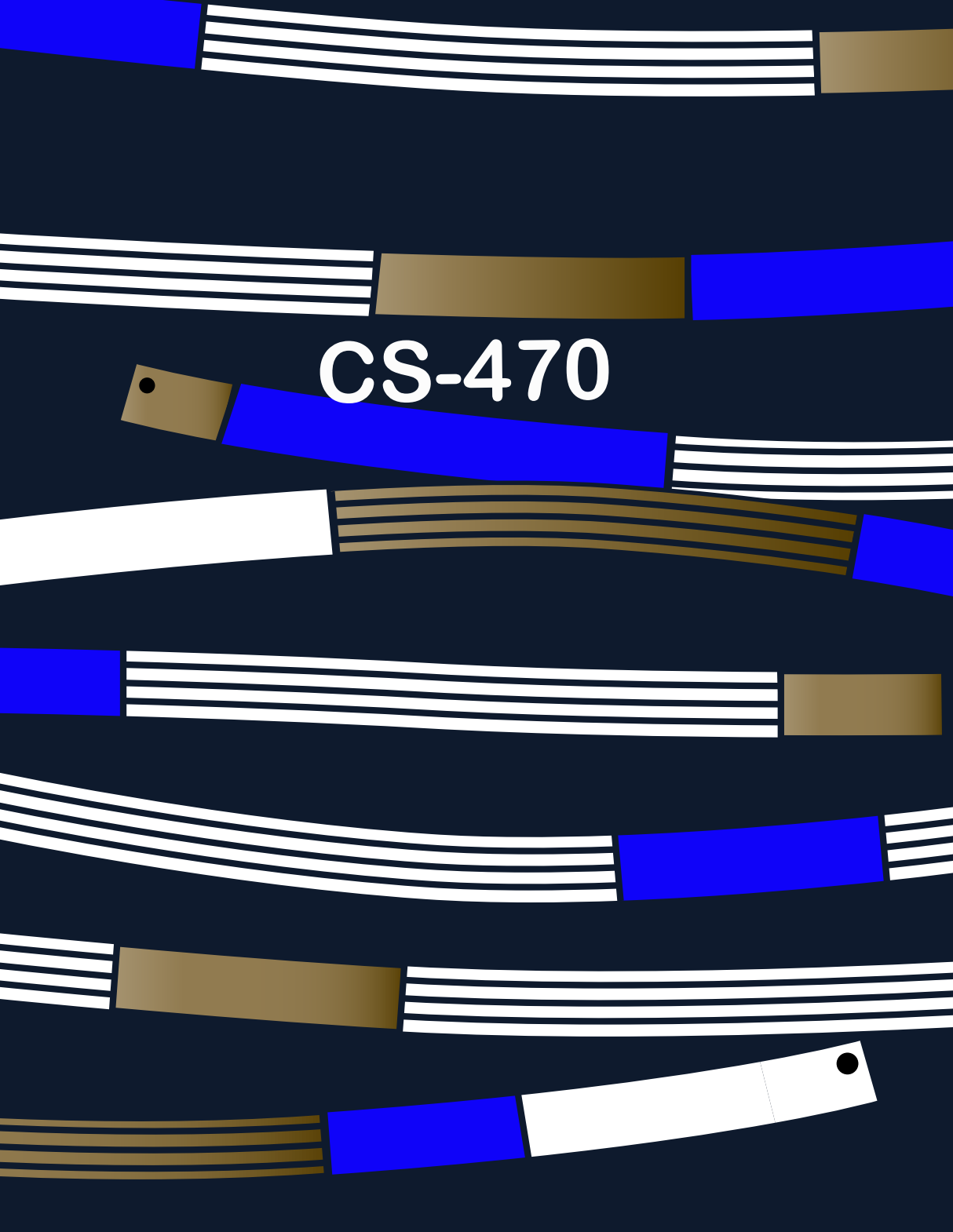

CS-470

Chapter 1: introduction

our goal:

- get "feel" and terminology
- more depth, detail *later* in course
- approach:
  - use Internet as example

overview:

- what's the Internet?
- what's a protocol?
- network edge; hosts, access net, physical media
- network core: packet/circuit switching, Internet structure
- performance: loss, delay, throughput
- security
- protocol layers, service models
- history

1

Chapter 1: roadmap

1.1 what *is* the Internet?
1.2 network edge
- end systems, access networks, links

1.3 network core
- packet switching, circuit switching, network structure

1.4 delay, loss, throughput in networks
1.5 protocol layers, service models
1.6 networks under attack: security
1.7 history

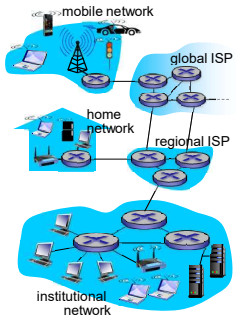
2

What's the Internet: "nuts and bolts" view

- billions of connected computing devices:
  - *hosts* = *end systems*
  - running *network apps*

- *communication links*
  - fiber, copper, radio, satellite
  - transmission rate: *bandwidth*

- *packet switches:* forward packets (chunks of data)
  - *routers* and *switches*



3

"Fun" Internet-connected devices


IP picture frame
http://www.ceiva.com/


Web-enabled toaster + weather forecaster


Tweet-a-watt: monitor energy use


Internet refrigerator


Slingbox: watch, control cable TV remotely

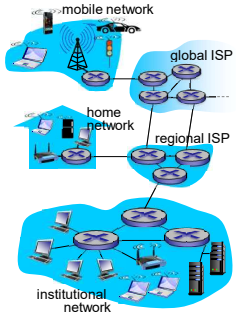

sensorized, bed mattress


Internet phones

4

What's the Internet: "nuts and bolts" view

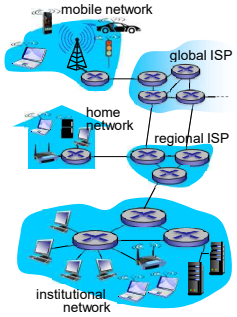
- *Internet:* "network of networks"
  - Interconnected ISPs
- *protocols* control sending, receiving of messages
  - e.g., TCP, IP, HTTP, Skype, 802.11
- *Internet standards*
  - RFC: Request for comments
  - IETF: Internet Engineering Task Force



5

What's the Internet: a service view

- *infrastructure that provides services to applications:*
  - Web, VoIP, email, games, e-commerce, social nets, …
- *provides programming interface to apps*
  - hooks that allow sending and receiving app programs to "connect" to Internet
  - provides service options, analogous to postal service



6

What's a protocol?

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

… specific messages sent
… specific actions taken when messages received, or other events

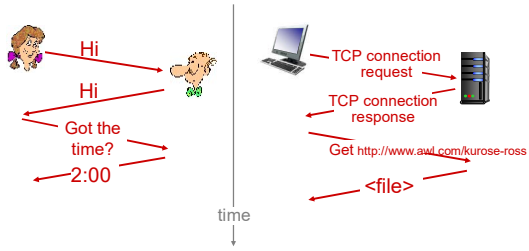
network protocols:
- machines rather than humans
- all communication activity in Internet governed by protocols

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

7

What's a protocol?

a human protocol and a computer network protocol:



Q: other human protocols?

8

Chapter 1: roadmap

1.1 what *is* the Internet?
1.2 network edge
- end systems, access networks, links

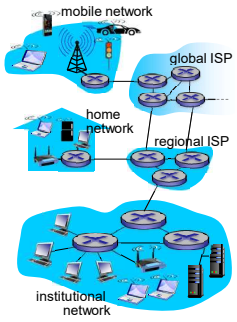
1.3 network core
- packet switching, circuit switching, network structure

1.4 delay, loss, throughput in networks
1.5 protocol layers, service models
1.6 networks under attack: security
1.7 history

9

A closer look at network structure:

- *network edge:*
  - hosts: clients and servers
  - servers often in data centers
- *access networks, physical media:* wired, wireless communication links
- *network core:*
  - interconnected routers
  - network of networks

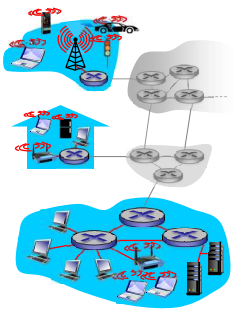


10

Access networks and physical media

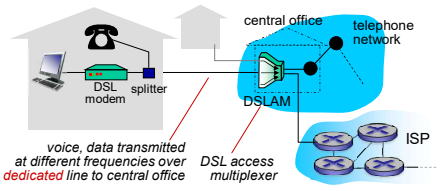
Q: How to connect end systems to edge router?
- residential access nets
- institutional access networks (school, company)
- mobile access networks

keep in mind:
- bandwidth (bits per second) of access network?
- shared or dedicated?



11

Access network: digital subscriber line (DSL)

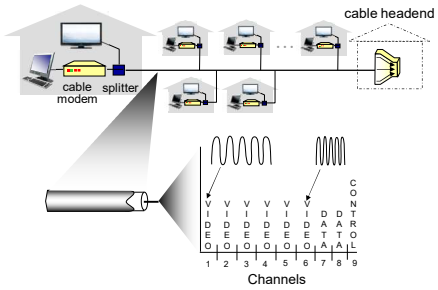


- use *existing* telephone line to central office DSLAM
  - data over DSL phone line goes to Internet
  - voice over DSL phone line goes to telephone net
- < 2.5 Mbps upstream transmission rate (typically < 1 Mbps)
- < 24 Mbps downstream transmission rate (typically < 10 Mbps)

http://www.ebay.com/gds/What-is-the-Difference-Between-a-DSL-and-a-Cable-Modem-/10000000177629327/g.html

12

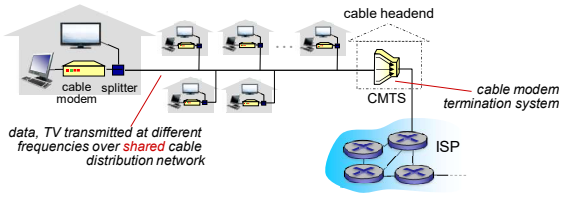
Access network: cable network



frequency division multiplexing: different channels transmitted in different frequency bands

13

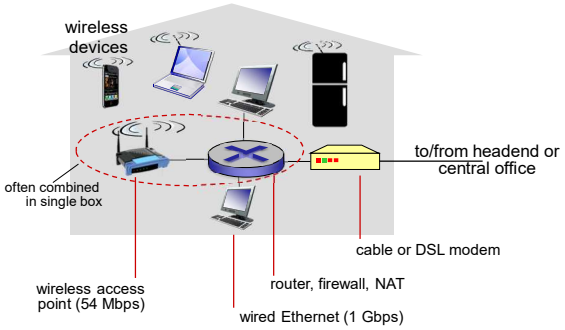
Access network: cable network



- HFC: hybrid fiber coax
  - asymmetric: up to 30Mbps downstream transmission rate, 2 Mbps upstream transmission rate
- network of cable, fiber attaches homes to ISP router
  - homes *share access network* to cable headend
  - unlike DSL, which has dedicated access to central office

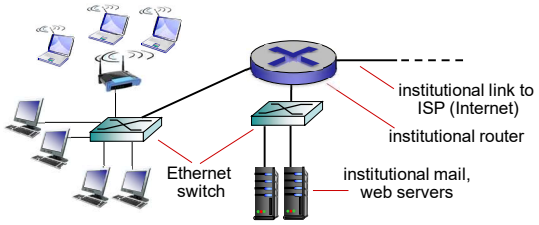
14

Access network: home network



15

Enterprise access networks (Ethernet)



- typically used in companies, universities, etc.
- 10 Mbps, 100Mbps, 1Gbps, 10Gbps transmission rates
- today, end systems typically connect into Ethernet switch

16

Wireless access networks

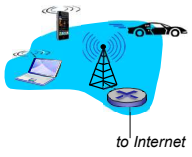
- shared *wireless* access network connects end system to router
  - via base station aka "access point"

wireless LANs:

- within building (100 ft.)
- 802.11b/g/n (WiFi): 11, 54, 450 Mbps transmission rate

wide-area wireless access

- provided by telco (cellular) operator, 10's km
- between 1 and 10 Mbps
- 3G, 4G: LTE

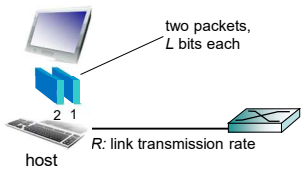


17

Host: sends *packets* of data

host sending function:

- takes application message
- breaks into smaller chunks, known as *packets*, of length *L* bits
- transmits packet into access network at *transmission rate R*
  - link transmission rate, aka link *capacity, aka link bandwidth*



$$\text{packet transmission delay} = \text{time needed to transmit } L\text{-bit packet into link} = \frac{L \text{ (bits)}}{R \text{ (bits/sec)}}$$

18

Physical media

- bit: propagates between transmitter/receiver pairs
- physical link: what lies between transmitter & receiver
- guided media:
  - signals propagate in solid media: copper, fiber, coax
- unguided media:
  - signals propagate freely, e.g., radio

twisted pair (TP)

- two insulated copper wires
  - Category 5: 100 Mbps, 1 Gbps Ethernet
  - Category 6: 10Gbps

19

Physical media: coax, fiber

coaxial cable:

- two concentric copper conductors
- bidirectional
- broadband:
  - multiple channels on cable
  - HFC

fiber optic cable:

- glass fiber carrying light pulses, each pulse a bit
- high-speed operation:
  - high-speed point-to-point transmission (e.g., 10' s-100' s Gbps transmission rate)
- low error rate:
  - repeaters spaced far apart
  - immune to electromagnetic noise

20

Physical media: radio

- signal carried in electromagnetic spectrum
- no physical "wire"
- bidirectional
- propagation environment effects:
  - reflection
  - obstruction by objects
  - interference

radio link types:

- terrestrial microwave
  - e.g. up to 45 Mbps channels
- LAN (e.g., WiFi)
  - 54 Mbps
- wide-area (e.g., cellular)
  - 4G cellular: ~ 10 Mbps
- satellite
  - Kbps to 45Mbps channel (or multiple smaller channels)
  - 270 msec end-end delay
  - geosynchronous versus low altitude

21

Chapter 1: roadmap

1.1 what *is* the Internet?
1.2 network edge
- end systems, access networks, links

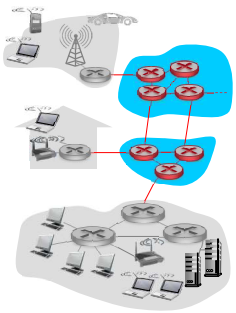
1.3 network core
- packet switching, circuit switching, network structure

1.4 delay, loss, throughput in networks
1.5 protocol layers, service models
1.6 networks under attack: security
1.7 history

22 Router is layer 3 device: network layer

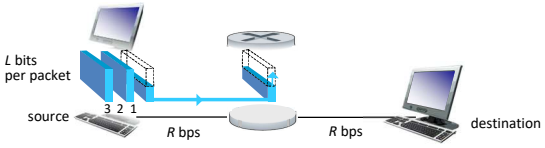
The network core

- mesh of interconnected routers
- packet-switching: hosts break application-layer messages into *packets*
  - forward packets from one router to the next, across links on path from source to destination
  - each packet transmitted at full link capacity



23

Packet-switching: store-and-forward



- takes *L*/*R* seconds to transmit (push out) *L*-bit packet into link at *R* bps
- *store and forward:* entire packet must arrive at router before it can be transmitted on next link
- end-end delay = 2*L*/*R* (assuming zero propagation delay)

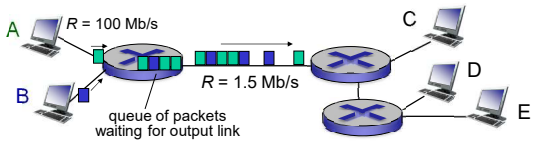
one-hop numerical example:

- *L* = 7.5 Mbits
- *R* = 1.5 Mbps
- one-hop transmission delay = 5 sec

} more on delay shortly …

24

Packet Switching: queueing delay, loss



queuing and loss:

- if arrival rate (in bits) to link exceeds transmission rate of link for a period of time:
  - packets will queue, wait to be transmitted on link
  - packets can be dropped (lost) if memory (buffer) fills up

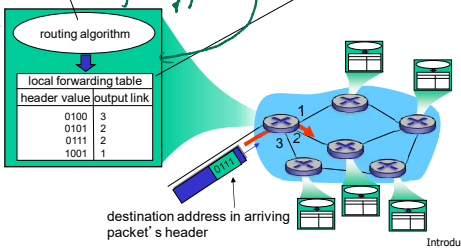
25

Routing forwarding

Two key network-core functions

routing: determines source-destination route taken by packets
- *routing algorithms*

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



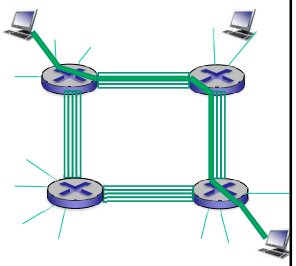
26

routing algorithms does not do routing - only produce table.
Forwarding is not intelligent operation

Alternative core: circuit switching

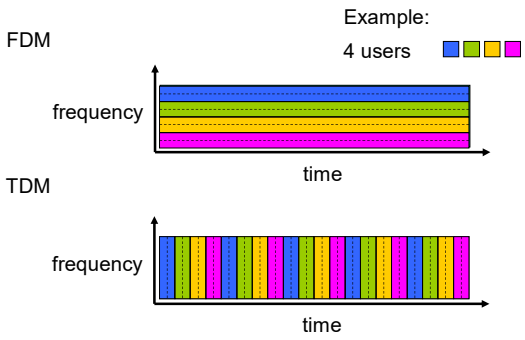
end-end resources allocated to, reserved for "call" between source & dest:

- in diagram, each link has four circuits.
  - call gets 2nd circuit in top link and 1st circuit in right link.
- dedicated resources: no sharing
  - circuit-like (guaranteed) performance
- circuit segment idle if not used by call *(no sharing)*
- commonly used in traditional telephone networks



27

Circuit switching: FDM versus TDM

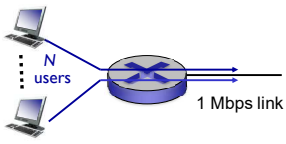


28

Packet switching versus circuit switching

packet switching allows more users to use network!

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



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

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

29

Packet switching versus circuit switching

is packet switching a "slam dunk winner?"

- great for bursty data
  - resource sharing
  - simpler, no call setup
- excessive congestion possible: packet delay and loss
  - protocols needed for reliable data transfer, congestion control
- *Q:* How to provide circuit-like behavior?
  - bandwidth guarantees needed for audio/video apps
  - still an unsolved problem (chapter 7)

Q: human analogies of reserved resources (circuit switching) versus on-demand allocation (packet-switching)?

30

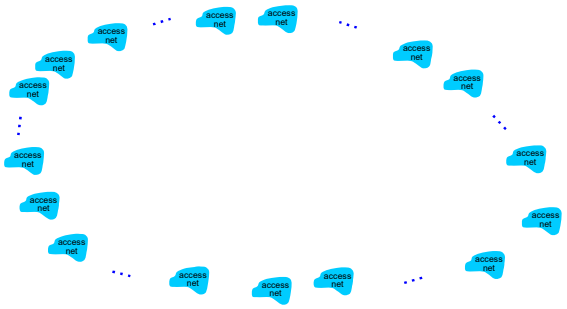
Internet structure: network of networks

- End systems connect to Internet via access ISPs (Internet Service Providers)
  - residential, company and university ISPs
- Access ISPs in turn must be interconnected.
  - so that any two hosts can send packets to each other
- Resulting network of networks is very complex
  - evolution was driven by economics and national policies
- Let's take a stepwise approach to describe current Internet structure

31

Internet structure: network of networks

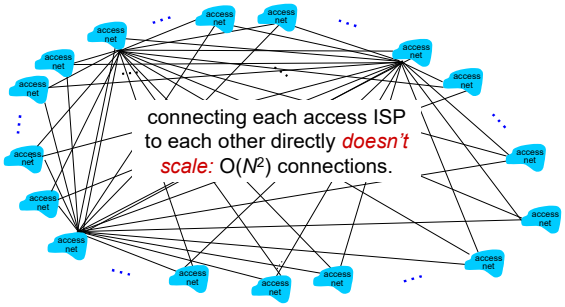
Question: given *millions* of access ISPs, how to connect them together?



32

Internet structure: network of networks

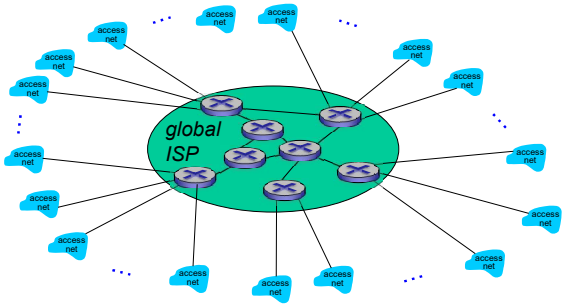
Option: connect each access ISP to every other access ISP?



33

Internet structure: network of networks

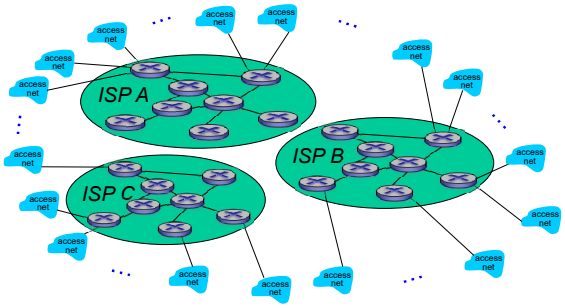
Option: connect each access ISP to one global transit ISP?
Customer and provider ISPs have economic agreement.



34

Internet structure: network of networks

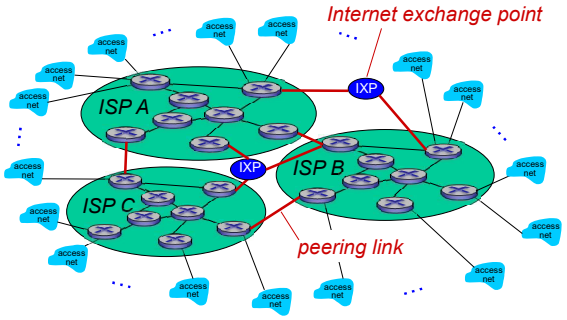
But if one global ISP is viable business, there will be competitors ….



35

Internet structure: network of networks

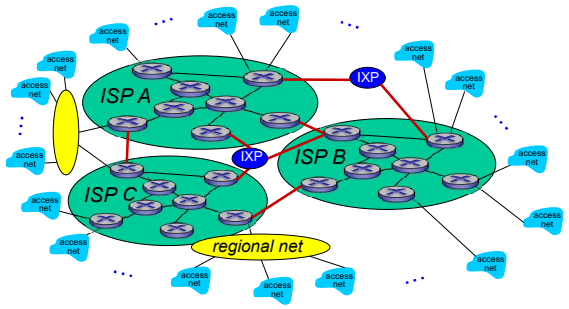
But if one global ISP is viable business, there will be competitors …. which must be interconnected



36

Internet structure: network of networks

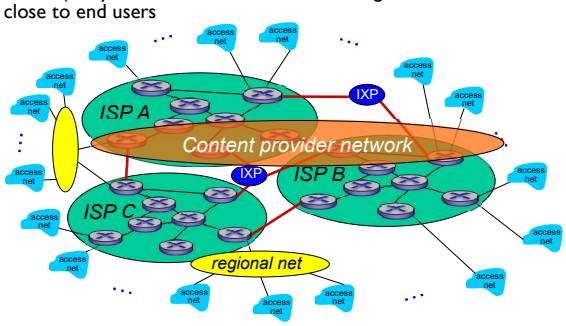
… and regional networks may arise to connect access nets to ISPs



37

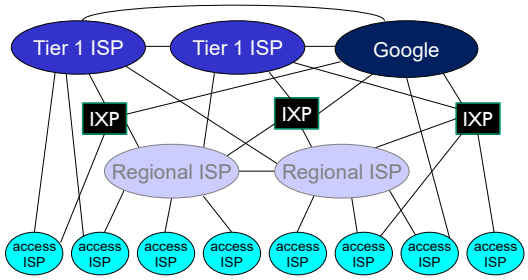
Internet structure: network of networks

… and content provider networks (e.g., Google, Microsoft, Akamai) may run their own network, to bring services, content close to end users



38

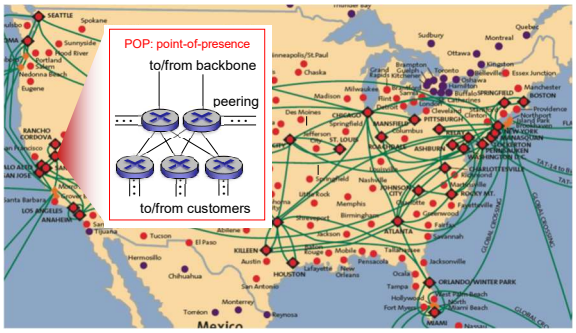
Internet structure: network of networks



- at center: small # of well-connected large networks
  - "tier-1" commercial ISPs (e.g., Level 3, Sprint, AT&T, NTT), national & international coverage
  - content provider network (e.g., Google): private network that connects it data centers to Internet, often bypassing tier-1, regional ISPs

39

Tier-1 ISP: e.g., Sprint



40

Chapter 1: roadmap

1.1 what *is* the Internet?
1.2 network edge
- end systems, access networks, links

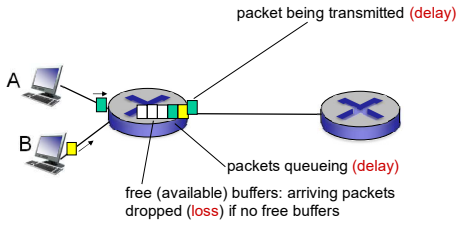
1.3 network core
- packet switching, circuit switching, network structure

1.4 delay, loss, throughput in networks
1.5 protocol layers, service models
1.6 networks under attack: security
1.7 history

41

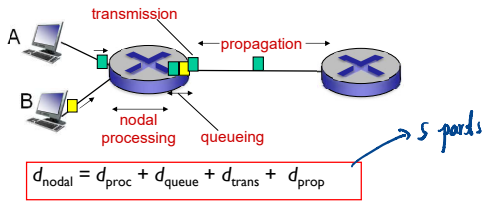
How do loss and delay occur?

packets *queue* in router buffers
- packet arrival rate to link (temporarily) exceeds output link capacity
- packets queue, wait for turn



42

Four sources of packet delay



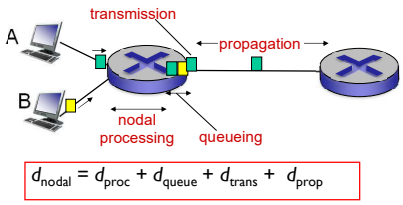
$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$

d_{proc}: nodal processing
- check bit errors
- determine output link
- typically < msec

d_{queue}: queueing delay
- time waiting at output link for transmission
- depends on congestion level of router

43

Four sources of packet delay



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

d_{trans}: transmission delay:
- L: packet length (bits)
- R: link *bandwidth (bps)*
- $d_{trans} = L/R$

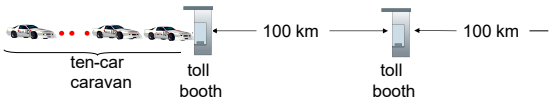
d_{trans} and d_{prop} *very* different

d_{prop}: propagation delay:
- d: length of physical link
- s: propagation speed (~2×10^8 m/sec)
- $d_{prop} = d/s$

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/
* Check out the Java applet for an interactive animation on trans vs. prop delay

44

Caravan analogy

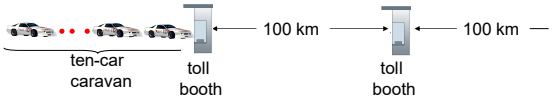


- cars "propagate" at 100 km/hr
- toll booth takes 12 sec to service car (bit transmission time)
- car ~ bit; caravan ~ packet
- *Q:* How long until caravan is lined up before 2nd toll booth?

- time to "push" entire caravan through toll booth onto highway = 12*10 = 120 sec
- time for last car to propagate from 1st to 2nd toll both: 100km/(100km/hr)= 1 hr
- *A:* 62 minutes

45

Caravan analogy (more)



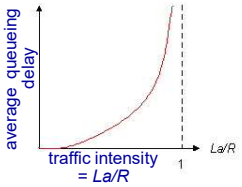
- suppose cars now "propagate" at 1000 km/hr
- and suppose toll booth now takes one min to service a car
- *Q:* Will cars arrive to 2nd booth before all cars serviced at first booth?
  - *A: Yes!* after 7 min, first car arrives at second booth; three cars still at first booth

46

参考后面的notes

Queueing delay (revisited)

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



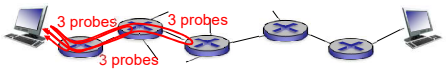
- $La/R \sim 0$: avg. queueing delay small
- $La/R \to 1$: avg. queueing delay large
- $La/R > 1$: more "work" arriving than can be serviced, average delay infinite!

* Check online interactive animation on queuing and loss

47

"Real" Internet delays and routes

- what do "real" Internet delay & loss look like?
- `traceroute` program: provides delay measurement from source to router along end-end Internet path towards destination. For all *i:*
  - sends three packets that will reach router *i* on path towards destination
  - router *i* will return packets to sender
  - sender times interval between transmission and reply.



https://gsuite.tools/traceroute

https://centralops.net/co/

48

"Real" Internet delays, routes

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

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

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

trans-oceanic link

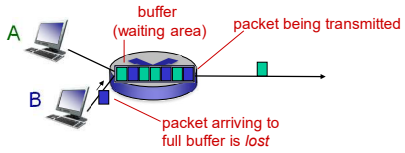
* means no response (probe lost, router not replying)

* Do some traceroutes from exotic countries at www.traceroute.org

49

Packet loss

- queue (aka buffer) preceding link in buffer has finite capacity
- packet arriving to full queue dropped (aka lost)
- lost packet may be retransmitted by previous node, by source end system, or not at all

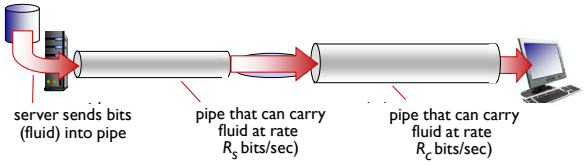


* Check out the Java applet for an interactive animation on queuing and loss

50

Throughput

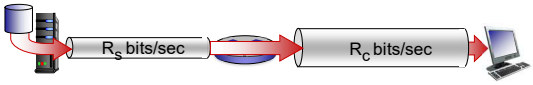
- *throughput:* rate (bits/time unit) at which bits transferred between sender/receiver
  - *instantaneous:* rate at given point in time
  - *average:* rate over longer period of time



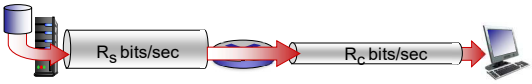
51

Throughput (more)

- $R_s < R_c$ What is average end-end throughput?



- $R_s > R_c$ What is average end-end throughput?



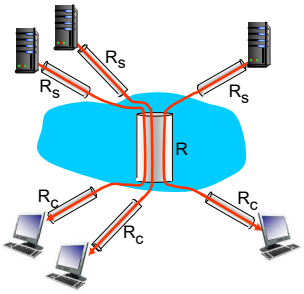
bottleneck link

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

52

Throughput: Internet scenario

- per-connection end-end throughput: $min(R_c, R_s, R/10)$
- in practice: R_c or R_s is often bottleneck



10 connections (fairly) share backbone bottleneck link *R* bits/sec

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

53

Chapter 1: roadmap

1.1 what *is* the Internet?
1.2 network edge
- end systems, access networks, links

1.3 network core
- packet switching, circuit switching, network structure

1.4 delay, loss, throughput in networks
1.5 protocol layers, service models
1.6 networks under attack: security
1.7 history

54

Protocol "layers"

Networks are complex, with many "pieces":

- hosts
- routers
- links of various media
- applications
- protocols
- hardware, software

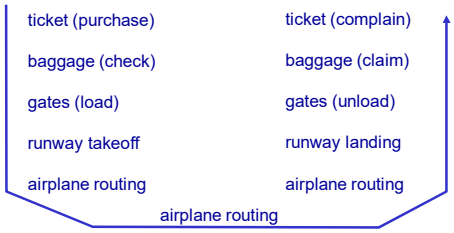
Question:

is there any hope of *organizing* structure of network?

…. or at least our discussion of networks?

55

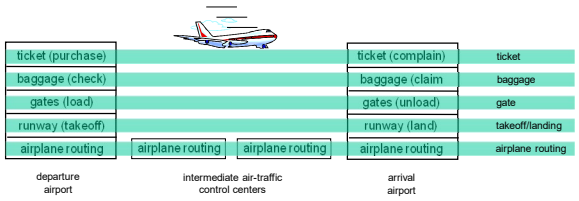
Organization of air travel



- a series of steps

56

Layering of airline functionality



layers: each layer implements a service

- via its own internal-layer actions
- relying on services provided by layer below

57

Why layering?

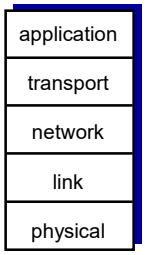
dealing with complex systems:

- explicit structure allows identification, relationship of complex system's pieces
  - layered *reference model* for discussion
- modularization eases maintenance, updating of system
  - change of implementation of layer's service transparent to rest of system
  - e.g., change in gate procedure doesn't affect rest of system
- layering considered harmful?

58

Internet protocol stack

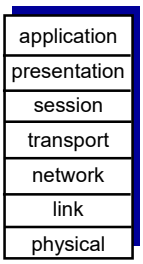
- *application:* supporting network applications
  - FTP, SMTP, HTTP
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.111 (WiFi), PPP
- *physical:* bits "on the wire"



59

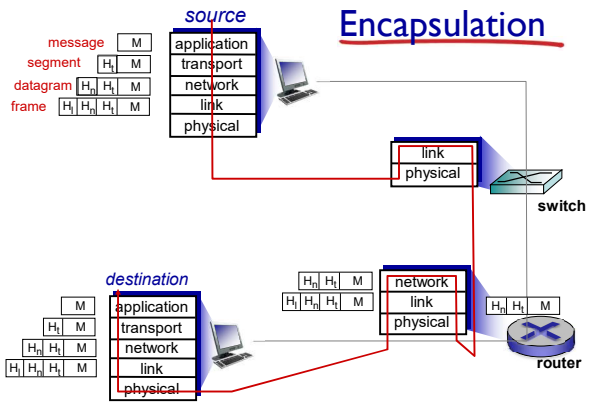
ISO/OSI reference model

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



60

Encapsulation



61

Chapter 1: roadmap

1.1 what *is* the Internet?
1.2 network edge
- end systems, access networks, links

1.3 network core
- packet switching, circuit switching, network structure

1.4 delay, loss, throughput in networks
1.5 protocol layers, service models
1.6 networks under attack: security
1.7 history

62

Network security

- field of network security:
  - how bad guys can attack computer networks
  - how we can defend networks against attacks
  - how to design architectures that are immune to attacks
- Internet not originally designed with (much) security in mind
  - *original vision:* "a group of mutually trusting users attached to a transparent network" ☺
  - Internet protocol designers playing "catch-up"
  - security considerations in all layers!

63

Bad guys: put malware into hosts via Internet

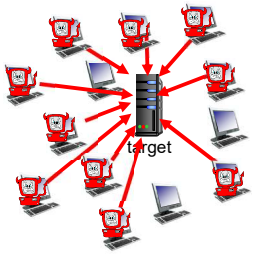
- malware can get in host from:
  - *virus:* self-replicating infection by receiving/executing object (e.g., e-mail attachment)
  - *worm:* self-replicating infection by passively receiving object that gets itself executed
- spyware malware can record keystrokes, web sites visited, upload info to collection site
- infected host can be enrolled in botnet, used for spam. DDoS attacks

64

Bad guys: attack server, network infrastructure

Denial of Service (DoS): attackers make resources (server, bandwidth) unavailable to legitimate traffic by overwhelming resource with bogus traffic

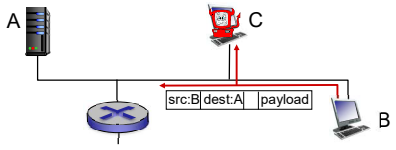
1. select target
2. break into hosts around the network (see botnet)
3. send packets to target from compromised hosts



65

Bad guys can sniff packets

packet "sniffing":
- broadcast media (shared Ethernet, wireless)
- promiscuous network interface reads/records all packets (e.g., including passwords!) passing by

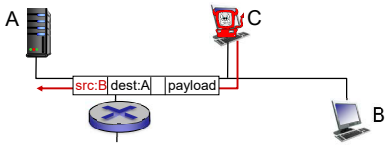


- wireshark software used for end-of-chapter labs is a (free) packet-sniffer

66

Bad guys can use fake addresses

IP spoofing: send packet with false source address



… lots more on security (throughout, Chapter 8)

67

Chapter 1: roadmap

1.1 what *is* the Internet?
1.2 network edge
- end systems, access networks, links

1.3 network core
- packet switching, circuit switching, network structure

1.4 delay, loss, throughput in networks
1.5 protocol layers, service models
1.6 networks under attack: security
1.7 history

68

Internet history

1961-1972: Early packet-switching principles

- 1961: Kleinrock - queueing theory shows effectiveness of packet-switching
- 1964: Baran - packet-switching in military nets
- 1967: ARPAnet conceived by Advanced Research Projects Agency
- 1969: first ARPAnet node operational
- 1972:
  - ARPAnet public demo
  - NCP (Network Control Protocol) first host-host protocol
  - first e-mail program
  - ARPAnet has 15 nodes

69

Internet history

1972-1980: Internetworking, new and proprietary nets

- 1970: ALOHAnet satellite network in Hawaii
- 1974: Cerf and Kahn - architecture for interconnecting networks
- 1976: Ethernet at Xerox PARC
- late70's: proprietary architectures: DECnet, SNA, XNA
- late 70's: switching fixed length packets (ATM precursor)
- 1979: ARPAnet has 200 nodes

Cerf and Kahn's internetworking principles:
- minimalism, autonomy - no internal changes required to interconnect networks
- best effort service model
- stateless routers
- decentralized control

define today's Internet architecture

70

Internet history

1980-1990: new protocols, a proliferation of networks

- 1983: deployment of TCP/IP
- 1982: smtp e-mail protocol defined
- 1983: DNS defined for name-to-IP-address translation
- 1985: ftp protocol defined
- 1988: TCP congestion control
- new national networks: CSnet, BITnet, NSFnet, Minitel
- 100,000 hosts connected to confederation of networks

71

Internet history

1990, 2000's: commercialization, the Web, new apps

- early 1990's: ARPAnet decommissioned
- 1991: NSF lifts restrictions on commercial use of NSFnet (decommissioned, 1995)
- early 1990s: Web
  - hypertext [Bush 1945, Nelson 1960's]
  - HTML, HTTP: Berners-Lee
  - 1994: Mosaic, later Netscape
  - late 1990's: commercialization of the Web

late 1990's – 2000's:
- more killer apps: instant messaging, P2P file sharing
- network security to forefront
- est. 50 million host, 100 million+ users
- backbone links running at Gbps

72

Internet history

2005-present

- ~5B devices attached to Internet (2016)
  - smartphones and tablets
- aggressive deployment of broadband access
- increasing ubiquity of high-speed wireless access
- emergence of online social networks:
  - Facebook: ~ one billion users
- service providers (Google, Microsoft) create their own networks
  - bypass Internet, providing "instantaneous" access to search, video content, email, etc.
- e-commerce, universities, enterprises running their services in "cloud" (e.g., Amazon EC2)

73

Introduction: summary

covered a "ton" of material!

- Internet overview
- what's a protocol?
- network edge, core, access network
  - packet-switching versus circuit-switching
  - Internet structure
- performance: loss, delay, throughput
- layering, service models
- security
- history

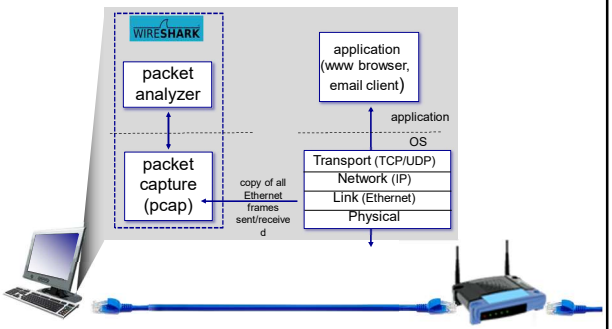
you now have:

- context, overview, "feel" of networking
- more depth, detail *to follow!*

74

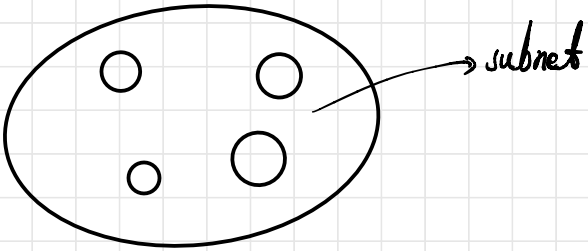
Chapter 1
Additional Slides

75



76

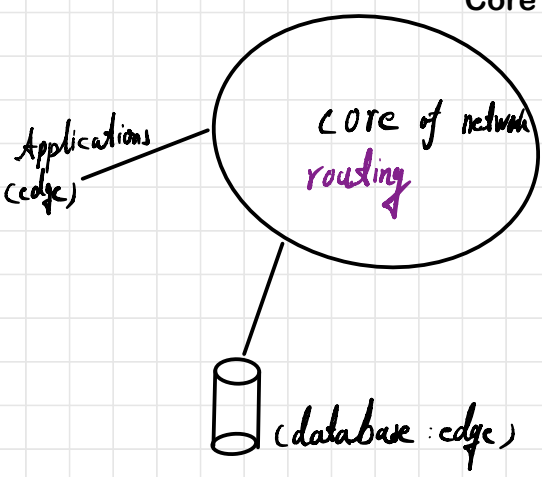
1/11/2022



·Network backbone: IETF(Internet Engineering Task Force)

- Major designer of internet: CISCO

Core is doing one job: routing.



Flag day: The day to reboot network.(never gonna happen).

- **Modem: Mojolater and DEMojolater.**

- **Protocol:**

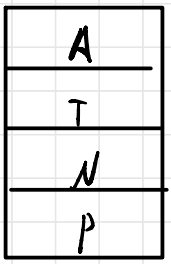
> *protocols define format, order of messages sent and received among network entities, and actions taken on message transmission, receipt*

Packet: chunk of data.

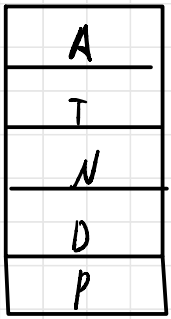
Transmission rate: how fast your pc dump data on wires.

Propagation rate: how fast the single transfer on wire.

1/13/2021



Application
Transport
Network
physical



Application
Transport
Network
Data link
physical

decapsulate P layer, frame, find out next router based on logical IP address.
Then encapsulate the next router H/W address

Host : A

Host B

①

(encapsulation)

Application: P (gives a packet)

Transport: T P (gives a port number)

Network: N T P (gives a logical ip address)

physical: H N T P T (gives a physical ip address) (called frame)

hardware address of next router

②

(decapsulation)

Application: P (We have packet)

Transport: T P (give up a port number)

Network: N T P (give up a logical ip address)

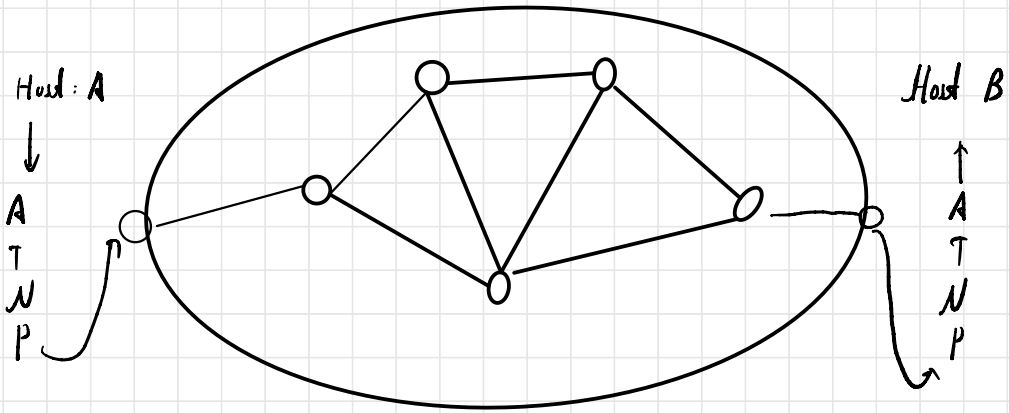
physical: H N T P T (give up a physical ip address)

hardware address

- dynamic IP address (logical address)
- physic address (Mac address)

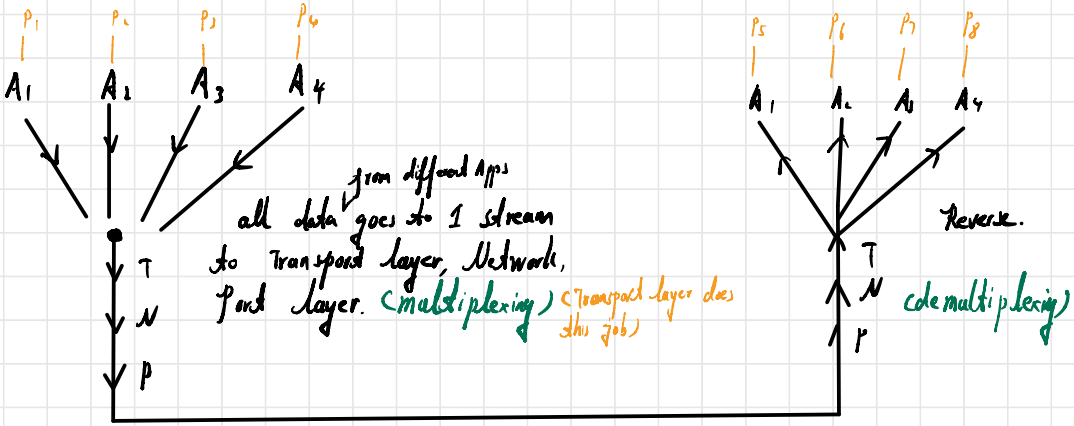
next router = next hop

01/18/2021



It is possible to skip Transport layer, but it turns out pc can only run one application, no multiple programs.

Mud & Demux: multiplexing and demultiplexing.



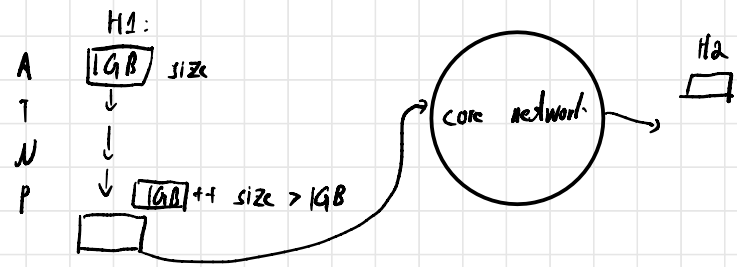
- Q: Does the transport need to know the destination port ahead?
  Yes, it must know it before you can communicate that machine.

- Q: Could A1 and A5 use same port number like P1?
  Yes, since the port number is unique on one computer.
  But you can not let two different app runs on same pc and use same port number, that will be disaster.

- Loop back IP address: 127.0.0.1
  When we send this data to this IP address, it comes back to you for testing purpose.

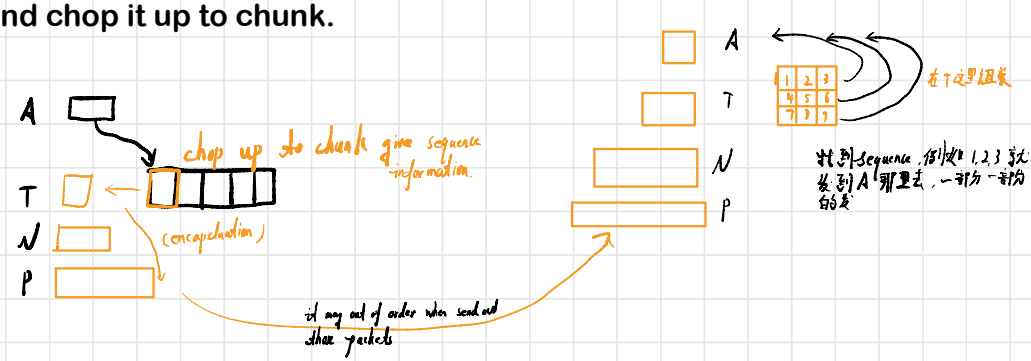
- Three switching methods.

  - Message switching: No limited for data size which sent out to network.

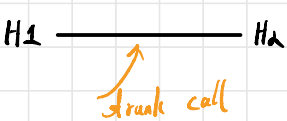


Bad things: ① If the connection in the network disconnect then the whole data will be discarded.
② Fair share of internet will be interrupted.

  - Packet switching: There is limit for data size which sent out to network. And chop it up to chunk.



- Circuit switching: guarantee the resources is from h1 to h2, but low source reservation.



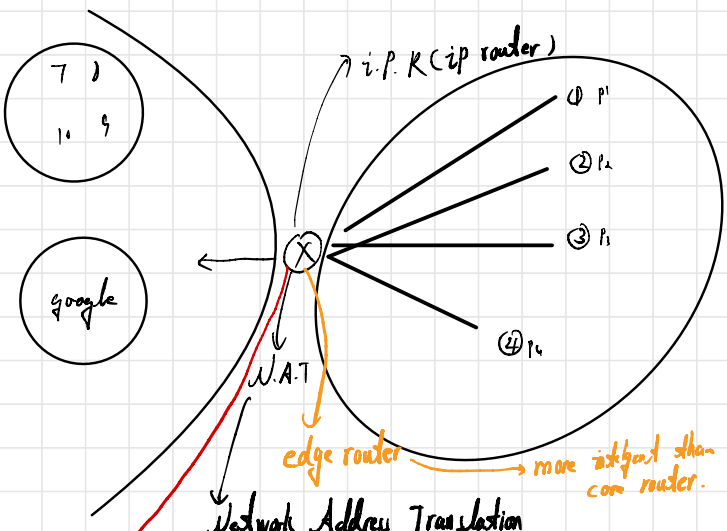
Switch: dump router, never change the connect automatically. (static)

eg: Truman WiFi accessing point, WiFi Extender

Router: intelligent, and dynamic automatically change the links. Make decisions where to forward.

eg: Truman network outgoing router, home WiFi router, NAT

Home network:



private ip: 10/192

public ip: Globally unique.

edge router may not always be NAT

eg: Truman: Gate ✓, NAT ✗
Home: Gate ✓, NAT ✓

change P4's packet's source IP from private to I.P.R (public IP)

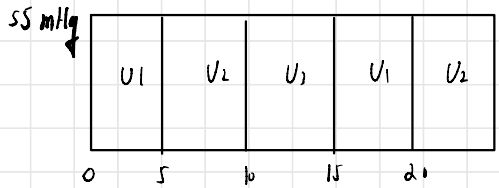
change packet's destination IP from IPR to private IP address

- N.A.T
- Router
- Fire Wall
- WiFi access point
- ip address saviour: save ip address

TDM(Time division multiplexing)

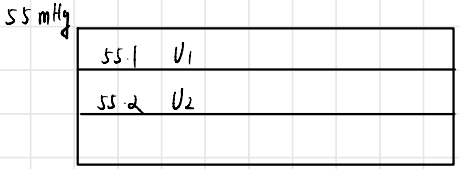
Collision will happen if multiple users use same frequency at same time.

Same Frequency different time

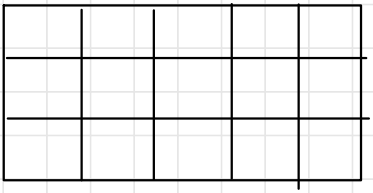


Time line : interval 5 ms

FDM(Frequency devision multiplexing)



If more users then combine fdm and tdm.

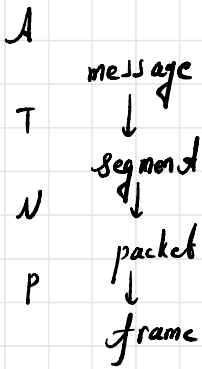


1/20/2022

Some notes are on slides.

Segmentation and reassembly

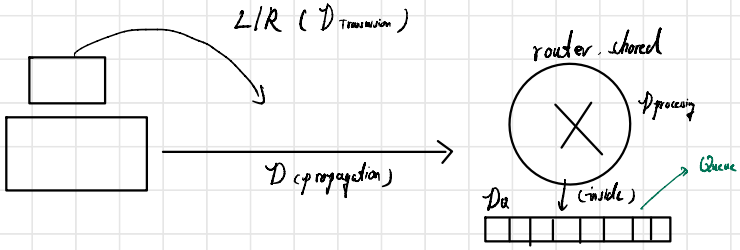
- segmentation: Transport layer segement message into packets
- reassembly: reassemble all packets by using sequence information



- *store and forward:* entire packet must arrive at router before it can be transmitted on next link

The router need to wait whole packet to arrive even they can read the ip information. Why?

Because the packet may have error and router will know until whole packet has arrived. Thats why we need store and forward.



if all queue spot is full coming data is dropped/lost

whole delay: $D_T + D_{prop} + D_q + D_{proc} = D_{nodal}$

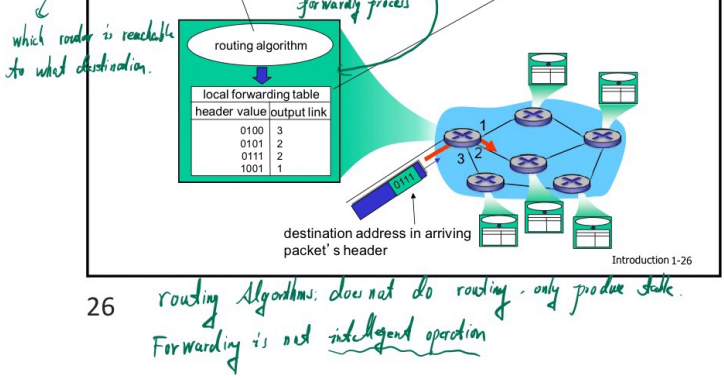
Routering / Forwarding

Routing forwarding

Two key network-core functions

routing: determines source-destination route taken by packets
- routing algorithms

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



Introduction 1-26

26

routing algorithms: does not do routing, only produce table.
Forwarding is not intelligent operation

Internet traffic = bursty traffic

Packet switching is good for public internet, like bursty data, and it could be shared for many users.

Circuit switching: is good for dedicated link communication like government or special comms.

01/25/2022

R: Link bandwidth(processing capacity)

L: Packet length.

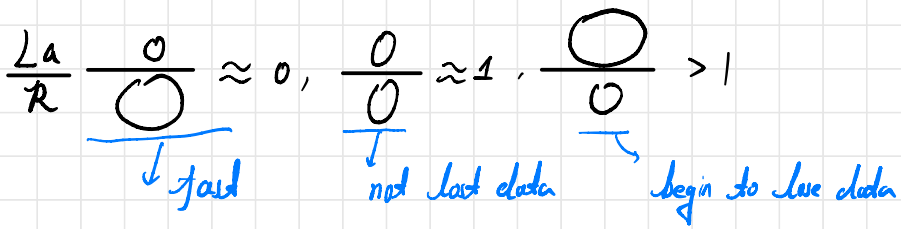
a: packet arrival rate.

Bps = Bytes per second

bps = bits per second

congestion: 拥塞

Traffic intensity = La / R



Conclusion: No matter how to decrease the a or L or increase the R, there is no way to decline the congestion.

- That's why we have congestion control and install it on Transport layer.

Because network layer is used to prepare routing, physical layer doesn't make any decisions. Install on application will require more effort from programmers. Only transport is left.

- Congestion control: Slow down the network speed that won't be noticed by users but enough for router to process datas.

best protocal

TCP is more sensitive, and UDP doesn't care about congestion.

TTL

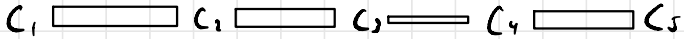
Max-hop-count: max number of intermediate router your packet should visit till destination in order to make sure the packet will not stuck in loop of routers.

If the count is to 0, then router drop the packet.

ICMP: internet control message protocol: used to determine if the packet reached to the destination in timely manner. And not every router has it.

Review traceroute

Throughout:



The effective internet speed is affected by the smallest intermediate link rate.

短板效应 (bottle neck)

ISO/OSI model: 7 layers could be found on slides.

Why do we have layers?

Why layering?

dealing with complex systems:

- explicit structure allows identification, relationship of complex system's pieces
  - layered *reference model* for discussion
- modularization eases maintenance, updating of system
  - change of implementation of layer's service transparent to rest of system
  - e.g., change in gate procedure doesn't affect rest of system
- layering considered harmful?

} separate the network into specific areas

Job of Application layer: syntax semantic meaning of transport message.

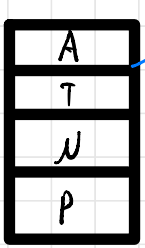
Transport layer: Deliver packet from one application to application.

Network: host to host delivery.

Physical layer: node to node delivery.

- TLS: Transport Layer Security.

Used to be SSL but now is TLS.



TLS is here, encrypt the whole message, not single packet since it's more economical and faster

Chapter 2

Chapter 2: outline

2.1 principles of network applications
2.2 Web and HTTP
2.3 electronic mail
- SMTP, POP3, IMAP

2.4 DNS
2.5 P2P applications
2.6 video streaming and content distribution networks
2.7 socket programming with UDP and TCP

1

Chapter 2: application layer

our goals:
- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
  - content distribution networks
- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- creating network applications
  - socket API

2

Some network apps

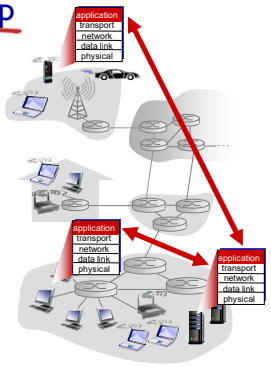
- e-mail
- web
- text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- …
- …

3

Creating a network app

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

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



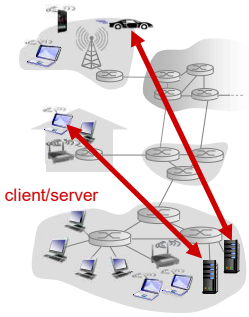
4

Application architectures

possible structure of applications:
- client-server
- peer-to-peer (P2P)

5

Client-server architecture



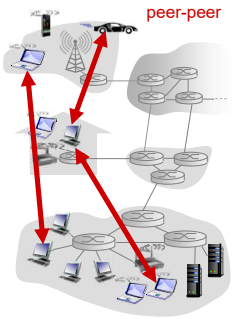
server:
- always-on host
- permanent IP address
- data centers for scaling

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

6

P2P architecture

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



7

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

clients, servers

client process: process that initiates communication

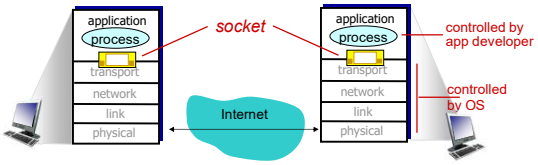
server process: process that waits to be contacted

- aside: applications with P2P architectures have client processes & server processes

8

Sockets

- 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 to deliver message to socket at receiving process



9

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…

10

Data is generated by App layer

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

open protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype

11

就是说需要完整的data，但是速度不快：gmail

需要速度，但是可能掉包，说的就是fps游戏

T offer services like this

What transport service does an app need?

data integrity

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

timing

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

throughput 吞吐量

- 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, …

12

Transport service requirements: common apps

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

13

TCP and UDP protocol:

Internet transport protocols services

TCP service:

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

UDP service:

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

Q: why bother? Why is there a UDP?

14

Internet apps: application, transport protocols

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

15

Securing TCP

TCP & UDP

- no encryption
- cleartext passwds sent into socket traverse Internet in cleartext

SSL

- provides encrypted TCP connection
- data integrity
- end-point authentication

SSL is at app layer

- apps use SSL libraries, that "talk" to TCP

SSL socket API

- cleartext passwords sent into socket traverse Internet encrypted
- see Chapter 8

16

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

2.7 socket programming with UDP and TCP

17

Web and HTTP

First, a review…

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

`www.someschool.edu/someDept/pic.gif`

host name | path name

18

HTTP overview

HTTP: hypertext transfer protocol

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

19

HTTP overview (continued)

uses TCP:

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

HTTP is "stateless"

- server maintains no information about past client requests

aside

protocols that maintain "state" are complex!

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

20

HTTP connections

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

- multiple objects can be sent over single TCP connection between client, server

21

Non-persistent HTTP

suppose user enters URL:
`www.someSchool.edu/someDepartment/home.index`
(contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

22

Non-persistent HTTP (cont.)

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

time

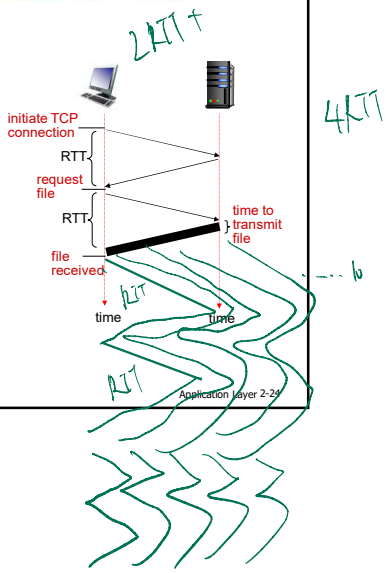
23

Non-persistent HTTP: response time

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

HTTP response time:

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



24

Persistent HTTP

non-persistent HTTP issues:

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

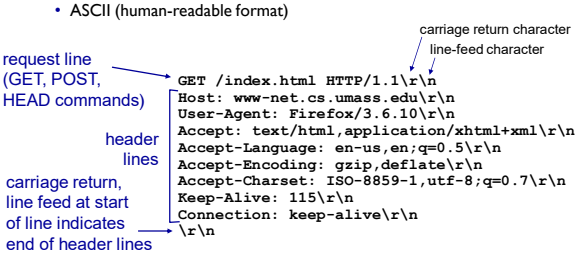
persistent HTTP:

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

25

HTTP request message

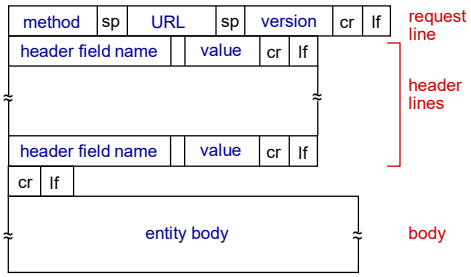
- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

26

HTTP request message: general format



27

Uploading form input

POST method:

- web page often includes form input
- input is uploaded to server in entity body

URL method:

- uses GET method
- input is uploaded in URL field of request line:

```
www.somesite.com/animalsearch?monkeys&banana
```


28

Method types

HTTP/1.0:

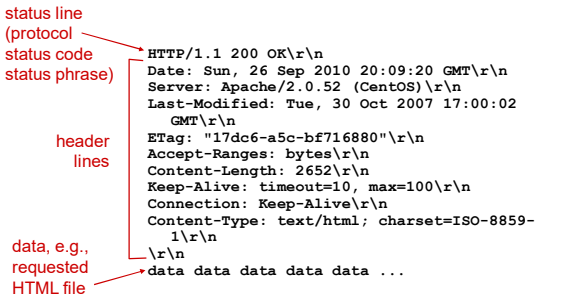
- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

29

HTTP response message



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

30

HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

200 OK
- request succeeded, requested object later in this msg

301 Moved Permanently
- requested object moved, new location specified later in this msg (Location:)

400 Bad Request
- request msg not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported

31

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet gaia.cs.umass.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at gaia.cs.umass. edu. anything typed in will be sent to port 80 at gaia.cs.umass.edu

2. type in a GET HTTP request:

```
GET /kurose_ross/interactive/index.php HTTP/1.1
Host: gaia.cs.umass.edu
```

by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!
(or use Wireshark to look at captured HTTP request/response)

32

User-server state: cookies

many Web sites use cookies

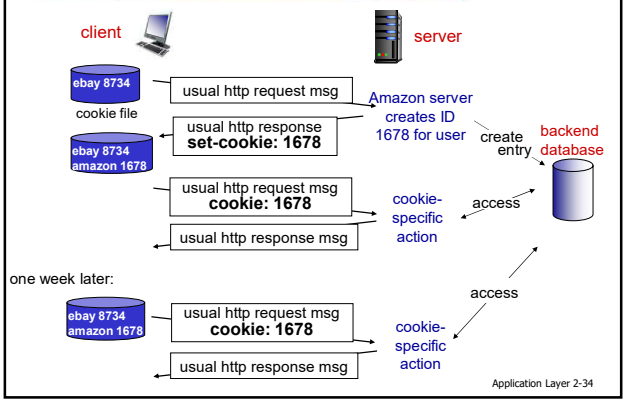
four components:

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

example:
- Susan always access Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

33

Cookies: keeping "state" (cont.)



34

Cookies (continued)

what cookies can be used for:
- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

aside

cookies and privacy:
- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

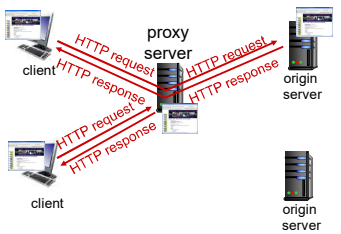
how to keep "state":
- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

35

Web caches (proxy server)

goal: satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



36

More about Web caching

- cache acts as both client and server
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link
- Internet dense with caches: enables "poor" content providers to effectively deliver content (so too does P2P file sharing)

Application Layer 2-37

37

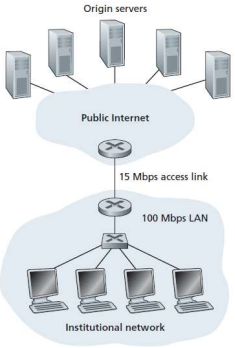
Caching example:

assumptions:

- avg object size: 1M bits
- avg request rate from browsers to origin servers: 15/sec
- RTT from institutional router to any origin server: 2 sec
- LAN rate: 100 Mbps
- access link rate: 15 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = 100% *problem!*
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs



Application Layer 2-38

38

Caching example:

consequences:

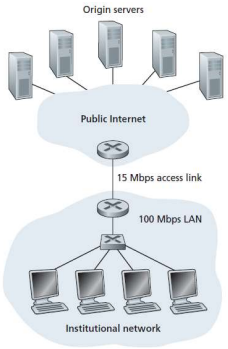
- LAN utilization: 15% (Traffic Intensity)

(15 requests/sec) * (1 Mbits/request)/(100 Mbps) = 0.15

- access link utilization = 100% (Traffic Intensity)

(15 requests/sec)* (1 Mbits/request)/(15 Mbps) = 1

- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs



Application Layer 2-39

39

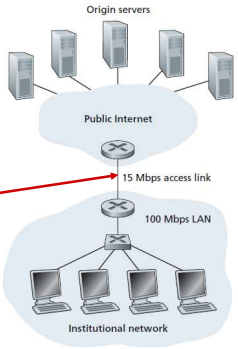
Caching example:

assumptions:

- avg object size: 1M bits
- avg request rate from browsers to origin servers: 15/sec
- RTT from institutional router to any origin server: 2 sec
- LAN rate: 100 Mbps
- access link rate: 100 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = 15%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + usecs + usecs



Cost: increased access link speed (not cheap!)

Application Layer 2-40

40

Caching example: install local cache

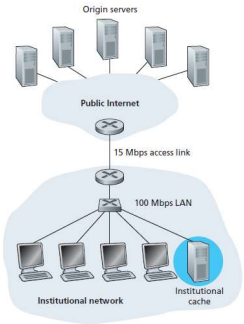
assumptions:

- avg object size: 1M bits
- avg request rate from browsers to origin servers: 15/sec
- RTT from institutional router to any origin server: 2 sec
- access link rate: 15 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = 100%
- total delay = Internet delay + LAN delay

How to compute link utilization, delay?



Cost: web cache (cheap!)

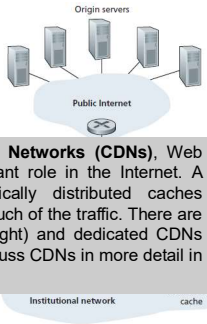
Application Layer 2-41

41

Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin
- access link utilization:



Through the use of **Content Distribution Networks (CDNs)**, Web caches are increasingly playing an important role in the Internet. A CDN company installs many geographically distributed caches throughout the Internet, thereby localizing much of the traffic. There are shared CDNs (such as Akamai and Limelight) and dedicated CDNs (such as Google and Microsoft). We will discuss CDNs in more detail in Chapter 7.

Why CDN's are getting so popular?

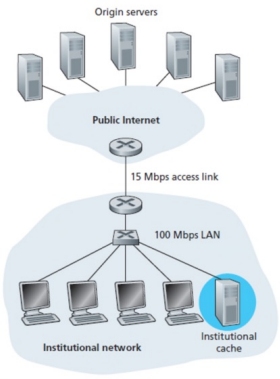
Application Layer 2-42

42

Caching example: install local cache

Calculating access link utilization, delay with cache:

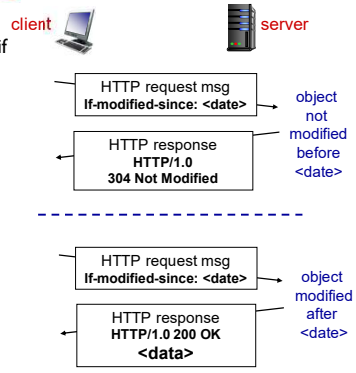
- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin
- access link utilization:
  - 60% of requests use access link
  - Traffic Intensity at access link: 0.6
  - Utilization: 60%
- total delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs
  - less than with 154 Mbps link (and cheaper too!)



Why CDN's are getting so popular?

Conditional GET

- *Goal:* don't send object if cache has up-to-date cached version
  - no object transmission delay
  - lower link utilization
- *cache:* specify date of cached copy in HTTP request
  `If-modified-since: <date>`
- *server:* response contains no object if cached copy is up-to-date:
  `HTTP/1.0 304 Not Modified`



43

Chapter 2: outline

2.1 principles of network applications
2.2 Web and HTTP
2.3 electronic mail
- SMTP, POP3, IMAP

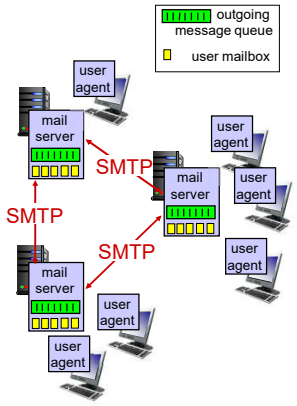
2.4 DNS
2.5 P2P applications
2.6 video streaming and content distribution networks
2.7 socket programming with UDP and TCP

44

Electronic mail

Three major components:
- user agents
- mail servers
- simple mail transfer protocol: SMTP

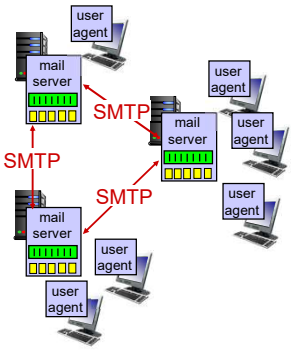
User Agent
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



45

Electronic mail: mail servers

mail servers:
- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages
  - client: sending mail server
  - "server": receiving mail server



46

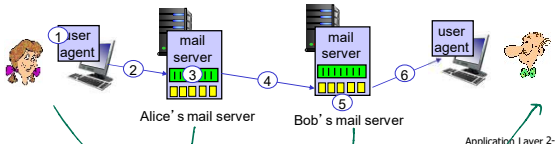
Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction (like HTTP)
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCI

47

Scenario: Alice sends message to Bob

1) Alice uses UA to compose message "to" `bob@someschool.edu`
2) Alice's UA sends message to her mail server; message placed in message queue
3) client side of SMTP opens TCP connection with Bob's mail server
4) SMTP client sends Alice's message over the TCP connection
5) Bob's mail server places the message in Bob's mailbox
6) Bob invokes his user agent to read message



48

eg: gmail server, outlook server

Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```


49

Try SMTP interaction for yourself:

- **`telnet servername 25`**
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

50

SMTP: final words

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

comparison with HTTP:

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

51

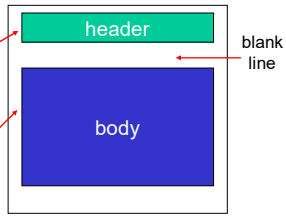
Mail message format

SMTP: protocol for exchanging email messages

RFC 822: standard for text message format:

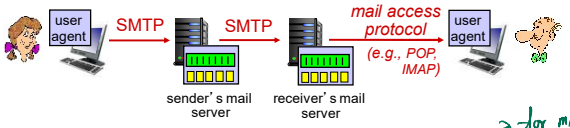
- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from SMTP MAIL FROM, RCPT TO: commands!
- Body: the "message"
  - ASCII characters only



52

Mail access protocols



- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]: authorization, download
  - IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored messages on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

for mailbox

53

POP3 protocol

authorization phase

- client commands:
  - **`user:`** declare username
  - **`pass:`** password
- server responses
  - **`+OK`**
  - **`-ERR`**

transaction phase, client:

- **`list:`** list message numbers
- **`retr:`** retrieve message by number
- **`dele:`** delete
- **`quit`**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```


54

POP3 (more) and IMAP

more about POP3

- previous example uses POP3 "download and delete" mode
  - Bob cannot re-read e-mail if he changes client
- POP3 "download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

IMAP

- keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name

55

2.4:

Chapter 2: outline

2.1 principles of network applications
2.2 Web and HTTP
2.3 electronic mail
- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications
2.6 video streaming and content distribution networks
2.7 socket programming with UDP and TCP

56

Based on UDP protocol because it's fast

DNS: domain name system

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

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.yahoo.com - used by humans

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

Domain Name System:

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

57

DNS: services, structure

DNS services

- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

why not centralize DNS?

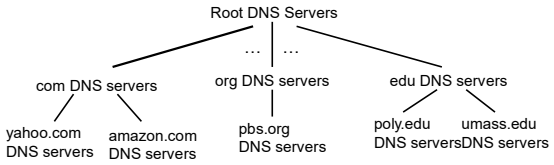
- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: doesn't scale!

58

Knows all translation

DNS: a distributed, hierarchical database



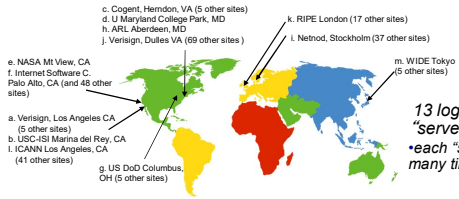
client wants IP for www.amazon.com; 1st approximation:

- client queries root server to find com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

59

DNS: root name servers

- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server



13 logical root name "servers" worldwide
•each "server" replicated many times

60

TLD, authoritative servers

top-level domain (TLD) servers:
- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

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

61

Local DNS name server

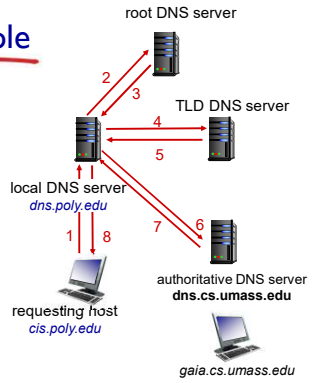
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

62

DNS name resolution example

- host at cis.poly.edu wants IP address for gaia.cs.umass.edu

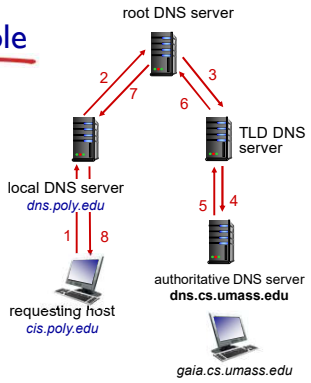
iterated query:
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



63

DNS name resolution example

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



64

DNS: caching, updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire
- update/notify mechanisms proposed IETF standard
  - RFC 2136

65

DNS records

DNS: distributed database storing resource records (RR)

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

type=A
- **name** is hostname
- **value** is IP address

type=NS
- **name** is domain (e.g., foo.com)
- **value** is hostname of authoritative name server for this domain

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

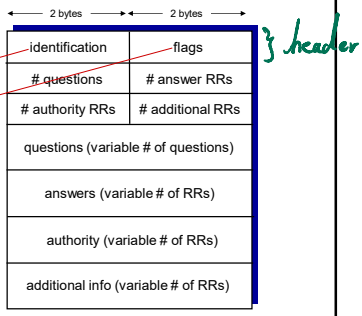
type=MX
- **value** is name of mailserver associated with **name**

66

DNS protocol, messages

- *query* and *reply* messages, both with same *message format*

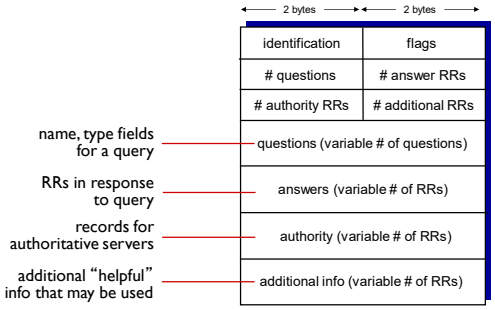
message header
- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



header

67

DNS protocol, messages



68

Inserting records into DNS

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

69

Attacking DNS

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

redirect attacks
- man-in-middle
  - Intercept queries
- DNS poisoning
  - Send bogus relies to DNS server, which caches

exploit DNS for DDoS
- send queries with spoofed source address: target IP
- requires amplification

70

example of P2P: Gmail, u Torrent, Bit Torrent, skype

P2P:

Chapter 2: outline

2.1 principles of network applications
2.2 Web and HTTP
2.3 electronic mail
- SMTP, POP3, IMAP

2.4 DNS

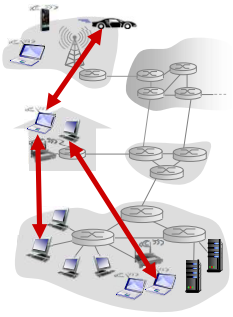
2.5 P2P applications
2.6 video streaming and content distribution networks
2.7 socket programming with UDP and TCP

71

Pure P2P architecture

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

examples:
- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



72

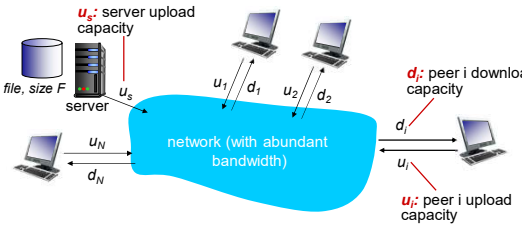
Why P2P: No third person read this information.
And only this protocol use upload speed

File distribution:

File distribution: client-server vs P2P

Question: how much time to distribute file (size *F*) from one server to *N* *peers*?

- peer upload/download capacity is limited resource

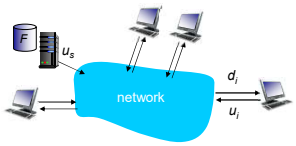


73

file upload speed : u_s

File distribution time: client-server

- *server transmission:* must sequentially send (upload) *N* file copies:
  - time to send one copy: F/u_s
  - time to send *N* copies: NF/u_s
- *client:* each client must download file copy
  - d_{min} = min client download rate
  - min client download time: F/d_{min} → download time



time to distribute F to N clients using client-server approach $D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$

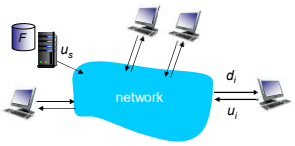
increases linearly in N

74

all of us upload file to everyone
Theoretically, P2P is faster than server-client.

File distribution time: P2P

- *server transmission:* must upload at least one copy
  - time to send one copy: F/u_s
- *client:* each client must download file copy
  - min client download time: F/d_{min}
- *clients:* as aggregate must download *NF* bits
  - max upload rate (limiting max download rate) is $u_s + \Sigma u_i$



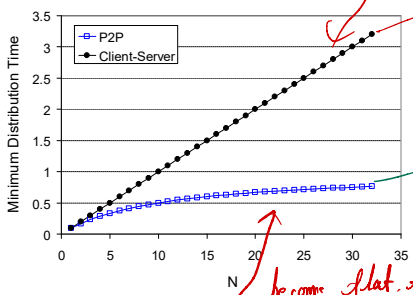
time to distribute F to N clients using P2P approach $D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$

increases linearly in *N* …
… but so does this, as each peer brings service capacity

75

Client-server vs. P2P: example

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



this line controlled by server
by client
become flat, then faster

76

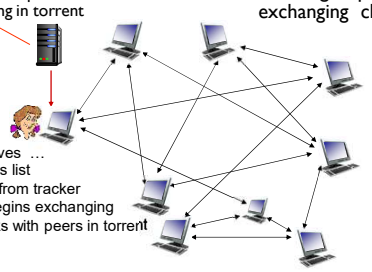
Tracker: users who have file
Seeds: chunks, users have
Torrent:
or: seeds

P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file

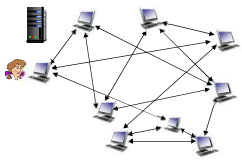


77

We need a server for tracking purpose not data purpose

P2P file distribution: BitTorrent

- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")



- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

78

BitTorrent: requesting, sending file chunks

requesting chunks:

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

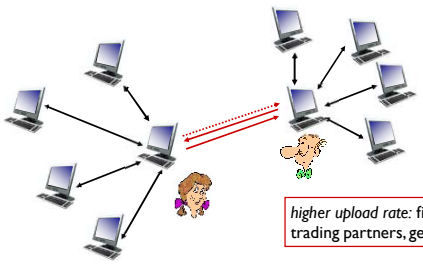
sending chunks: tit-for-tat

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

79

BitTorrent: tit-for-tat

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



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

80

2.6 CDNs

Chapter 2: outline

2.1 principles of network applications
2.2 Web and HTTP
2.3 electronic mail
  - SMTP, POP3, IMAP
2.4 DNS
2.5 P2P applications
2.6 video streaming and content distribution networks (CDNs)
2.7 socket programming with UDP and TCP

81

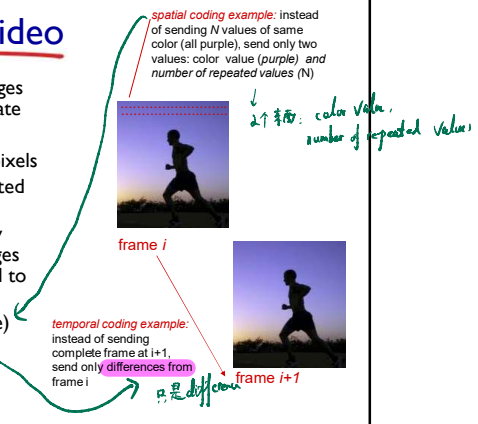
Video Streaming and CDNs: context

- video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
  - ~1B YouTube users, ~75M Netflix users
- challenge: scale - how to reach ~1B users?
  - single mega-video server won't work (why?)
- challenge: heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- *solution:* distributed, application-level infrastructure

82

Multimedia: video

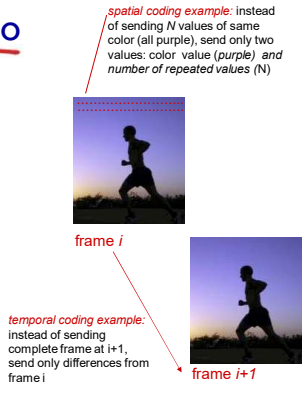
- video: sequence of images displayed at constant rate
  - e.g., 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits
- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)



83

Multimedia: video

- CBR: (constant bit rate): video encoding rate fixed
- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
- examples:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, < 1 Mbps)



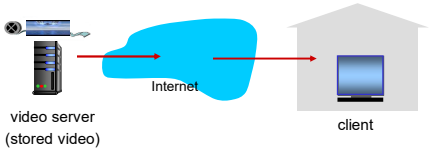
84

Audio and video is separated

Since you can not record video with audio

Streaming stored video:

simple scenario:



85

Streaming multimedia: DASH

- *DASH: D*ynamic, *A*daptive *S*treaming over *H*TTP
- *server:*
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - *manifest file:* provides URLs for different chunks
- *client:*
  - periodically measures server-to-client bandwidth
  - consulting manifest, requests one chunk at a time
    - chooses maximum coding rate sustainable given current bandwidth
    - can choose different coding rates at different points in time (depending on available bandwidth at time)

86

one video could be stored multiple version at different rate for different users. who has different bandwidth or resolution

Streaming multimedia: DASH

- *DASH: D*ynamic, *A*daptive *S*treaming over *H*TTP
- *"intelligence"* at client: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is "close" to client or has high available bandwidth)

so fucking smart to choose which URL of chunk

the complexity is pushed onto edge, so server doesn't have to deal with it.

87

Content distribution networks

- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?
- *option 1:* single, large "mega-server" 肯定不行
  - single point of failure
  - point of network congestion
  - long path to distant clients
  - multiple copies of video sent over outgoing link

….quite simply: this solution *doesn't scale*

88

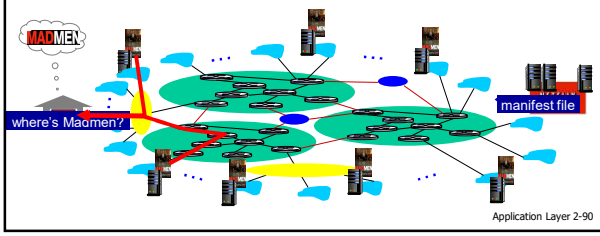
Content distribution networks

- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of
- Akamai: 100,000+ servers in 1000+ clusters in 1000+ networks in 70+ countries serving trillions of requests a day.
  - *enter deep:* push CDN servers deep into many access networks
    - close to users
    - used by Akamai, 1700 locations
  - *bring home:* smaller number (10's) of larger clusters in POPs (point of presence) near (but not within) access networks
    - used by Limelight

89

Content Distribution Networks (CDNs)

- CDN: stores copies of content at CDN nodes
  - e.g. Netflix stores copies of MadMen
- subscriber requests content from CDN
  - directed to nearby copy, retrieves content
  - may choose different copy if network path congested



90

Content distribution networks

- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

- *option 2:* store/serve multiple copies of videos at multiple geographically distributed sites *(CDN)*
  - *enter deep:* push CDN servers deep into many access networks
    - close to users
    - used by Akamai, 1700 locations
  - *bring home:* smaller number (10's) of larger clusters in POPs (point of presence) near (but not within) access networks
    - used by Limelight

when size of data is small

smaller servers, but many

CDN

google does, larger servers, but fewer

when size of data is large

Content Distribution Networks (CDNs)

OTT challenges: coping with a congested Internet

- from which CDN node to retrieve content?
- viewer behavior in presence of congestion?
- what content to place in which CDN node?

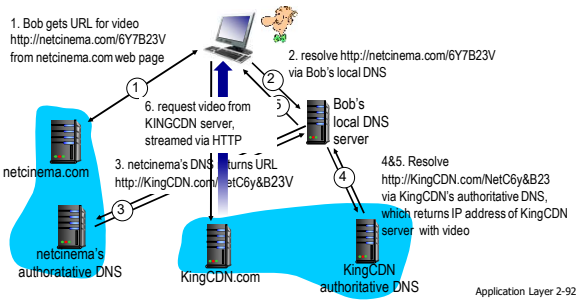
more .. in chapter 7

91

CDN content access: a closer look

Bob (client) requests video http://netcinema.com/6Y7B23V

- video stored in CDN at http://KingCDN.com/NetC6y&B23V

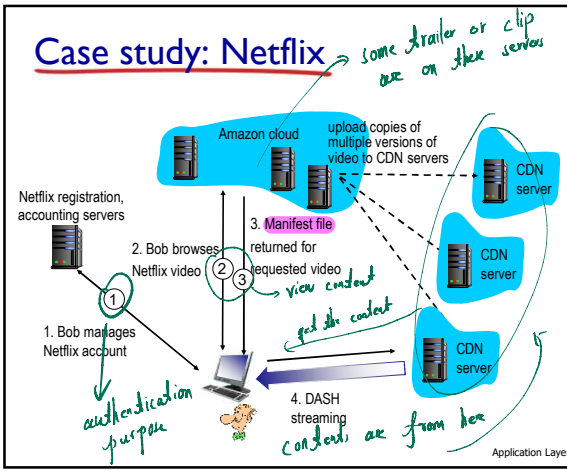


Application Layer 2-92

92

2.7:

Case study: Netflix



Application Layer 2-93

93

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

2.7 socket programming with UDP and TCP

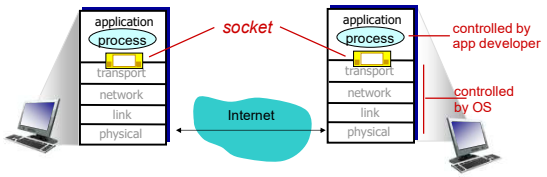
Application Layer 2-94

94

Socket programming

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



Application Layer 2-95

95

Socket programming

Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream-oriented

Application Example:

1. client reads a line of characters (data) from its keyboard and sends data to server
2. server receives the data and converts characters to uppercase
3. server sends modified data to client
4. client receives modified data and displays line on its screen

Application Layer 2-96

96

Socket programming *with UDP*

UDP: no "connection" between client & server

- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

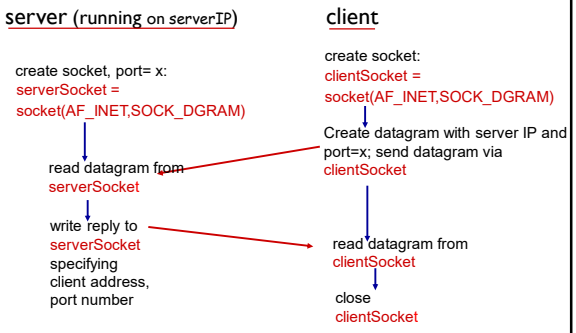
UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes ("datagrams") between client and server

97

Client/server socket interaction: UDP



98

Example app: UDP client

Python UDPClient

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message.encode(),
                    (serverName, serverPort))
modifiedMessage, serverAddress =
                    clientSocket.recvfrom(2048)
print modifiedMessage.decode()
clientSocket.close()
```

include Python's socket library
create UDP socket for server
get user keyboard input
Attach server name, port to message; send into socket
read reply characters from socket into string
print out received string and close socket

99

Example app: UDP server

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print ("The server is ready to receive")
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(),
                        clientAddress)
```

create UDP socket
bind socket to local port number 12000
loop forever
Read from UDP socket into message, getting client's address (client IP and port)
send upper case string back to this client

100

Socket programming *with TCP*

client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

client contacts server by:

- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

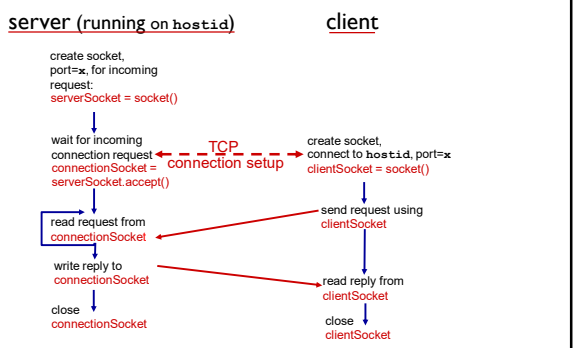
- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint:

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

101

Client/server socket interaction: TCP



102

Example app: TCP client

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET,SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server, remote port 12000

No need to attach server name, port

Example app: TCP server

Python TCPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while True:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                          encode())
    connectionSocket.close()
```

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from socket (but not address as in UDP)

close connection to this client (but *not* welcoming socket)

Chapter 2: summary

our study of network apps now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - HTTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent
- video streaming, CDNs
- socket programming: TCP, UDP sockets

Chapter 2: summary

most importantly: learned about protocols!

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - *headers*: fields giving info about data
  - *data:* info(payload) being communicated

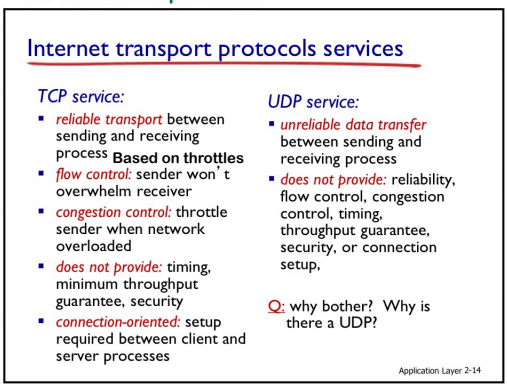
important themes:

- control vs. messages
  - in-band, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable message transfer
- "complexity at network edge"

Two protocol services offered from T to A:

TCP and UDP: check the details o on slides.

TCP and UDP protocol:



Internet transport protocols services

TCP service:

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

UDP service:

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

Q: why bother? Why is there a UDP?

Application Layer 2-14

14

Congestion: could be only control by sender, which need to be sensed by TCP since it would be worse if we add feedback feature.

Stream delivery offered by TCP: If send 1, 2, 3, 4 then receiver will receive 1,2,3,4 not 4,3,2,1 by sequence number feature.

UDP: Does not care about anything.

If UDP dropped packet, then it will do nothing.

It has no chop-up feature for example: send1GB packet it wouldn't chop up, has no segmentation and reassemble feature.

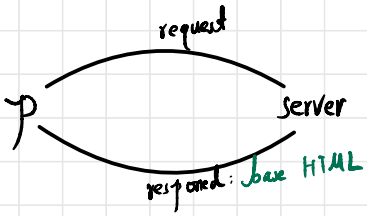
Simple, Fast

TCP: Not fast, but reliable, send the packet again when dropped.
Has segmentation and reassembly.

Section 2:HTML and Web

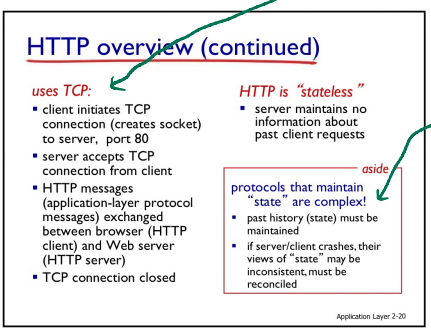
Browser only knows one language: HTML

When1st time P request Server.



Base html contains objects: JS, Vedio….links, then P keep going request server for more objects.

TCP is stateful protocol: can save data



20

Why HTTP should be stateless: Too much information.

continue

TCP协议对应于传输层，而**HTTP**协议对应于应用层，从本质上来说，二者没有可比性。**Http**协议是建立在**TCP**协议基础之上的，当浏览器需要从服务器获取网页数据的时候，会发出一次**Http**请求。**Http**会通过**TCP**建立起一个到服务器的连接通道，当本次请求需要的数据完毕后，**Http**会立即将**TCP**连接断开，这个过程是很短的。所以**Http**连接是一种短连接，是一种无状态的连接。所谓的无状态，是指浏览器每次向服务器发起请求的时候，不是通过一个连接，而是每次都建立一个新的连接。如果是一个连接的话，服务器进程中就能保持住这个连接并且在内存中记住一些信息状态。而每次请求结束后，连接就关闭，相关的内容就释放了，所以记不住任何状态，成为无状态连接。

HTTP versions: 1.0, 1.1, 2.0, 3.0

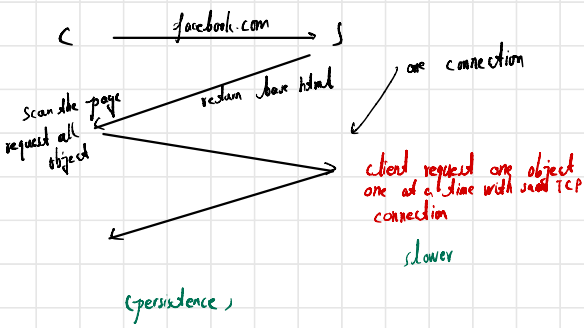
spdy → Quick

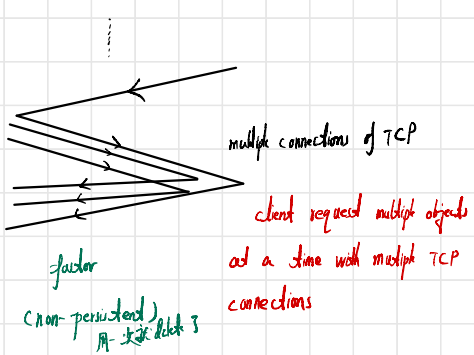
made by google

most used

and do stand use yall

When browser scan the base page it has two ways to request objects from servers. Persistence and Non-persistence.

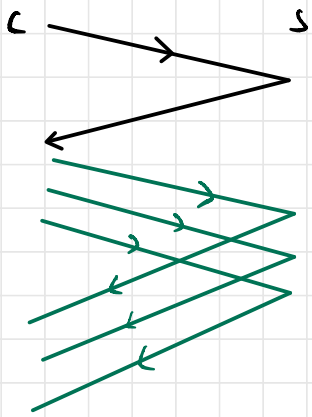




No one is better than other, since non-persistent need to maintain multiple TCP even it is fast.

Heavy, complex, need max sources

Hybrid mode:



green are multiple persistent connects

maximum is and 6

02/01/2022 CS-470

Web Proxy

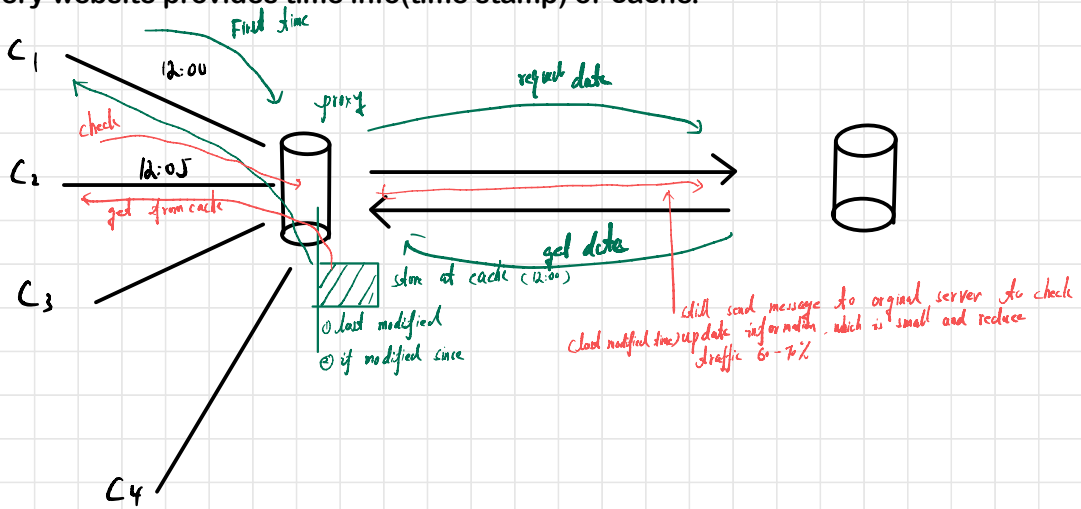
Gate way router is a multiplexers and Demultiplexer.

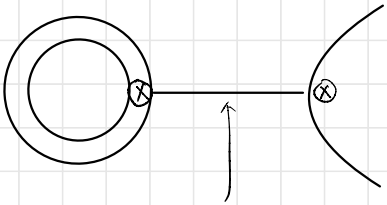
On site cache server: server records what request user made and respond users got, whenever the receive the same request then it doesn't go to original server,instead give you the data in cache server.

Why need cache server: faster, higher response time, reduce redundant of traffic network.

Session log site: where cache server is useless because everyone's data is different.

TTL: Every website provides time info(time stamp) of cache.





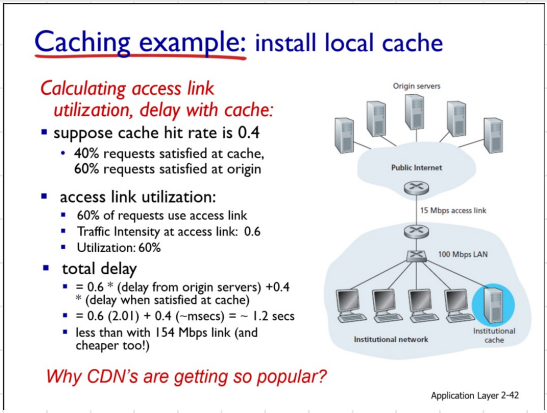
Access link: the connection between your network and to the public network

Local network is faster than public

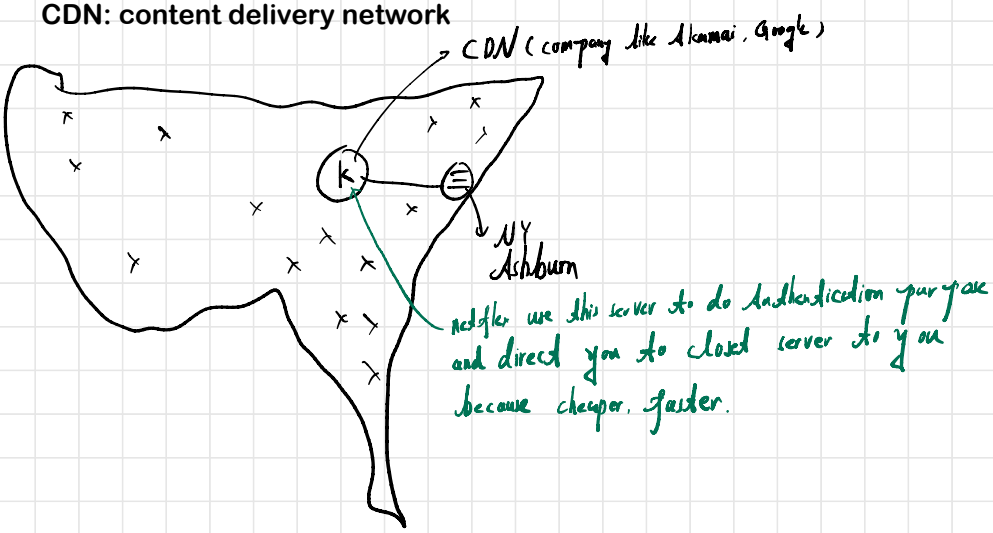
- Size is small
- Connection is good

Why is not sustainable: More students, more traffic,more congestion, pay more
It is like black hole

Install local cache



CDN: content delivery network



DNS: Convert domain name to IP address.

IP address: unique identifier of a machine on any network.

02/03/2022

{ open public DNS:
  Google DNS

Why DNS shouldn't be centralized: Fast response , preventing one down the whole system down.

DNS: is a edge server. It is a application layer protocol, which implements DNS.
It is a core internet functionality but implemented as application protocol.

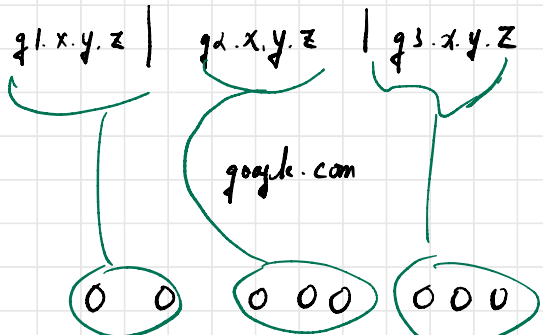
Edge layers:Application and transport.

Alias and Canonical name: If we want IP address then we need to know canonical name.

eg: wiki.org (easy to remember) → d1.T1.wiki.org (actual name) → And convert cname to IP

DNS also convert alias to canonical name

Interesting phenomenon:



同一个domain name 他出现不同的ip address

TLD:

TLD, authoritative servers

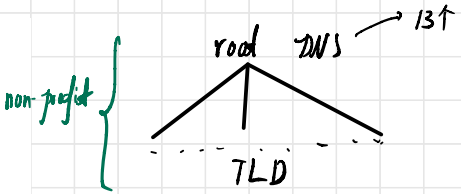
top-level domain (TLD) servers:

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

authoritative DNS servers:

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

Application Layer 2-61



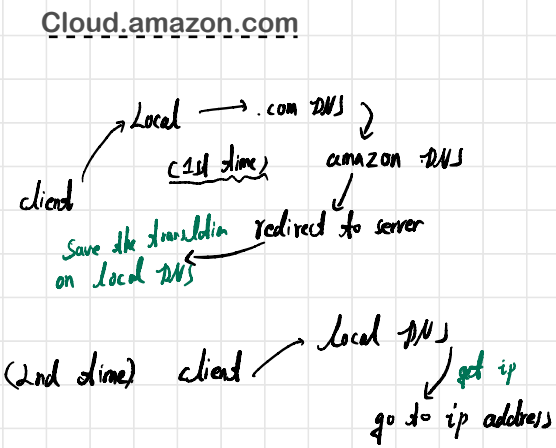
61

Local DNS:

Local DNS name server

- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

Application Layer 2-62

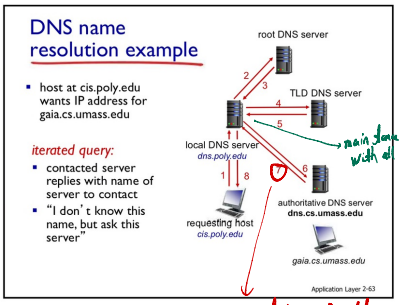


当然如果突然ip变了，不再是原来的了，那这个local的存的ip就用不了了，所以每个translation都有个 TTL. (Time To Live)
当过了TTL，它会自己discard。
除了那些非常常用的网站，不如不会自己refresh。要不然就等再cache一遍。

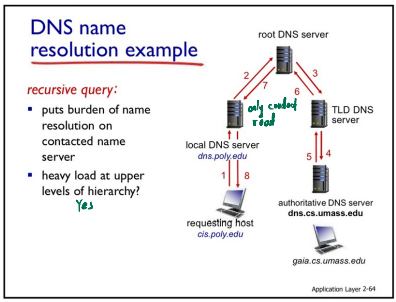
Authoritative dns balance the loads

DNS can convert ip to domain name as well.

Two types of resolutions:



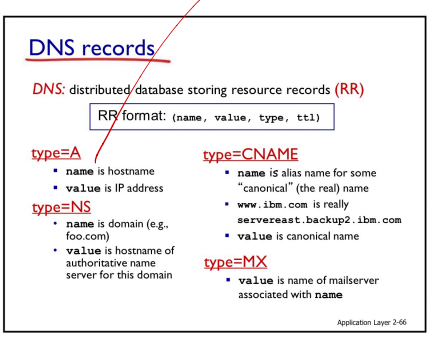
63



64

If the server is down, the authoritative dns will know first, and the local dns won't know until it refreshes or TTL expires.

DNS Records



Authoritative server gives Type A record and CName

TLD will gives type NS and A

Root gives type A and NS

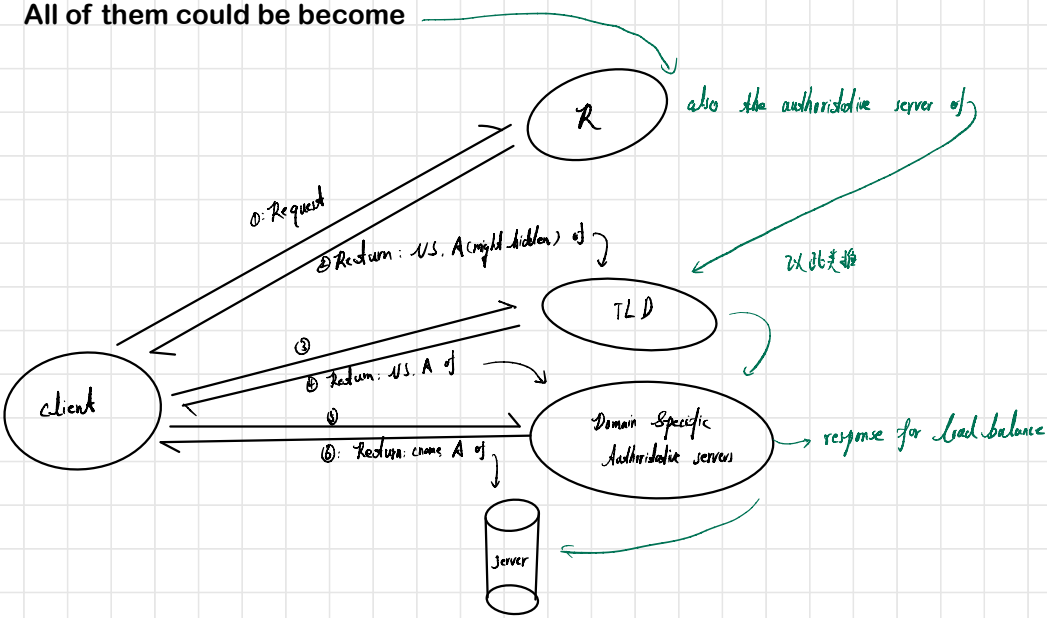
02/08/2022

Iterated and recursive, which is one is better.

Both has pro and cons, but Iterative is slightly better since no pressure on top servers.

Could Root, TLD, and authoritative server become authoritative servers?

All of them could be become



Build own website: no need for root to know. TLD is enough.

DNS use UDP to transport: Why?

The DNS has to be the fastest, and TCP will slow down DNS transmission.

DNS poisoning. External learning. Since DNS use UDP has no secure measurement.

Why we still have UDP and TCP?

UDT has all features of UCP but more secured and safe.

DNS will be the crucial content on EXAN and quiz

E-mail:

The protocol email uses: SMTP, POP3, IMAP.

Simple Mail Transport Protocol:

HTTP is a pool protocol and SMTP is a punch protocol

POP3: used for military.

02/10/2022

Most of notes will be on slides

Why P2P: No third person read this information.
And only this protocol use upload speed

P2P is not favored by those companies: Since they can not make money like Netflix

File distribution: on slides

02/15/2022

都把在 PPT 上

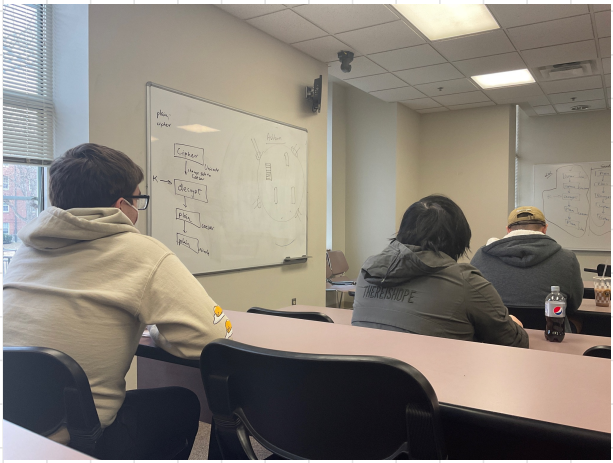
(~08:25)

Bob example

(~ 12:21) 2.6 end

preference 发音

(~19:09) DNS message on slides



Network OS: NOS

HAC: High Availability Cluster (~23:48)

(30:45~) project

(44:43~)

UDP server class

socket = new Dat..... (9876)

incomingPack.get...( )

Quiz 2 summery:

What server will the url goes first?

Local dns is a cache which is not a server. Thus the correct answer is root dns.

Chapter 3

Chapter 3: Transport Layer

our goals:

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

1

Chapter 3 outline

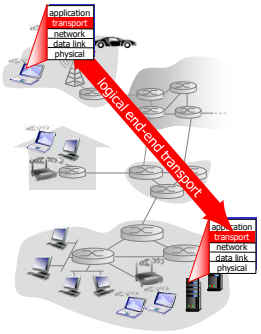
3.1 transport-layer services
3.2 multiplexing and demultiplexing
3.3 connectionless transport: UDP
3.4 principles of reliable data transfer
3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control
3.7 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:

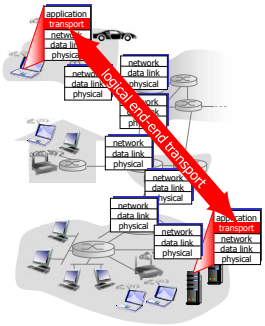
12 kids in Ann's house sending letters to 12 kids in Bill's house:

- hosts = houses
- processes = kids
- app messages = letters in envelopes
- transport protocol = Ann and Bill who demux to in-house siblings
- 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

Chapter 3 outline

3.1 transport-layer services
3.2 multiplexing and demultiplexing
3.3 connectionless transport: UDP
3.4 principles of reliable data transfer
3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

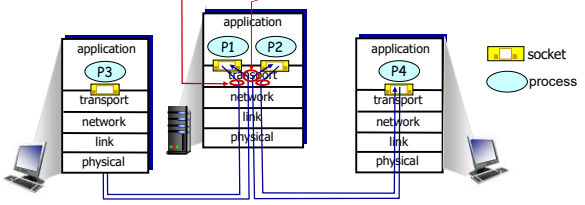
3.6 principles of congestion control
3.7 TCP congestion control

6

Multiplexing/demultiplexing

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

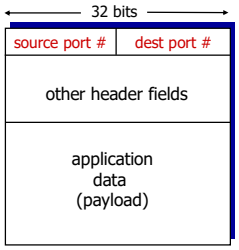
demultiplexing at receiver: use header info to deliver received segments to correct socket



7

How demultiplexing works

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



TCP/UDP segment format

8

Connectionless demultiplexing

- *recall:* created socket has host-local port #:

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

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

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

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

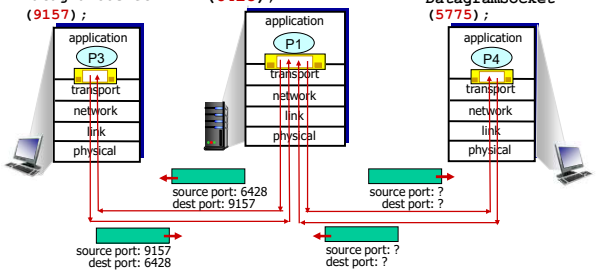
9

Connectionless demux: example

```
DatagramSocket
mySocket2 = new
DatagramSocket
(9157);
```

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

```
DatagramSocket
mySocket1 = new
DatagramSocket
(5775);
```



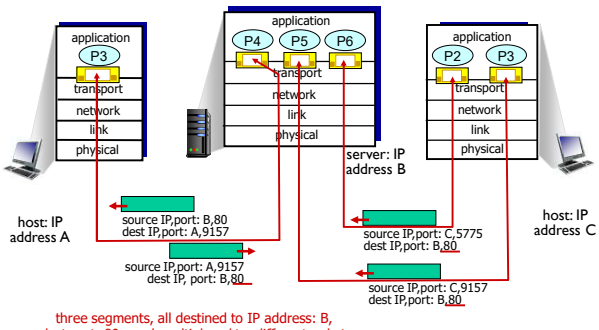
10

Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver 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

11

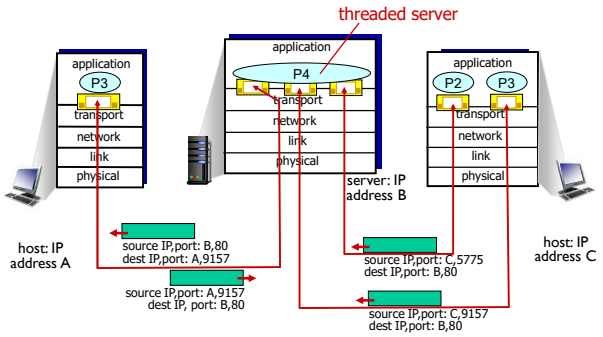
Connection-oriented demux: example



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

12

Connection-oriented demux: example



13

TCP: bidirectional
UDP: unidirection

Chapter 3 outline

3.1 transport-layer services
3.2 multiplexing and demultiplexing
3.3 connectionless transport: UDP
3.4 principles of reliable data transfer
3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control
3.7 TCP congestion control

14

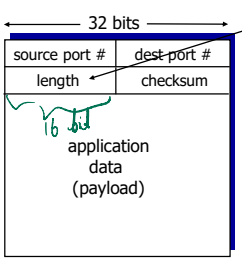
UDP: User Datagram Protocol [RFC 768]

- "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

- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

15

UDP: segment header



UDP segment format

why is there a UDP?
- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control: UDP can blast away as fast as desired

16

only check the error, not fix

UDP checksum

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

sender:
- treat segment contents, including header fields, as sequence of 16-bit integers
- checksum: addition (one'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 ….

17

Internet checksum: example

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
```

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

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

18

02/17/2022

Chapter 3

- TCP: complicated. slow. reliable.
- UDP: easy. fast, unreliable

"Jack steak and kingdom"

A can't directly connect another A without using T.N.P but only for one App. expensive

Logical pipe:

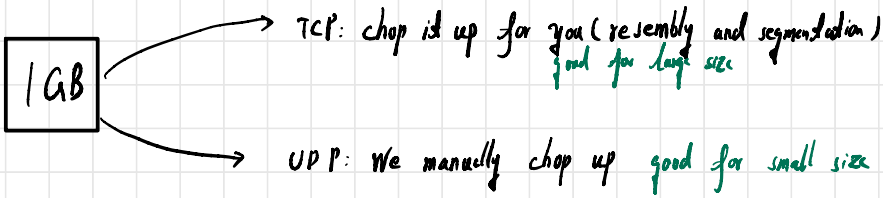
Why transport protocol runs on end system, not in intermediate router?
Because intermediate router not connect application, but only the host.

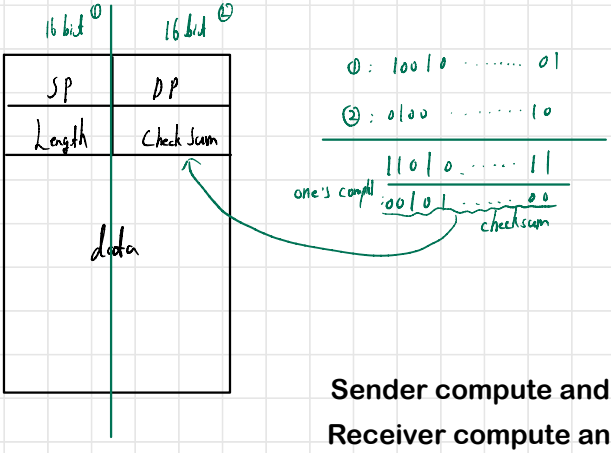
Transport: reassemble and chop up data

Stream delivery = sequence number delivery

TCP requires a hand shake.

Transport: App to app delivery.





Sender compute and poplulate Check sum

Receiver compute and compare the checksum

一直加多余的1直到只有十六个bits

Chapter 3 outline

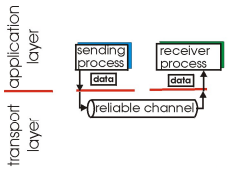
3.1 transport-layer services
3.2 multiplexing and demultiplexing
3.3 connectionless transport: UDP
3.4 principles of reliable data transfer
3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control
3.7 TCP congestion control

19

Principles of reliable data transfer

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



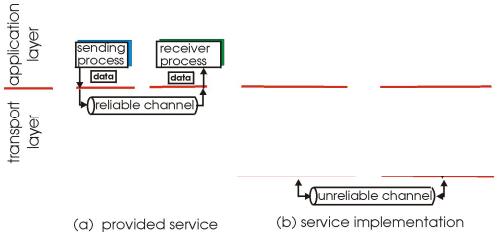
(a) provided service

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

20

Principles of reliable data transfer

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



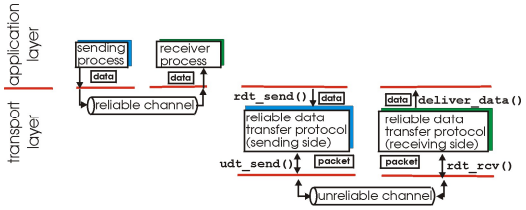
(a) provided service (b) service implementation

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

21

Principles of reliable data transfer

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



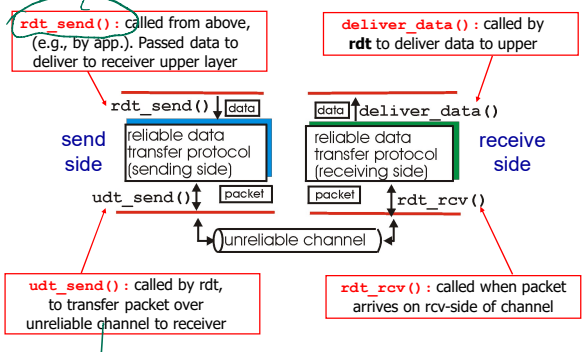
(a) provided service (b) service implementation

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

22

rdt_send, called by only A layer resides in T layer

Reliable data transfer: getting started



23

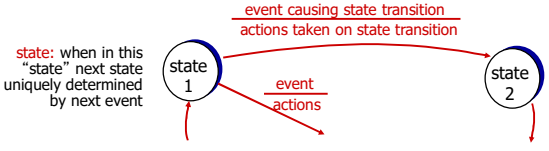
can't be called by rdt in A

udt_send is in network layer, called by T layer.

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



24

02/22/2022

Why do we have reliabile system on non-reliable network.

1: speed.

2: network guarantee something. Like TCP

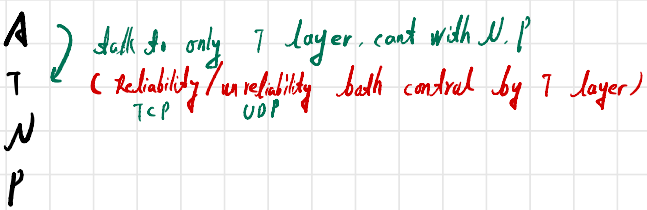
3: simplicity. Do not want reliability on top of network layer.

Reliable network:

Reliability: A guarantee the packet will be delivered to B

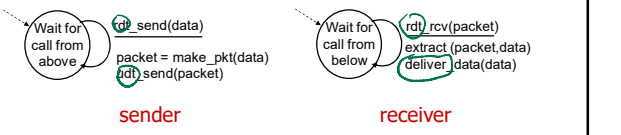
Integrity: If a data will be deliver to B it will be exact the A send.

check sum mechnism.



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 reads data from underlying channel



sender receiver

25

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:

How do humans recover from "errors" during conversation?

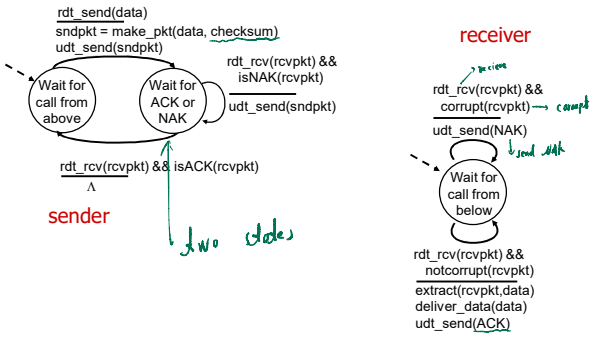
26

rdt2.0: channel with bit errors

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

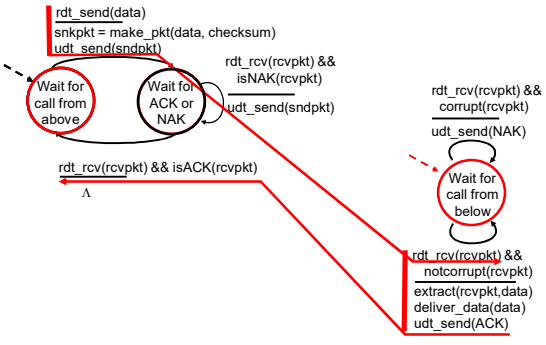
27

rdt2.0: FSM specification



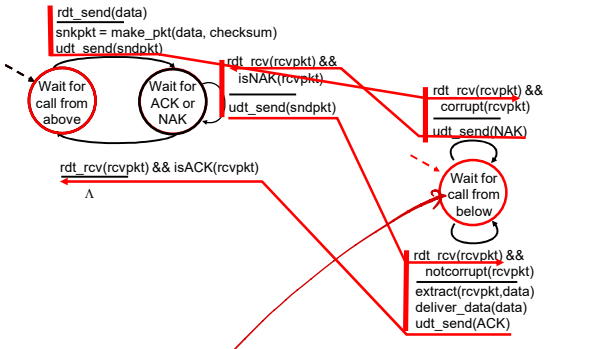
28

rdt2.0: operation with no errors



29

rdt2.0: error scenario



30

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 pkt if ACK/NAK corrupted
- sender adds *sequence number* to each pkt
- receiver discards (doesn't deliver up) duplicate pkt

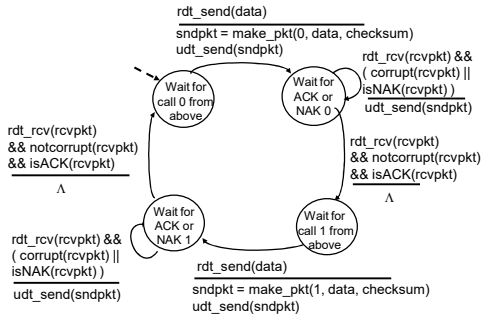
stop and wait
sender sends one packet, then waits for receiver response

31

sender and reciever will wait forever!!!!

stop & wait protocal

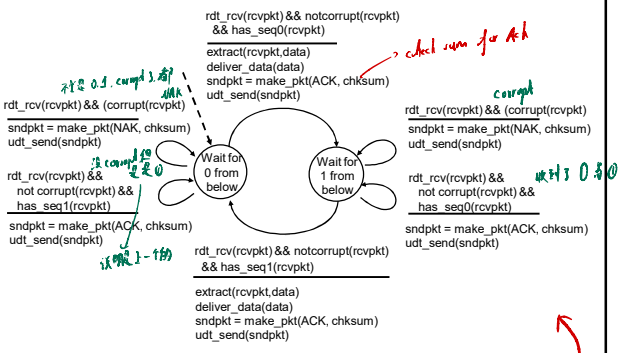
rdt2.1: sender, handles garbled ACK/NAKs



32

1,2 sequence no. is enough.

rdt2.1: receiver, handles garbled ACK/NAKs



33

Too slow !!!!

rdt2.1: discussion

for duplication

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

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

34

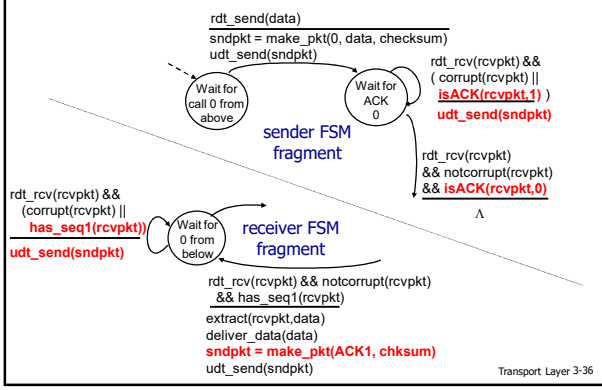
rdt2.2: a NAK-free protocol

- 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*

35

faster

rdt2.2: sender, receiver fragments



36

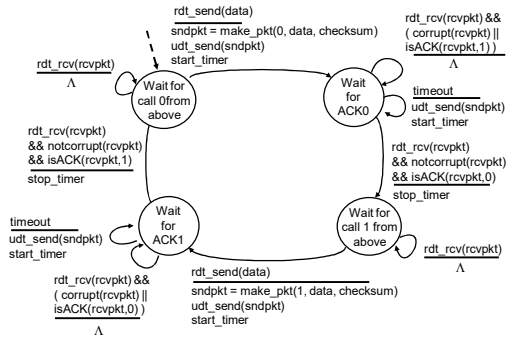
rdt3.0: channels with errors *and* loss

new assumption: underlying channel can also lose packets (data, 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 seq. #'s already handles this
  - receiver must specify seq # of pkt being ACKed
- requires countdown timer

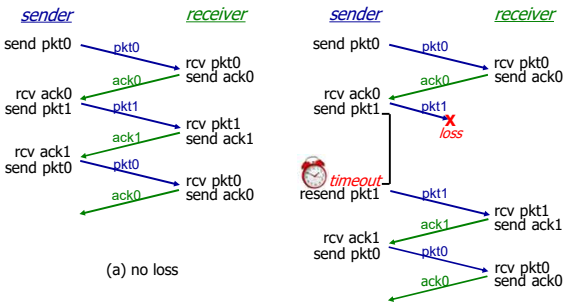
37

rdt3.0 sender



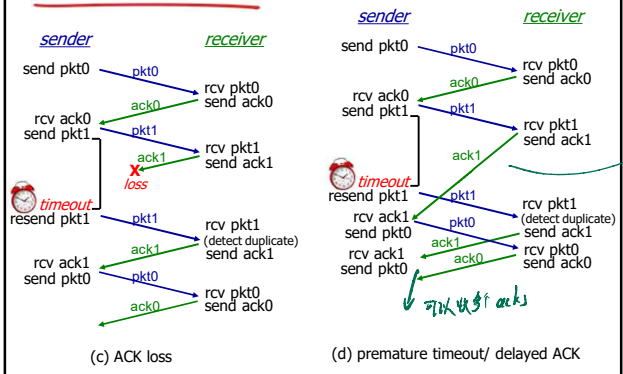
38

rdt3.0 in action



39

rdt3.0 in action



40

Performance of rdt3.0

- rdt3.0 is correct, but performance stinks
- e.g.: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

$$D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$$

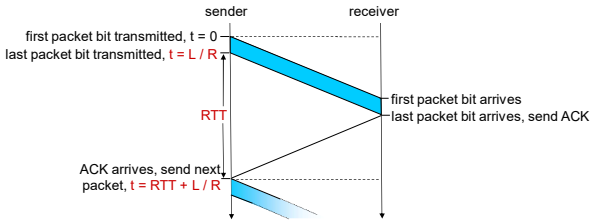
- U_{sender}: *utilization* – fraction of time sender busy sending

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

- if RTT=30 msec, 1KB pkt every 30 msec: 33kB/sec thruput over 1 Gbps link
- network protocol limits use of physical resources!

41

rdt3.0: stop-and-wait operation



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

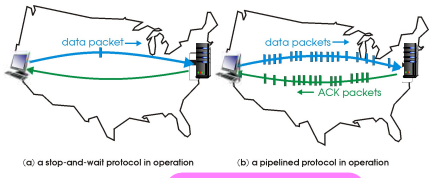
42

pipeline: multiple in-flight pkt

Pipelined 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

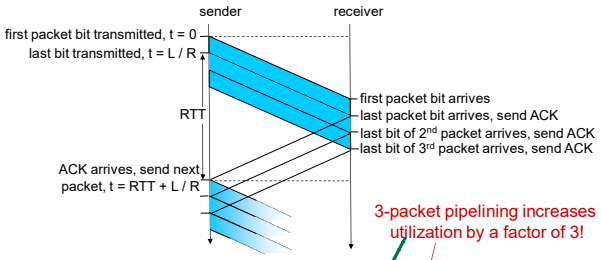


- two generic forms of pipelined protocols: *go-Back-N, selective repeat*

43

pipelined protocols: { go-back-n, selective repeat

Pipelining: increased utilization



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

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

44

sending time increased

Pipelined protocols: overview

Go-back-N:

- sender can have up to N unacked packets in pipeline
- receiver only sends *cumulative ack*
  - doesn't ack packet if there's a gap
- sender has timer for oldest unacked packet
  - when timer expires, retransmit *all* unacked packets

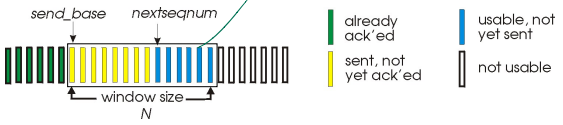
Selective Repeat:

- sender can have up to N unack'ed packets in pipeline
- rcvr sends *individual ack* for each packet
- sender maintains timer for each unacked packet
  - when timer expires, retransmit only that unacked packet

45

Go-Back-N: sender

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



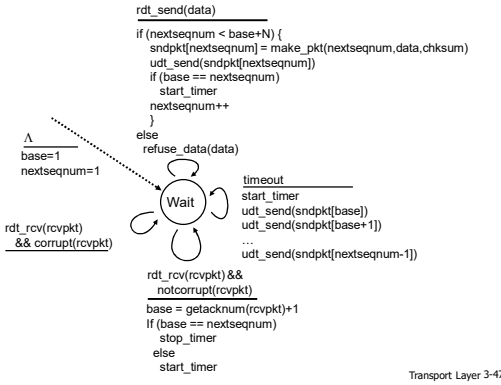
- ACK(n): ACKs all pkts up to, including seq # n - *"cumulative ACK"*
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- *timeout(n)*: retransmit packet n and all higher seq # pkts in window

46

this not been sent yet

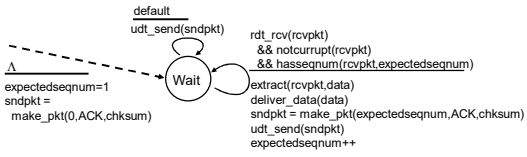
sender required buffering

GBN: sender extended FSM



47

GBN: receiver extended FSM



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

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

- out-of-order pkt:
  - discard (don't buffer): *no receiver buffering!*
  - re-ACK pkt with highest in-order seq #

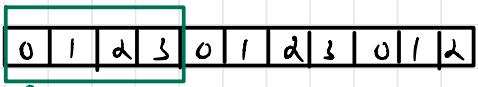
not required for receiver usually is window-size

48

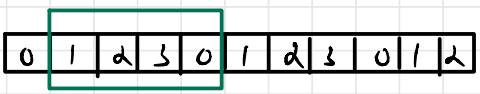
02/24/2022

Go-back-N: Window size: 4

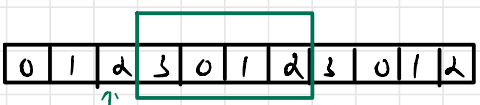
seq #:



oldest packet timer, if time out, send all packet in the window



0 is acked, then slide window to next



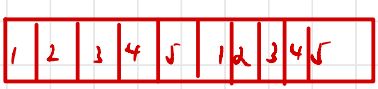
2 is acked, then slide the window since ack is cumulative

Problem:

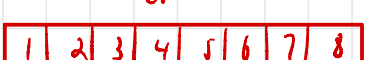
①: what determines seq # : window size?

①: issue of go-back-to N

②: the seq # in GBN is



or



when window slice, will seq # change?

size of window should $\frac{1}{2}$ seq'd region.

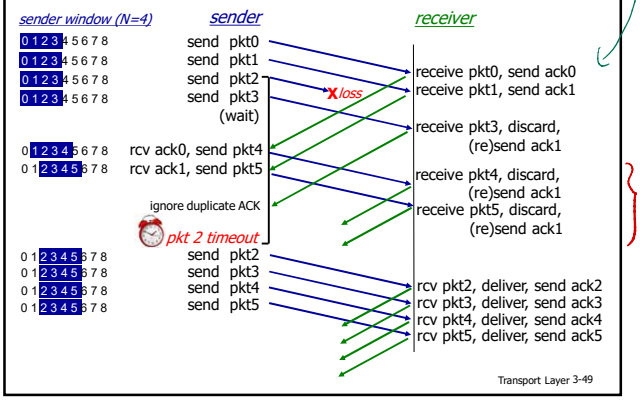
smaller.

The size of window is dynamic The more congested, the smaller window size

The size of window is at least half of the range of sequence numbers

Window size has threshold:

GBN in action



Transport Layer 3-49

49

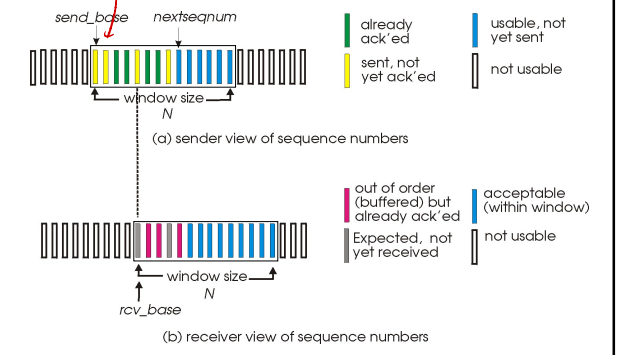
Selective repeat

- 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
  - limits seq #s of sent, unACKed pkts

Transport Layer 3-50

50

Selective repeat: sender, receiver windows



Transport Layer 3-51

51

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

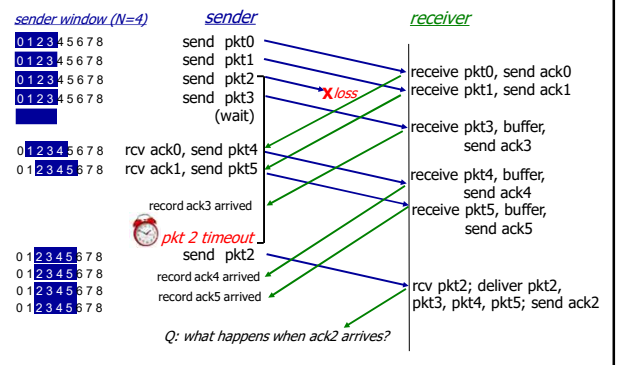
pkt n in [rcvbase-N,rcvbase-1]
- ACK(n)

otherwise:
- ignore

Transport Layer 3-52

52

Selective repeat in action



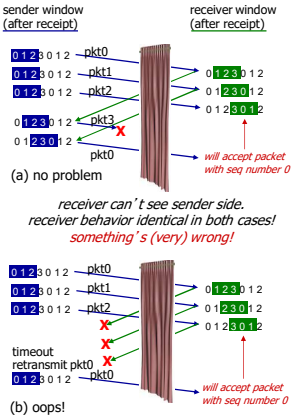
Transport Layer 3-53

53

Selective repeat: dilemma

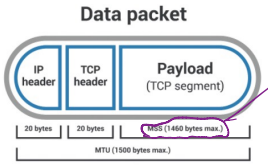
example:
- seq #'s: 0, 1, 2, 3
- window size=3
- receiver sees no difference in two scenarios!
- duplicate data accepted as new in (b)

Q: what relationship between seq # size and window size to avoid problem in (b)?



Transport Layer 3-54

54



Chapter 3 outline

3.1 transport-layer services
3.2 multiplexing and demultiplexing
3.3 connectionless transport: UDP
3.4 principles of reliable data transfer
3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control
3.7 TCP congestion control

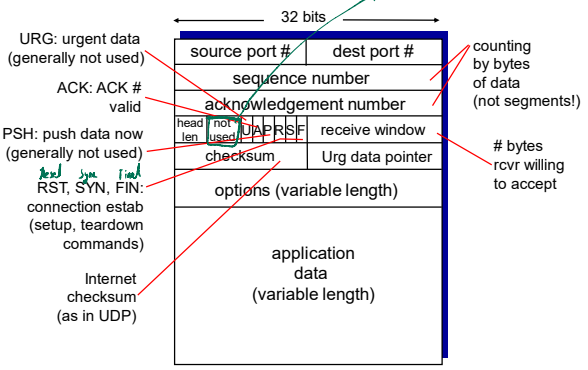
55

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

56

TCP segment structure



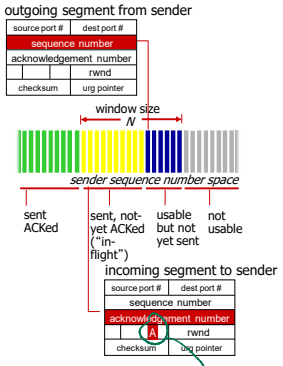
57

TCP seq. numbers, ACKs

sequence numbers:
- byte stream "number" of first byte in segment's data

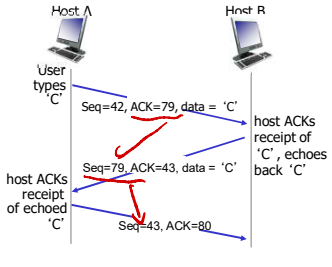
acknowledgements:
- seq # of next byte expected from other side
- cumulative ACK

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



58

TCP seq. numbers, ACKs



simple telnet scenario

59

TCP round trip time, 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

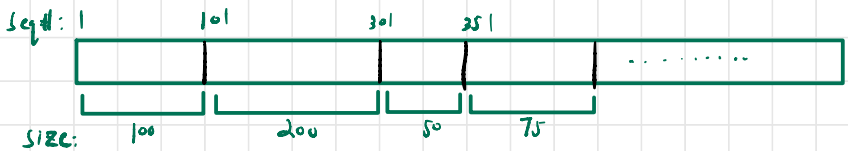
Q: how to estimate RTT?
- **SampleRTT**: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **SampleRTT** will vary, want estimated RTT "smoother"
  - average several *recent* measurements, not just current **SampleRTT**

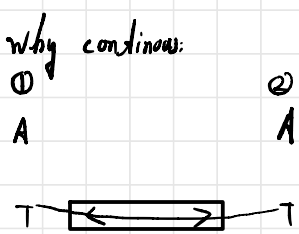
60

Sequence #
- GBN: seq # for packets
- TCP: seq # for byte-stream

TCP assumes data is continuous not discrete. That's why called stream.

The size of segment in TCP is different.





Q: how receiver handles out-of-order segments

- A: TCP spec doesn't say, - up to implementor

- TCP: like Ed, cars. wash machine may not have large memory. (don't buffer)
- PC

either way (buffer)

⚠ why not measure instantaneous RTT to determine time-out?

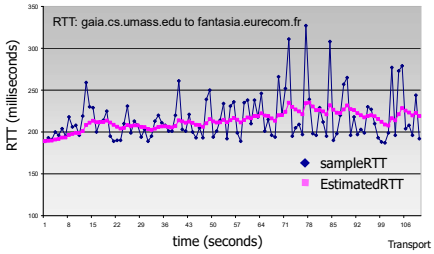
Smoothen increase, not only influence by bursty data only, but also the previous EstimatedRTT, the history

Timeout should based on smoothen RTT.

TCP round trip time, timeout

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

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



Some of RTTs, not all of them. Otherwise you're killing the system.

61

TCP round trip time, timeout

- timeout interval: `EstimatedRTT` plus "safety margin"
  - large variation in `EstimatedRTT` -> larger safety margin
- estimate SampleRTT deviation from EstimatedRTT:

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

(DevRTT: Deviation RTT)

(typically, $\beta = 0.25$)

in order to be safe

`TimeoutInterval = EstimatedRTT + 4*DevRTT`

estimated RTT | "safety margin"

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

62

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

63

TCP reliable data transfer

- TCP creates rdt service on top of IP's unreliable service
  - pipelined segments
  - cumulative acks
  - single retransmission timer
- retransmissions triggered by:
  - timeout events
  - duplicate acks

let's initially consider simplified TCP sender:
- ignore duplicate acks
- ignore flow control, congestion control

64

TCP sender events:

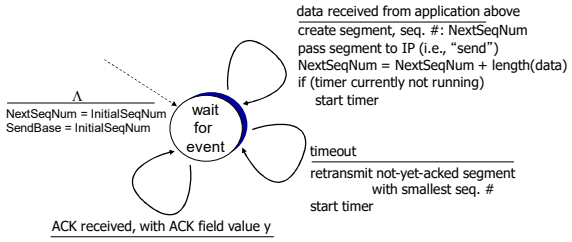
data rcvd from app:
- create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running
  - think of timer as for oldest unacked segment
  - expiration interval: `TimeOutInterval`

timeout:
- retransmit segment that caused timeout
- restart timer

ack rcvd:
- if ack acknowledges previously unacked segments
  - update what is known to be ACKed
  - start timer if there are still unacked segments

65

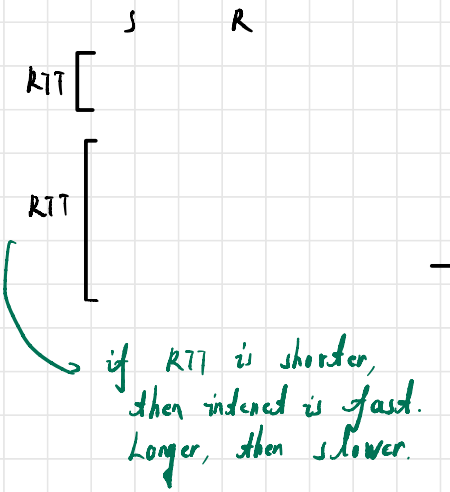
TCP sender (simplified)

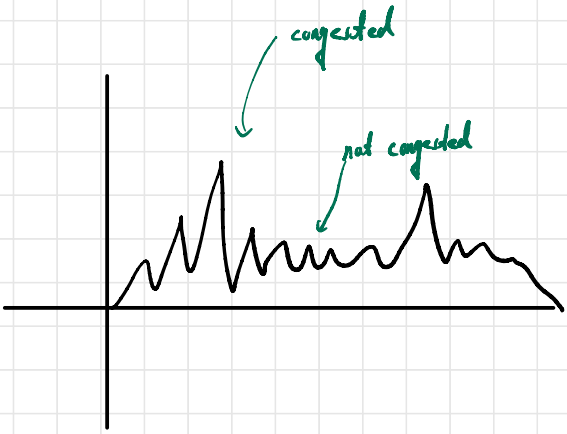


66

03/01/2022

internet is bursty data.

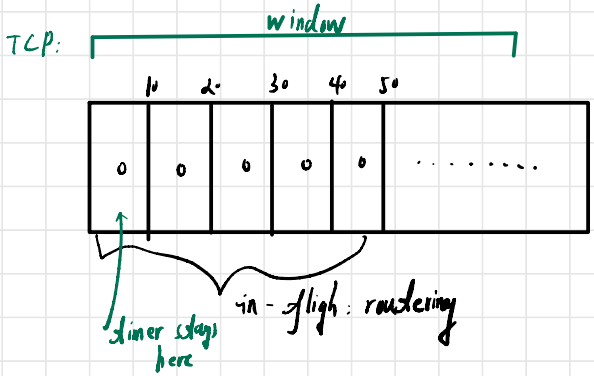




If rtt is shorter, then we may have timer wait less time. If rtt is longer or bigger, then we should have the timer wait more time.

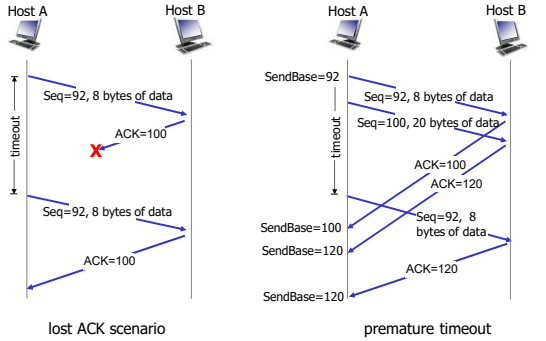
RTT should not be larger than time-out

It is good to spend time on sending rather than waiting to increase utilization.



If time out, only send one segment, not whole window, then restart timer.

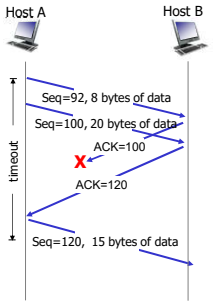
TCP: retransmission scenarios



lost ACK scenario premature timeout

67

TCP: retransmission scenarios



cumulative ACK

68

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

69

TCP 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.

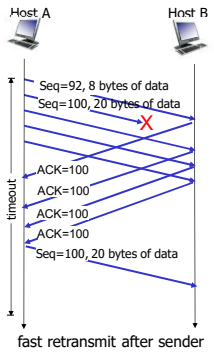
TCP fast retransmit

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

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

70

TCP fast retransmit



fast retransmit after sender receipt of triple duplicate ACK

71

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

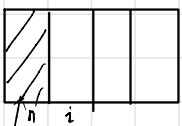
72

03/03/2022 color of using

① optimistic to wait up to 500ms if the n recived in order

Last ack will be ack n if n is expected.

②

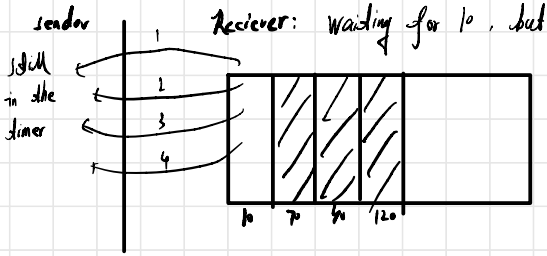


pending ack n if i arrives then immediatly send ack i.

③:

expected 30 but 50, 70 recived in out of order.
Then immediatly send duplicate of ack 30
Since we assume 30 is lost.

Fast-retransmission:



Reciever: waiting for 10, but got 70, 50, 120, Recriver send ack 10 4 times. The send will not wait the sender time-out. But sends ack 10 immediatly. Because these acks are telling sender network is not congested.

03/03/2022

Sender

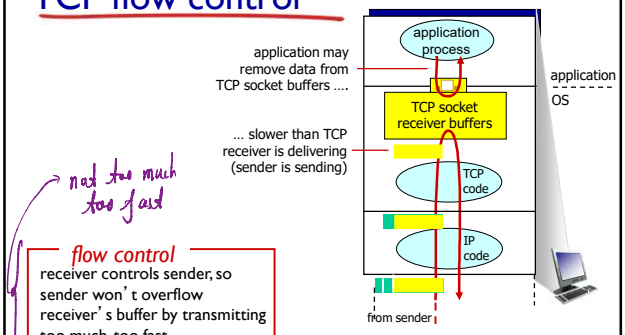
exam:

Reciever

If the timer is time-out and receives nothing
That means network is congest.

If receives duplicated acks the it means network is fine but the sent segment is lost. And not need to wait timer then fast retransmit.

TCP flow control



not too much too fast

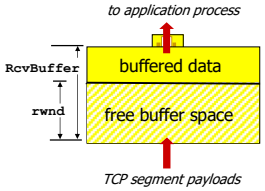
flow control
receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

73

Receiver window : rwnd is dynamic, not static.

TCP flow control

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



receiver-side buffering

has a limit: 64 kBytes (could larger than 64 KB) Because make sure Fair share network

74

rwnd: is a value based on RcvBuffer. Network property
RcvBuffer: is system property
rwnd could be smaller than RcvBuffer

Chapter 3 outline

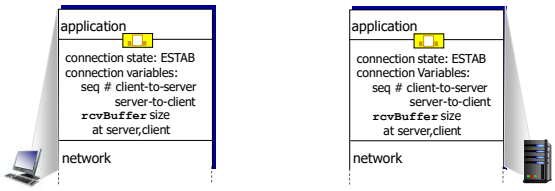
3.1 transport-layer services
3.2 multiplexing and demultiplexing
3.3 connectionless transport: UDP
3.4 principles of reliable data transfer
3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control
3.7 TCP congestion control

75

Connection Management

before exchanging data, sender/receiver "handshake":
- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters




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

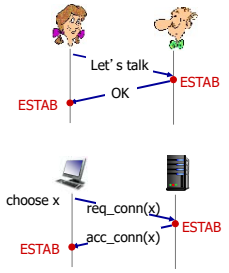
```
Socket connectionSocket =
  welcomeSocket.accept();
```


76

快过，无细讲

Agreeing to establish a connection

2-way handshake:

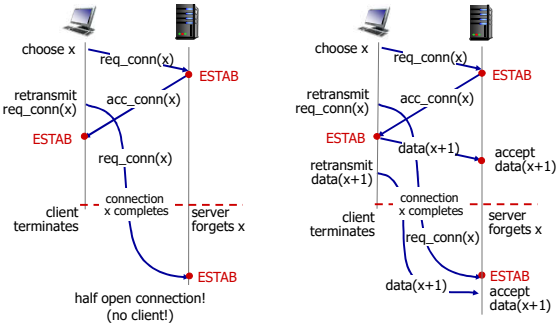


Q: will 2-way handshake always work in network?
- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

77

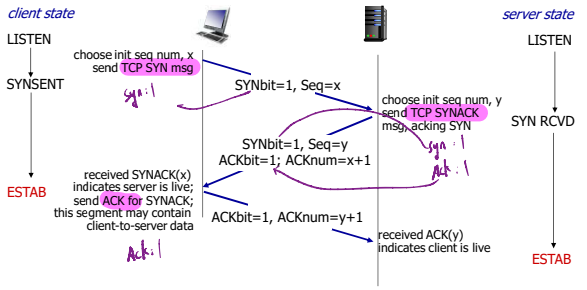
Agreeing to establish a connection

2-way handshake failure scenarios:



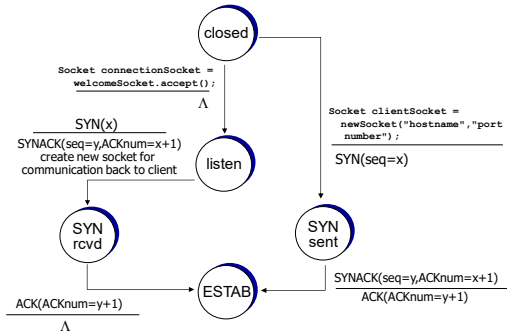
78

TCP 3-way handshake



79

TCP 3-way handshake: FSM



80

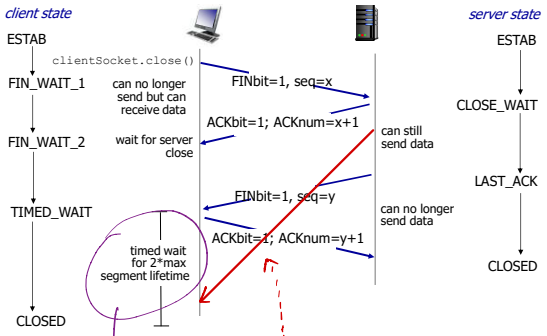
TCP: closing a connection

- client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN
- simultaneous FIN exchanges can be handled

81

Fin 之后还能send data 3

TCP: closing a connection



82

why wait ? In case some data packed is delayed.

Based on estimated RTT

Chapter 3 outline

3.1 transport-layer services
3.2 multiplexing and demultiplexing
3.3 connectionless transport: UDP
3.4 principles of reliable data transfer
3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control
3.7 TCP congestion control

83

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!

This is receiver

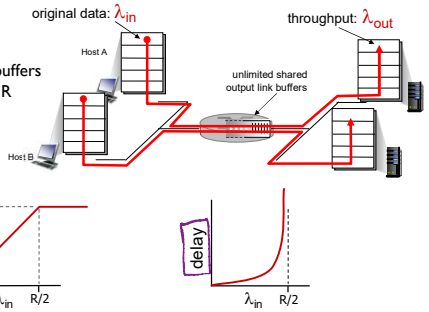
Based on receive window

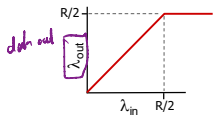
84

03/03/2022

Causes/costs of congestion: scenario 1

- two senders, two receivers
- one router, infinite buffers
- output link capacity: R
- no retransmission





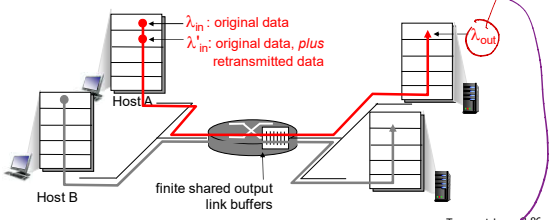
- maximum per-connection throughput: R/2

- large delays as arrival rate, λ_{in}, approaches capacity

85

Causes/costs of congestion: scenario 2

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

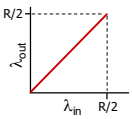


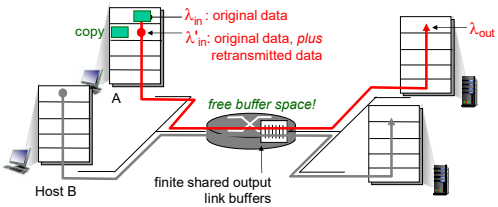
86

Causes/costs of congestion: scenario 2

idealization: perfect knowledge

- sender sends only when router buffers available



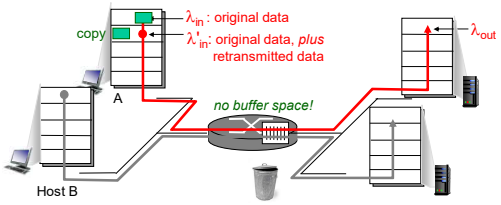


87

Causes/costs of congestion: scenario 2

Idealization: known loss
packets can be lost, dropped at router due to full buffers

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

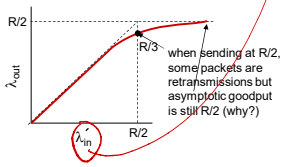


88

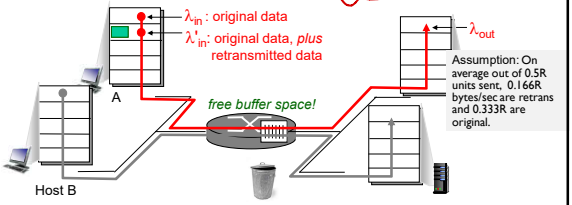
Causes/costs of congestion: scenario 2

Idealization: known loss
packets can be lost, dropped at router due to full buffers

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



when sending at R/2, some packets are retransmissions but asymptotic goodput is still R/2 (why?)



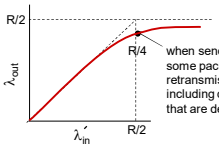
Assumption: On average out of 0.5R units sent, 0.166R bytes/sec are retrans and 0.333R are original.

89

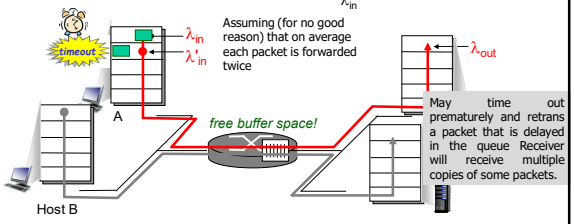
Causes/costs of congestion: scenario 2

Realistic: duplicates

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



when sending at R/2, some packets are retransmissions including duplicated that are delivered!



Assuming (for no good reason) that on average each packet is forwarded twice

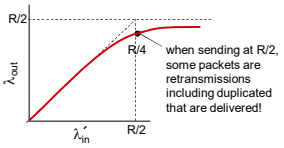
May time out prematurely and retrans a packet that is delayed in the queue Receiver will receive multiple copies of some packets.

90

Causes/costs of congestion: scenario 2

Realistic: duplicates

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



"costs" of congestion:

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

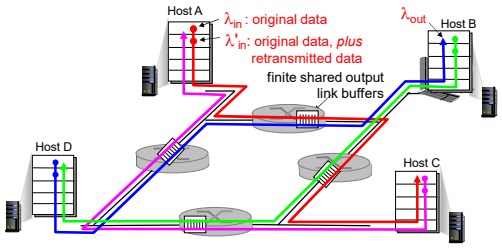
91

Causes/costs of congestion: scenario 3

- four senders
- multihop paths
- timeout/retransmit

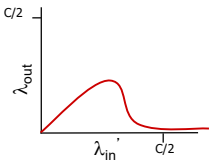
Q: what happens as λ_{in} and λ_{in}' increase ?

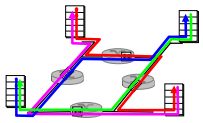
A: as red λ_{in}' increases, all arriving blue pkts at upper queue are dropped, blue throughput → 0



92

Causes/costs of congestion: scenario 3





another "cost" of congestion:

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

93

Chapter 3 outline

3.1 transport-layer services
3.2 multiplexing and demultiplexing
3.3 connectionless transport: UDP
3.4 principles of reliable data transfer
3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control
3.7 TCP congestion control

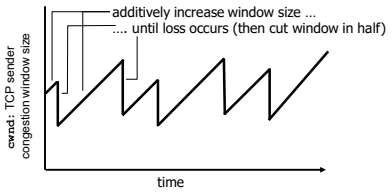
94

rwnd and cwnd controls size of sliding window. And the minimal one of them decide will decides the size.

TCP congestion control: additive increase multiplicative decrease

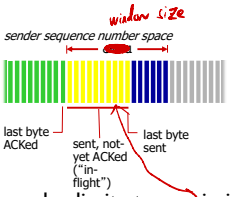
- *approach:* sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase `cwnd (congestion window)` by 1 MSS every RTT until loss detected
  - *multiplicative decrease:* cut `cwnd` in half after loss

AIMD saw tooth behavior: probing for bandwidth



95

TCP Congestion Control: details



- sender limits transmission:

`LastByteSent-LastByteAcked` ≤ `cwnd`

- `cwnd` is dynamic, function of perceived network congestion

TCP sending rate:

- *roughly:* send cwnd bytes, wait RTT for ACKS, then send more bytes

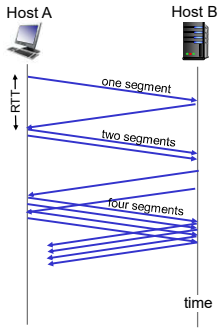
$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

`LastByteSent-LastByteAcked` ≤ `min {cwnd, rwnd}`

96

TCP Slow Start

- when connection begins, increase rate exponentially until first loss event:
  - initially `cwnd` = 1 MSS
  - double `cwnd` every RTT
  - done by incrementing `cwnd` for every ACK received
- *summary:* initial rate is slow but ramps up exponentially fast



97

TCP: detecting, reacting to loss

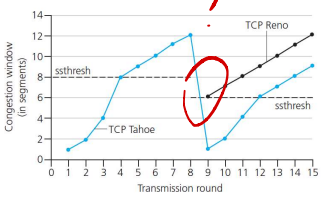
- loss indicated by timeout:
  - `cwnd` set to 1 MSS;
  - window then grows exponentially (as in slow start) to threshold, then grows linearly
- loss indicated by 3 duplicate ACKs: TCP RENO
  - dup ACKs indicate network capable of delivering some segments
  - `cwnd` is cut in half window then grows linearly
- TCP Tahoe always sets `cwnd` to 1 (timeout or 3 duplicate acks)

98

TCP: switching from slow start to CA (Congestion Avoidance)

Q: when should the exponential increase switch to linear?

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

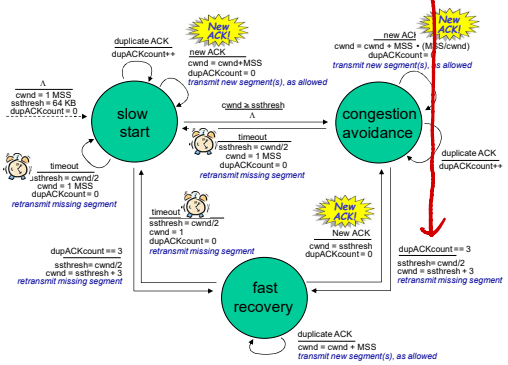


Implementation:

- variable `ssthresh`
- on loss event, `ssthresh` *(slow start threshold)* is set to 1/2 of `cwnd` just before loss event
- Default ssthresh: 65536 bytes
- When TCP performs a fast restransmit (and did not timeout), then set SSThresh = CWS/2 and set CWS = SSThresh + 3 * MSS

99

Summary: TCP Congestion Control

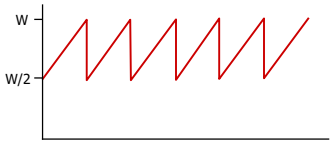


100

TCP throughput

- avg. TCP thruput as function of window size, RTT?
  - ignore slow start, assume always data to send
- W: window size (measured in bytes) where loss occurs
  - avg. window size (# in-flight bytes) is ¾ W
  - avg. thruput is 3/4W per RTT

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



101

TCP Futures: TCP over "long, fat pipes"

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

$$\text{TCP throughput} = \frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

➜ to achieve 10 Gbps throughput, need a loss rate of $L = 2 \cdot 10^{-10}$ – *a very small loss rate!*

- new versions of TCP for high-speed

102

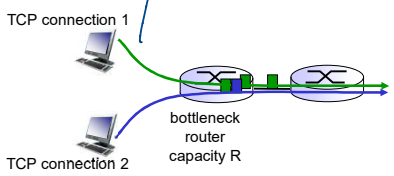
03/22/2022 color :

congestion control is defined by what?

Review questions

TCP Fairness

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

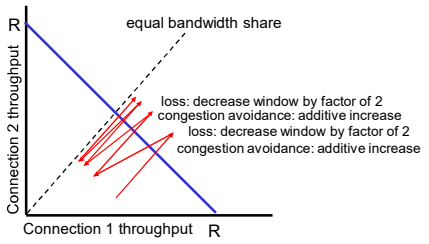


103

Why is TCP fair?

two competing sessions:

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



104

Fairness (more)

Fairness and UDP

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

Fairness, parallel TCP connections

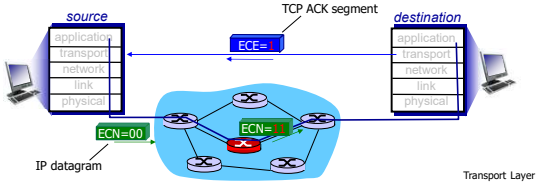
- application can open multiple parallel connections between two hosts
- web browsers do this
- e.g., link of rate R with 9 existing connections:
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2

105

Explicit Congestion Notification (ECN)

network-assisted congestion control:

- two bits in IP header (ToS field, type of service) marked *by network router* to indicate congestion
- congestion indication carried to receiving host
- receiver (seeing congestion indication in IP datagram)) sets ECE bit on receiver-to-sender ACK segment to notify sender of congestion



106

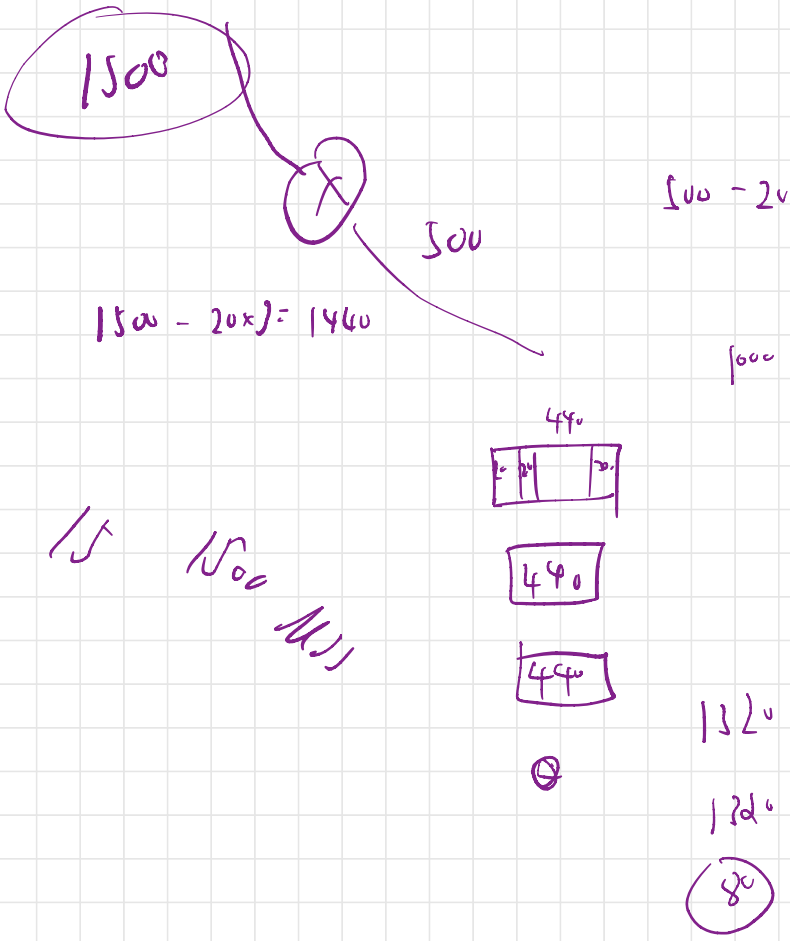
Chapter 3: summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation, implementation in the Internet
  - UDP
  - TCP

next:

- leaving the network "edge" (application, transport layers)
- into the network "core"
- two network layer chapters:
  - data plane
  - control plane

107


1500
X
500
1500 - 20x = 1460
500 - 20x = 440
1000
440
20
30
490
440
1320
1320
80

Chapter 4

Chapter 4: outline

4.1 Overview of Network layer
- data plane
- control plane

4.2 What's inside a router

4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

4.4 Generalized Forward and SDN
- match
- action
- OpenFlow examples of match-plus-action in action

1

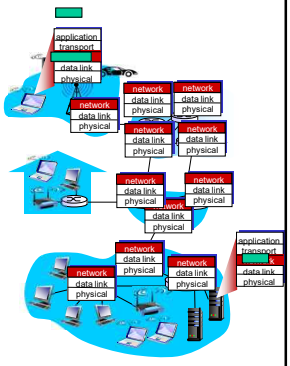
Chapter 4: network layer

chapter goals:
- understand principles behind network layer services, focusing on data plane:
  - network layer service models
  - forwarding versus routing
  - how a router works
  - generalized forwarding
- instantiation, implementation in the Internet

2

Network layer

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



3

Two key network-layer functions

network-layer functions:
- *forwarding:* move packets from router's input to appropriate router output
- *routing:* determine route taken by packets from source to destination
  - *routing algorithms*

analogy: taking a trip
- *forwarding:* process of getting through single interchange
- *routing:* process of planning trip from source to destination

4

Network layer: data plane, control plane

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

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

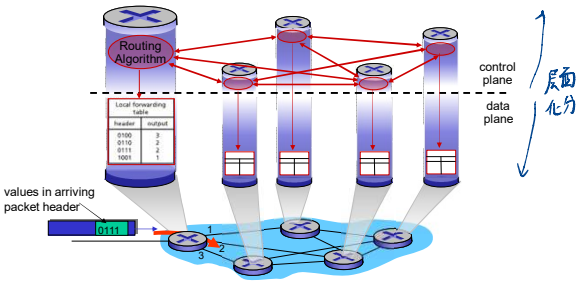
5

forwarding

routing

Per-router control plane

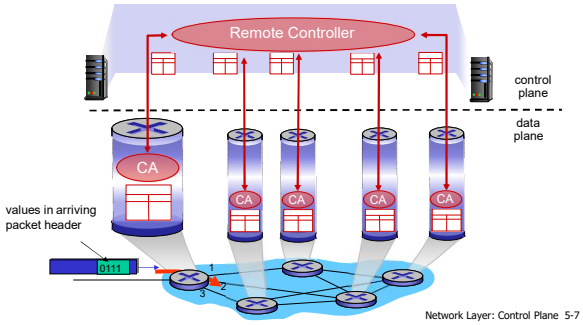
Individual routing algorithm components *in each and every router* interact in the control plane



6

Logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs)



7

Network service model

Q: What *service model* for "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

8

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

9

Chapter 4: outline

4.1 Overview of Network layer
- data plane
- control plane

4.2 What's inside a router

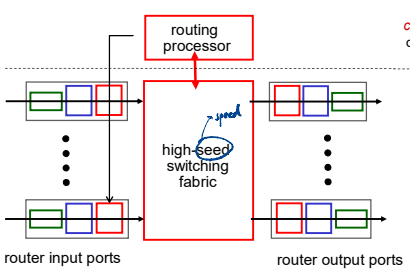
4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

4.4 Generalized Forward and SDN
- match
- action
- OpenFlow examples of match-plus-action in action

10

Router architecture overview

- high-level view of generic router architecture:

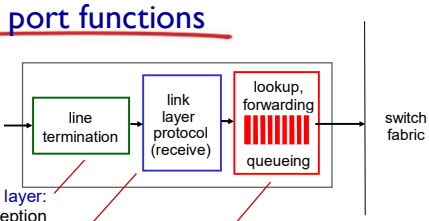


routing, management control plane (software) operates in millisecond time frame

forwarding data plane (hardware) operttes in nanosecond timeframe

11

Input port functions



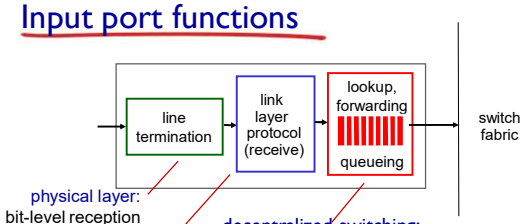
physical layer: bit-level reception

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

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

12

Input port functions



physical layer: bit-level reception

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

decentralized switching:

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

13

Destination-based forwarding

forwarding table

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

Q: but what happens if ranges don't divide up so nicely?

14

Longest prefix matching

longest prefix matching

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

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

examples:

DA: 11001000 00010111 00010110 10100001 which interface?

DA: 11001000 00010111 00011000 10101010 which interface?

15

Longest prefix matching

- we'll see *why* longest prefix matching is used shortly, when we study addressing
- longest prefix matching: often performed using ternary content addressable memories (TCAMs)
  - *content addressable:* present address to TCAM: retrieve address in one clock cycle, regardless of table size
  - Cisco Catalyst: can up ~1M routing table entries in TCAM

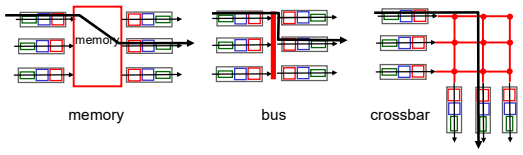
16

Router must has more than two interfaces

03/24/2022 starts:

Switching fabrics

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

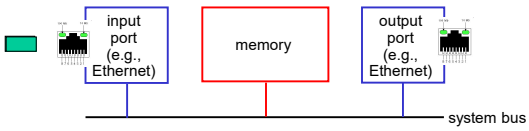


17

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)

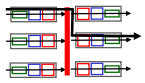


Cisco's Catalyst 8500 series switches [Cisco 8500 2012]

18

Switching via a bus

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

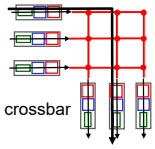


The Cisco 5600
[Cisco Switches 2012]

19

Switching via interconnection network

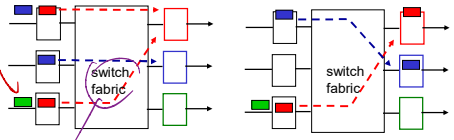
- overcome bus bandwidth limitations
- banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor
- advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco 12000: switches 60 Gbps through the interconnection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



20

Input port queuing

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



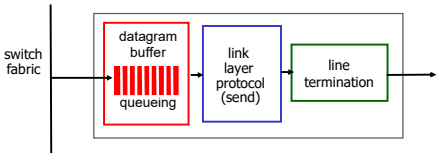
output port contention: only one red datagram can be transferred. *lower red packet is blocked*

one packet time later: green packet experiences HOL blocking

21

Output ports

This slide in HUGELY important!



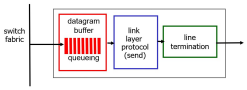
- *buffering* required from fabric faster rate

  Datagram (packets) can be lost due to congestion, lack of buffers

- *scheduling* datagrams

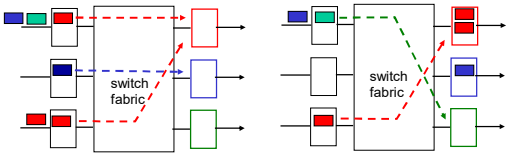
  Priority scheduling – who gets best performance, network neutrality

22



- *buffering* required when datagrams arrive from fabric faster than the transmission rate
- *scheduling discipline* chooses among queued datagrams for transmission

Output port queueing



at *t*, packets more from input to output

one packet time later

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

23

How much buffering?

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

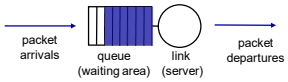
$$\frac{RTT \cdot C}{\sqrt{N}}$$

24

The data may get higher priority: government, routing info data.

Scheduling mechanisms

- *scheduling:* choose next packet to send on link
- *FIFO (first in first out) scheduling:* send in order of arrival to queue
  - real-world example?
  - *discard policy:* if packet arrives to full queue: who to discard?
    - *tail drop:* drop arriving packet
    - *priority:* drop/remove on priority basis
    - *random:* drop/remove randomly

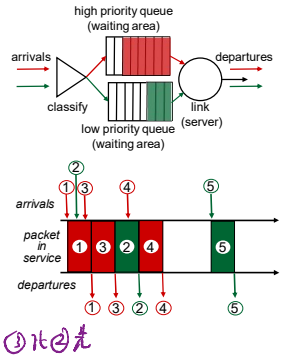


25

Scheduling policies: priority

priority scheduling: send highest priority queued packet

- multiple *classes*, with different priorities
  - class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc.
  - real world example?

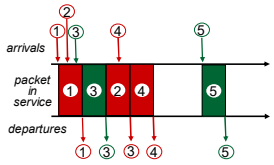


26

Scheduling policies: still more

Round Robin (RR) scheduling:

- multiple classes
- cyclically scan class queues, sending one complete packet from each class (if available)
- real world example?

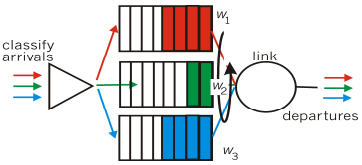


27

Scheduling policies: still more

Weighted Fair Queuing (WFQ):

- generalized Round Robin
- each class gets weighted amount of service in each cycle
- real-world example?



28

Chapter 4: outline

4.1 Overview of Network layer
- data plane
- control plane

4.2 What's inside a router

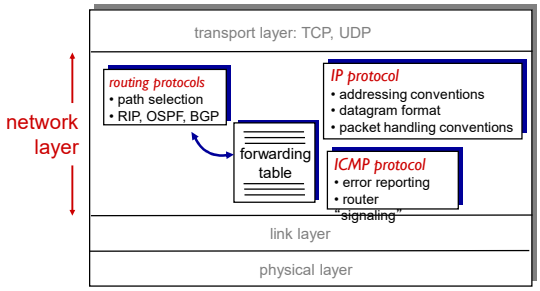
4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

4.4 Generalized Forward and SDN
- match
- action
- OpenFlow examples of match-plus-action in action

29

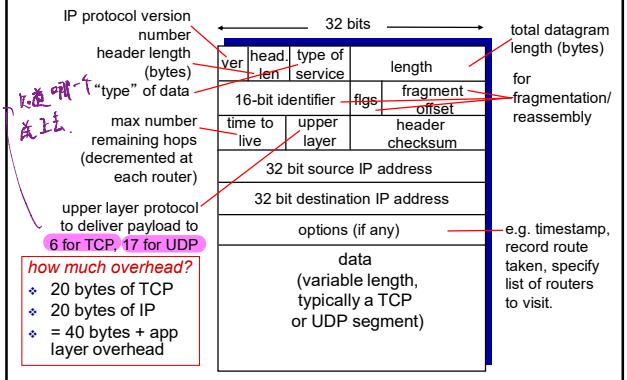
The Internet network layer

host, router network layer functions:



30

IP datagram format

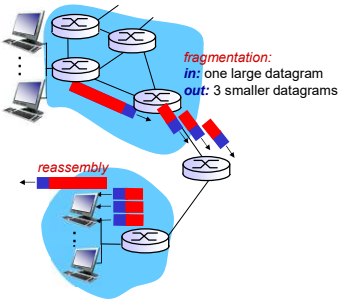


how much overhead?
- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead

31

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

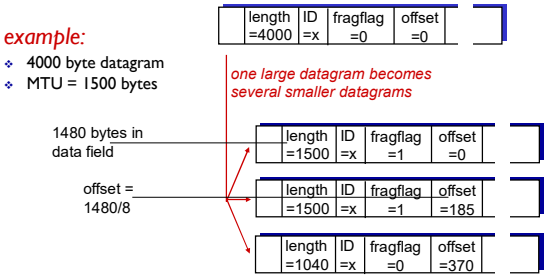


32

03/29 / 2022 : color : start from ↓

IP fragmentation, reassembly

example:
- 4000 byte datagram
- MTU = 1500 bytes



33

Chapter 4: outline

4.1 Overview of Network layer
- data plane
- control plane

4.2 What's inside a router

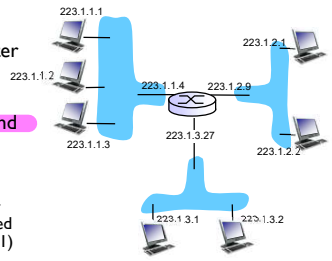
4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

4.4 Generalized Forward and SDN
- match
- action
- OpenFlow examples of match-plus-action in action

34

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 or two interfaces (e.g., wired Ethernet, wireless 802.11)
- *IP addresses associated with each interface*



223.1.1.1 = 11011111 00000001 00000001 00000001

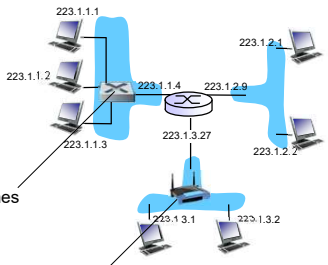
223 1 1 1

35

IP addressing: introduction

Q: how are interfaces actually connected?

A: we'll learn about that in chapter 5, 6.



A: wired Ethernet interfaces connected by Ethernet switches

For now: don't need to worry about how one interface is connected to another (with no intervening router)

A: wireless WiFi interfaces connected by WiFi base station

36

03/24/2022

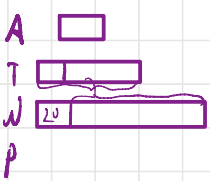
- Why we need queue at input port.
- Because we need queue to buffer the incoming packet if switch speed is slower than incoming packet rate. And if that happens without queue, then packet will loss

- Why we need queue at output port?
- If output port packet rate is faster than switch output going port speed, then packet may get lost without queue.

Higher priority packets: the packet is router information and generated by router, not users. Government packet for security from FBI, NSL.

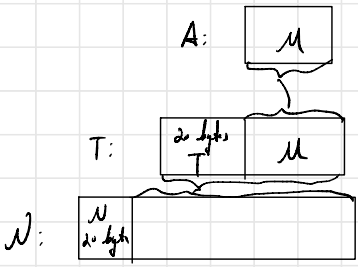
☆ The priority queue will be on the exam. Carefully review.

Why there is no packet length in transport layer?
Because Network layer has the packet length, and use it minus 20 bytes get the payload, which is the length of TCP packet length.



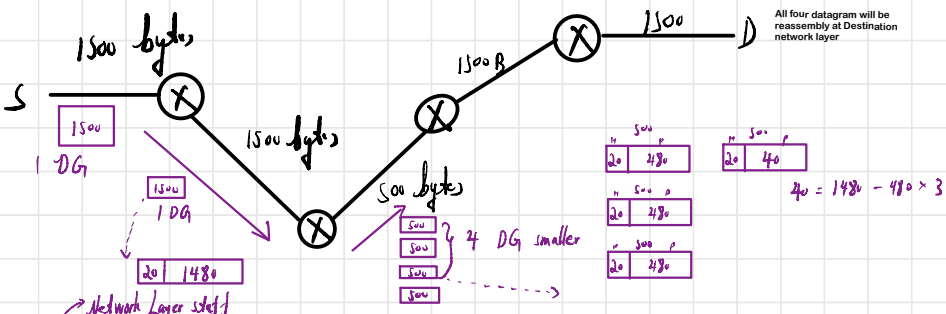
Length of N packet − 20 bytes = Segment Length

03/24/2022



★：方便
Should I expected fragmentation at ending system?
Chop up is never be the job of network layer, it is transport layer work. but it is capable to do it. Network layer job is host to host delivery, and we should give the job to transport layer. And much pressure will be on end system.

Fragmentation and reassembly: The below is network layer links

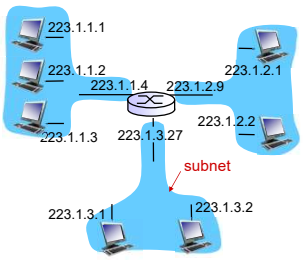


Typical MTU size: 1500 bytes, best guess not a hard number. Possible network layer packet has to be chopped up. (同时是整条路上最小的取值)

Offset calculation refer to slides.

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*



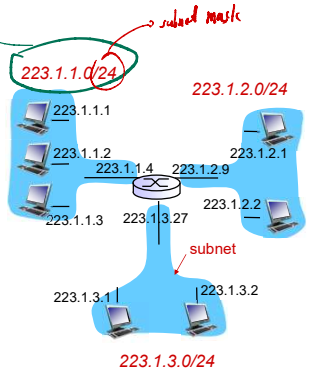
network consisting of 3 subnets

37

Subnets

recipe

- to determine the subnets, detach each interface from its host or router, creating islands of isolated networks
- each isolated network is called a *subnet*

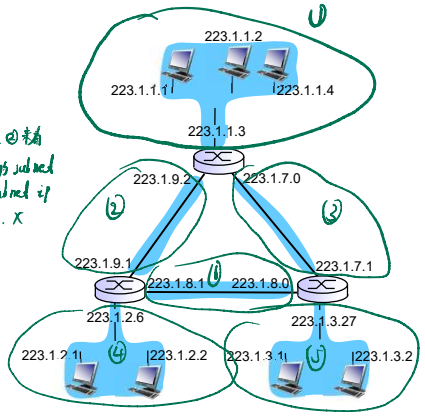


subnet mask: /24

38

Subnets

how many?

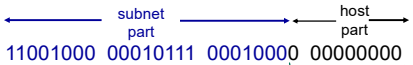


39

IP addressing: CIDR

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



11001000 00010111 00010000 00000000

200.23.16.0/23

40

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 as server
  - "plug-and-play"

41

DHCP: Dynamic Host Configuration Protocol

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

- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/"on")
- support for mobile users who want to join network (more shortly)

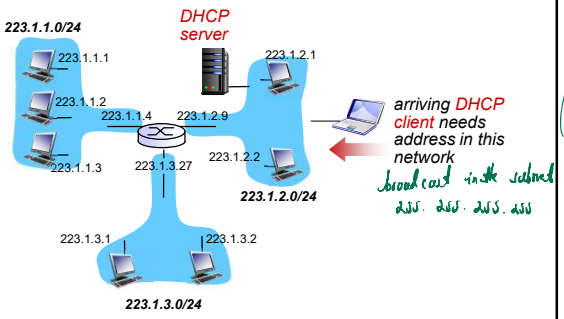
DHCP overview:

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

42

Scenario: Lease problem. DHCP gives you a lease to use the IP address. But it may reject you if the ip has been given to otheres

DHCP client-server scenario

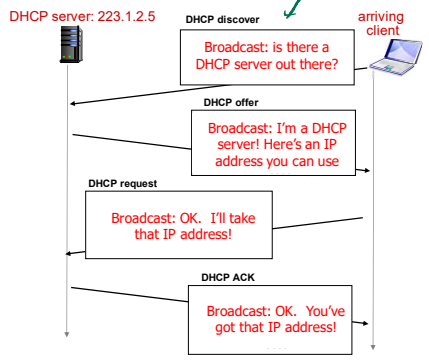


Network Layer: Data Plane 4-43

43

If you have a lease you don't have to go through first 2 steps

DHCP client-server scenario



Network Layer: Data Plane 4-44

44

Cache 数量和gateway相同

DHCP: more than IP addresses

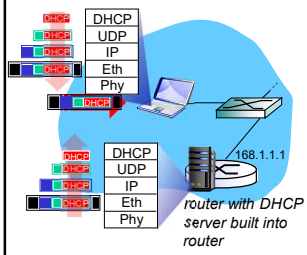
DHCP can return more than just allocated IP address on subnet:

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

Network Layer: Data Plane 4-45

45

DHCP: example

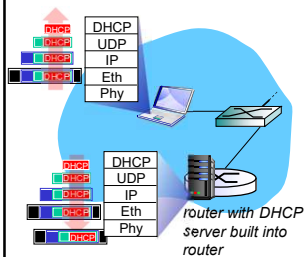


- connecting laptop needs its IP address, addr of first-hop router, addr of DNS server: use DHCP
- DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.1 Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demuxed to IP demuxed, UDP demuxed to DHCP

Network Layer: Data Plane 4-46

46

DHCP: example



- DCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation of DHCP server, frame forwarded to client, demuxing up to DHCP at client
- client now knows its IP address, name and IP address of DSN server, IP address of its first-hop router

Network Layer: Data Plane 4-47

47

DHCP: Wireshark output (home LAN)

request

```
Message type: Boot Request (1)
Hardware type: Ethernet
Hardware address length: 6
Hops: 0
Transaction ID: 0x6b3a11b7
Seconds elapsed: 0
Bootp flags: 0x0000 (Unicast)
Client IP address: 0.0.0.0 (0.0.0.0)
Your (client) IP address: 0.0.0.0 (0.0.0.0)
Next server IP address: 0.0.0.0 (0.0.0.0)
Relay agent IP address: 0.0.0.0 (0.0.0.0)
Client MAC address: Wistron_23:68:8a (00:16:d3:23:68:8a)
Server host name not given
Boot file name not given
Magic cookie: (OK)
Option: (t=53,l=1) DHCP Message Type = DHCP Request
Option: (61) Client identifier
    Length: 7; Value: 010016D323688A;
    Hardware type: Ethernet
    Client MAC address: Wistron_23:68:8a (00:16:d3:23:68:8a)
Option: (t=50,l=4) Requested IP Address = 192.168.1.101
Option: (t=12,l=5) Host Name = "nomad"
Option: (55) Parameter Request List
    Length: 11; Value: 010F03062C2E2F1F21F92B
    1 = Subnet Mask; 15 = Domain Name
    3 = Router; 6 = Domain Name Server
    44 = NetBIOS over TCP/IP Name Server
    .....
```

reply

```
Message type: Boot Reply (2)
Hardware type: Ethernet
Hardware address length: 6
Hops: 0
Transaction ID: 0x6b3a11b7
Seconds elapsed: 0
Bootp flags: 0x0000 (Unicast)
Client IP address: 192.168.1.101 (192.168.1.101)
Your (client) IP address: 0.0.0.0 (0.0.0.0)
Next server IP address: 192.168.1.1 (192.168.1.1)
Relay agent IP address: 0.0.0.0 (0.0.0.0)
Client MAC address: Wistron_23:68:8a (00:16:d3:23:68:8a)
Server host name not given
Boot file name not given
Magic cookie: (OK)
Option: (t=53,l=1) DHCP Message Type = DHCP ACK
Option: (t=54,l=4) Server Identifier = 192.168.1.1
Option: (t=1,l=4) Subnet Mask = 255.255.255.0
Option: (t=3,l=4) Router = 192.168.1.1
Option: (6) Domain Name Server
    Length: 12; Value: 445747E2445749F244574092;
    IP Address: 68.87.71.226;
    IP Address: 68.87.73.242;
    IP Address: 68.87.64.146
Option: (t=15,l=20) Domain Name = "hsd1.ma.comcast.net."
```

Network Layer: Data Plane 4-48

48

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

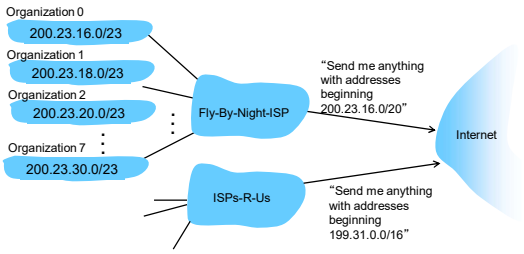
eg: How many IP ISP bought? 2^{12}

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

49

Hierarchical addressing: route aggregation

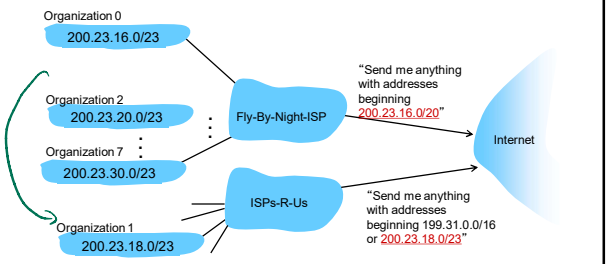
hierarchical addressing allows efficient advertisement of routing information:



50

Hierarchical addressing: more specific routes

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



51

IP addressing: the last word...

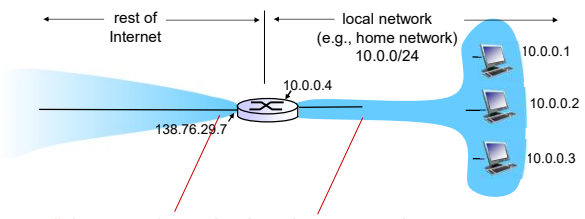
Q: how does an ISP get block of addresses?

A: ICANN: Internet Corporation for Assigned Names and Numbers http://www.icann.org/

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

52

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)

53

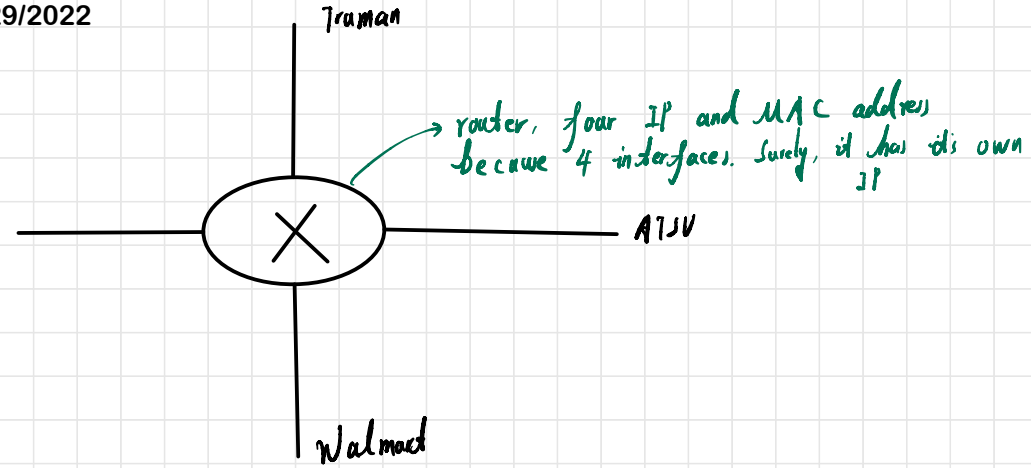
NAT: network address translation

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)

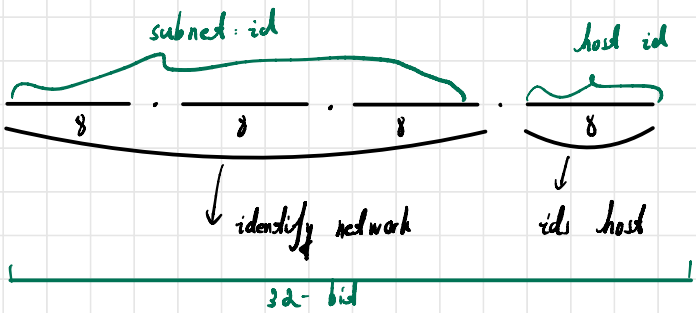
54

03/29/2022

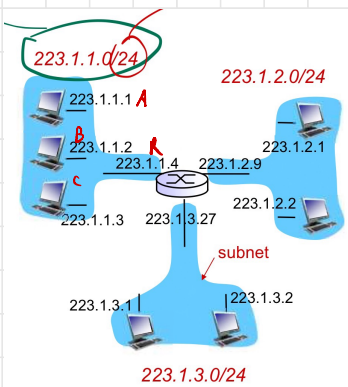


Subnet mask:

IP address:



eg: 192. 6. 223. 9 / 24, 前3段是IP of subnet
, 后一段(8 bit)是 subnet mask



A can directly talk to B, C, R, because they have the same subnet IP

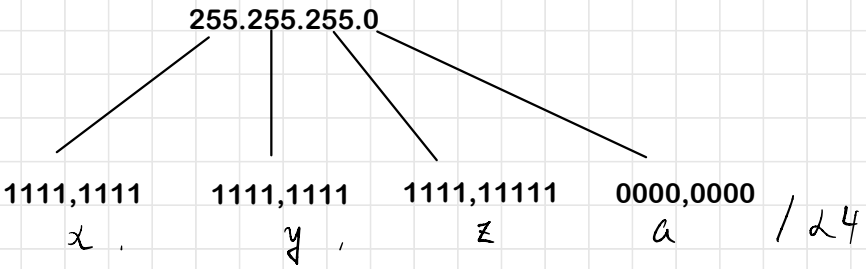
☆提

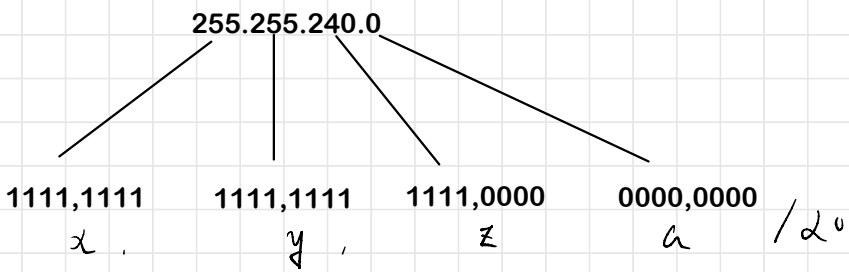
We need IP address to analysis if two pc are under same subnet, but only use MAC address to communicate.

When it sends data by MAC address it, all node in the Wi-Fi receives it but only the pc with the MAC address accept it.

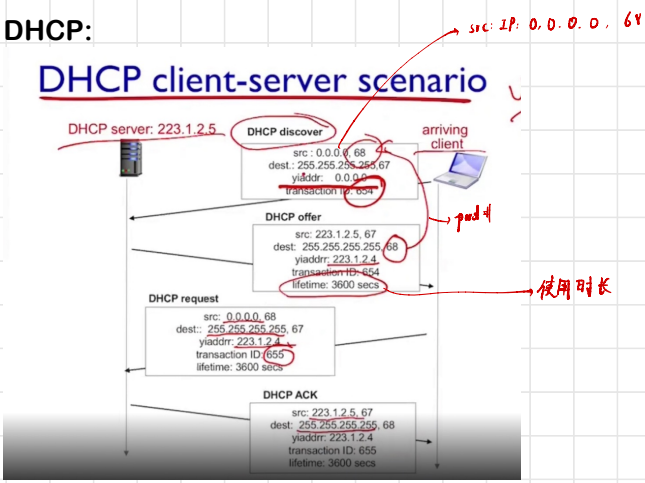
03/29/2022

CIDR: real example for subnet mask



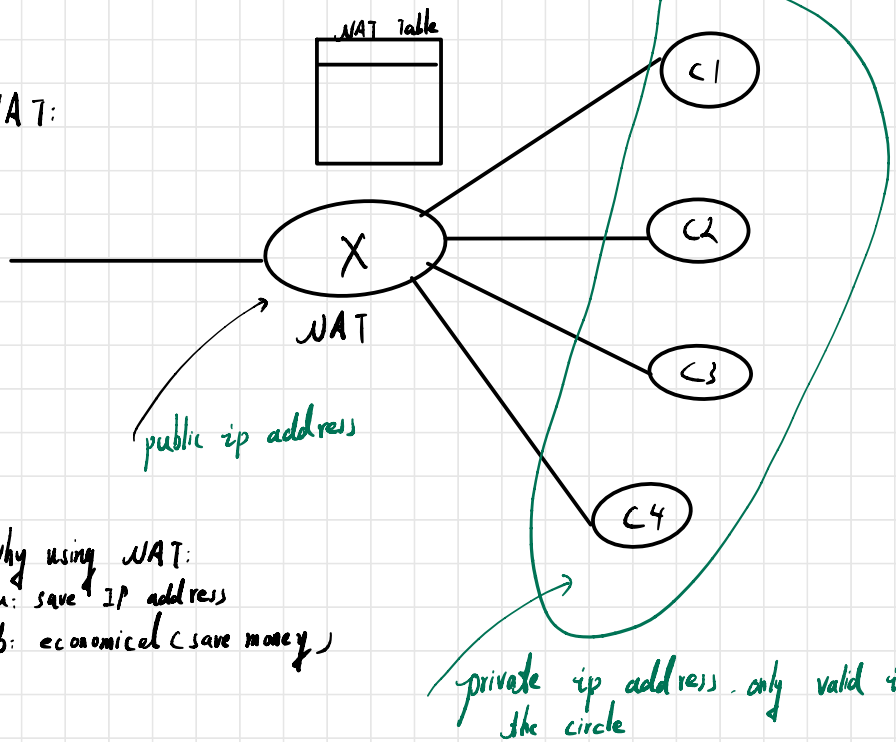


DHCP:



03/29/2022

NAT:



Q: Why using NAT:

a: save IP address

b: economical (save money)

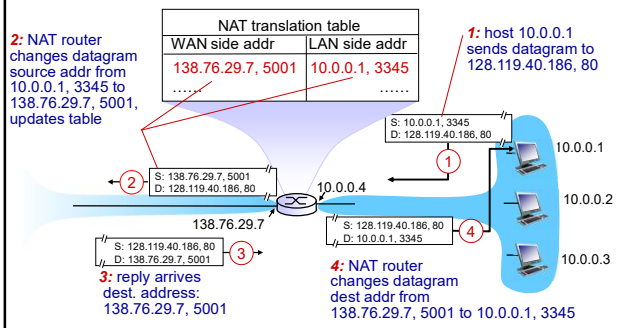
NAT: network address translation

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

55

NAT: network address translation



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

56

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 (CP)
  - address shortage should be solved by IPv6
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, e.g., P2P applications
  - NAT traversal: what if client wants to connect to server behind NAT?

57

NAT is working up to T

T
P
N

part of wording

Chapter 4: outline

4.1 Overview of Network layer
- data plane
- control plane

4.2 What's inside a router

4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

4.4 Generalized Forward and SDN
- match
- action
- OpenFlow examples of match-plus-action in action

58

IPv6: motivation

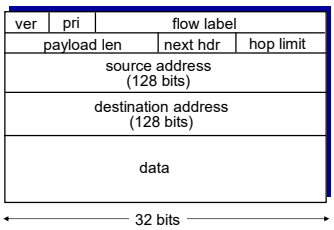
- *initial motivation:* 32-bit address space soon to be completely allocated.
- additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

IPv6 datagram format:
- fixed-length 40 byte header
- no fragmentation allowed

59

IPv6 datagram format

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



60

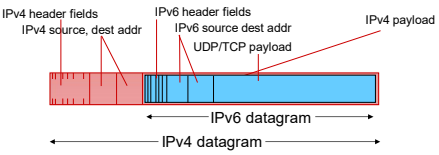
Other changes from IPv4

- *checksum:* removed entirely to reduce processing time at each hop
- *options:* allowed, but outside of header, indicated by "Next Header" field
- *ICMPv6:* new version of ICMP
  - additional message types, e.g. "Packet Too Big"
  - multicast group management functions

61

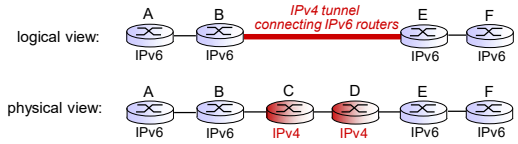
Transition from IPv4 to IPv6

- not all routers can be upgraded simultaneously
  - no "flag days"
  - how will network operate with mixed IPv4 and IPv6 routers?
- *tunneling:* IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers



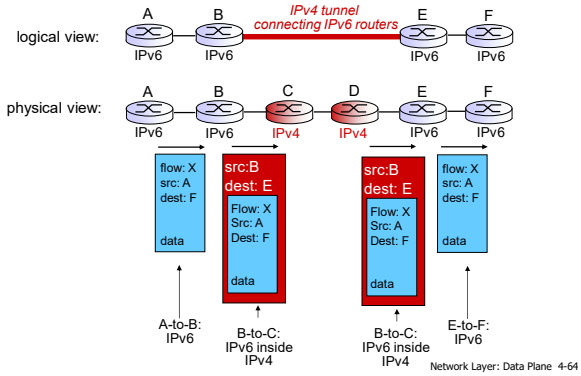
62

Tunneling



63

Tunneling



64

IPv6: adoption

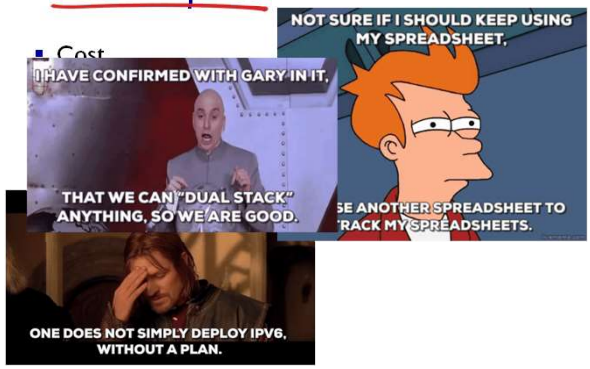
- Google: 8% of clients access services via IPv6
- NIST: 1/3 of all US government domains are IPv6 capable

- *Long (long!) time for deployment, use*
  - 20 years and counting!
  - think of application-level changes in last 20 years: WWW, Facebook, streaming media, Skype, …
  - *Why?*

65

IPv6: adoption

- Cost



66

Chapter 4: outline

4.1 Overview of Network layer
- data plane
- control plane

4.2 What's inside a router

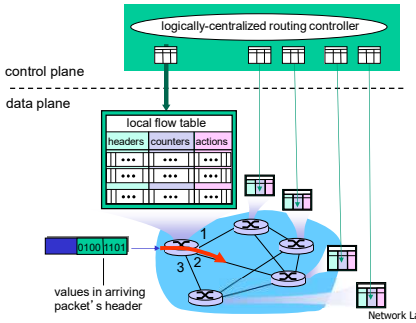
4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

4.4 Generalized Forward and SDN
- match
- action
- OpenFlow examples of match-plus-action in action

67

Generalized Forwarding and SDN

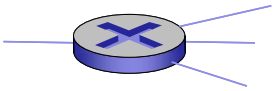
Each router contains a *flow table* that is computed and distributed by a *logically centralized* routing controller



68

OpenFlow data plane abstraction

- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
  - *Pattern:* match values in packet header fields
  - *Actions: for matched packet:* drop, forward, modify, matched packet or send matched packet to controller
  - *Priority*: disambiguate overlapping patterns
  - *Counters:* #bytes and #packets

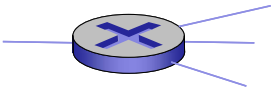


Flow table in a router (computed and distributed by controller) define router's match+action rules

69

OpenFlow data plane abstraction

- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
  - *Pattern:* match values in packet header fields
  - *Actions: for matched packet:* drop, forward, modify, matched packet or send matched packet to controller
  - *Priority*: disambiguate overlapping patterns
  - *Counters:* #bytes and #packets

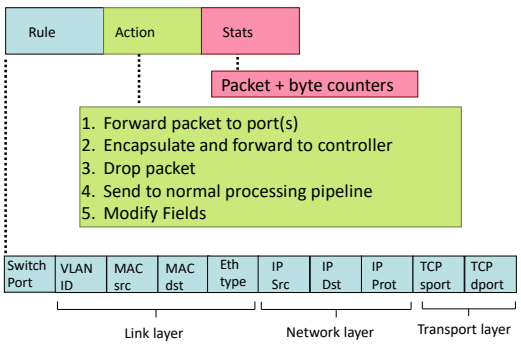


* : wildcard

1. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.*.* → forward(2)
3. src=10.1.2.3, dest=*.*.*.* → send to controller

70

OpenFlow: Flow Table Entries



71

Examples

Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	51.6.0.8	*	*	*	port6

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	*	*	*	22		drop

do not forward (block) all datagrams destined to TCP port 22

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	128.119.1.1	*	*	*	*	drop

do not forward (block) all datagrams sent by host 128.119.1.1

72

Examples

Destination-based layer 2 (switch) forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	22:A7:23: 11:E1:02	*	*	*	*	*	*	*	*	port3

layer 2 frames from MAC address 22:A7:23:11:E1:02 should be forwarded to output port 6

73

OpenFlow abstraction

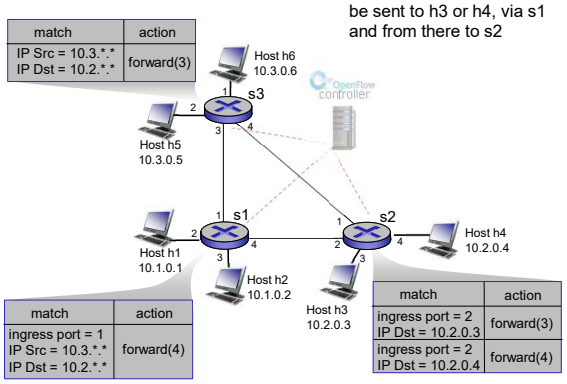
- *match+action:* unifies different kinds of devices

- Router
  - *match:* longest destination IP prefix
  - *action:* forward out a link
- Switch
  - *match:* destination MAC address
  - *action:* forward or flood
- Firewall
  - *match*: IP addresses and TCP/UDP port numbers
  - *action:* permit or deny
- NAT
  - *match:* IP address and port
  - *action:* rewrite address and port

74

OpenFlow example

Example: datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2



75

Chapter 4: *done!*

4.1 Overview of Network layer: data plane and control plane

4.2 What's inside a router

4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- NAT
- IPv6

4.4 Generalized Forward and SDN
- match plus action
- OpenFlow example

Question: how do forwarding tables (destination-based forwarding) or flow tables (generalized forwarding) computed?

Answer: by the control plane (next chapter)

76

Chapter-5

4/5/2022 starts here Continue network

Chapter 5: network layer control plane

chapter goals: understand principles behind network control plane
- traditional routing algorithms
- SDN controlllers
- Internet Control Message Protocol
- network management

and their instantiation, implementation in the Internet:
- OSPF, BGP, OpenFlow, ODL and ONOS controllers, ICMP, SNMP

1

Chapter 5: outline

5.1 introduction
5.2 routing protocols
- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF
5.4 routing among the ISPs: BGP
5.5 The SDN control plane
5.6 ICMP: The Internet Control Message Protocol
5.7 Network management and SNMP

2

Network-layer functions

Recall: two network-layer functions:
- *forwarding:* move packets from router's input to appropriate router output — *data plane*
- *routing:* determine route taken by packets from source to destination — *control plane*

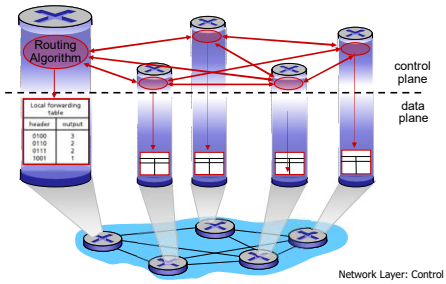
Two approaches to structuring network control plane:
- per-router control (traditional)
- logically centralized control (software defined networking)

3

network control planes: { per-router; logically centralized control }

Per-router control plane

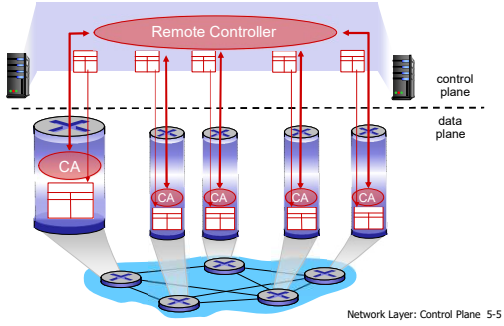
Individual routing algorithm components *in each and every router* interact with each other in control plane to compute forwarding tables



4

Logically centralized control plane or SDN

A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables



5

All the tables will be consistent.

Chapter 5: outline

5.1 introduction
5.2 routing protocols
- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF
5.4 routing among the ISPs: BGP
5.5 The SDN control plane
5.6 ICMP: The Internet Control Message Protocol
5.7 Network management and SNMP

6

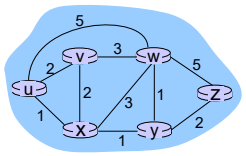
Routing protocols

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

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

7

Graph abstraction of the network



graph: G = (N,E)

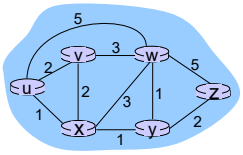
N = set of routers = { u, v, w, x, y, z }

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

aside: graph abstraction is useful in other network contexts, e.g., P2P, where *N* is set of peers and *E* is set of TCP connections

8

Graph abstraction: costs



c(x,x') = cost of link (x,x')
e.g., c(w,z) = 5

cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

cost of path $(x_1, x_2, x_3, \ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

key question: what is the least-cost path between u and z ?
routing algorithm: algorithm that finds that least cost path

9

Routing algorithm classification

Q: global or decentralized information?

global:
- all routers have complete topology, link cost info
- "link state" algorithms

decentralized:
- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- "distance vector" algorithms

Q: static or dynamic?

static:
- routes change slowly over time

dynamic:
- routes change more quickly
  - periodic update
  - in response to link cost changes

10

Chapter 5: outline

5.1 introduction
5.2 routing protocols
- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF
5.4 routing among the ISPs: BGP
5.5 The SDN control plane
5.6 ICMP: The Internet Control Message Protocol
5.7 Network management and SNMP

11

A link-state routing algorithm

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

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

12

Dijsktra's algorithm

```
1  Initialization:
2   N' = {u}
3   for all nodes v
4     if v adjacent to u
5        then D(v) = c(u,v)
6     else D(v) = ∞
7
8  Loop
9    find w not in N' such that D(w) is a minimum
10   add w to N'
11   update D(v) for all v adjacent to w and not in N' :
12      D(v) = min( D(v), D(w) + c(w,v) )
13   /* new cost to v is either old cost to v or known
14    shortest path cost to w plus cost from w to v */
15  until all nodes in N'
```

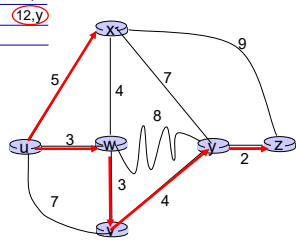

13

Dijkstra's algorithm: example

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

notes:

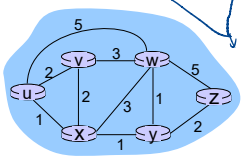
- construct shortest path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)



14

Dijkstra's algorithm: another example

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



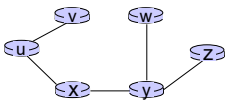
die 的时候随便选一个

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

15

Dijkstra's algorithm: example (2)

resulting shortest-path tree from u:



from u perspective 也就有其他的那就要重新一遍

resulting forwarding table in u:

destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

can y is not directed connecting to u

16

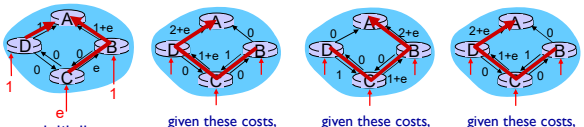
Dijkstra's algorithm, discussion

algorithm complexity: n nodes

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

oscillations possible:

- e.g., support link cost equals amount of carried traffic:



17

based on current path not previous one, and calculate it before updating it.

Chapter 5: outline

5.1 introduction

5.2 routing protocols
- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

18

Distance vector algorithm

Bellman-Ford equation (dynamic programming)

let

$d_x(y)$:= cost of least-cost path from x to y

then

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

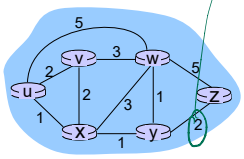
$c(x,v)$: cost to neighbor v

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

min taken over all neighbors v of x

19

Bellman-Ford example



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

B-F equation says:

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

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

20

Distance vector algorithm

- $D_x(y)$ = estimate of least cost from x to y
  - x maintains distance vector $\mathbf{D}_x = [D_x(y): y \in N]$
- node x:
  - knows cost to each neighbor v: c(x,v)
  - maintains its neighbors' distance vectors. For each neighbor v, x maintains $\mathbf{D}_v = [D_v(y): y \in N]$

21

Distance vector algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

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

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

22

Distance vector algorithm

iterative, asynchronous: each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed:

- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

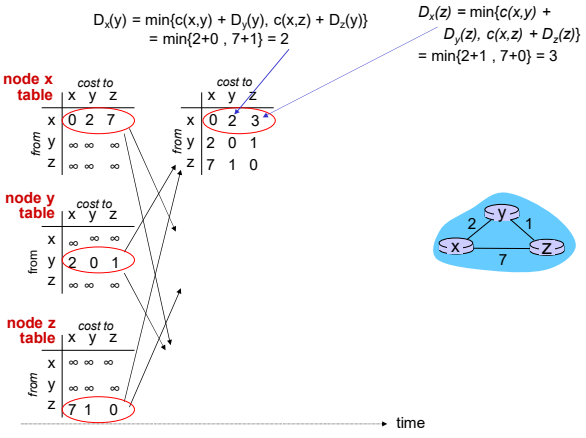
each node:

wait for (change in local link cost or msg from neighbor)

recompute estimates

if DV to any dest has changed, *notify* neighbors

23



24

- Routing is a P2P network.

Logically centralized control: pros: tables are consistent, less routing data transferring in network. Cons: central control is down.

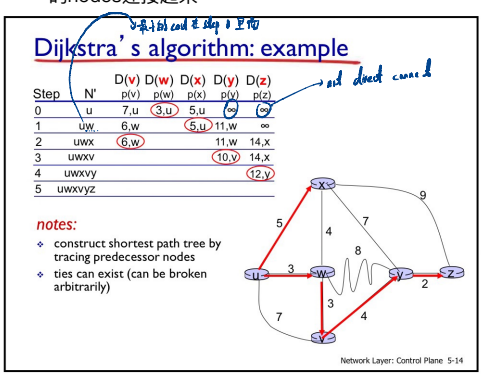
- Routing algorithms
  - Global: linked state, you know the entire topology.
    Good for: small or medium size of network.
    Broadcast for each router causing too much traffic. Router knows all others.
    Good for static network, since no too much traffic
  - Decentralized: distance vectors, you only knows neighbors
    Good for: large size network
    Good for dynamic network.

- Dijkstra's algorithm — Global, linked state

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

ex:

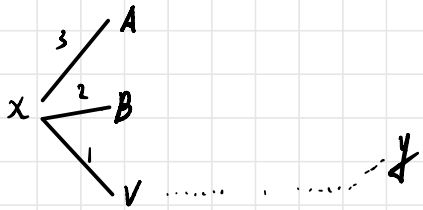
其实他每一次都是在找least cost的link 把所有的nodes连接起来



04/05/2022

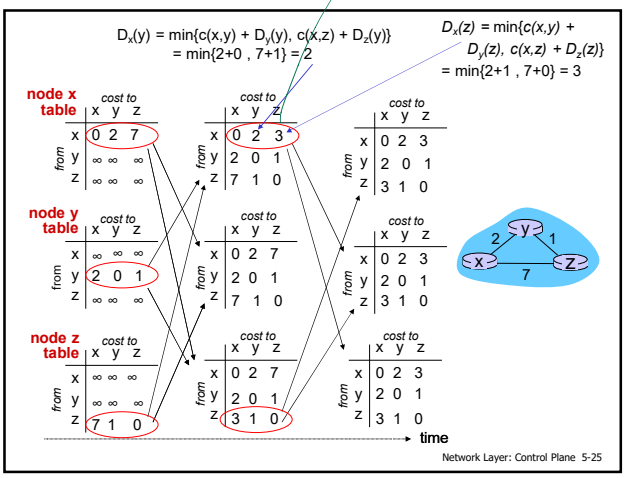
Bellman-Ford

concept:



选中x至v，至于怎么让他到y，那是v node的事情，这也是和Linked state的区别，linked state知道所有的node信息，dan sh

04/07/2022 starts here



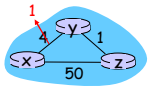
Network Layer: Control Plane 5-25

25

Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- updates routing info, recalculates distance vector
- if DV changes, notify neighbors



"good news travels fast"

t_0 : *y* detects link-cost change, updates its DV, informs its neighbors.

t_1 : *z* receives update from *y*, updates its table, computes new least cost to *x* , sends its neighbors its DV.

t_2 : *y* receives *z*'s update, updates its distance table. *y*'s least costs do *not* change, so *y* does *not* send a message to *z*.

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

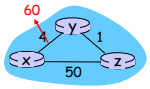
Network Layer: Control Plane 5-26

26

Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- *bad news travels slow* - "count to infinity" problem!
- 44 iterations before algorithm stabilizes: see text



What would have happened if the link cost $c(y,x)$ had changed from 4 to 10,000 and the cost $c(z, x)$ had been 9,999? Because of such scenarios, the problem we have seen is sometimes referred to as the countto-infinity problem.

poisoned reverse:

- If Z routes through Y to get to X :
  - Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- will this completely solve count to infinity problem?

Network Layer: Control Plane 5-27

27

Comparison of LS and DV algorithms

message complexity

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

speed of convergence

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

robustness: what happens if router malfunctions?

LS:

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

DV:

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

in 1997, a malfunctioning router in a small ISP provided national backbone routers with erroneous routing information. This caused other routers to flood the malfunctioning router with traffic and caused large portions of the Internet to become disconnected for up to several hours.

Network Layer: Control Plane 5-28

28

04/12/2022 starts here

Chapter 5: outline

5.1 introduction
5.2 routing protocols
- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF
5.4 routing among the ISPs: BGP
5.5 The SDN control plane
5.6 ICMP: The Internet Control Message Protocol
5.7 Network management and SNMP

Network Layer: Control Plane 5-29

29

Making routing scalable

our routing study thus far - idealized
- all routers identical
- network "flat"

… *not* true in practice

scale: with billions of destinations:
- can't store all destinations in routing tables!
- routing table exchange would swamp links!

administrative autonomy
- internet = network of networks
- each network admin may want to control routing in its own network

Network Layer: Control Plane 5-30

30

linked stats is not good for larger network (quiz question)

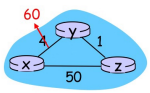
- **Higher frequency update, higher frequency of results sent, more number of broad cast.**

When does router in bellmen ford sends information?
Regular synchronization like each 12 hr sent information to make sure all router has synced information. And a new costs path is found and need to be updated.

Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- *bad news travels slow* - "count to infinity" problem!
- 44 iterations before algorithm stabilizes: see text

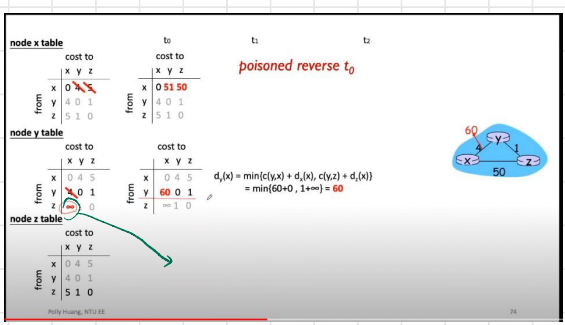


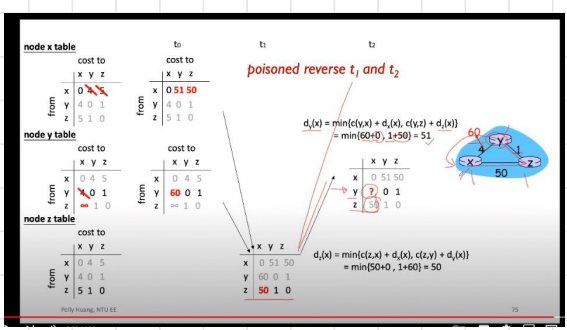
What would have happened if the link cost $c(y,x)$ had changed from 4 to 10,000 and the cost $c(z, x)$ had been 9,999? Because of such scenarios, the problem we have seen is sometimes referred to as the countto-infinity problem.

poisoned reverse:

- If Z routes through Y to get to X :
  - Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- will this completely solve count to infinity problem?

重点：由于在未改变costs 4之前，y知道z到x的least costs是4+1，然后在改变了xy的costs 到60之后，y就认为我需要从z送包是最近的，因为他是5，总比60好，随后他的新路线就是1+ 原来z的least costs是5=6. 问题就在于y不知道的是原来z的least cost路线是经过我自己的，傻逼了。好了这下z自己也要更新table了，一看y给的数据，y到x的least costs是6，那我就把包给y然后这样的话我z到x的least costs path就是6 +1 = 7. 然后又把这个数据告诉了y. 随后他们两个就不停的增加1然后update table。直到发现z到x的least costs有一条路是50，然后就停止了相互增加。





https://www.youtube.com/watch?v=UcTgNZwGmRU

HW: poisoned reverse 并没能解决问题，因为route 还对，更多折不行了.
reread book

Internet approach to scalable routing

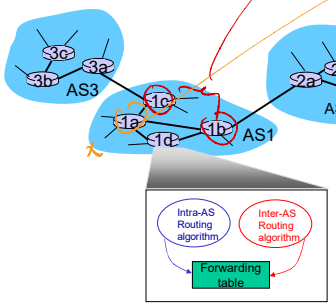
aggregate routers into regions known as "autonomous systems" (AS) (a.k.a. "domains")

intra-AS routing
- routing among hosts, routers in same AS ("network")
- all routers in AS must run *same* intra-domain protocol
- routers in *different* AS can run *different* intra-domain routing protocol
- gateway router: at "edge" of its own AS, has link(s) to router(s) in other AS'es

inter-AS routing
- routing among AS'es
- gateways perform inter-domain routing (as well as intra-domain routing)

31

Interconnected ASes



- forwarding table configured by both intra- and inter-AS routing algorithm
  - intra-AS routing determine entries for destinations within AS
  - inter-AS & intra-AS determine entries for external destinations

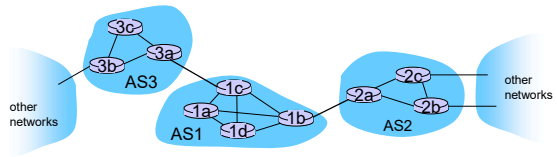
32

Inter-AS tasks

- suppose router in AS1 receives datagram destined outside of AS1:
  - router should forward packet to gateway router, but which one?

AS1 must:
1. learn which dests are reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1

job of inter-AS routing!



33

Intra-AS Routing

- also known as *interior gateway protocols (IGP)*
- most common intra-AS routing protocols:
  - RIP: Routing Information Protocol
  - OSPF: Open Shortest Path First (IS-IS protocol essentially same as OSPF) Intermediate System to Intermediate System
  - IGRP: Interior Gateway Routing Protocol (Cisco proprietary for decades, until 2016)

34

OSPF (Open Shortest Path First)

- "open": publicly available
- uses link-state algorithm
  - link state packet dissemination
  - topology map at each node
  - route computation using Dijkstra's algorithm
- router floods OSPF link-state advertisements to all other routers in *entire* AS
  - carried in OSPF messages directly over IP (rather than TCP or UDP
  - link state: for each attached link
- *IS-IS routing* protocol: nearly identical to OSPF

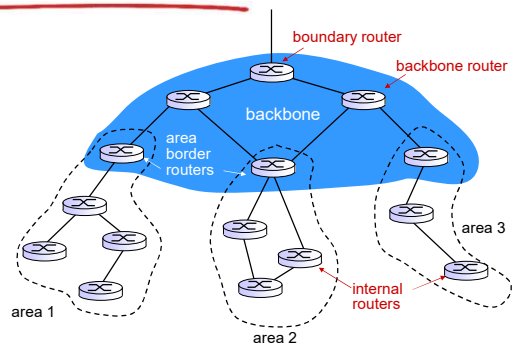
35

OSPF "advanced" features

- *security*: all OSPF messages authenticated (to prevent malicious intrusion)
- multiple same-cost paths allowed (only one path in RIP)
- for each link, multiple cost metrics for different TOS (e.g., satellite link cost set low for best effort ToS; high for real-time ToS)
- integrated uni- and multi-cast support:
  - Multicast OSPF (MOSPF) uses same topology data base as OSPF
- hierarchical OSPF in large domains.

36

Hierarchical OSPF



37

Hierarchical OSPF

- *two-level hierarchy:* local area, backbone.
  - link-state advertisements only in area
  - each nodes has detailed area topology; only know direction (shortest path) to nets in other areas.
- *area border routers:* "summarize" distances to nets in own area, advertise to other Area Border routers.
- *backbone routers:* run OSPF routing limited to backbone.
- *boundary routers:* connect to other AS' es.

38

Chapter 5: outline

5.1 introduction
5.2 routing protocols
- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF
5.4 routing among the ISPs: BGP
5.5 The SDN control plane
5.6 ICMP: The Internet Control Message Protocol
5.7 Network management and SNMP

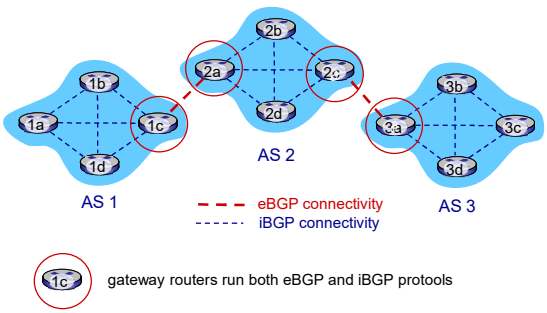
39

Internet inter-AS routing: BGP

- BGP (Border Gateway Protocol): *the* de facto inter-domain routing protocol
  - "glue that holds the Internet together"
- BGP provides each AS a means to:
  - eBGP: obtain subnet reachability information from neighboring ASes.
  - iBGP: propagate reachability information to all AS-internal routers.
  - determine "good" routes to other networks based on reachability information and *policy*
- allows subnet to advertise its existence to rest of Internet: *"I am here"*

40

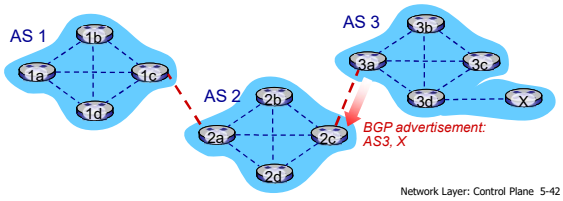
eBGP, iBGP connections



41

BGP basics

- BGP session: two BGP routers ("peers") exchange BGP messages over semi-permanent TCP connection:
  - advertising *paths* to different destination network prefixes (BGP is a "path vector" protocol)
- when AS3 gateway router 3a advertises path AS3,X to AS2 gateway router 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X



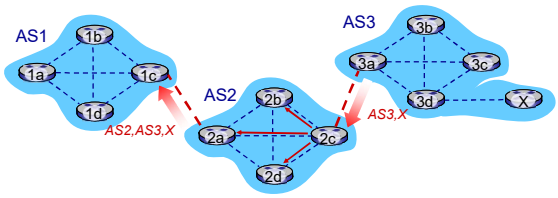
42

Path attributes and BGP routes

- advertised prefix includes BGP attributes
  - prefix + attributes = "route"
- two important attributes:
  - AS-PATH: list of ASes through which prefix advertisement has passed
  - NEXT-HOP: indicates specific internal-AS router to next-hop AS
- *Policy-based routing:*
  - gateway receiving route advertisement uses *import policy* to accept/decline path (e.g., never route through AS Y).
  - AS policy also determines whether to *advertise* path to other other neighboring ASes

43

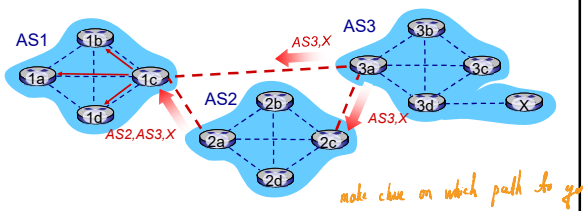
BGP path advertisement



- AS2 router 2c receives path advertisement AS3,X (via eBGP) from AS3 router 3a
- Based on AS2 policy, AS2 router 2c accepts path AS3,X, propagates (via iBGP) to all AS2 routers
- Based on AS2 policy, AS2 router 2a advertises (via eBGP) path AS2, AS3, X to AS1 router 1c

44

BGP path advertisement



gateway router may learn about multiple paths to destination:

- AS1 gateway router 1c learns path *AS2,AS3,X* from 2a
- AS1 gateway router 1c learns path *AS3,X* from 3a
- Based on policy, AS1 gateway router 1c chooses path *AS3,X, and advertises path within AS1 via iBGP*

45

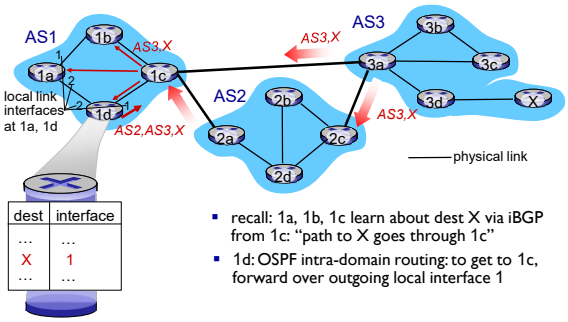
BGP messages

- BGP messages exchanged between peers over TCP connection
- BGP messages:
  - OPEN: opens TCP connection to remote BGP peer and authenticates sending BGP peer
  - UPDATE: advertises new path (or withdraws old)
  - KEEPALIVE: keeps connection alive in absence of UPDATES; also ACKs OPEN request
  - NOTIFICATION: reports errors in previous msg; also used to close connection

46

BGP, OSPF, forwarding table entries

Q: how does router set forwarding table entry to distant prefix?

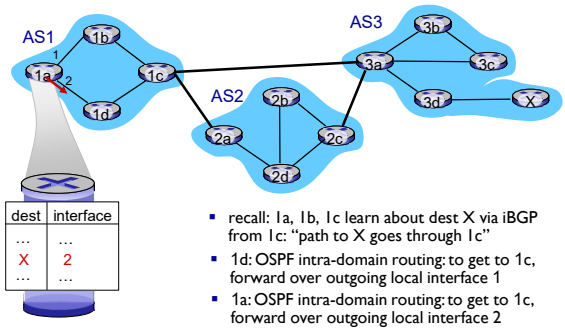


- recall: 1a, 1b, 1c learn about dest X via iBGP from 1c: "path to X goes through 1c"
- 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1

47

BGP, OSPF, forwarding table entries

Q: how does router set forwarding table entry to distant prefix?

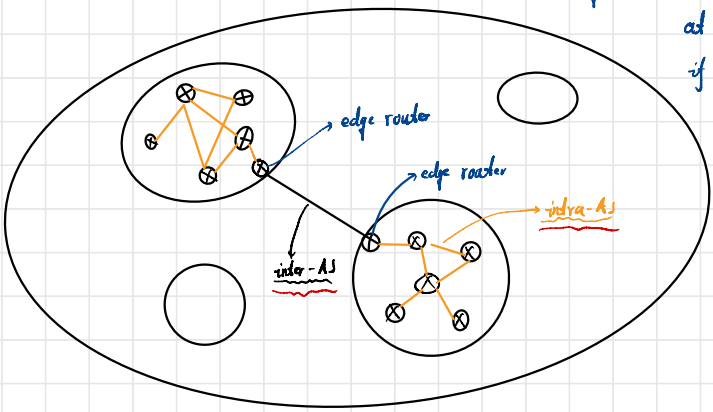


- recall: 1a, 1b, 1c learn about dest X via iBGP from 1c: "path to X goes through 1c"
- 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1
- 1a: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 2

48

04/12/2022

edge router: at least 2 routing algorithms: 1 intra, 1 inter
at most 3 algorithms: 1 intra, 2 inter
if more outer link connecting to the router.



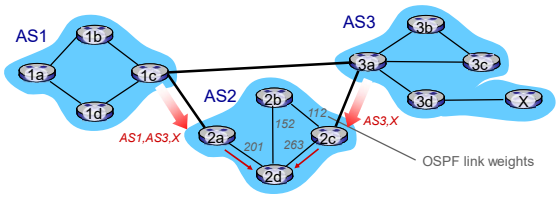
04/14/2022 starts here

BGP route selection

- router may learn about more than one route to destination AS, selects route based on:
  1. local preference value attribute: policy decision
  2. shortest AS-PATH
  3. closest NEXT-HOP router: hot potato routing
  4. additional criteria

49

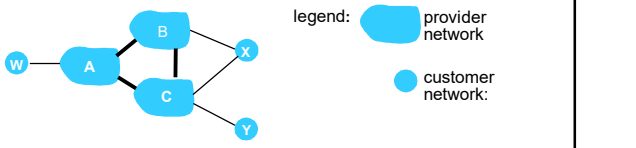
Hot Potato Routing



- 2d learns (via iBGP) it can route to X via 2a or 2c
- *hot potato routing:* choose local gateway that has least intra-domain cost (e.g., 2d chooses 2a, even though more AS hops to X): don't worry about inter-domain cost!

50

BGP: achieving policy via advertisements

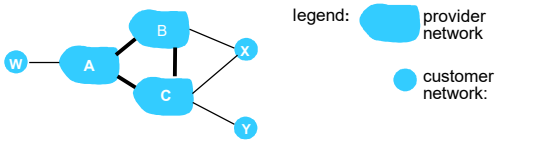


Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- A advertises path Aw to B and to C
- B *chooses not to advertise* BAw to C:
  - B gets no "revenue" for routing CBAw, since none of C, A, w are B's customers
  - C does not learn about CBAw path
- C will route CAw (not using B) to get to w

51

BGP: achieving policy via advertisements



Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- A,B,C are *provider networks*
- X,W,Y are customer (of provider networks)
- X is *dual-homed:* attached to two networks
- *policy to enforce:* X does not want to route from B to C via X
  - .. so X will not advertise to B a route to C

52

Why different Intra-, Inter-AS routing ?

policy:

- inter-AS: admin wants control over how its traffic routed, who routes through its net.
- intra-AS: single admin, so no policy decisions needed

scale:

- hierarchical routing saves table size, reduced update traffic

performance:

- intra-AS: can focus on performance
- inter-AS: policy may dominate over performance

53

Chapter 5: outline

5.1 introduction
5.2 routing protocols
- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF
5.4 routing among the ISPs: BGP
5.5 The SDN control plane
5.6 ICMP: The Internet Control Message Protocol
5.7 Network management and SNMP

54

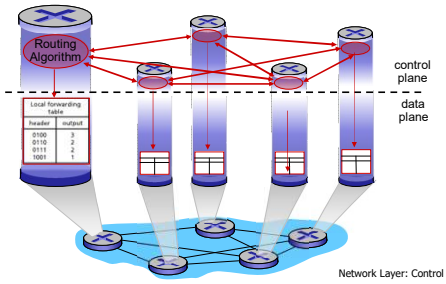
Software defined networking (SDN)

- Internet network layer: historically has been implemented via distributed, per-router approach
  - *monolithic* router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS)
  - different "middleboxes" for different network layer functions: firewalls, load balancers, NAT boxes, ..
- ~2005: renewed interest in rethinking network control plane

55

Recall: per-router control plane

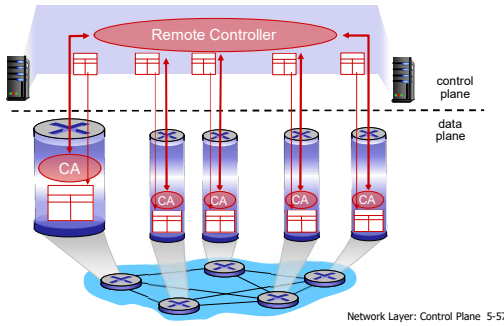
Individual routing algorithm components *in each and every router* interact with each other in control plane to compute forwarding tables



56

Recall: logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables



57

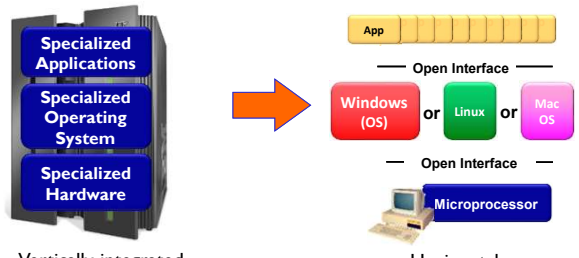
Software defined networking (SDN)

Why a *logically centralized* control plane?

- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding (recall OpenFlow API) allows "programming" routers
  - centralized "programming" easier: compute tables centrally and distribute
  - distributed "programming: more difficult: compute tables as result of distributed algorithm (protocol) implemented in each and every router
- open (non-proprietary) implementation of control plane

58

Analogy: mainframe to PC evolution*



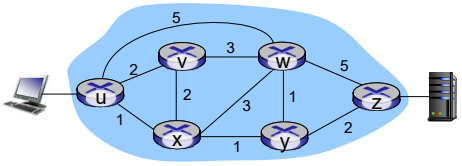
Vertically integrated
Closed, proprietary
Slow innovation
Small industry

Horizontal
Open interfaces
Rapid innovation
Huge industry

* Slide courtesy: N. McKeown

59

Traffic engineering: difficult traditional routing



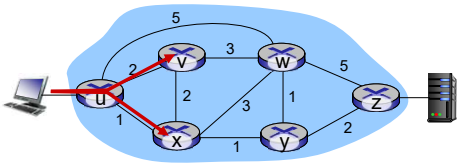
Q: what if network operator wants u-to-z traffic to flow along *uvwz*, x-to-z traffic to flow *xwyz*?

A: need to define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!

Link weights are only control "knobs": wrong!

60

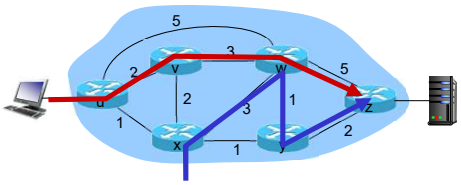
Traffic engineering: difficult



Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?
A: can't do it (or need a new routing algorithm)

61

Traffic engineering: difficult

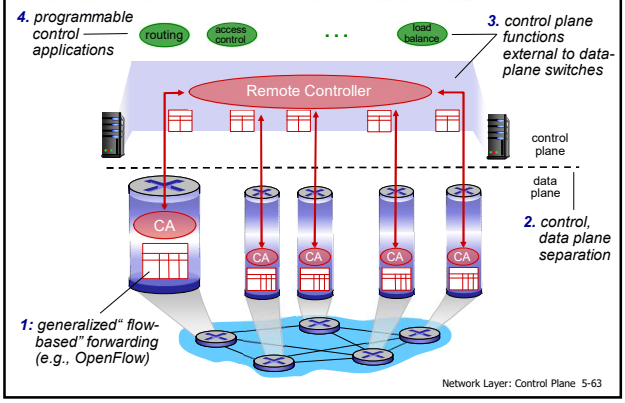


Q: what if w wants to route blue and red traffic differently?

A: can't do it (with destination based forwarding, and LS, DV routing)

62

Software defined networking (SDN)

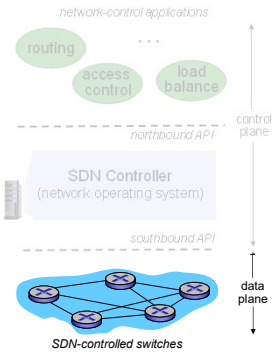


63

SDN perspective: data plane switches

Data plane switches

- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- switch flow table computed, installed by controller
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable and what is not
- protocol for communicating with controller (e.g., OpenFlow)

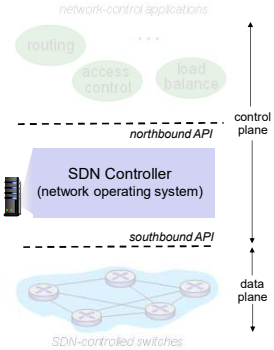


64

SDN perspective: SDN controller

SDN controller (network OS):

- maintain network state information
- interacts with network control applications "above" via northbound API
- interacts with network switches "below" via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness

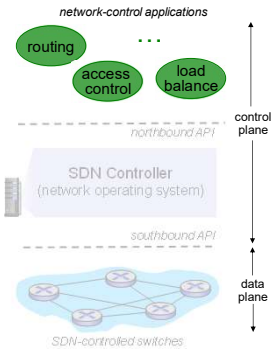


65

SDN perspective: control applications

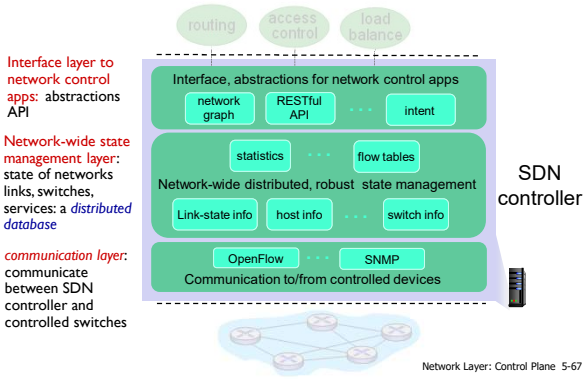
network-control apps:

- "brains" of control: implement control functions using lower-level services, API provided by SND controller
- *unbundled:* can be provided by 3rd party: distinct from routing vendor, or SDN controller



66

Components of SDN controller



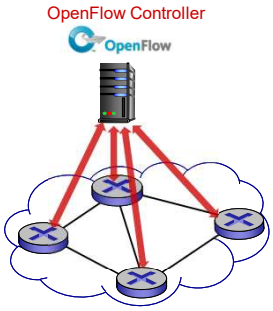
Interface layer to network control apps: abstractions API

Network-wide state management layer: state of networks links, switches, services: a *distributed database*

communication layer: communicate between SDN controller and controlled switches

67

OpenFlow protocol



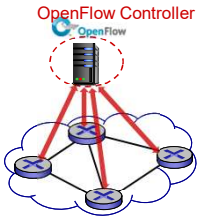
- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc)

68

OpenFlow: controller-to-switch messages

Key controller-to-switch messages

- *features:* controller queries switch features, switch replies
- *configure:* controller queries/sets switch configuration parameters
- *modify-state:* add, delete, modify flow entries in the OpenFlow tables
- *packet-out:* controller can send this packet out of specific switch port

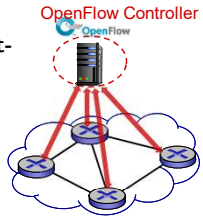


69

OpenFlow: switch-to-controller messages

Key switch-to-controller messages

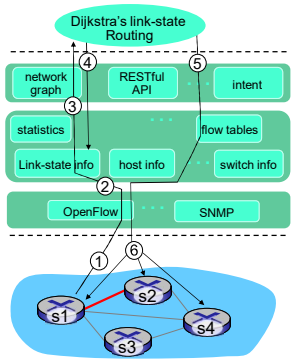
- *packet-in:* transfer packet (and its control) to controller. See packet-out message from controller
- *flow-removed:* flow table entry deleted at switch
- *port status:* inform controller of a change on a port.



Fortunately, network operators don't "program" switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

70

SDN: control/data plane interaction example



① S1, experiencing link failure using OpenFlow port status message to notify controller

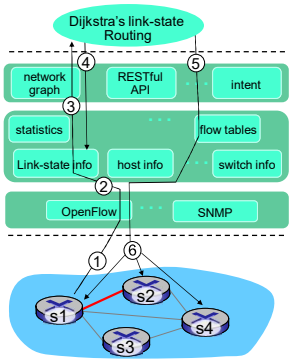
② SDN controller receives OpenFlow message, updates link status info

③ Dijkstra's routing algorithm application has previously registered to be called when ever link status changes. It is called.

④ Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes

71

SDN: control/data plane interaction example

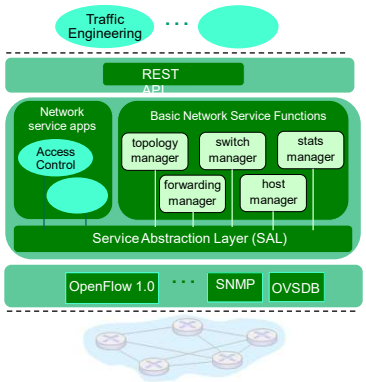


⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed

⑥ Controller uses OpenFlow to install new tables in switches that need updating

72

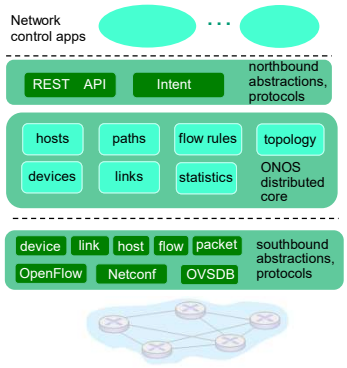
OpenDaylight (ODL) controller



- ODL Lithium controller
- network apps may be contained within, or be external to SDN controller
- Service Abstraction Layer: interconnects internal, external applications and services

73

ONOS controller



- control apps separate from controller
- intent framework: high-level specification of service: what rather than how
- considerable emphasis on distributed core: service reliability, replication performance scaling

74

SDN: selected challenges

- hardening the control plane: dependable, reliable, performance-scalable, secure distributed system
  - robustness to failures: leverage strong theory of reliable distributed system for control plane
  - dependability, security: "baked in" from day one?
- networks, protocols meeting mission-specific requirements
  - e.g., real-time, ultra-reliable, ultra-secure
- Internet-scaling

75

前面的内容被略过了

Chapter 5: outline

5.1 introduction

5.2 routing protocols
- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

76

Traceroute exploits ICMP ?
Application layer

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

77

implemented on top network layer
but not transport layer (middle between N and T)
use IP but not use TCP/UDP ★

Traceroute and ICMP

- source sends series of UDP segments to destination
  - first set has TTL =1
  - second set has TTL=2, etc.
  - unlikely port number
- when datagram in nth set arrives to nth router:
  - router discards datagram and sends source ICMP message (type 11, code 0)
  - ICMP message include name of router & IP address
- when ICMP message arrives, source records RTTs

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

78

use bad port # on purpose. not to be used by destination host.

Chapter 5: outline

5.1 introduction

5.2 routing protocols
- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

79

What is network management?

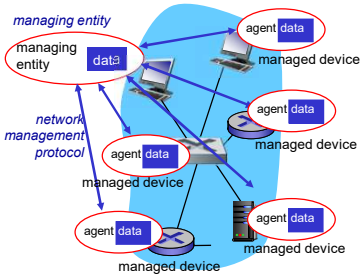
- autonomous systems (aka "network"): 1000s of interacting hardware/software components
- other complex systems requiring monitoring, control:
  - jet airplane
  - nuclear power plant
  - others?

"Network management includes the deployment, integration and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost."

80

Infrastructure for network management

definitions:

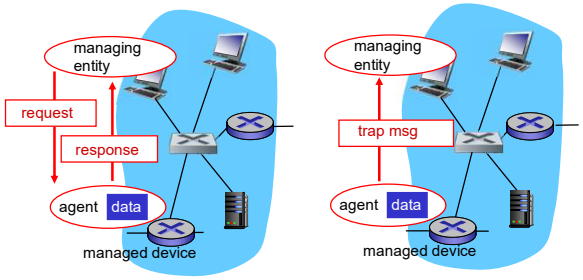


managed devices contain *managed objects* whose data is gathered into a *Management Information Base (MIB)*

81

SNMP protocol

Two ways to convey MIB info, commands:



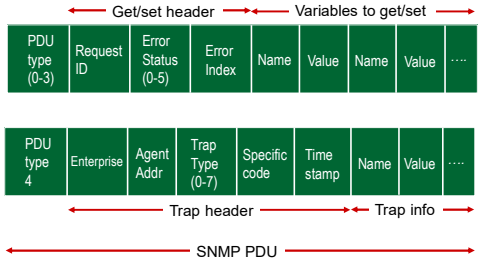
82

SNMP protocol: message types

Message type	Function
GetRequest GetNextRequest GetBulkRequest	manager-to-agent: "get me data" (data instance, next data in list, block of data)
InformRequest	manager-to-manager: here's MIB value
SetRequest	manager-to-agent: set MIB value
Response	Agent-to-manager: value, response to Request
Trap	Agent-to-manager: inform manager of exceptional event

83

SNMP protocol: message formats



More on network management: see earlier editions of text!

84

Chapter 5: summary

we've learned a lot!

- approaches to network control plane
  - per-router control (traditional)
  - logically centralized control (software defined networking)
- traditional routing algorithms
  - implementation in Internet: OSPF, BGP
- SDN controllers
  - implementation in practice: ODL, ONOS
- Internet Control Message Protocol
- network management

next stop: link layer!

Chapter 6

04/14/2022 continues

Chapter 6: Link layer and LANs

our goals:

- understand principles behind link layer services:
  - error detection, correction
  - sharing a broadcast channel: multiple access
  - link layer addressing
  - local area networks: Ethernet, VLANs
- instantiation, implementation of various link layer technologies

1

Link layer, LANs: outline

6.1 introduction, services
6.2 error detection, correction
6.3 multiple access protocols
6.4 LANs
- addressing, ARP
- Ethernet
- switches
- VLANS

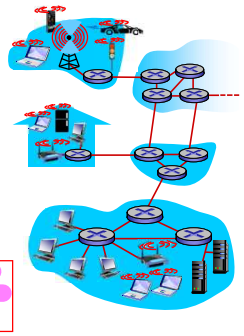
6.5 link virtualization: MPLS
6.6 data center networking
6.7 a day in the life of a web request

2

Link layer: introduction

terminology:

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



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

3

Link layer: context

- datagram transferred by different link protocols over different links:
  - e.g., Ethernet on first link, frame relay on intermediate links, 802.11 on last link
- each link protocol provides different services
  - e.g., may or may not provide rdt over link

transportation analogy:

- trip from Princeton to Lausanne
  - limo: Princeton to JFK
  - plane: JFK to Geneva
  - train: Geneva to Lausanne
- tourist = datagram
- transport segment = communication link
- transportation mode = link layer protocol
- travel agent = routing algorithm

4

Link layer 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, destination
    - 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 (uses Forward Error Corection, FEC)
    - *Q:* why both link-level and end-end reliability?

MAC address: 48 bit long or LAN address:

5

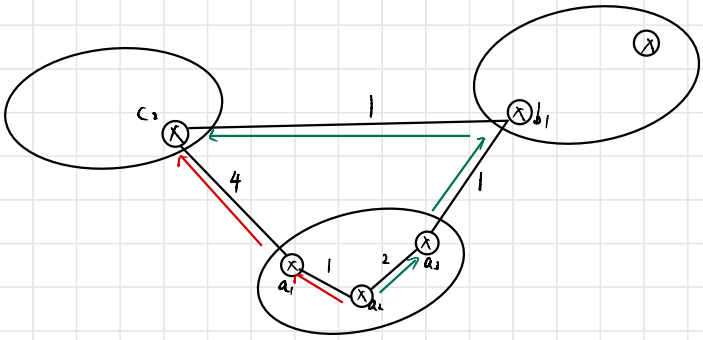
Link layer services (more)

- *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

单边发 双边同时发

6

04/14/2022 starts here



hot potato routing

从 a_2 出来到 C_2，用的是红线路线，而不是绿线，那相总的cost要小于红线，因为在a中我想要

- *hot potato routing:* choose local gateway that has least intra-domain cost (e.g., 2d chooses 2a, even though more AS hops to X): don't worry about inter-domain cost!

inter-As: dominated by policy

intra-As: dominated by performance

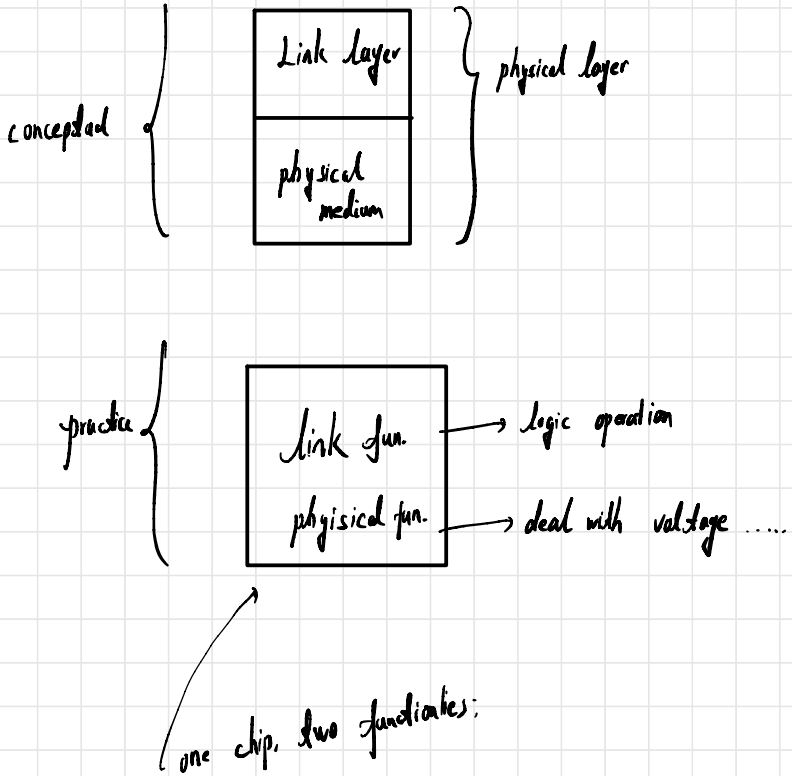
ch: 6:

edge
- A
- T: TCP/UDP protocal
- N: IP protocal

core

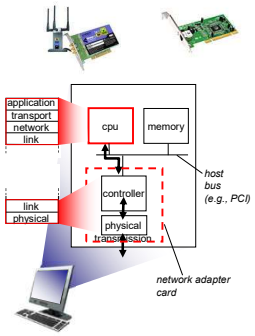
- P

Introduce reliability on physical layer is debatable: for example, fiber has low error rate, the reliability may slow it down, and Wi-Fi has relatively high errors rate and the reliability may slow the speed down further.



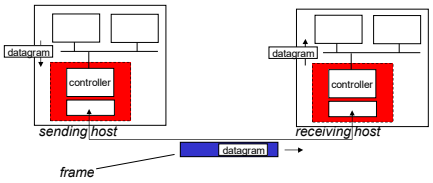
Where is the link layer implemented?

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



7

Adaptors communicating



- sending side:
  - encapsulates datagram in frame
  - adds error checking bits, rdt, flow control, etc.
- receiving side
  - looks for errors, rdt, flow control, etc.
  - extracts datagram, passes to upper layer at receiving side

8

Link layer, LANs: outline

6.1 introduction, services
6.2 error detection, correction
6.3 multiple access protocols
6.4 LANs
- addressing, ARP
- Ethernet
- switches
- VLANS

6.5 link virtualization: MPLS
6.6 data center networking
6.7 a day in the life of a web request

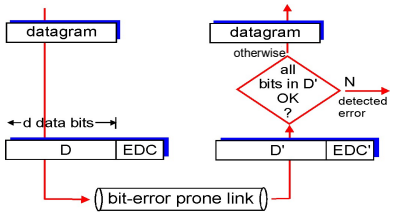
9

04/19/2022 starts here

Error detection

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

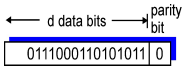
• Error detection not 100% reliable!
  • protocol may miss some errors, but rarely
  • larger EDC field yields better detection and correction



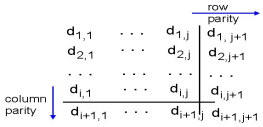
10

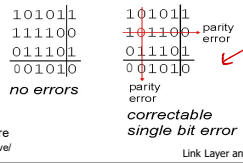
Parity checking

single bit parity:
- *d*etect single bit errors



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





* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

11

Internet checksum (review)

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

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

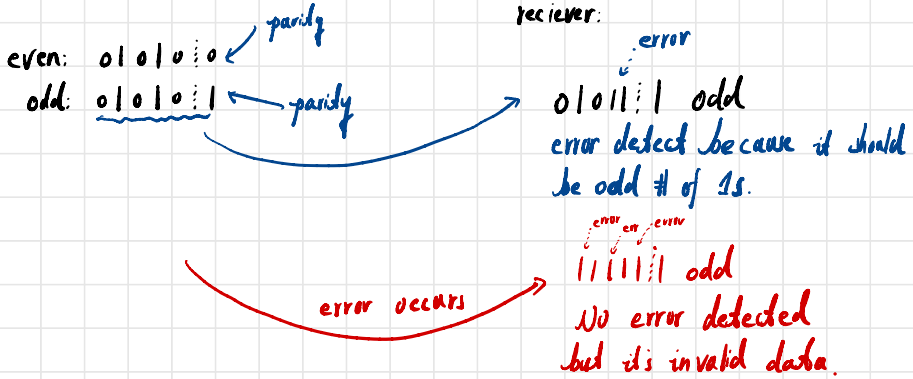
receiver:
- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?*

12

04/19/2022

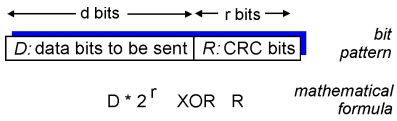
Parity check

Even parity and odd parity



Cyclic redundancy check

- more powerful error-detection coding
- 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)



13

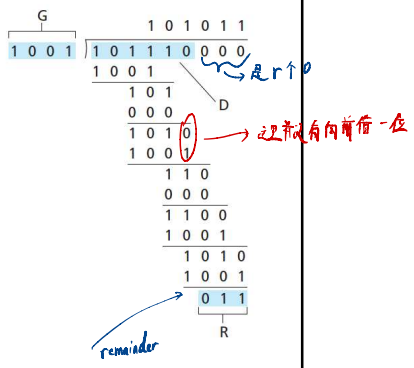
CRC example

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

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

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

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



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

14

Link layer, LANs: outline

6.1 introduction, services
6.2 error detection, correction
6.3 multiple access protocols
6.4 LANs
- addressing, ARP
- Ethernet
- switches
- VLANS

6.5 link virtualization: MPLS
6.6 data center networking
6.7 a day in the life of a web request

15

Multiple access links, protocols

two types of "links":

- point-to-point
  - PPP for dial-up access
  - point-to-point link between Ethernet switch, host
- *broadcast (shared wire or medium)*
  - old-fashioned Ethernet
  - upstream HFC
  - 802.11 wireless LAN

16

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

17

An ideal multiple access protocol

given: broadcast channel of rate R bps

desiderata:

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

18

MAC protocols: taxonomy

three broad classes:

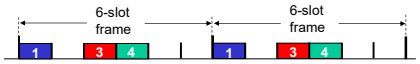
- *channel partitioning*
  - divide channel into smaller "pieces" (time slots, frequency, code)
  - allocate piece to node for exclusive use
- *random access*
  - channel not divided, allow collisions
  - "recover" from collisions
- *"taking turns"*
  - nodes take turns, but nodes with more to send can take longer turns

19

Channel partitioning MAC protocols: TDMA

TDMA: time division multiple access

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

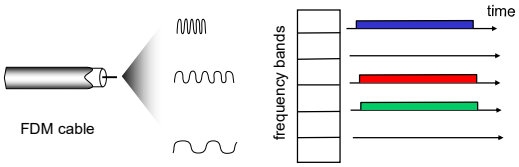


20

Channel partitioning MAC protocols: FDMA

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: 6-station LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle



21

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:
  - slotted ALOHA
  - ALOHA
  - CSMA, CSMA/CD, CSMA/CA

22

Slotted ALOHA

assumptions:

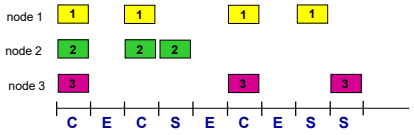
- all frames same size
- time divided into equal size slots (time to transmit 1 frame)
- nodes start to transmit only slot beginning
- nodes are synchronized
- if 2 or more nodes transmit in slot, all nodes detect collision

operation:

- when node obtains fresh frame, transmits in next slot
  - *if no collision:* node can send new frame in next slot
  - *if collision:* node retransmits frame in each subsequent slot with prob. p until success

23

Slotted ALOHA



Pros:

- single active node can continuously transmit at full rate of channel
- highly decentralized: only slots in nodes need to be in sync
- simple

Cons:

- collisions, wasting slots
- idle slots
- nodes may be able to detect collision in less than time to transmit packet
- clock synchronization

24

Slotted ALOHA: efficiency

efficiency: long-run fraction of successful slots (many nodes, all with many frames to send)

- *suppose:* N nodes with many frames to send, each transmits in slot with probability p
- prob that given node has success in a slot $= p(1-p)^{N-1}$
- prob that *any* node has a success $= Np(1-p)^{N-1}$

- max efficiency: find p^* that maximizes $Np(1-p)^{N-1}$
- for many nodes, take limit of $Np^*(1-p^*)^{N-1}$ as N goes to infinity, gives:

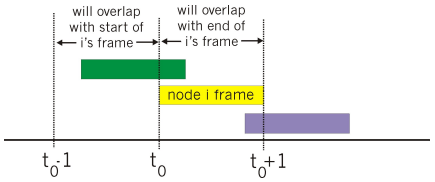
max efficiency $= 1/e = .37$

at best: channel used for useful transmissions 37% of time!

25

Pure (unslotted) ALOHA

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



26

Pure ALOHA efficiency

P(success by given node) = P(node transmits) ·
P(no other node transmits in $[t_0-1, t_0]$ ·
P(no other node transmits in $[t_0-1, t_0]$

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

… choosing optimum p and then letting $n \to \infty$

$= 1/(2e) = .18$

even *worse* than slotted Aloha!

27

CSMA (carrier sense multiple access)

CSMA: listen before transmit:

if channel sensed idle: transmit entire frame

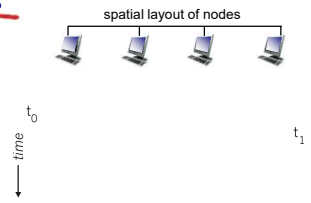
- if channel sensed busy, defer transmission

- human analogy: don't interrupt others!

28

CSMA collisions

- collisions *can* still occur: propagation delay means two nodes may not hear each other's transmission
- collision: entire packet transmission time wasted
  - distance & propagation delay play role in determining collision probability



29

CSMA/CD (collision detection)

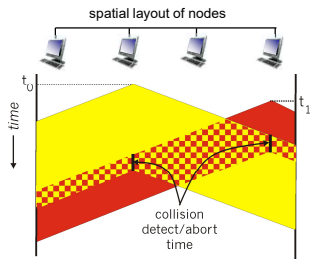
CSMA/CD: carrier sensing, deferral as in CSMA

- collisions *detected* within short time
- colliding transmissions aborted, reducing channel wastage

- collision detection:
  - easy in wired LANs: measure signal strengths, compare transmitted, received signals
  - difficult in wireless LANs: received signal strength overwhelmed by local transmission strength
- human analogy: the polite conversationalist

30

CSMA/CD (collision detection)



31

Ethernet CSMA/CD algorithm

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

32

size of LAN is designed for protocol

CSMA/CD efficiency

- T_{prop} = max prop delay between 2 nodes in LAN
- t_{trans} = time to transmit max-size frame

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

longer data you have better efficiency because if you are sending no one else will send

- efficiency goes to 1
  - as t_{prop} goes to 0
  - as t_{trans} goes to infinity
- better performance than ALOHA: and simple, cheap, decentralized!

smaller network, less t_{prop}

33

"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!

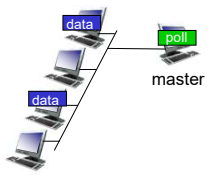
random access MAC protocols
- efficient at low load: single node can fully utilize channel
- high load: collision overhead

"taking turns" protocols
look for best of both worlds!

34

"Taking turns" MAC protocols

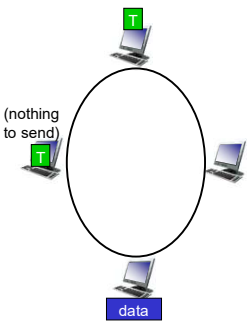
polling:
- master node "invites" slave nodes to transmit in turn
- typically used with "dumb" slave devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (master)



35

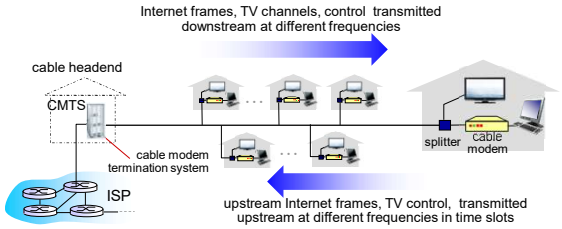
"Taking turns" MAC protocols

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



36

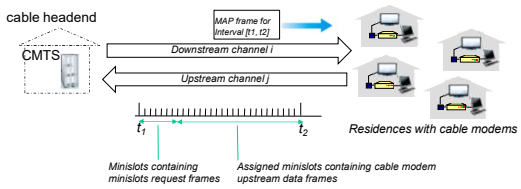
Cable access network



- multiple 40Mbps downstream (broadcast) channels
  - single CMTS transmits into channels
- multiple 30 Mbps upstream channels
  - multiple access: all users contend for certain upstream channel time slots (others assigned)

37

Cable access network



DOCSIS: data over cable service interface spec

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

38

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, token ring

39

Link layer, LANs: outline

6.1 introduction, services
6.2 error detection, correction
6.3 multiple access protocols
6.4 LANs
- addressing, ARP
- Ethernet
- switches
- VLANS

6.5 link virtualization: MPLS
6.6 data center networking
6.7 a day in the life of a web request

40

MAC addresses and ARP

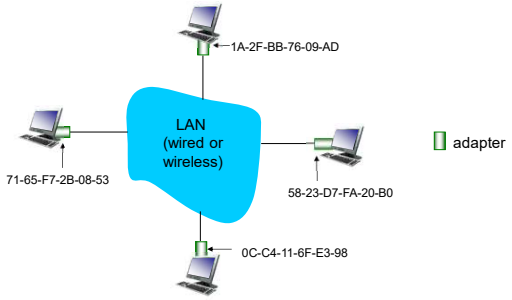
- 32-bit IP address:
  - *network-layer* address for interface
  - used for layer 3 (network layer) forwarding
- MAC (or LAN or physical or Ethernet) address:
  - function: *used 'locally" to get frame from one interface to another physically-connected interface (same network, in IP-addressing sense)*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

hexadecimal (base 16) notation (each "numeral" represents 4 bits)

41

LAN addresses and ARP

each adapter on LAN has unique *LAN* address



42

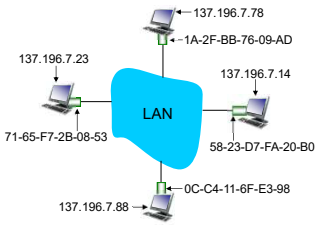
LAN addresses (more)

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

43

ARP: address resolution protocol

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



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

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

44

ARP protocol: same LAN

- A wants to send datagram to B
  - B's MAC address not in A's ARP table.
- A broadcasts ARP query packet, containing B's IP address
  - destination MAC address = FF-FF-FF-FF-FF-FF
  - all nodes on LAN receive ARP query
- B receives ARP packet, replies to A with its (B's) MAC address
  - frame sent to A's MAC address (unicast)

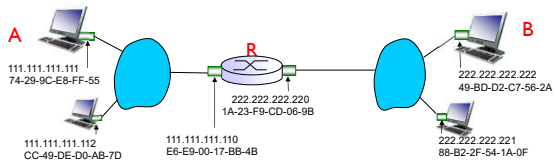
- A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)
  - soft state: information that times out (goes away) unless refreshed
- ARP is "plug-and-play":
  - nodes create their ARP tables *without intervention from net administrator*

45

Addressing: routing to another LAN

walkthrough: send datagram from A to B via R

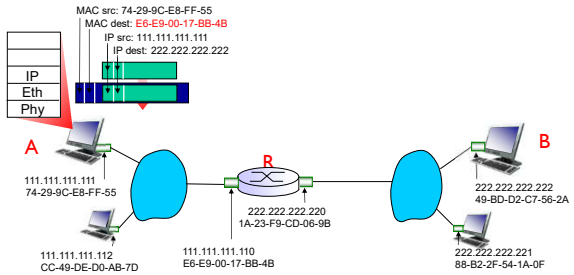
- focus on addressing – at IP (datagram) and MAC layer (frame)
- assume A knows B's IP address
- assume A knows IP address of first hop router, R (how?)
- assume A knows R's MAC address (how?)



46

Addressing: routing to another LAN

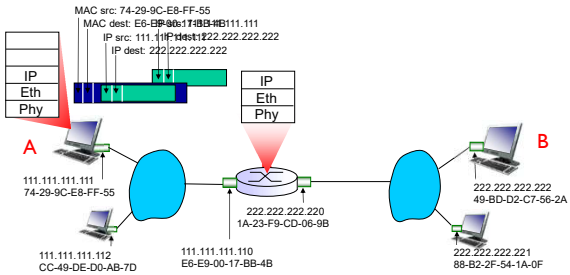
- A creates IP datagram with IP source A, destination B
- A creates link-layer frame with R's MAC address as destination address, frame contains A-to-B IP datagram



47

Addressing: routing to another LAN

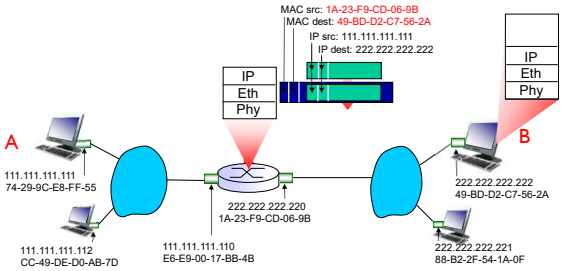
- frame sent from A to R
- frame received at R, datagram removed, passed up to IP



48

Addressing: routing to another LAN

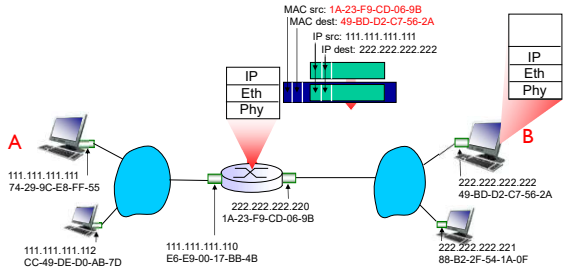
- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as destination address, frame contains A-to-B IP datagram



49

Addressing: routing to another LAN

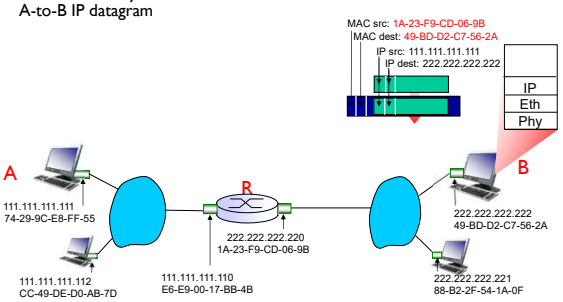
- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as destination address, frame contains A-to-B IP datagram



50

Addressing: routing to another LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as destination address, frame contains A-to-B IP datagram



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

51

Link layer, LANs: outline

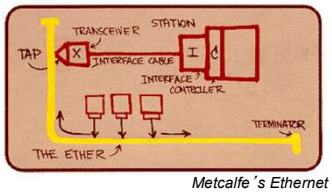
6.1 introduction, services
6.2 error detection, correction
6.3 multiple access protocols
6.4 LANs
- addressing, ARP
- Ethernet
- switches
- VLANS

6.5 link virtualization: MPLS
6.6 data center networking
6.7 a day in the life of a web request

52

Ethernet

"dominant" wired LAN technology:
- single chip, multiple speeds (e.g., Broadcom BCM5761)
- first widely used LAN technology
- simpler, cheap
- kept up with speed race: 10 Mbps – 10 Gbps

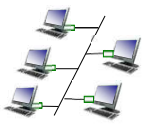


Metcalfe's Ethernet sketch

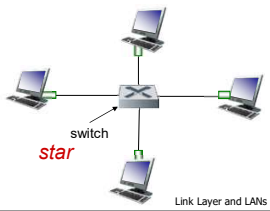
53

Ethernet: physical topology

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



bus: coaxial cable

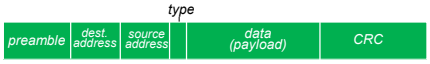


star

54

Ethernet frame structure

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



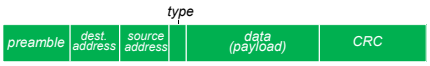
preamble:

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

55

Ethernet frame structure (more)

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



56

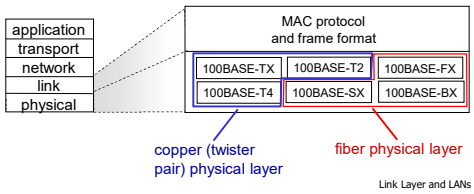
Ethernet: unreliable, connectionless

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

57

802.3 Ethernet standards: link & physical layers

- *many* different Ethernet standards
  - common MAC protocol and frame format
  - different speeds: 2 Mbps, 10 Mbps, 100 Mbps, 1Gbps, 10 Gbps, 40 Gbps
  - different physical layer media: fiber, cable



58

Link layer, LANs: outline

6.1 introduction, services
6.2 error detection, correction
6.3 multiple access protocols
6.4 LANs
- addressing, ARP
- Ethernet
- switches
- VLANS

6.5 link virtualization: MPLS
6.6 data center networking
6.7 a day in the life of a web request

59

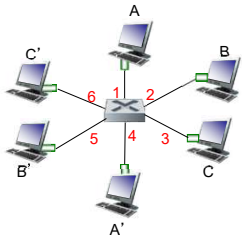
Ethernet switch

- link-layer device: takes an *active* role
  - store, forward Ethernet frames
  - examine incoming frame's MAC address, selectively forward frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment
- *transparent*
  - hosts are unaware of presence of switches
- *plug-and-play, self-learning*
  - switches do not need to be configured

60

Switch: *multiple* simultaneous transmissions

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



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

61

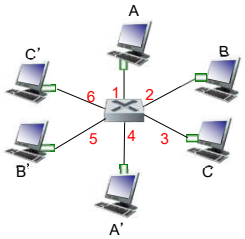
Switch forwarding table

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

- A: each switch has a switch table, each entry:
  - (MAC address of host, interface to reach host, time stamp)
  - looks like a routing table!

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

- something like a routing protocol?

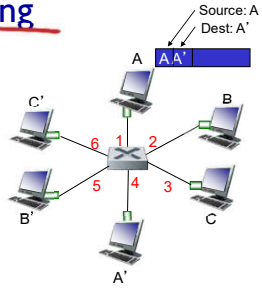


switch with six interfaces (1,2,3,4,5,6)

62

Switch: self-learning

- switch *learns* which hosts can be reached through which interfaces
  - when frame received, switch "learns" location of sender: incoming LAN segment
  - records sender/location pair in switch table



MAC addr	interface	TTL
A	*1*	*60*

Switch table (initially empty)

63

Switch: frame filtering/forwarding

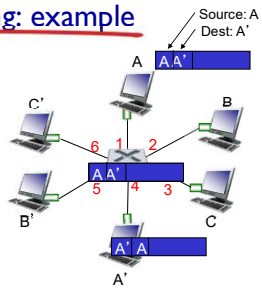
when frame received at switch:

1. record incoming link, MAC address of sending host
2. index switch table using MAC destination address
3. if entry found for destination
   then {
   if destination on segment from which frame arrived
   then drop frame
   else forward frame on interface indicated by entry
   }
   else flood /* forward on all interfaces except arriving interface */

64

Self-learning, forwarding: example

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



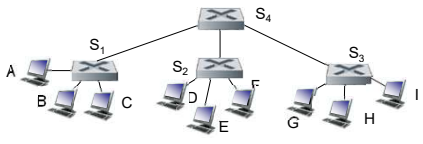
MAC addr	interface	TTL
A	*1*	*60*
A'	*4*	*60*

switch table (initially empty)

65

Interconnecting switches

self-learning switches can be connected together:



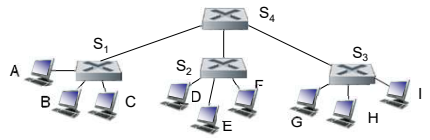
Q: sending from A to G - how does S_1 know to forward frame destined to G via S_4 and S_3?

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

66

Self-learning multi-switch example

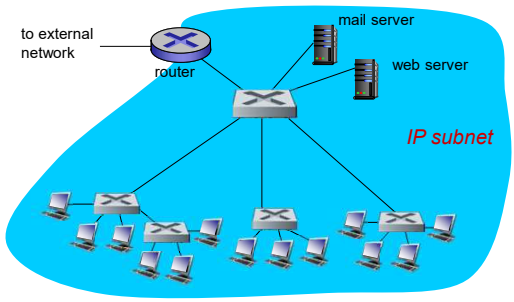
Suppose C sends frame to I, I responds to C



- Q: show switch tables and packet forwarding in S_1, S_2, S_3, S_4

67

Institutional network



68

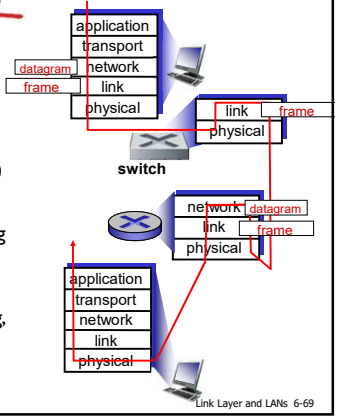
Switches vs. routers

both are store-and-forward:

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

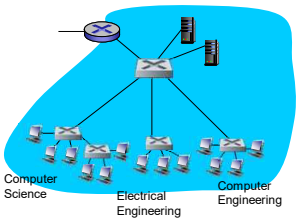
both have forwarding tables:

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



69

VLANs: motivation



consider:

- CS user moves office to EE, but wants connect to CS switch?
- single broadcast domain:
  - all layer-2 broadcast traffic (ARP, DHCP, unknown location of destination MAC address) must cross entire LAN
  - security/privacy, efficiency issues

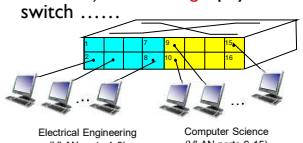
70

VLANs

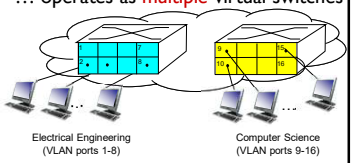
Virtual Local Area Network

switch(es) supporting VLAN capabilities can be configured to define multiple ***virtual*** LANS over single physical LAN infrastructure.

port-based VLAN: switch ports grouped (by switch management software) so that *single* physical switch ……



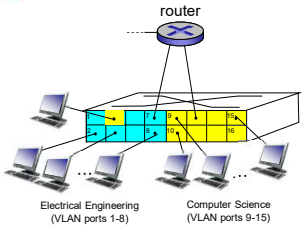
… operates as multiple virtual switches



71

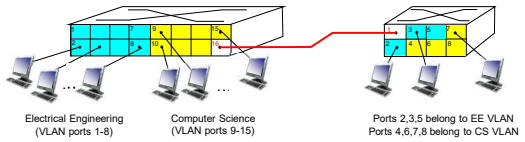
Port-based VLAN

- *traffic isolation:* frames to/from ports 1-8 can *only* reach ports 1-8
  - can also define VLAN based on MAC addresses of endpoints, rather than switch port
- dynamic membership: ports can be dynamically assigned among VLANs
- forwarding between VLANS: done via routing (just as with separate switches)
  - in practice vendors sell combined switches plus routers



72

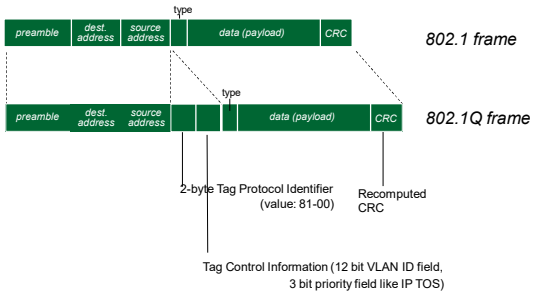
VLANS spanning multiple switches



- *trunk port:* carries frames between VLANS defined over multiple physical switches
  - frames forwarded within VLAN between switches can't be vanilla 802.1 frames (must carry VLAN ID info)
  - 802.1q protocol adds/removed additional header fields for frames forwarded between trunk ports

73

802.1Q VLAN frame format



74

Link layer, LANs: outline

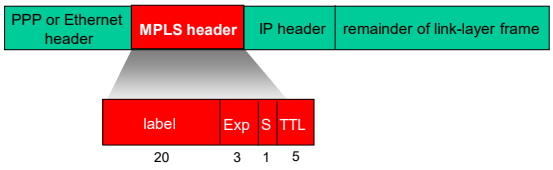
6.1 introduction, services
6.2 error detection, correction
6.3 multiple access protocols
6.4 LANs
- addressing, ARP
- Ethernet
- switches
- VLANS

6.5 link virtualization: MPLS
6.6 data center networking
6.7 a day in the life of a web request

75

Multiprotocol label switching (MPLS)

- initial goal: high-speed IP forwarding using fixed length label (instead of IP address)
  - fast lookup using fixed length identifier (rather than shortest prefix matching)
  - borrowing ideas from Virtual Circuit (VC) approach
  - but IP datagram still keeps IP address!



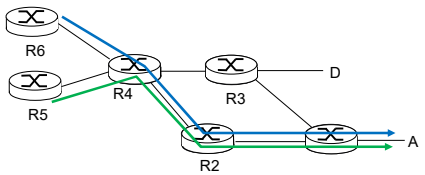
76

MPLS capable routers

- a.k.a. label-switched router
- forward packets to outgoing interface based only on label value (*don't inspect IP address*)
  - MPLS forwarding table distinct from IP forwarding tables
- *flexibility:* MPLS forwarding decisions can *differ* from those of IP
  - use destination *and* source addresses to route flows to same destination differently (traffic engineering)
  - re-route flows quickly if link fails: pre-computed backup paths (useful for VoIP)

77

MPLS versus IP paths

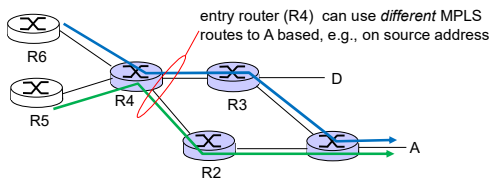


- *IP routing: path to destination determined by destination address alone*

IP router

78

MPLS versus IP paths



- *IP routing:* path to destination determined by destination address alone
- *MPLS routing:* path to destination can be based on source *and* destination address
  - *fast reroute:* precompute backup routes in case of link failure

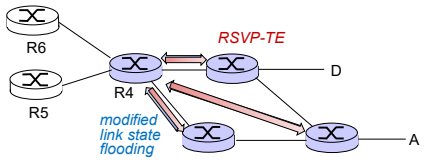
IP-only router

MPLS and IP router

79

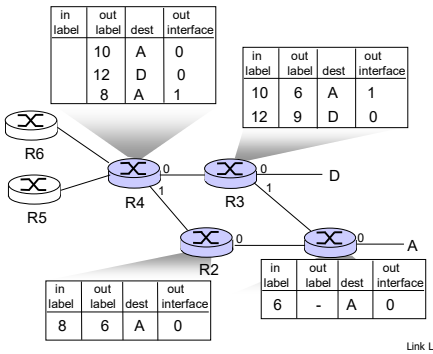
MPLS signaling

- modify OSPF, IS-IS link-state flooding protocols to carry info used by MPLS routing,
  - e.g., link bandwidth, amount of "reserved" link bandwidth
- *entry MPLS router uses RSVP-TE signaling protocol to set up MPLS forwarding at downstream routers*



80

MPLS forwarding tables



81

Link layer, LANs: outline

6.1 introduction, services
6.2 error detection, correction
6.3 multiple access protocols
6.4 LANs
- addressing, ARP
- Ethernet
- switches
- VLANS

6.5 link virtualization: MPLS
6.6 data center networking
6.7 a day in the life of a web request

82

Data center networks

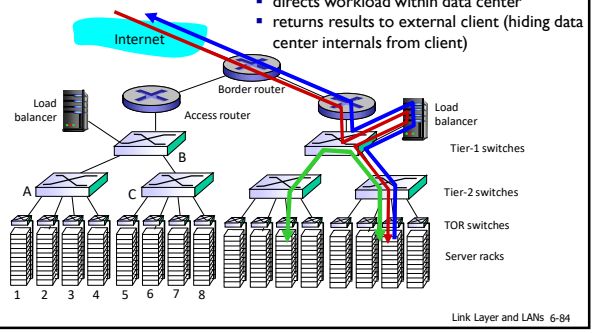
- 10's to 100's of thousands of hosts, often closely coupled, in close proximity:
  - e-business (e.g. Amazon)
  - content-servers (e.g., YouTube, Akamai, Apple, Microsoft)
  - search engines, data mining (e.g., Google)
- challenges:
  - multiple applications, each serving massive numbers of clients
  - managing/balancing load, avoiding processing, networking, data bottlenecks

Inside a 40-ft Microsoft container, Chicago data center

83

Data center networks

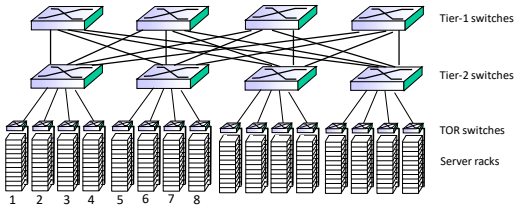
load balancer: application-layer routing
- receives external client requests
- directs workload within data center
- returns results to external client (hiding data center internals from client)



84

Data center networks

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



85

Link layer, LANs: outline

6.1 introduction, services
6.2 error detection, correction
6.3 multiple access protocols
64 LANs
- addressing, ARP
- Ethernet
- switches
- VLANS

6.5 link virtualization: MPLS
6.6 data center networking
6.7 a day in the life of a web request

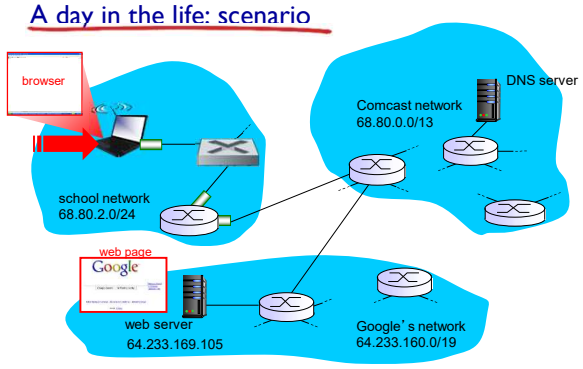
86

Synthesis: a day in the life of a web request

- journey down protocol stack complete!
  - application, transport, network, link
- putting-it-all-together: synthesis!
  - *goal:* identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page
  - *scenario:* student attaches laptop to campus network, requests/receives www.google.com

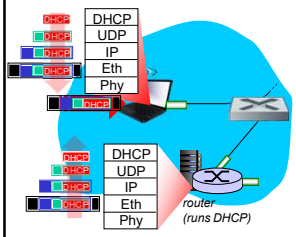
87

A day in the life: scenario



88

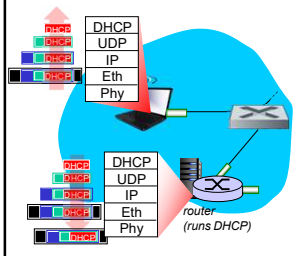
A day in the life… connecting to the Internet



- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use *DHCP*
- DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.3 Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demuxed to IP demuxed, UDP demuxed to DHCP

89

A day in the life… connecting to the Internet

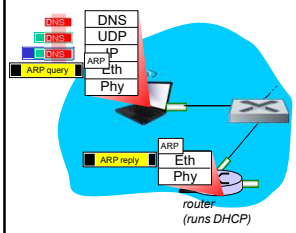


- DHCP server formulates *DHCP ACK* containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation at DHCP server, frame forwarded (switch learning) through LAN, demultiplexing at client
- DHCP client receives DHCP ACK reply

Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

90

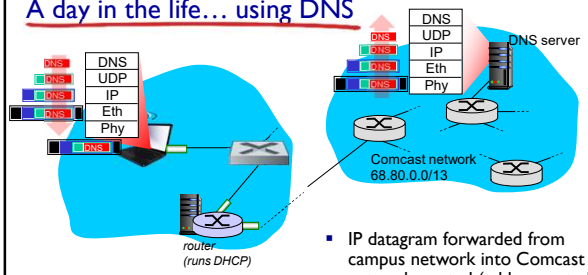
A day in the life… ARP (before DNS, before HTTP)



- before sending *HTTP* request, need IP address of www.google.com: *DNS*
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: ARP
- ARP query broadcast, received by router, which replies with ARP reply giving MAC address of router interface
- client now knows MAC address of first hop router, so can now send frame containing DNS query

91

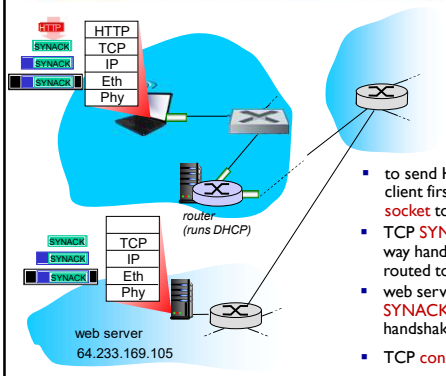
A day in the life… using DNS



- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router
- IP datagram forwarded from campus network into Comcast network, routed (tables created by RIP, OSPF, IS-IS and/or BGP routing protocols) to DNS server
- demuxed to DNS server
- DNS server replies to client with IP address of www.google.com

92

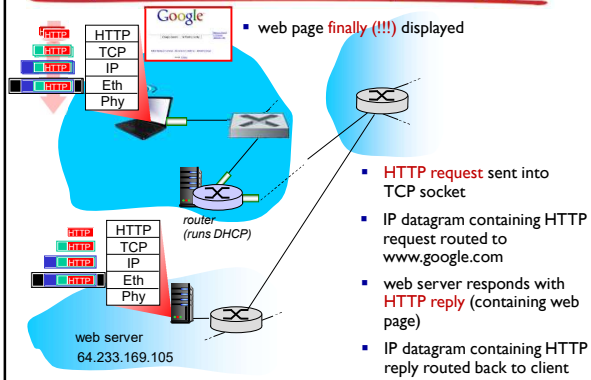
A day in the life…TCP connection carrying HTTP



- to send HTTP request, client first opens TCP socket to web server
- TCP SYN segment (step 1 in 3-way handshake) inter-domain routed to web server
- web server responds with TCP SYNACK (step 2 in 3-way handshake)
- TCP connection established!

93

A day in the life… HTTP request/reply



- web page finally (!!!) displayed
- HTTP request sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with HTTP reply (containing web page)
- IP datagram containing HTTP reply routed back to client

94

Chapter 6: Summary

- principles behind data link layer services:
  - error detection, correction
  - sharing a broadcast channel: multiple access
  - link layer addressing
- instantiation and implementation of various link layer technologies
  - Ethernet
  - switched LANS, VLANs
  - virtualized networks as a link layer: MPLS
- synthesis: a day in the life of a web request

95

Chapter 6: let's take a breath

- journey down protocol stack *complete* (except PHY)
- solid understanding of networking principles, practice
- ….. could stop here …. but *lots* of interesting topics!
  - wireless
  - multimedia
  - security

96

4/19/2022

6.1 Link layer serives: PPt (review)

6.2 error checksum

Parity Checking: ppt (~6:51)

(~12:27) one error is fine, multiple error is no good.

CRC example

★: (21:11) exam Q:

(23:15) Mac protocals: (PPt)

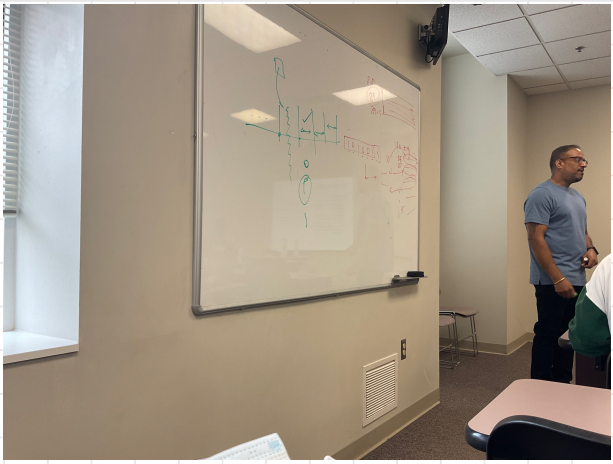
channel partitioning Mac protocal:

fair, not efficient (26:40)

Random Access Protocal (30:30~)

slotted ALOHA

time synchronized.



P ↑ , ↓ collision. (~38:00)

slotted ALOHA : efficiency 개념 (43:00~)

pure ALOHA

⋮

lower even

CSMA collisions

장점 collision

CSMA/CD