Jaringan Komputer

Pertemuan 3

Prodi Informatika

1

Outlines

- Web dan HTTP
- Cookies dan Web Cache
- Email
- DNS
- Overview Transport Layer

Prodi Informatika

2

Web and HTTP

- *web page* terdiri *objects*
- object bisa HTML file, JPEG image, Java applet, audio file,…
- web page berisi *base HTML-file* yang didalamnya terdapat *referenced objects*
- Setiap objek memiliki alamat dengan sebuah *URL,* e.g.,

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

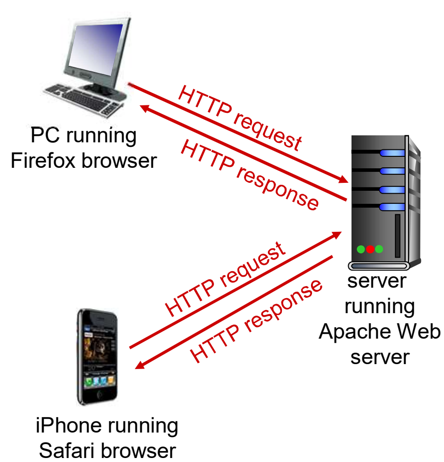
Application Layer 2-3 Prodi Informatika

3

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



Application Layer 2-4 Prodi Informatika

4

HTTP overview (continued)

uses TCP:

- client menginisiasi TCP connection (creates socket) ke server, port 80
- server menerima TCP connection dari client
- Pesan HTTP (application-layer protocol messages) dipertukarkan diantara browser (HTTP client) dan Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server tidak menyimpan informasi tentang permintaan klien sebelumnya

aside

protocols that maintain "state" are complex!

- past history (state) harus dipertahankan
- jika server/client crashes, status bisa inconsistent, sehingga harus direkonsiliasi

Application Layer 2-5 Prodi Informatika

5

HTTP connections

non-persistent HTTP

- paling banyak satu objek yang dikirim melalui koneksi TCP, *koneksi kemudian ditutup*
- mengunduh banyak objek memerlukan banyak koneksi

persistent HTTP

- beberapa objek dapat dikirim melalui koneksi TCP tunggal antara klien, server

Application Layer 2-6 Prodi Informatika

6

Non-persistent HTTP

suppose user enters URL:
www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 jpeg images)

1a. HTTP client **inisialisasi TCP connection** ke HTTP server (process) pada 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

Application Layer 2-7
Prodi Informatika

7

Non-persistent HTTP (cont.)

4. HTTP server closes TCP connection.

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

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

time

Application Layer 2-8
Prodi Informatika

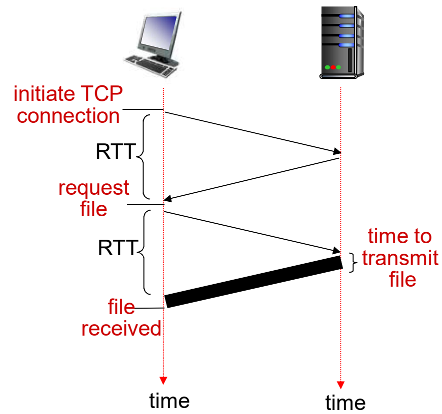
8

Non-persistent HTTP: response time

RTT (round trip time): waktu untuk perjalanan paket dari client ke server dan sebaliknya

HTTP response time:

- satu RTT untuk menginisiasi TCP connection
- satu RTT untuk HTTP request dan beberapa byte pertama dari HTTP response dikembalikan
- file transmission time
- **non-persistent HTTP response time** = 2RTT+ file transmission time



Prodi Informatika

9

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 membiarkan koneksi terbuka setelah mengirim respons
- pesan HTTP selanjutnya antara klien / server yang sama dikirim melalui koneksi terbuka
- klien mengirim permintaan segera setelah menemukan objek yang direferensikan
- sedikitnya satu RTT untuk semua objek yang direferensikan

Application Layer 2-10

Prodi Informatika

10

HTTP request message

- Dua tipe HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)

carriage return character

line-feed character

request line (GET, POST, HEAD commands)

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

header lines

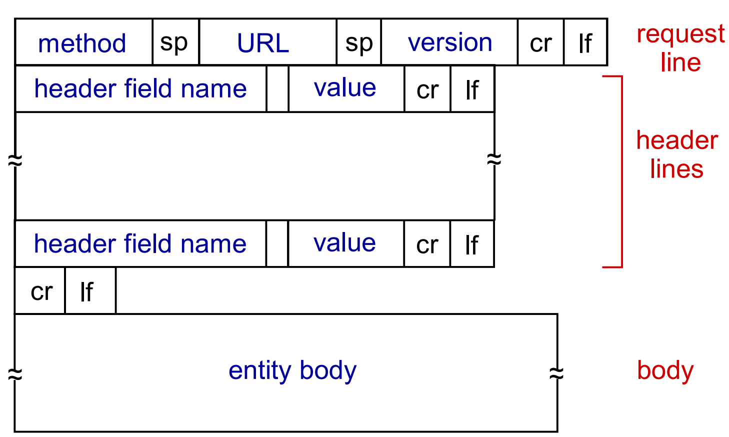
carriage return, line feed at start of line indicates end of header lines

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

Application Layer 2-11 Prodi Informatika

11

HTTP request message: general format



Application Layer 2-12 Prodi Informatika

12

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`

Application Layer 2-13 Prodi Informatika

13

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

Application Layer 2-14 Prodi Informatika

14

HTTP response message

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

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

Application Layer 2-15 Prodi Informatika

15

HTTP response status codes

- kode status muncul di baris pertama dalam pesan respons server-ke-klien.
- beberapa kode contoh:

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

Application Layer 2-16 Prodi Informatika

16

User-server state: cookies

Banyak website menggunakan 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

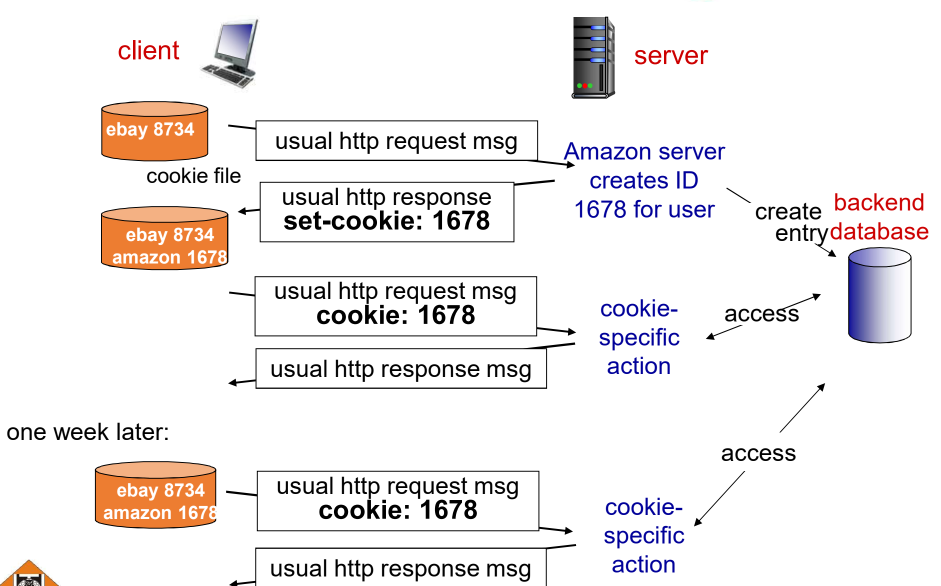
example:

- 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

Application Layer 2-17 Prodi Informatika

17

Cookies: keeping "state" (cont.)



Application Layer 2-18 Prodi Informatika

18

Cookies (continued)

what cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

how to keep "state":

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

aside

cookies and privacy:

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

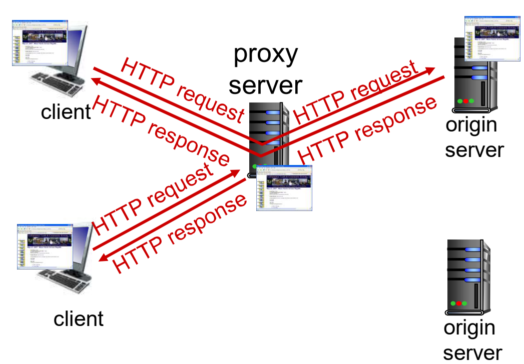
Application Layer 2-19 Prodi Informatika

19

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



Application Layer 2-20 Prodi Informatika

20

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)

21

Caching example:

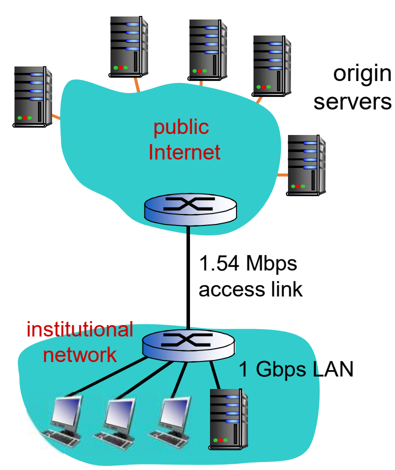
assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

problem!

- LAN utilization: 15%
- access link utilization = 99%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs



22

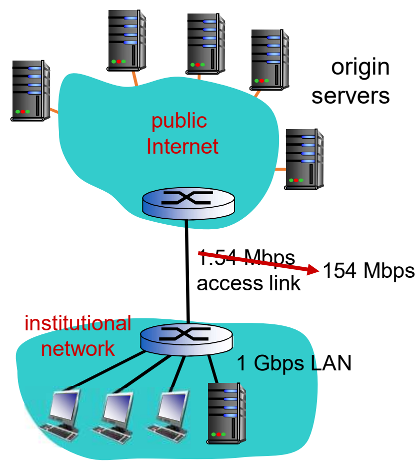
Caching example: fatter access link

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: ~~1.54 Mbps~~ 154 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ~~99%~~ 9.9%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + ~~minutes~~ msecs + usecs



Cost: increased access link speed (not cheap!)

Application Layer 2-23 Prodi Informatika

23

Caching example: install local cache

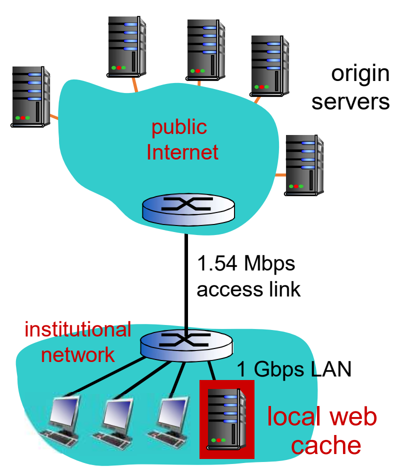
assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ?
- total delay = ?

How to compute link utilization, delay?



Cost: web cache (cheap!)

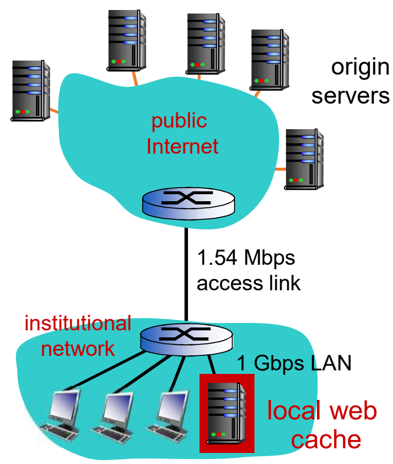
Application Layer 2-24 Prodi Informatika

24

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
- data rate to browsers over access link = 0.6*1.50 Mbps = .9 Mbps
  - utilization = 0.9/1.54 = .58
- 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!)



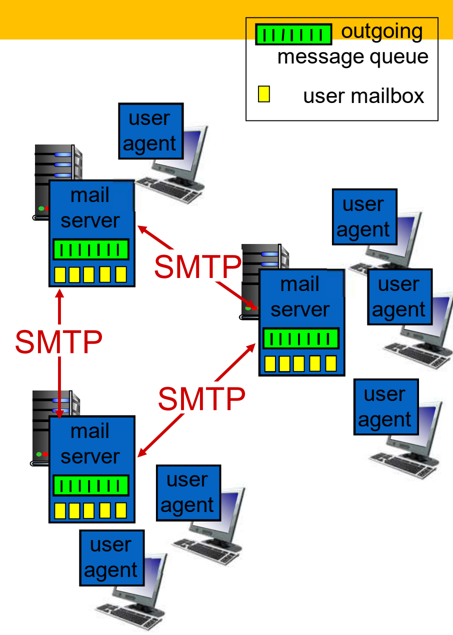
Electronic mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

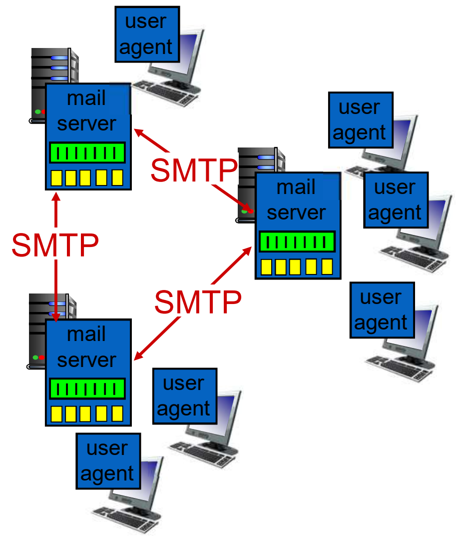
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



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

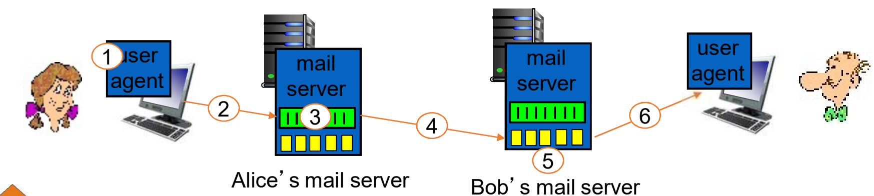


Electronic Mail: SMTP [RFC 2821]

- 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

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



Application Layer 2-29 Prodi Informatika

29

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
```

Application Layer 2-30 Prodi Informatika

30

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

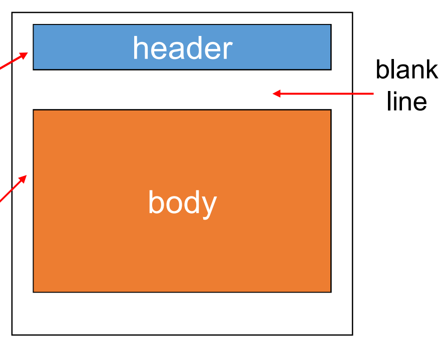
Mail message format

SMTP: protocol for exchanging email messages

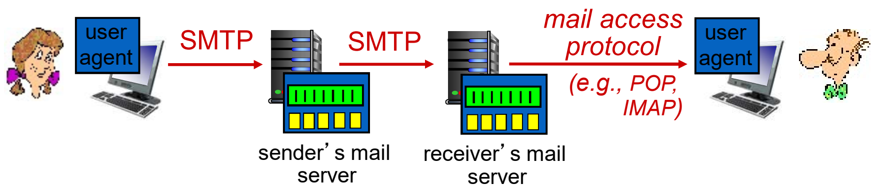
RFC 822: standard for text message format:

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

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



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.

Application Layer 2-33 Prodi Informatika

33

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
```

Application Layer 2-34 Prodi Informatika

34

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

Application Layer 2-35 Prodi Informatika

35

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"

Application Layer 2-36 Prodi Informatika

36

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?

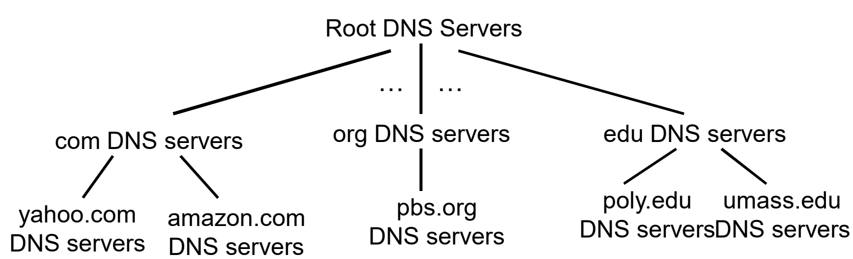
- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: doesn't scale!

Application Layer 2-37 Prodi Informatika

37

DNS: a distributed, hierarchical database



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

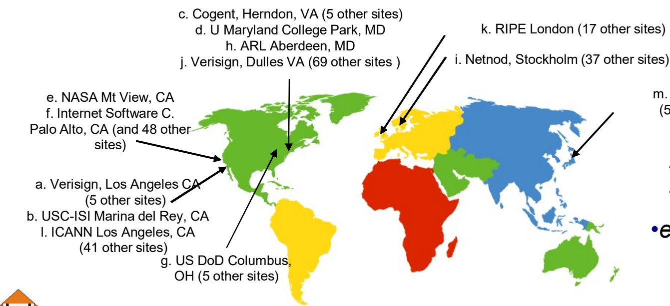
- client queries root server to find com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

Prodi Informatika

38

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

Application Layer 2-39 Prodi Informatika

39

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 2-40 Prodi Informatika

40

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 2-41 Prodi Informatika

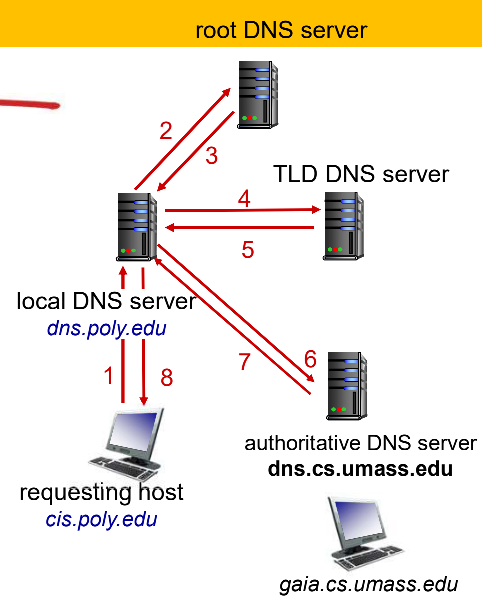
41

DNS name

- 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”



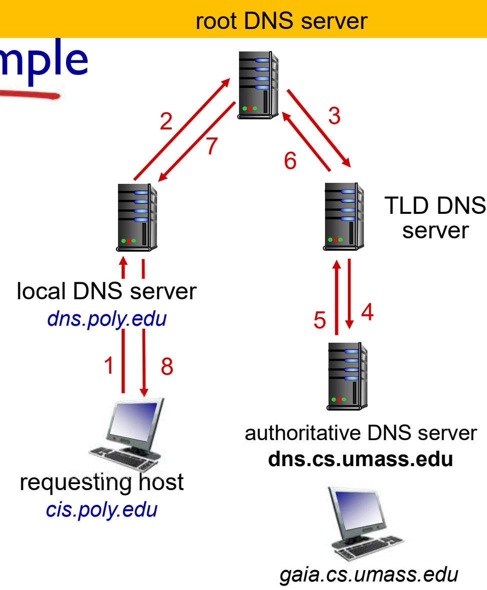
Application Layer 2-42 Prodi Informatika

42

DNS name resolution example

recursive query:

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



43

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

44

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**

Application Layer 2-45 Prodi Informatika

45

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

2 bytes	2 bytes
identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

Application Layer 2-46 Prodi Informatika

46

DNS protocol, messages

2 bytes	2 bytes
identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

name, type fields for a query — questions (variable # of questions)

RRs in response to query — answers (variable # of RRs)

records for authoritative servers — authority (variable # of RRs)

additional "helpful" info that may be used — additional info (variable # of RRs)

Application Layer 2-47 Prodi Informatika

47

Transport Layer

our goals:

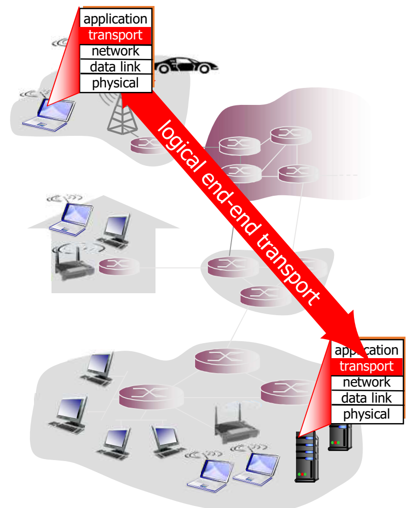
- understand principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- learn about Internet transport layer protocols:
  - UDP: connectionless transport
  - TCP: connection-oriented reliable transport
  - TCP congestion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

Transport Layer 3-48 Prodi Informatika

48

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



Transport vs. network layer

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

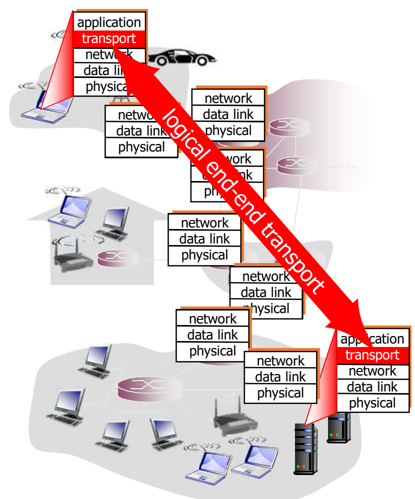
household analogy:

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

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



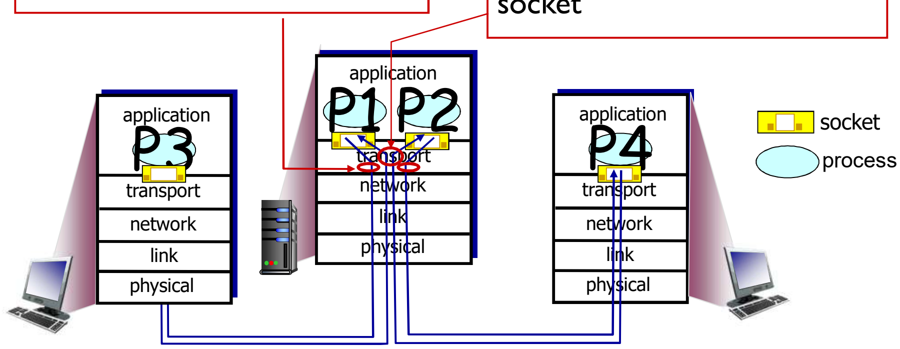
Transport Layer 3-51
Prodi Informatika

51

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

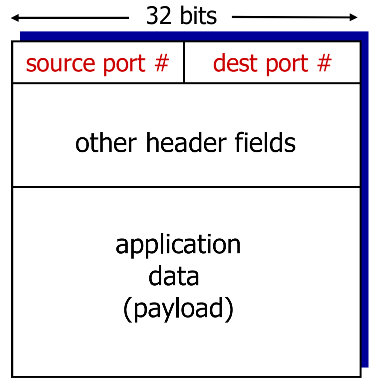


Transport Layer 3-52
Prodi Informatika

52

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

Transport Layer 3-53
Prodi Informatika

53

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 #

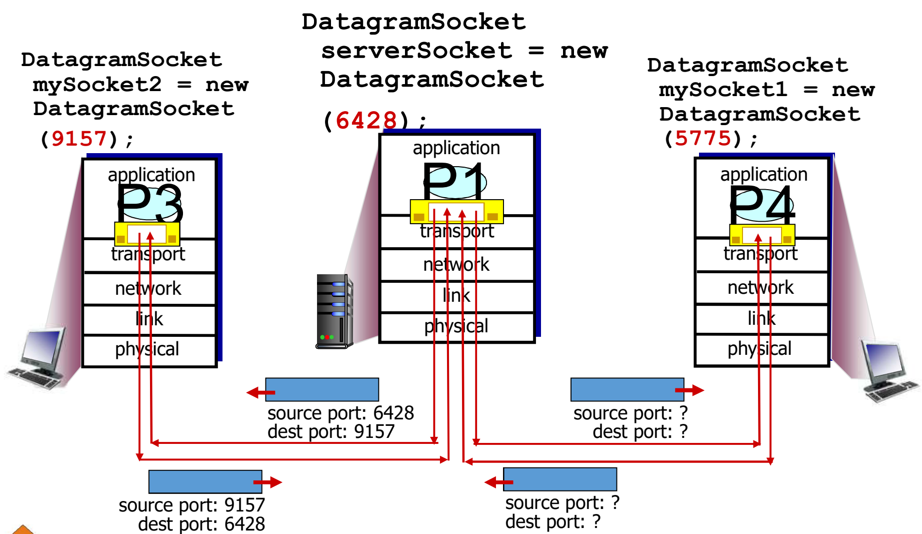
- when host receives UDP segment:
  - checks destination port # in segment
  - directs UDP segment to socket with that port #

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

Transport Layer 3-54
Prodi Informatika

54

Connectionless demux: example



Transport Layer 3-55 Prodi Informatika

55

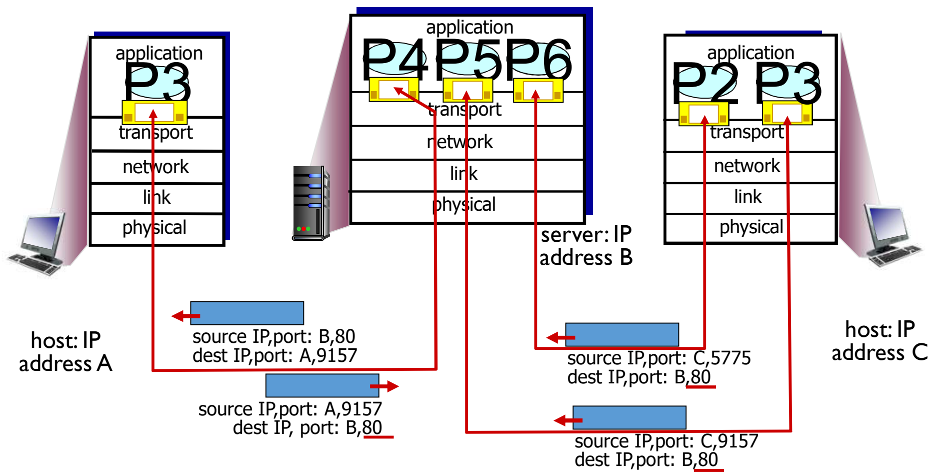
Connection-oriented demux

- 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

Transport Layer 3-56 Prodi Informatika

56

Connection-oriented demux: example

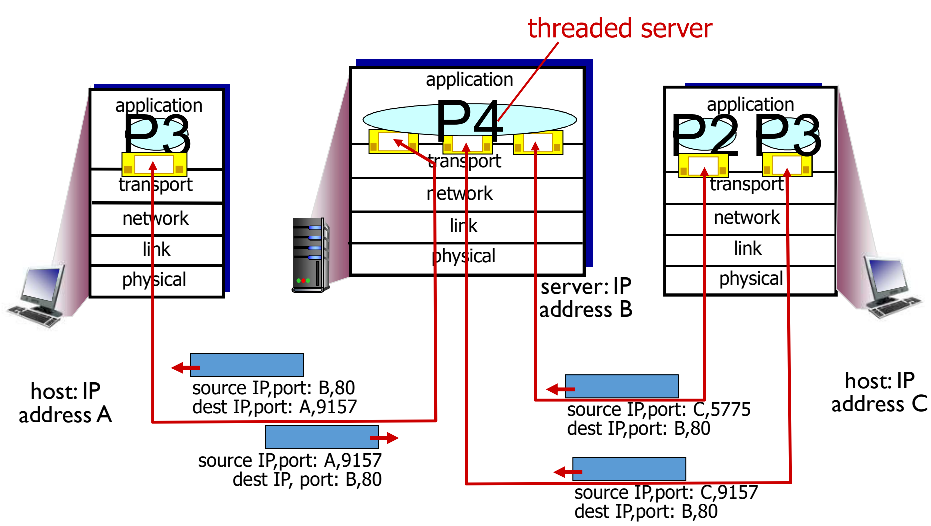


three segments, all destined to IP address: B, dest port: 80 are demultiplexed to *different* sockets

Transport Layer 3-57

57

Connection-oriented demux: example



Transport Layer 3-58

58