current value
- If you never send more than your current rate, you won't know if more bandwidth is available

Possible variations: (in terms of change per RTT)

- Multiplicative increase or decrease:

  cwnd→ cwnd * / a

- Additive increase or decrease:

  cwnd→ cwnd +- b

106

Four alternatives

- AIAD: gentle increase, gentle decrease
- AIMD: gentle increase, drastic decrease
- MIAD: drastic increase, gentle decrease
  - too many losses: eliminate
- MIMD: drastic increase and decrease

107

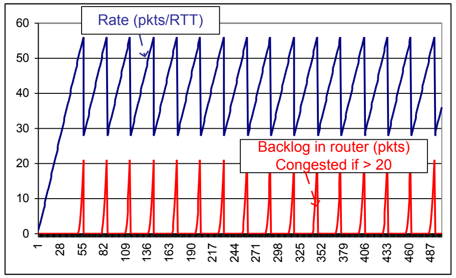
Problem #3: Multiple Flows

- Want steady state to be "fair"
- Many notions of fairness, but here just require two identical flows to end up with the same bandwidth
- This eliminates MIMD and AIAD
  - As we shall see…
- AIMD is the only remaining solution!
  - Not really, but close enough….

108

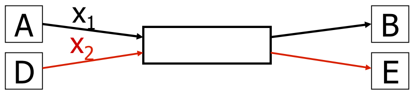
Recall Buffer and Window Dynamics

- No congestion → x increases by one packet/RTT every RTT
- Congestion → decrease x by factor 2

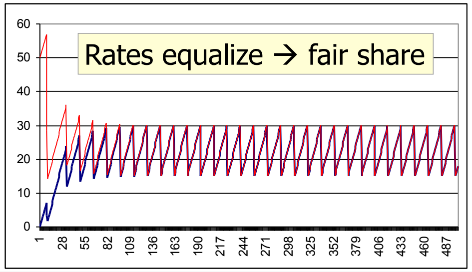


109

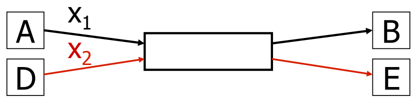
AIMD Sharing Dynamics



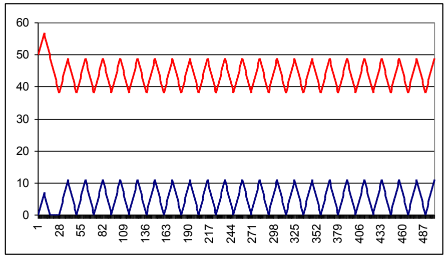
- No congestion → rate increases by one packet/RTT every RTT
- Congestion → decrease rate by factor 2



AIAD Sharing Dynamics

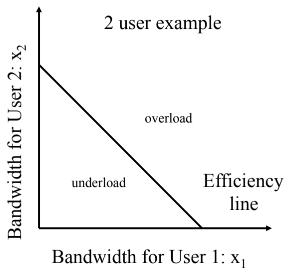


- No congestion → x increases by one packet/RTT every RTT
- Congestion → decrease x by 1



Simple Model of Congestion Control

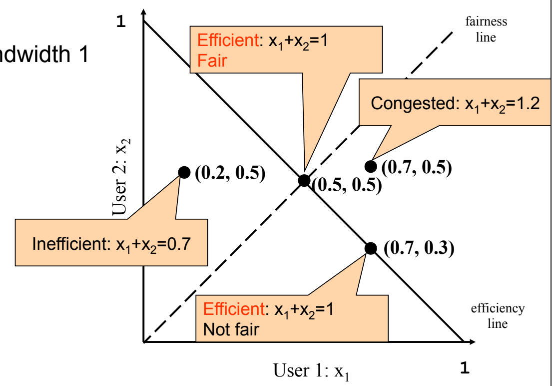
- Two TCP connections
  - Rates x_1 and x_2
- Congestion when sum>1
- Efficiency: sum near 1
- Fairness: x's converge



112

Example

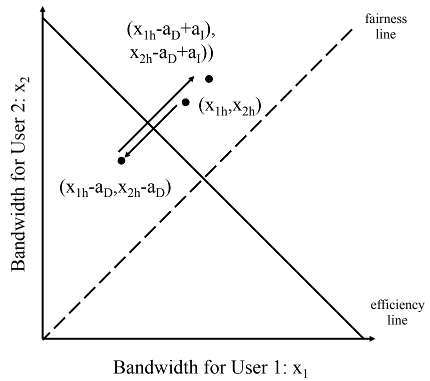
- Total bandwidth 1



113

AIAD

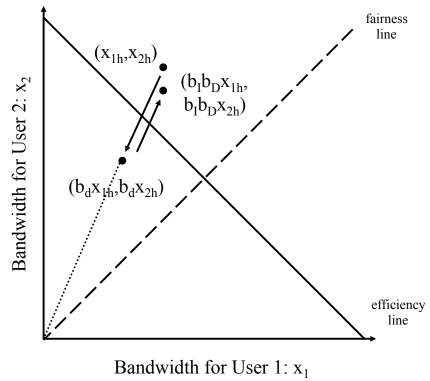
- Increase: $x + a_I$
- Decrease: $x - a_D$
- Does not converge to fairness



114

MIMD

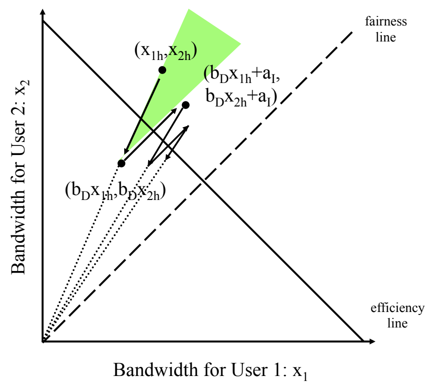
- Increase: $x*b_I$
- Decrease: $x*b_D$
- Does not converge to fairness



115

AIMD

- Increase: $x+a_D$
- Decrease: $x*b_D$
- Converges to fairness

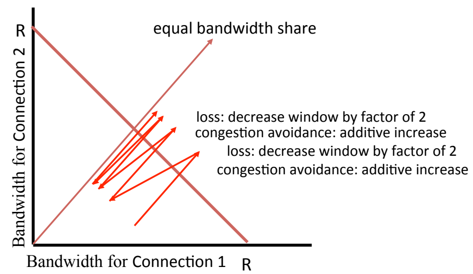


116

Why is AIMD fair?
(a pretty animation…)

Two competing sessions:

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



117

Fairness (more)

Fairness and UDP

- Multimedia apps may not use TCP
  - do not want rate throttled by congestion control
- Instead use UDP:
  - pump audio/video at constant rate, tolerate packet loss
- (Ancient yet ongoing) Research area: TCP friendly

Fairness and parallel TCP connections

- nothing prevents app from opening parallel connections between 2 hosts.
- Web browsers do this
- Example: link of rate R supporting 9 connections;
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2 !
- ***Recall*** Multiple browser sessions (and the potential for syncronized loss)

118

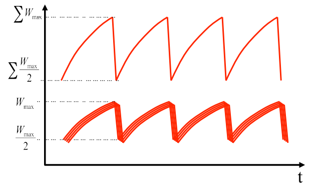
Some TCP issues outstanding…

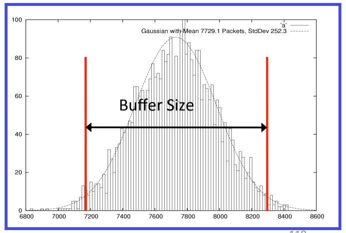
Synchronized Flows

- 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





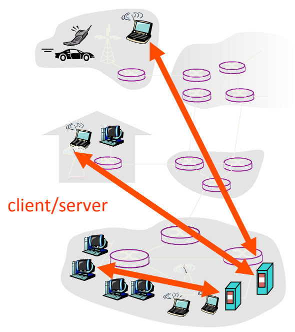
119

Topic 6 – Applications

- Traditional Applications (web)
- Infrastructure Services (DNS)
- Multimedia Applications (SIP)
- P2P Networks

2

Client-server architecture



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

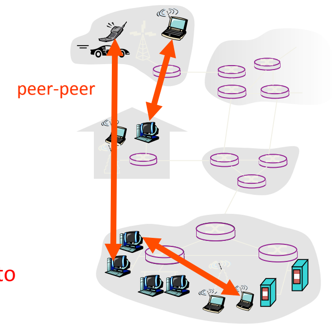
clients:
- – communicate with server
- – may be intermittently connected
- – may have dynamic IP addresses
- – do not communicate directly with each other

3

Pure P2P architecture

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

Highly scalable but difficult to manage



4

Hybrid of client-server and P2P

Skype
- – voice-over-IP P2P application
- – centralized server: finding address of remote party:
- – client-client connection: direct (not through server)

Instant messaging
- – chatting between two users is P2P
- – centralized service: client presence detection/ location
  - user registers its IP address with central server when it comes online
  - user contacts central server to find IP addresses of buddies

5

Processes communicating

Process: program running within a host.

- within same host, two processes communicate using inter-process communication (defined by OS).
- processes in different hosts communicate by exchanging messages

Client process: process that initiates communication

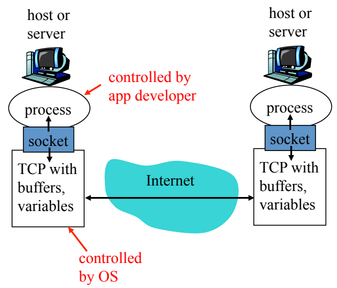
Server process: process that waits to be contacted

❒ Note: applications with P2P architectures have client processes & server processes

6

Sockets – an abstraction hiding layers

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process



❒ Socket API: (1) choice of transport protocol; (2) ability to fix a few parameters

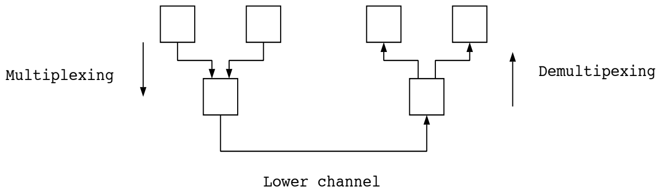
7

Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- Q: does IP address of host on which process runs suffice for identifying the process?
  - A: No, *many* processes can be running on same host
- *identifier* includes both IP address and port numbers associated with process on host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - Port number: 80
- more shortly…

8

Recall: Multiplexing is a service provided by (each) layer too!



Application: one web-server multiple sets of content
Host: one machine multiple services
Network: one physical box multiple addresses (like vns.cl.cam.ac.uk)
….

UNIX: /etc/protocols = examples of different transport-protocols on top of IP

UNIX: /etc/services = examples of different (TCP/UDP) services – by port

(THESE FILES ARE EXAMPLES OF NAME SERVICES)

App-layer protocol defines

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

Public-domain protocols:
- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

Proprietary protocols:
- e.g., Skype

10

What transport service does an app need?

Data loss
- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer

Timing
- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

Throughput
- ❒ some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- ❒ other apps ("elastic apps") make use of whatever throughput they get

Security
- ❒ Encryption, data integrity, …

Mysterious secret of *Transport*
- There is more than sort of *transport* layer

Shocked?
I seriously doubt it…

We call the two most common TCP and UDP

11

Naming

- Internet has one global system of addressing: IP
  - By explicit design
- And one global system of naming: DNS
  - Almost by accident
- At the time, only items worth naming were hosts
  - A mistake that causes many painful workarounds
- Everything is now named relative to a host
  - Content is most notable example (URL structure)

12

Logical Steps in Using Internet

- Human has name of entity she wants to access
  - Content, host, etc.
- Invokes an application to perform relevant task
  - Using that name
- App invokes DNS to translate name to address
- App invokes transport protocol to contact host
  - Using address as destination

13

Addresses vs Names

- Scope of relevance:
  - App/user is primarily concerned with names
  - Network is primarily concerned with addresses
- Timescales:
  - Name lookup once (or get from cache)
  - Address lookup on each packet
- When moving a host to a different subnet:
  - The address changes
  - The name does not change
- When moving content to a differently named host
  - Name and address both change!

14

Relationship Betw'n Names/ Addresses

- Addresses can change underneath
  - Move www.cnn.com to 4.125.91.21
  - Humans/Apps should be unaffected
- Name could map to multiple IP addresses
  - www.cnn.com 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.cnn.com and cnn.com
  - Mnemonic stable name, and dynamic canonical name
    - Canonical name = actual name of host

15

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.cnn.com

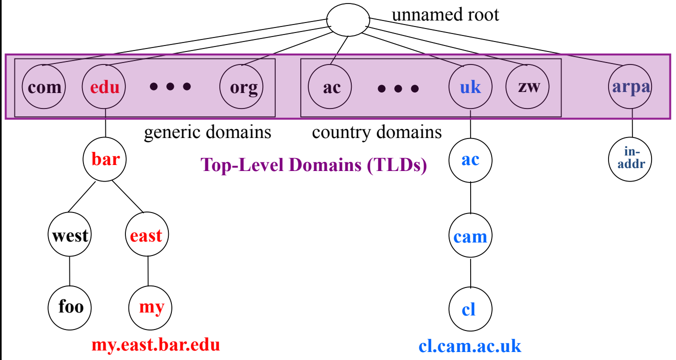
16

Domain Name System (DNS)

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

17

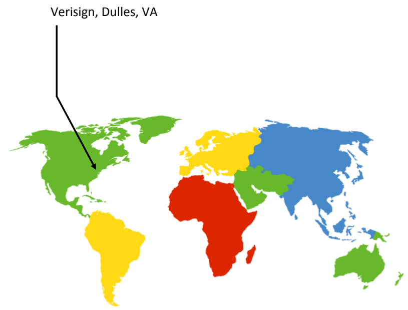
Distributed Hierarchical Database



18

DNS Root

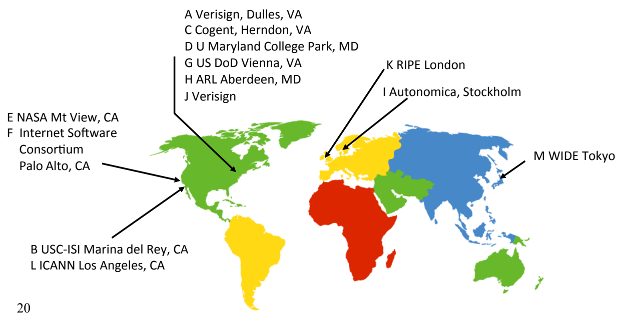
- Located in Virginia, USA
- How do we make the root scale?



19

DNS Root Servers

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



20

DNS Root Servers

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

21

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

22

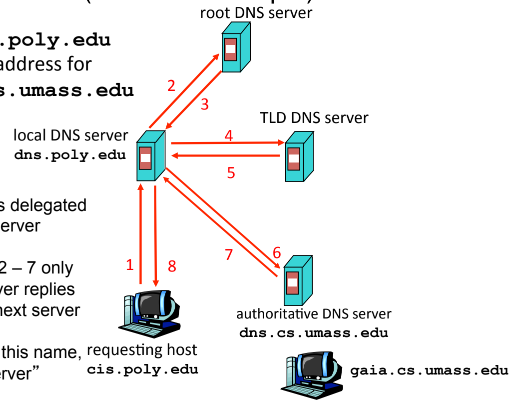
How Does Resolution Happen?

(**Iterative** example)

Host at **cis.poly.edu** wants IP address for **gaia.cs.umass.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"

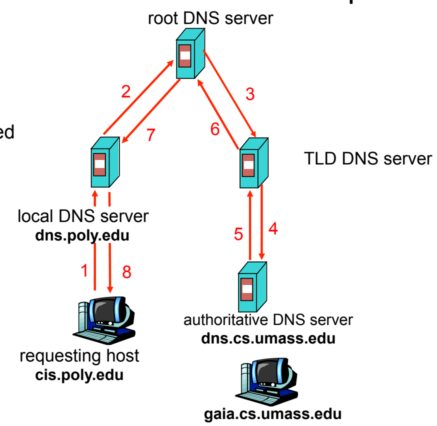


23

DNS name resolution **recursive** example

recursive query:

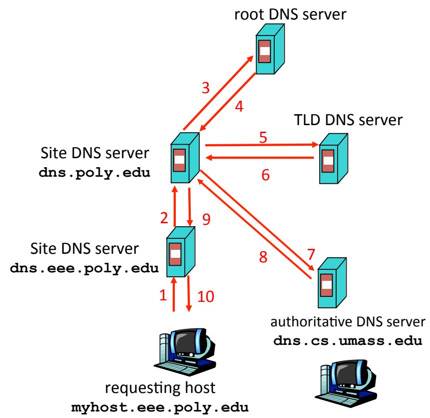
- puts burden of name resolution on contacted name server
- heavy load?



24

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



25

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.cnn.com) 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

26

Negative Caching

- Remember things that don't work
  - Misspellings like *www.cnn.comm* and *www.cnnn.com*
  - 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

27

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

28

DNS Measurements (MIT data from 2000)

- What is being looked up?
  - ~60% requests for A records
  - ~25% for PTR records
  - ~5% for MX records
  - ~6% for ANY records
- How long does it take?
  - Median ~100msec (but 90th percentile ~500msec)
  - 80% have no referrals; 99.9% have fewer than four
- Query packets per lookup: ~2.4
  - But this is misleading….

29

DNS Measurements (MIT data from 2000)

- Does DNS give answers?
  - ~23% of lookups fail to elicit an answer!
  - ~13% of lookups result in NXDOMAIN (or similar)
    - Mostly reverse lookups
  - Only ~64% of queries are successful!
    - *How come the web seems to work so well?*
- ~ 63% of DNS packets in unanswered queries!
  - Failing queries are frequently retransmitted
  - 99.9% successful queries have ≤2 retransmissions

30

DNS Measurements (MIT data from 2000)

- Top 10% of names accounted for ~70% of lookups
  - Caching should really help!
- 9% of lookups are unique
  - Cache hit rate can never exceed 91%
- Cache hit rates ~ 75%
  - But caching for more than 10 hosts doesn't add much

31

A Common Pattern.....

- Distributions of various metrics (file lengths, access patterns, etc.) often have two properties:
  - Large fraction of total metric in the top 10%
  - Sizable fraction (~10%) of total fraction in low values
- Not an exponential distribution
  - Large fraction is in top 10%
  - But low values have very little of overall total
- Lesson: have to pay attention to both ends of dist.
- Here: caching helps, but not a panacea

32

Moral of the Story

- If you design a highly resilient system, many things can be going wrong without you noticing it!

and this is a **good** thing

33

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!

34

Cache Poisoning

- Suppose you are a Bad Guy and you control the name server for foobar.com. You receive a request to resolve www.foobar.com and

```
;; QUESTION SECTION:
;www.foobar.com.          IN    A

;; ANSWER SECTION:
www.foobar.com.     300   IN    A     212.44.9.144

;; AUTHORITY SECTION:
foobar.com.         600   IN    NS    dns1.foobar.com.
foobar.com.         600   IN    NS    google.com.

;; ADDITIONAL SECTION:
google.com.         5     IN    A     212.44.9.155
```

Evidence of the attack disappears 5 seconds later!

A foobar.com machine, *not* google.com

35

The Web – Precursor

Ted Nelson

- **1967**, Ted Nelson, Xanadu:
  - A world-wide publishing network that would allow information to be stored not as separate files but as connected literature
  - Owners of documents would be automatically paid via electronic means for the virtual copying of their documents
- Coined the term "Hypertext"
  - Influenced research community
    - Who then missed the web.....

36

The Web – History

Tim Berners-Lee

- CS grad turned physicist trying to solve real problem
  - Distributed access to data
- World Wide Web (WWW): a distributed database of "pages" linked through Hypertext Transport Protocol (HTTP)
  - First HTTP implementation - 1990
    - Tim Berners-Lee at CERN
  - HTTP/0.9 – 1991
    - Simple GET command for the Web
  - HTTP/1.0 –1992
    - Client/Server information, simple caching
  - HTTP/1.1 - 1996

37

Why Didn't CS Research Invent Web?

HTML is precisely what we were trying to PREVENT— ever-breaking links, links going outward only, quotes you can't follow to their origins, no version management, no rights management.

– Ted Nelson

Academics get paid for being clever, not for being right.

–Don Norman

38

Why So Successful?

- What do the web, youtube, fb 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....

39

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!

40

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)

41

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`

42

HyperText Transfer Protocol (HTTP)

- Request-response protocol
- Reliance on a global namespace
- Resource *metadata*
- *Stateless*
- ASCII format

% telnet www.icir.org 80
GET /jdoe/ HTTP/1.0
<blank line, i.e., CRLF>

43

Steps in HTTP Request

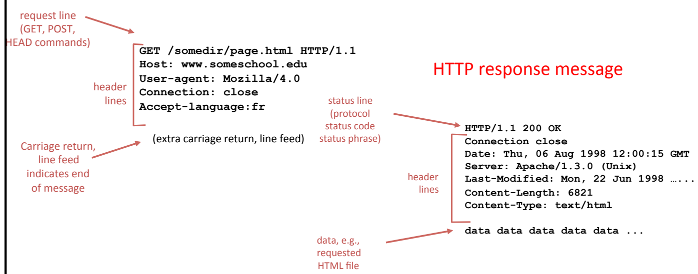
- HTTP Client initiates TCP connection to server
  - SYN
  - SYNACK
  - ACK
- Client sends HTTP request to server
  - Can be piggybacked on TCP's ACK
- HTTP Server responds to request
- Client receives the request, terminates connection
- TCP connection termination exchange

How many RTTs for a single request?

44

Client-Server Communication

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



45

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

46

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

47

HTTP is *Stateless*

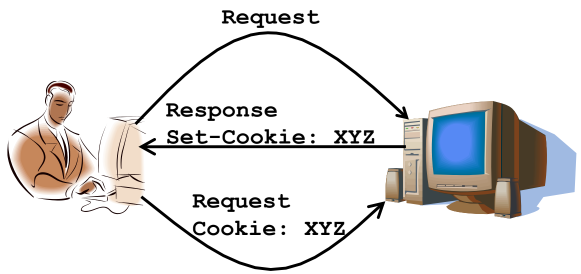
- Each request-response treated independently
  - Servers *not* required to retain state
- **Good**: Improves scalability on the server-side
  - Failure handling is easier
  - Can handle higher rate of requests
  - Order of requests doesn't matter
- **Bad**: Some applications need persistent state
 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

48

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



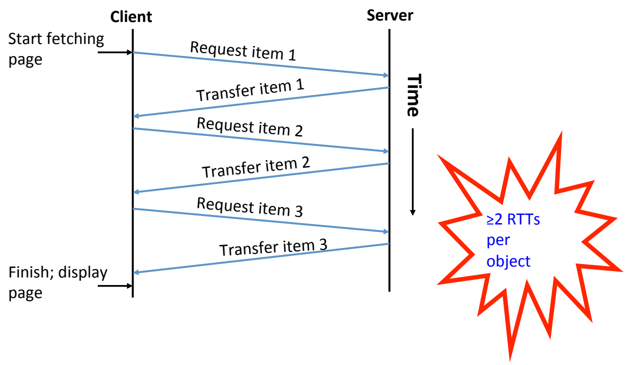
49

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*

50

Fetch HTTP Items: Stop & Wait

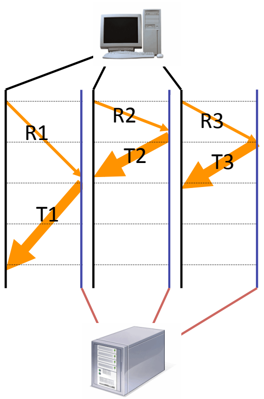


51

Improving HTTP Performance:

Concurrent Requests & Responses

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



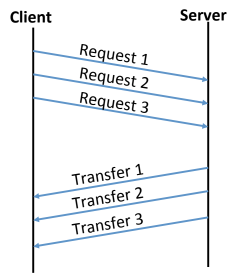
- Client = ☺
- Server = ☺
- Network = ☹ Why?

52

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



53

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

54

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

55

Scorecard: Getting n Large Objects

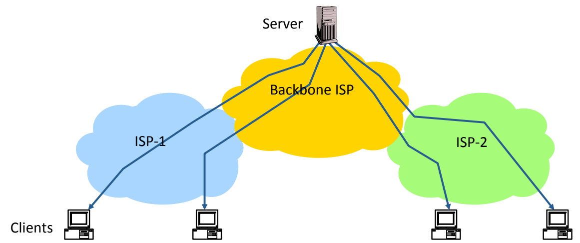
Time dominated by bandwidth

- 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..

56

Improving HTTP Performance:
Caching

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



57

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

58

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

59

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 /~ee122/fa07/ HTTP/1.1
Host: inst.eecs.berkeley.edu
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

60

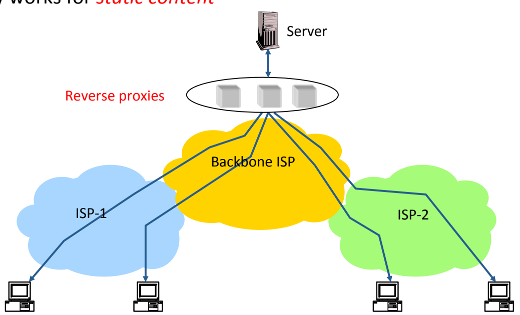
Improving HTTP Performance:

Caching with Reverse Proxies

Cache documents close to **server**

→ decrease server load

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

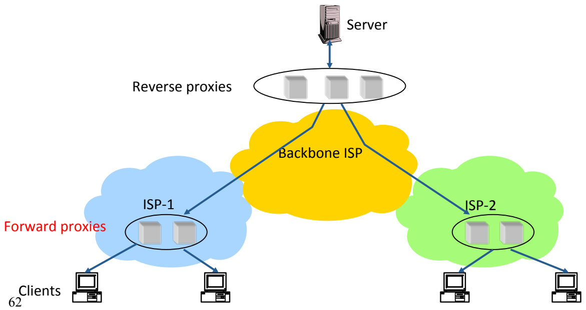


61

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



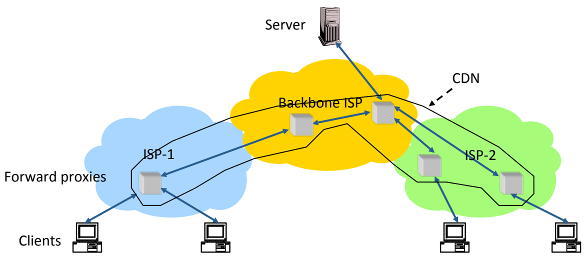
62

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*

63

Improving HTTP Performance:
Caching with CDNs (cont.)



64

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.cnn.com/image-of-the-day.gif* becomes *http://a128.g.akamai.net/image-of-the-day.gif*
- *Requests now sent to CDN's infrastructure…*

65

CDN examples

Hosting: Multiple Sites Per Machine

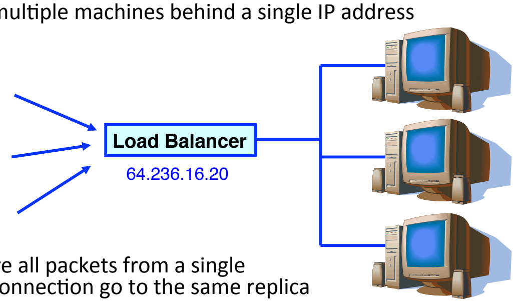
- Multiple Web sites on a single machine
  - Hosting company runs the Web server on behalf of multiple sites (*e.g.*, www.foo.com and www.bar.com)
- Problem: `GET /index.html`
  - `www.foo.com/index.html` or `www.bar.com/index.html`?
- Solutions:
  - Multiple server processes on the same machine
   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

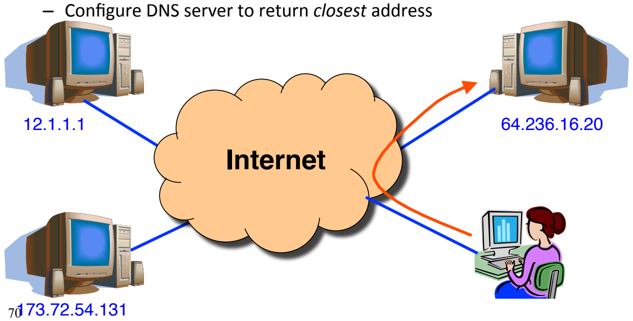
- Single IP address, multiple machines
  - Run multiple machines behind a single IP address



  - Ensure all packets from a single TCP connection go to the same replica

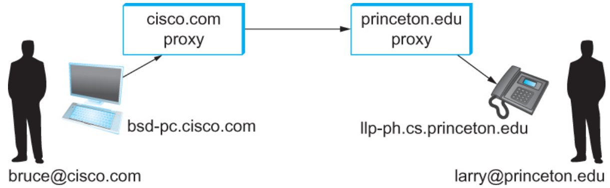
Multi-Hosting at Several Locations

- Multiple addresses, multiple machines
  - Same name but different addresses for all of the replicas
  - Configure DNS server to return *closest* address



70

SIP - VoIP



Establishing communication through SIP proxies.

71

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")

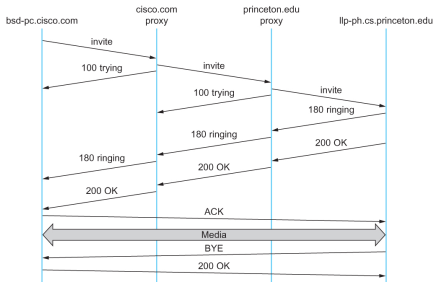
72

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

73

Multimedia Applications

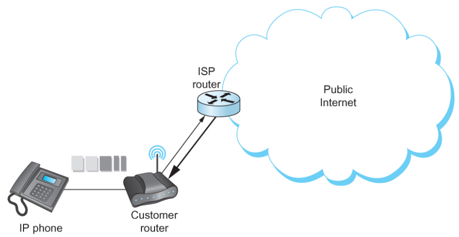


Message flow for a basic SIP session

74

The (still?) missing piece:

Resource Allocation for Multimedia Applications

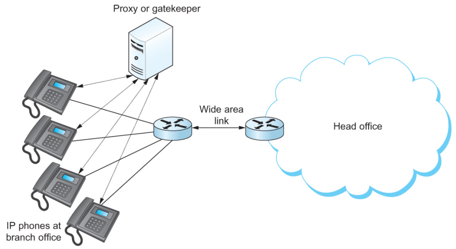


I can 'differentiate' VoIP from data but…
I can only control data going into the Internet

75

Multimedia Applications

- Resource Allocation for Multimedia Applications



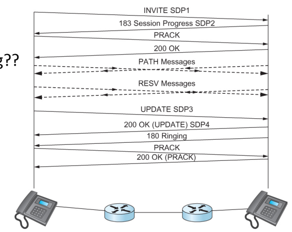
Admission control using session control protocol.

76

Resource Allocation for Multimedia Applications

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

Co-ordination of SIP signaling and resource reservation.



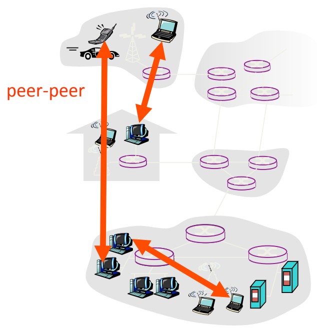
So where does it happen?
Inside single institutions or *domains of control…..*
(Universities, Hospitals, big corp…)

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……

77

Pure P2P architecture

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

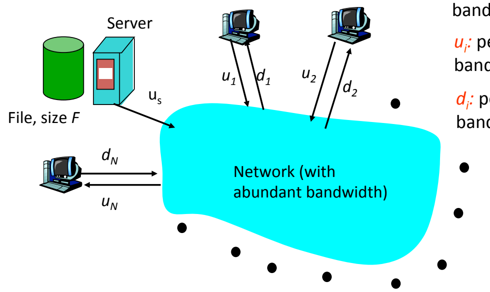


- Three topics:
  - File distribution
  - Searching for information
  - Case Study: Skype

78

File Distribution: Server-Client vs P2P

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



u_s: server upload bandwidth

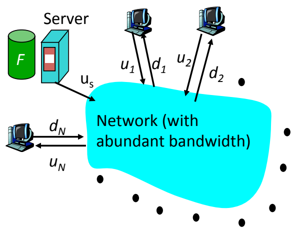
u_i: peer i upload bandwidth

d_i: peer i download bandwidth

79

File distribution time: server-client

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



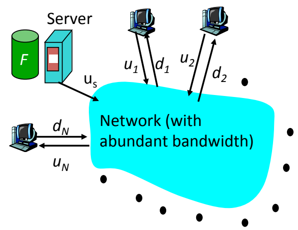
Time to distribute *F* to *N* clients using client/server approach $= d_{cs} = \max_i \{ NF/u_s, F/\min_i(d_i) \}$

increases linearly in N (for large N)

80

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: $u_s + \sum u_i$

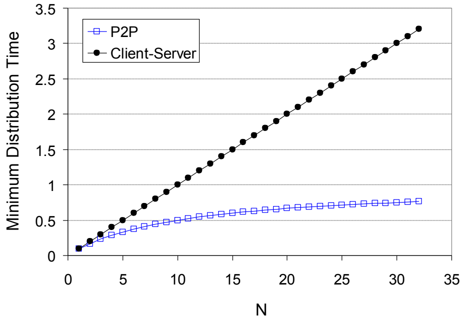


$d_{P2P} = \max \{ F/u_s, F/\min_i(d_i), NF/(u_s + \sum_i u_i) \}$

81

Server-client vs. P2P: example

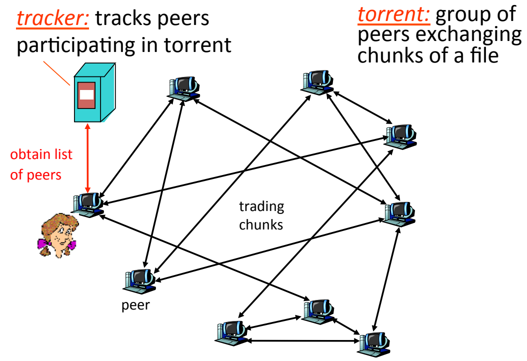
Client upload rate = u, F/u = 1 hour, u_s = 10u, $d_{min} \geq u_s$



82

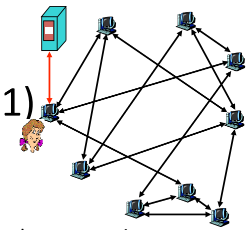
File distribution: BitTorrent

❒ P2P file distribution



83

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

84

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

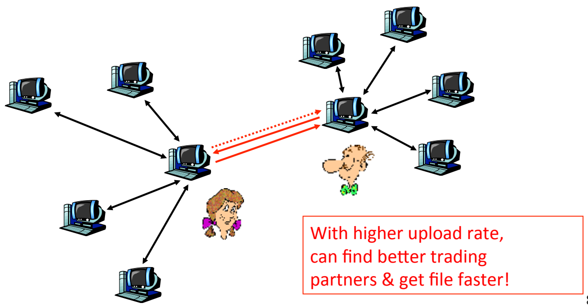
❒ Alice sends chunks to four neighbors currently sending her chunks *at the highest rate*
  - ❖ re-evaluate top 4 every 10 secs

❒ every 30 secs: randomly select another peer, starts sending chunks
  - ❖ newly chosen peer may join top 4
  - ❖ "optimistically unchoke"

85

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



86

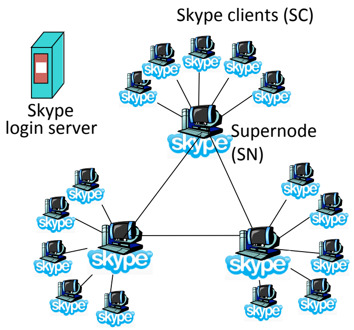
Distributed Hash Table (DHT)

- 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

87

P2P Case study: Skype

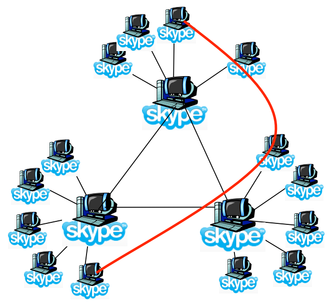
- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SNs
- Index maps usernames to IP addresses; distributed over SNs



88

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



89

Topic 7: The Datacenter

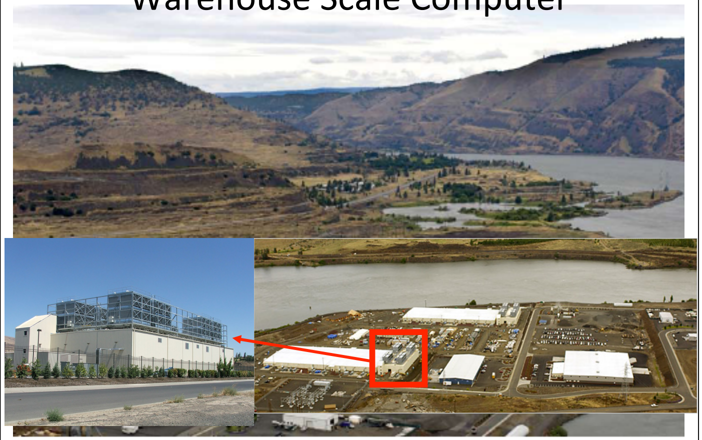
(DC/Data Center/Data Centre/….)

Our goals:

- Datacenters are the new Internet; regular Internet has become mature (ossified); datacenter along with wireless are a leading edge of new problems and new solutions
- Architectures and thoughts
  - Where do we start?
  - old ideas are new again: VL2
  - c-Through, Flyways, and all that jazz
- Transport layer obsessions:
  - TCP for the Datacenter (DCTCP)
  - recycling an idea (Multipath TCP)
  - Stragglers and Incast

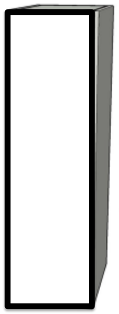
2

Google Oregon Warehouse Scale Computer

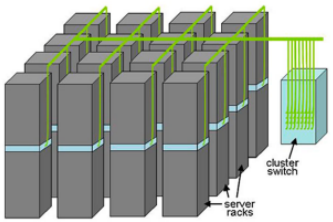


Equipment Inside a WSC

Server (in rack format):
1 ¾ inches high "1U", x 19 inches x 16-20 inches: 8 cores, 16 GB DRAM, 4x1 TB disk



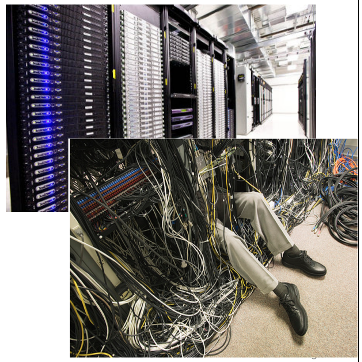
7 foot Rack: 40-80 servers + Ethernet local area network (1-10 Gbps) switch in middle ("rack switch")



Array (aka cluster): 16-32 server racks + larger local area network switch ("array switch") 10X faster => cost 100X: cost f(N^2)

4

Server, Rack, Array



Data warehouse?

If you have a Petabyte,
you might have a datacenter

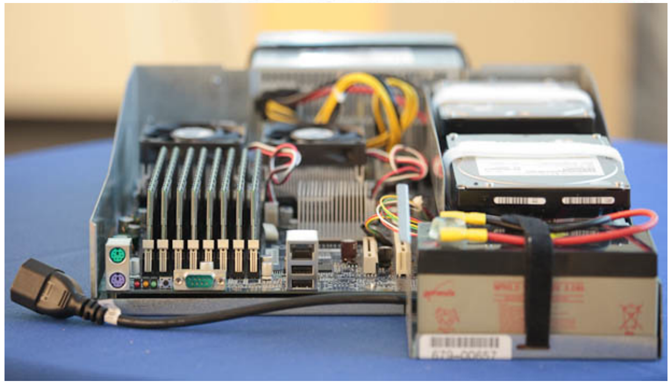
If your paged at 3am because you only have a few Petabyte left,
you might have a data warehouse

Luiz Barroso (Google), 2009

The slide most likely to get out of date…

6

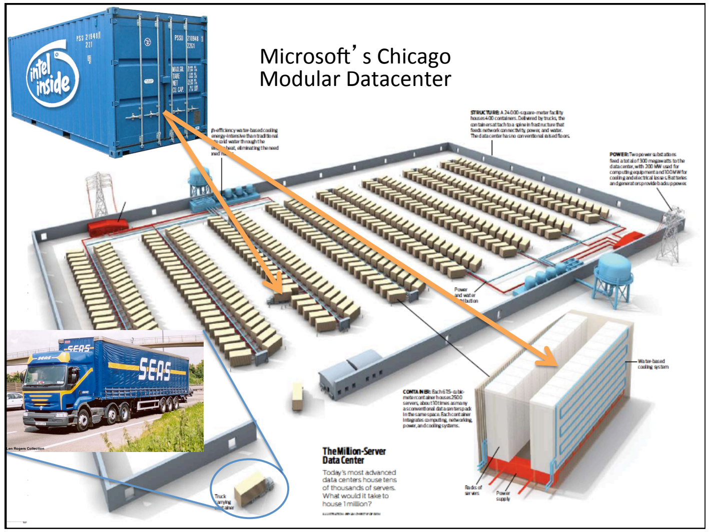
Google Server Internals



7

8

Microsoft's Chicago
Modular Datacenter



Some Differences Between Commodity DC Networking and Internet/WAN

Characteristic	Internet/WAN	Commodity Datacenter
Latencies	Milliseconds to Seconds	Microseconds
Bandwidths	Kilobits to Megabits/s	Gigabits to 10's of Gbits/s
Causes of loss	Congestion, link errors, ...	Congestion
Administration	Distributed	Central, single domain
Statistical Multiplexing	Significant	Minimal, 1-2 flows dominate links
Incast	Rare	Frequent, due to synchronized responses

10

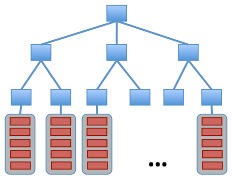
Coping with Performance in Array

Lower latency to DRAM in another server than local disk
Higher bandwidth to local disk than to DRAM in another server

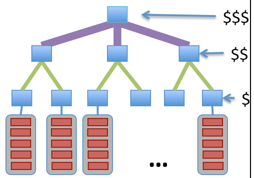
	Local	Rack	Array
Racks	--	1	30
Servers	1	80	2400
Cores (Processors)	8	640	19,200
DRAM Capacity (GB)	16	1,280	38,400
Disk Capacity (GB)	4,000	320,000	9,600,000
DRAM Latency (microseconds)	0.1	100	300
Disk Latency (microseconds)	10,000	11,000	12,000
DRAM Bandwidth (MB/sec)	20,000	100	10
Disk Bandwidth (MB/sec)	200	100	10

Datacenter design 101

- Naive topologies are tree-based
  same boxes, and same b/w links
  - Poor performance
  - Not fault tolerant



- An early solution; speed hierarchy (fewer expensive boxes at the top)
  - Boxes at the top run out of *capacity (bandwidth)*
  - but even the $ boxes needed
    $$$ abilities (forwarding table size)

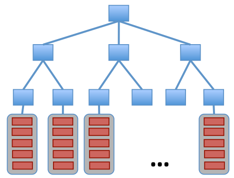


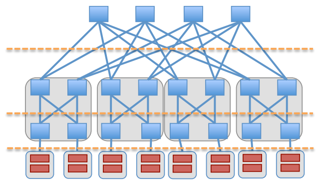
12

Data Center 102

- Tree leads to FatTree
- All bi-sections have same bandwidth

This is not the only solution…



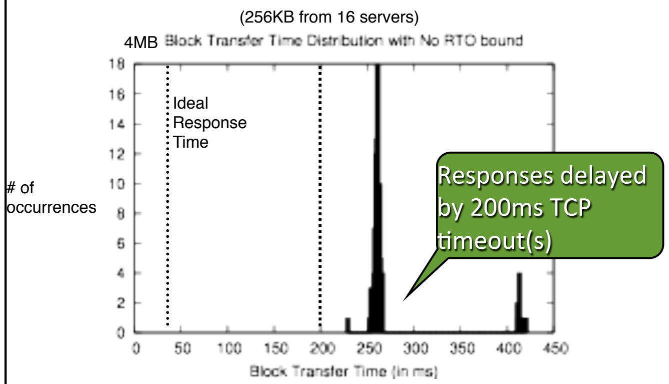


13

Latency-sensitive Apps

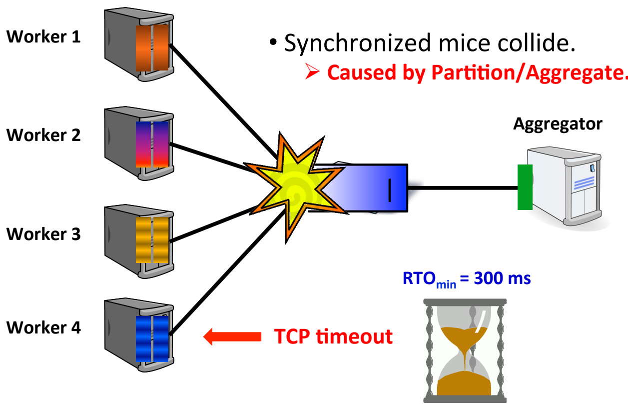
- Request for 4MB of data sharded across 16 servers (256KB each)
- How long does it take for all of the 4MB of data to return?

Timeouts Increase Latency



Incast

- Synchronized mice collide.
  - **Caused by Partition/Aggregate.**



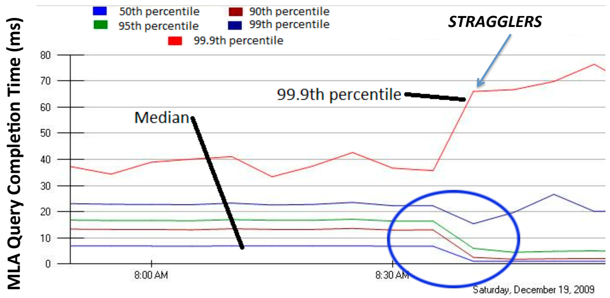
Applications Sensitive to 200ms TCP Timeouts

- "Drive-bys" affecting single-flow request/ response
- Barrier-Sync workloads
  - Parallel cluster filesystems (Incast workloads)
  - Massive multi-server queries
    - Latency-sensitive, customer-facing

The ***last result delivered*** is referred to as a ***straggler***.

Stragglers can be caused by one off (drive-by) events but also by incast congestion which may occur for every map-reduce or database record retrieve or distributed filesystem read….

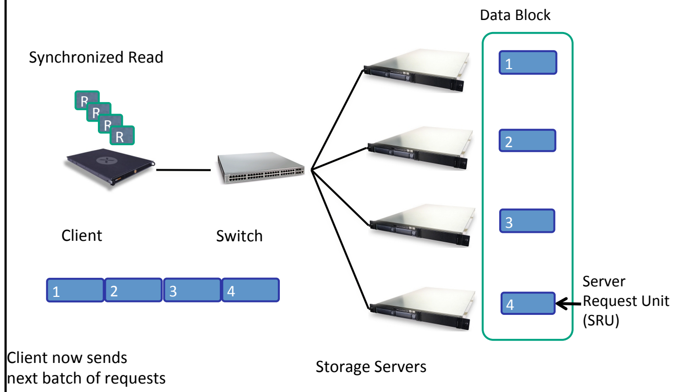
Incast Really Happens



Jittering **99.9th percentile is being tracked.** **ntiles.**

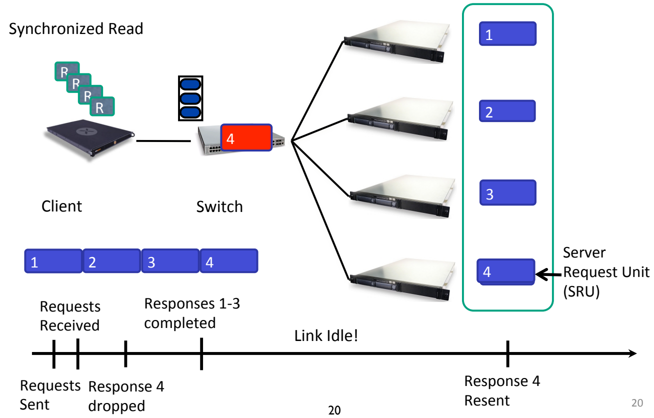
18

The Incast Workload



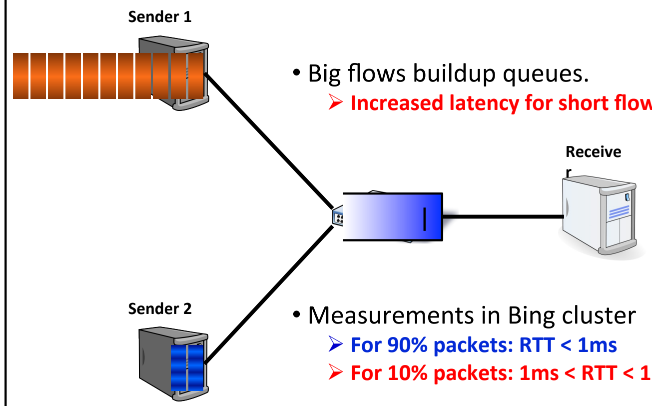
19

Incast Workload Overfills Buffers



20

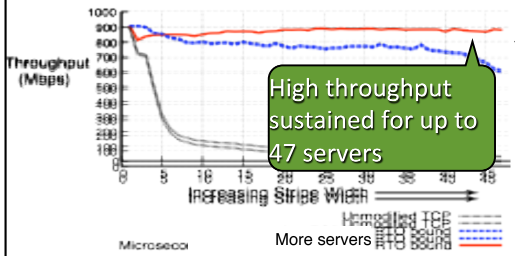
Queue Buildup



- Big flows buildup queues.
  - **Increased latency for short flows.**
- Measurements in Bing cluster
  - **For 90% packets: RTT < 1ms**
  - **For 10% packets: 1ms < RTT < 15ms**

21

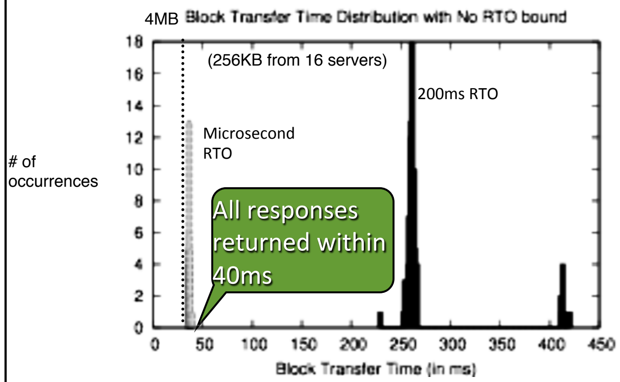
Microsecond timeouts are necessary



22

22

Improvement to Latency



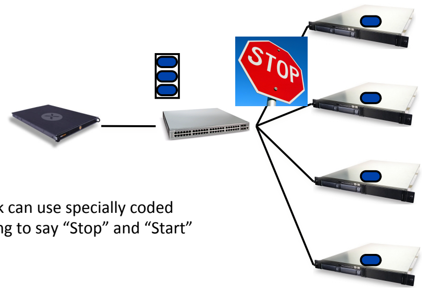
23

23

Link-Layer Flow Control

Common between switches but this is flow-control to the end host too…

- Another idea to reduce incast is to employ Link-Layer Flow Control…..

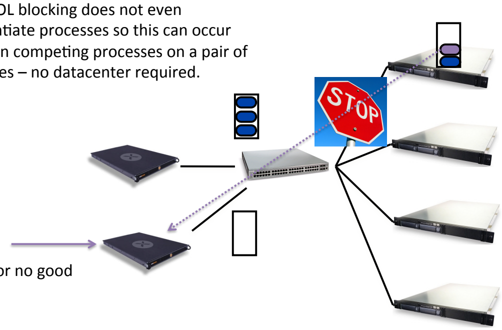


Recall: the Data-Link can use specially coded symbols in the coding to say "Stop" and "Start"

24

Link Layer Flow Control – The Dark side Head of Line Blocking….

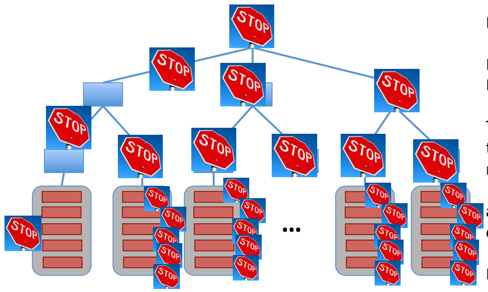
Such HOL blocking does not even differentiate processes so this can occur between competing processes on a pair of machines – no datacenter required.



Waiting for no good reason….

25

Link Layer Flow Control
But its worse that you imagine....



Double down on trouble....

Did I mention this is Link-Layer!

That means no (IP) control traffic, no routing messages....

a whole system waiting for one machine

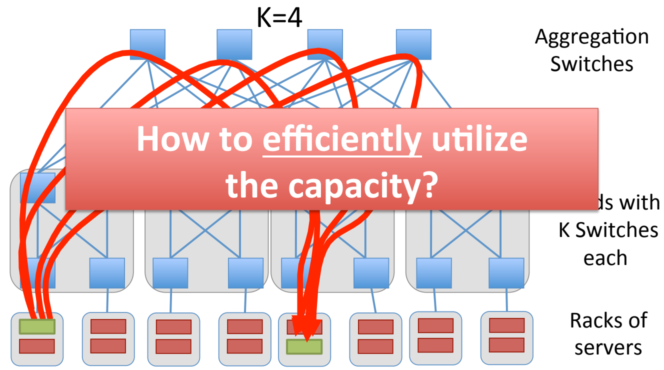
Incast is very unpleasant.

Reducing the impact of HOL in Link Layer Flow Control can be done through priority queues and *overtaking*....

26

Fat Tree Topology

(Fares et al., 2008; Clos, 1953)

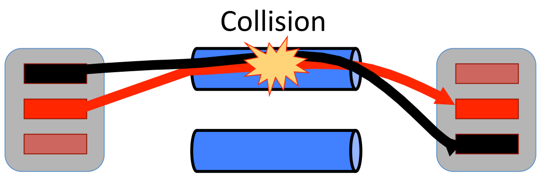


27

State of the Art

(as discussed in Hedera)

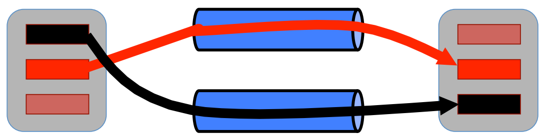
Statically stripe flows across available paths using ECMP



ECMP: Equal Cost Multi-Path Routing is common in Data Centers but network ninjas may be required to configure it correctly...

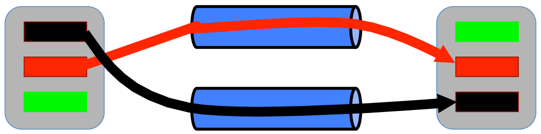
28

How about mapping each flow to a different path?

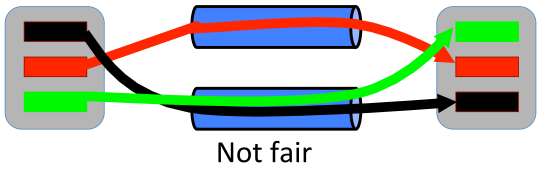


29

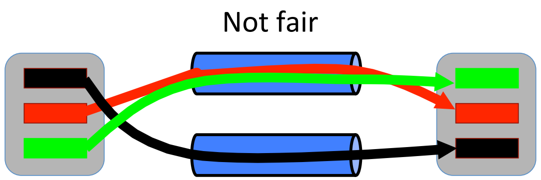
How about mapping each flow to a different path?



How about mapping each flow to a different path?

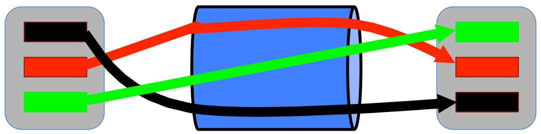


How about mapping each flow to a different path?



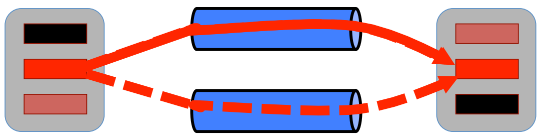
Mapping each flow to a path is the wrong approach!

Instead, <u>pool</u> capacity from links



Multipath TCP Primer
(IETF MPTCP WG)

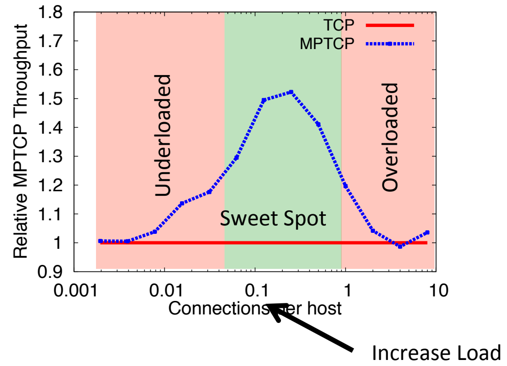
- A drop in replacement for TCP
- Spreads application data over multiple sub flows



- For each ACK on sub-flow r, increase the window w_r by $\min(\alpha/w_{total}, 1/w_r)$
- For each loss on sub-flow r, decrease the window w_r by $w_r/2$

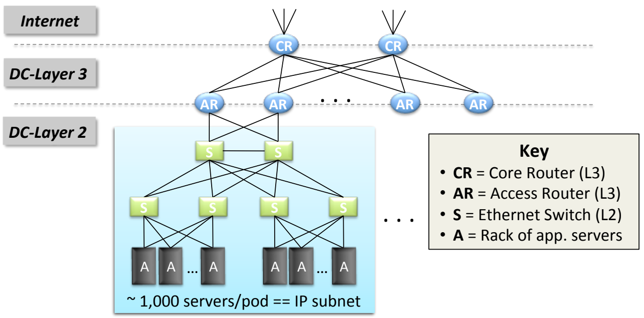
34

Performance improvements depend on traffic matrix



35

DC: lets add some of redundancy
The 3-layer Data Center

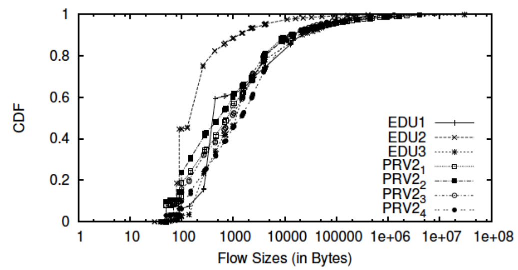


Reference – "Data Center: Load balancing Data Center Services", Cisco 2004

36

Understanding Datacenter Traffic

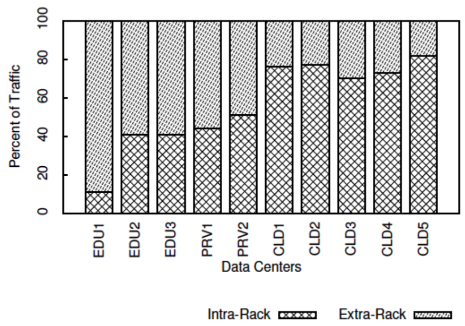
"Most flows in the data centers are **small in size** (<10KB)...". In other words, elephants are a very small fraction.



37

Understanding Datacenter Traffic

Majority of the traffic in cloud datacenters stay within the rack.



38

Today, Computation Constrained by Network*

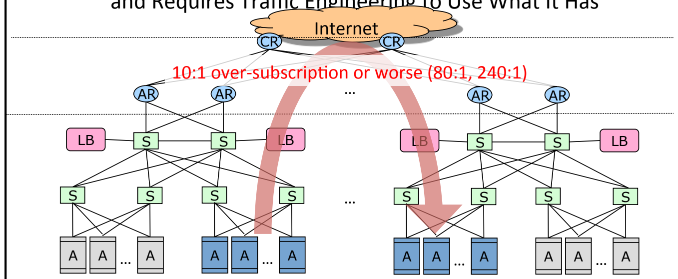
Figure: ln(Bytes/10sec) between servers in operational cluster

- Great efforts required to place communicating servers under the same ToR → Most traffic lies on the diagonal (w/o log scale all you see is the diagonal)
- Stripes show there is need for inter-ToR communication

*Kandula, Sengupta, Greenberg, Patel

39

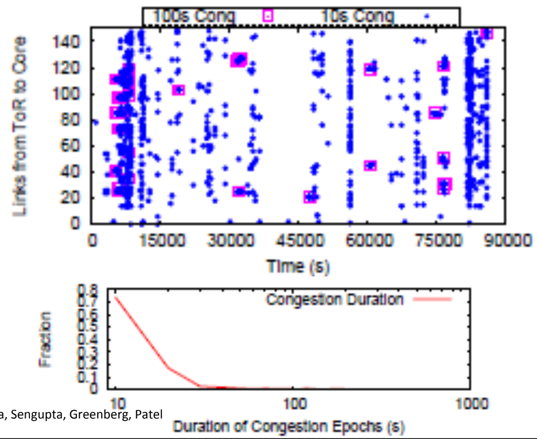
Network has Limited Server-to-Server Capacity, and Requires Traffic Engineering to Use What It Has



- Data centers run two kinds of applications:
  - Outward facing (serving web pages to users)
  - Internal computation (computing search index – think HPC)

40

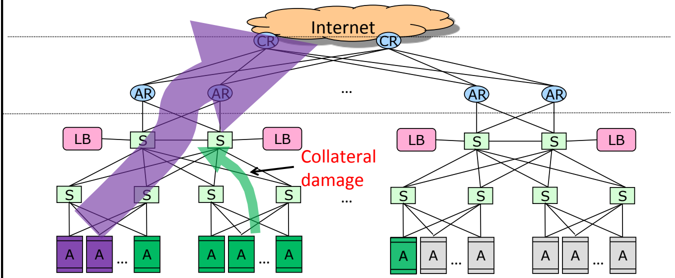
Congestion: Hits Hard When it Hits*



*Kandula, Sengupta, Greenberg, Patel

41

No Performance Isolation

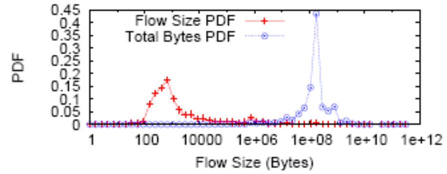


- VLANs typically provide reachability isolation only
- One service sending/receiving too much traffic hurts all services sharing its subtree

42

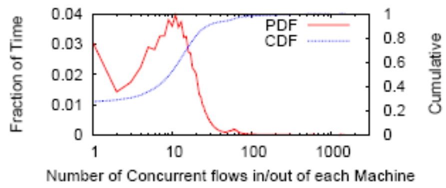
Flow Characteristics

DC traffic != Internet traffic



Most of the flows: various mice

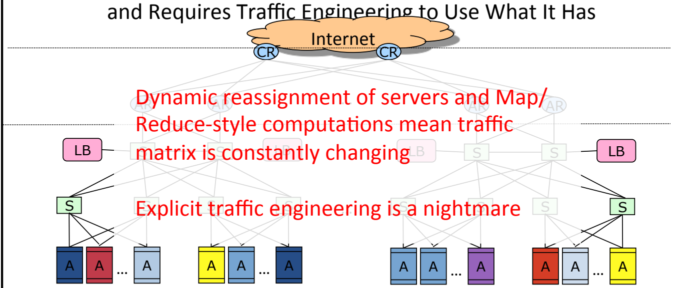
Most of the bytes: within 100MB flows



Median of 10 concurrent flows per server

43

Network Needs Greater Bisection BW, and Requires Traffic Engineering to Use What It Has



Dynamic reassignment of servers and Map/Reduce-style computations mean traffic matrix is constantly changing

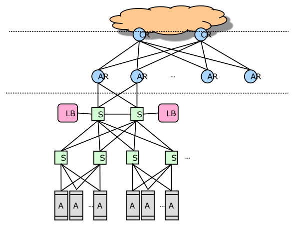
Explicit traffic engineering is a nightmare

- Data centers run two kinds of applications:
  - Outward facing (serving web pages to users)
  - Internal computation (computing search index – think HPC)

44

What Do Data Center Faults Look Like?

- Need very high reliability near top of the tree
  - Very hard to achieve
    - Example: failure of a temporarily unpaired core switch affected ten million users for four hours
  - 0.3% of failure events knocked out all members of a network redundancy group



Ref: Data Center: Load Balancing Data Center Services, Cisco 2004

45

Data Center: Challenges

- From a large cluster used for data mining and identified distinctive traffic patterns
- Traffic patterns are **highly volatile**
  – A large number of distinctive patterns even in a day
- Traffic patterns are **unpredictable**
  – Correlation between patterns very weak

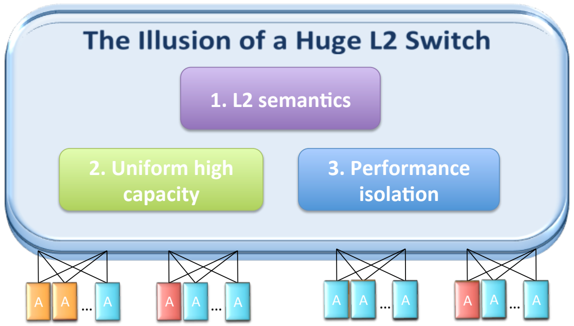
46

Data Center: Opportunities

- DC controller knows **everything** about **hosts**
- Host OS's are easily **customizable**
- **Probabilistic** flow distribution would work well enough, because …
  – Flows are numerous and not huge – no elephants!
  – Commodity switch-to-switch links are substantially thicker (~ 10x) than the maximum thickness of a flow

47

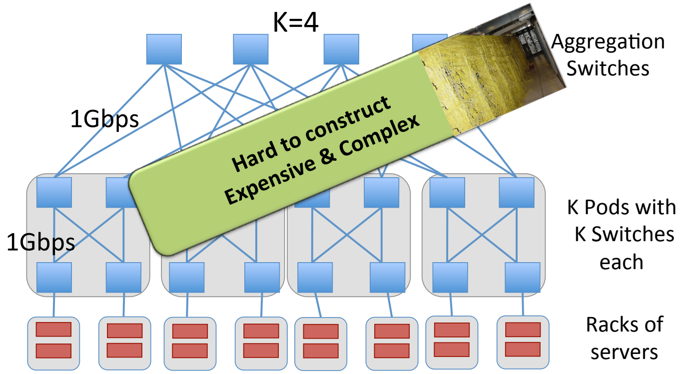
Sometimes we just wish we had a huge L2 switch (L2: data-link)



48

Fat Tree Topology

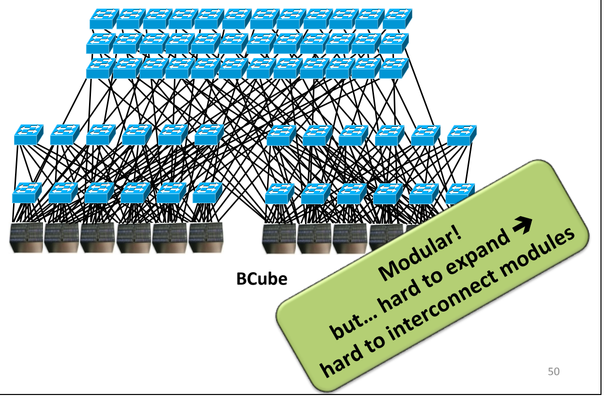
(Fares et al., 2008; Clos, 1953)



49

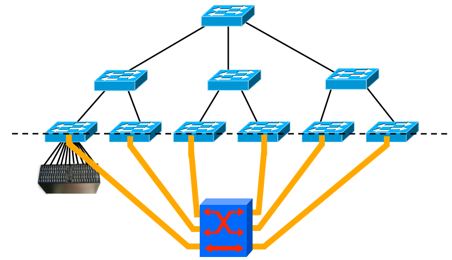
B-Tree Topology

(Guo et al., 2009)



50

An alternative: hybrid packet/circuit switched data center network



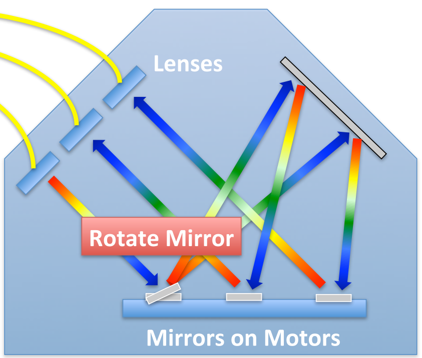
- **Goal of this work:**
  - Feasibility: software design that enables efficient use of optical circuits
  - Applicability: application performance over a hybrid network

51

Optical Circuit Switch

Output
Output
Input

Glass Fiber Bundle



- **Does not decode packets**
- **Needs take time to reconfigure**

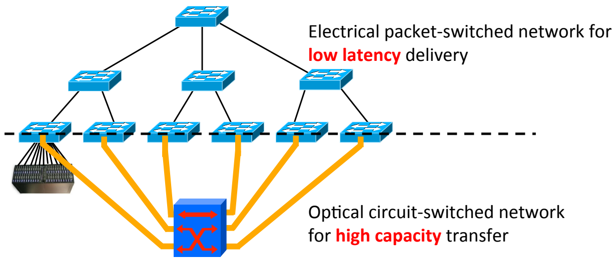
52

Optical circuit switching v.s. Electrical packet switching

	Electrical packet switching	Optical circuit switching
Switching technology	Store and forward	Circuit switching
Switching capacity	16x40Gbps at high end e.g. Cisco CRS-1	320x100Gbps on market, e.g. Calient FiberConnect
Switching time	Packet granularity	Less than 10ms
Switching traffic	For bursty, uniform traffic	For stable, pair-wise traffic

53
53

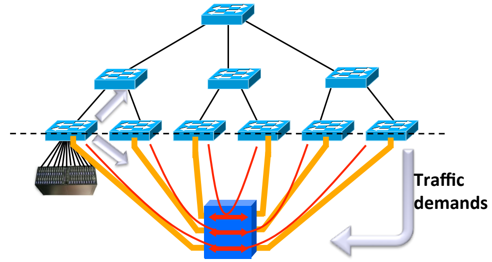
Hybrid packet/circuit switched network architecture



- Optical paths are provisioned rack-to-rack
  - A simple and cost-effective choice
  - Aggregate traffic on per-rack basis to better utilize optical circuits

54

Design requirements



- Control plane:
  - Traffic demand estimation
  - Optical circuit configuration
- Data plane:
  - Dynamic traffic de-multiplexing
  - Optimizing circuit utilization (optional)

55

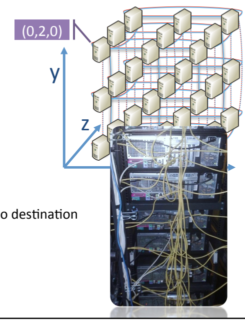
An alternative to Optical links: Link Racks by radio

(60GHz gives about 2Gbps at 5m)

56

CamCube

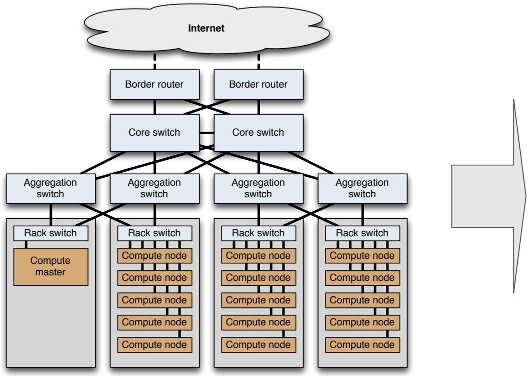
- Nodes are commodity x86 servers with local storage
  - Container-based model 1,500-2,500 servers
- Direct-connect 3D torus topology
  - Six Ethernet ports / server
  - Servers have (x,y,z) coordinates
    - Defines coordinate space
  - Simple 1-hop API
    - Send/receive packets to/from 1-hop neighbours
    - Not using TCP/IP
- Everything is a service
  - Run on all servers
- Multi-hop routing is a service
  - Simple link state protocol
  - Route packets along shortest paths from source to destination



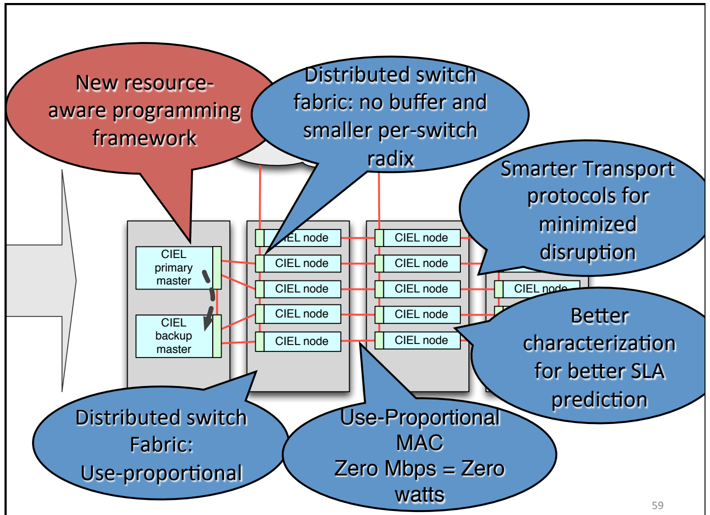
Two Downsides (there are others):
- complex to use (programming models…)
- messy to wire…

57

MRC²: A ***new*** data center approach



58



59

Other problems

Special class problems/Solutions?

Datacenters are computers too…

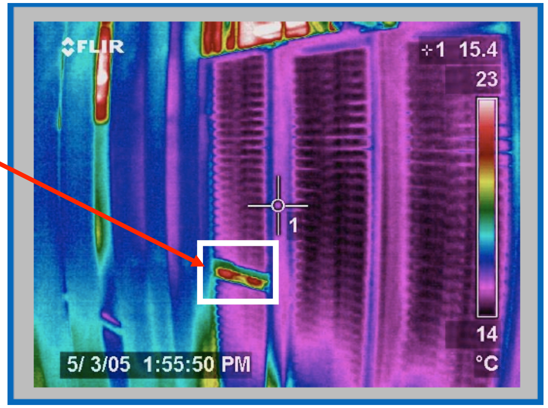
What do datacenters to anyway?

- Special class problems
- Special class data-structures
- Special class languages
- Special class hardware
- Special class operating systems
- Special class networks ✔
- Special class day-to-day operations

60

Google Oregon Warehouse Scale

Thermal Image of Typical Data Centre Rack



M. K. Patterson, A. Pratt, P. Kumar,
"From UPS to Silicon: an end-to-end evaluation of datacenter efficiency", Intel Corporation

62

DC futures

Warehouse-Scale Computing: Entering the Teenage Decade

Luiz Barroso (Google)

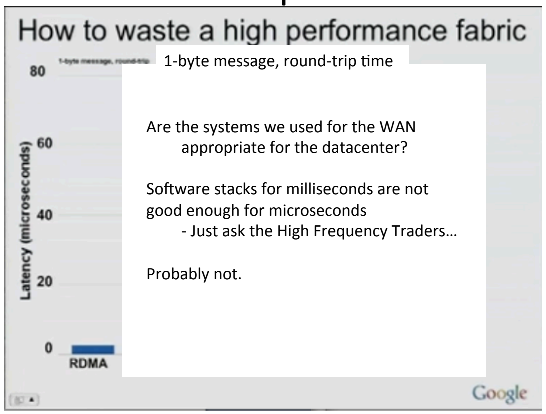
ISCA Keynote 2011

http://dl.acm.org/citation.cfm?id=2019527

It's a video. Watch it.

63

Sample



Are the systems we used for the WAN appropriate for the datacenter?

Software stacks for milliseconds are not good enough for microseconds
- Just ask the High Frequency Traders…

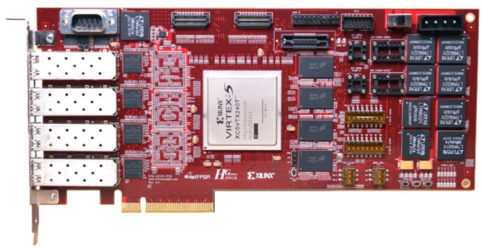
Probably not.

L. Barroso's excellent ISCA2011 keynote – go watch it! http://dl.acm.org/citation.cfm?id=2019527

64

Fancy doing somet exciting computer science? at 10Gbps? 40Gbps? 240Gbps?
Talk to me.

NetFPGA



65