


Application Layer

CS 3516 - Computer Networks






Chapter 2: Application Layer


Goals:

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



Some network apps

- e-mail
- web
- instant messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video clips
- social networks
- voice over IP
- real-time video conferencing
- grid computing



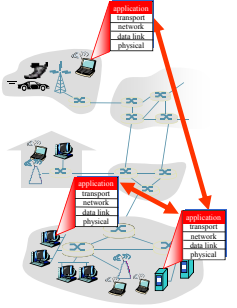

Creating a Network App

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

Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Socket programming with UDP
- 2.8 Socket programming with TCP

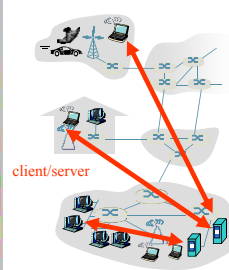


Application architectures

- Client-server (CS)
 - Including data centers / cloud computing
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P



Client-server Architecture



- server:**
- always-on host
 - permanent IP address
 - server farms for scaling
- clients:**
- communicate with server
 - may be intermittently connected
 - may have dynamic IP addresses
 - do not communicate directly with each other



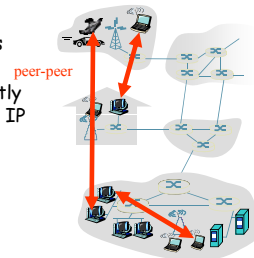
Server Example - Google Data Centers

- Estimated cost of data center: \$600M
- Google spent \$2.4B in 2007 on new data centers



Pure P2P Architecture

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



Highly scalable but difficult to manage



Hybrid of Client-server and P2P

- E.g. Skype
 - voice-over-IP P2P application
 - centralized server: finding address of remote party
 - client-client connection: often direct (not through server)
- E.g. Instant messaging
 - chatting between two users is P2P
 - centralized service: client presence detection/location
 - user registers its IP address with central server when it comes online
 - user contacts central server to find IP addresses of buddies



Processes Communicating

- Process:** program running within a host.
- Within same host, two processes communicate using **inter-process communication** (defined by OS).
 - Processes in different hosts communicate by exchanging **messages**

Client process: process that initiates communication

Server process: process that waits to be contacted

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



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 which brings message to socket at receiving process
- API: (1) choice of transport protocol; (2) ability to fix a few parameters (see Sockets slide deck)

Addressing Processes

- To receive messages, process must have **identifier**
- Host device has unique 32-bit IP address
- Exercise:** use ipconfig from command prompt to get your IP address (Windows)
- Q:** does IP address of host on which process runs suffice for identifying the process?
 - **A:** No, many processes can be running on same
- Identifier** includes both **IP address** and **port numbers** associated with process on host.
- Example port numbers:
 - HTTP server: 80
 - Mail server: 25

App-layer Protocol Defines

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

Public-domain protocols:

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

Proprietary protocols:

- e.g., Skype, ppstream

What Transport Service Does an App Need?

Data loss

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

Timing

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

Throughput

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

Security

- encryption, data integrity, ...

Transport Service Requirements of 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	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

Internet Transport Protocols Services

TCP service:

- connection-oriented:** setup required between client and server processes
- 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 guarantees, security

UDP service:

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

Q: why bother? Why is there a UDP?

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 (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP



Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Socket programming with UDP
- 2.8 Socket programming with TCP



Web and HTTP

First some jargon

- 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**
- Example URL:

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

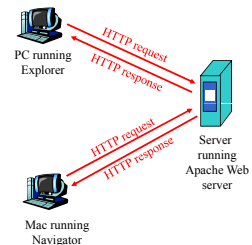
 host name path name



HTTP Overview

HTTP: hypertext transfer protocol

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



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

Protocols that maintain "state" are complex!

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



HTTP connections

Nonpersistent HTTP

- At most one object is sent over a TCP connection.

Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.



Nonpersistent HTTP

(contains text, references to 10 jpeg images)

Suppose user enters URL
www.someSchool.edu/someDepartment/home.index

- 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

Nonpersistent 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

Nonpersistent HTTP: Response time

Definition of RTT: time for a small packet to travel from client to server and back.

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

total = 2RTT+transmit time

Persistent HTTP

Nonpersistent 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

HTTP request message

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

```

request line
(GET, POST, HEAD commands)
GET /somedir/page.html HTTP/1.1
header lines
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
Carriage return line feed indicates end of message
(extra carriage return, line feed)
  
```

HTTP request message: general format

method	sp	URL	sp	version	cr	lf	request line
header field name	:	value	cr	lf	} header lines		
:	value	cr	lf				
cr	lf						
Entity Body							

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



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



HTTP response message

status line
(protocol
status code
status phrase)

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

header
lines

data, e.g.,
requested
HTML file

```
data data data data data ...
```



HTTP response status codes

In first line in server->client response message.
A few sample codes:

- 200 OK**
 - request succeeded, requested object later in this message
- 301 Moved Permanently**
 - requested object moved, new location specified later in this message (Location:)
- 400 Bad Request**
 - request message not understood by server
- 404 Not Found**
 - requested document not found on this server
- 505 HTTP Version Not Supported**



Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

Opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. Anything typed in sent to port 80 at cis.poly.edu

2. Type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.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!



User-server State: Cookies

Example:

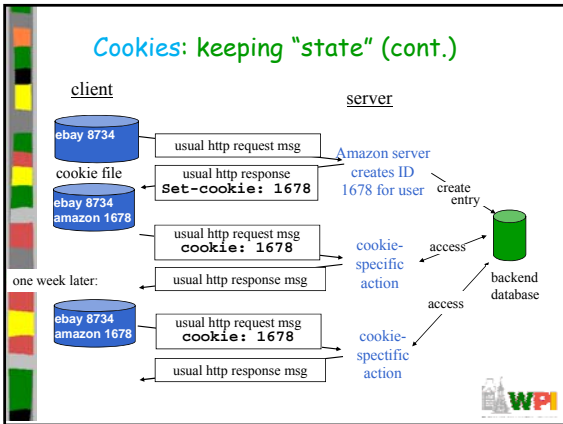
Many major Web sites use **cookies**

Four components:

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

- Susan always access Internet always 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





Cookies (continued)

What cookies can bring:

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

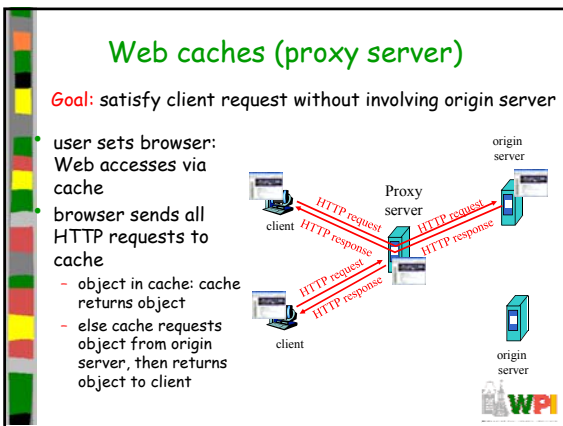
How to keep "state":

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

Cookies and privacy:

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

WPI



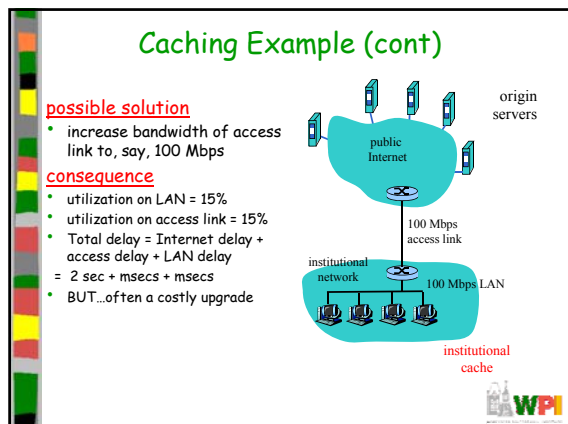
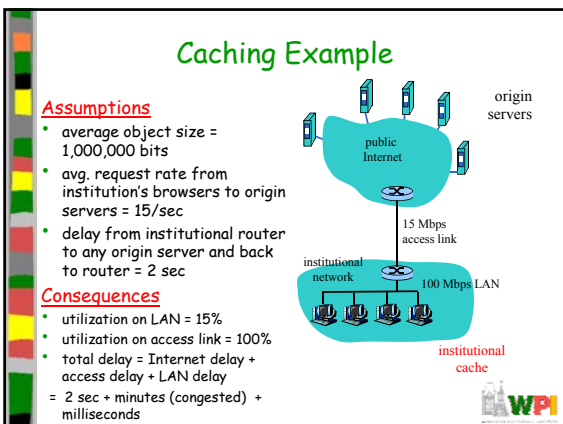
More About Web Caching

- Cache acts as both client and 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 (but so does P2P file sharing)

WPI



Caching example (cont)

possible solution: install cache

- suppose hit rate is 0.4

consequence

- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total avg delay = Internet delay + access delay + LAN delay = $.6 * (2.01 \text{ secs}) + .4 * \text{milliseconds} < 1.4 \text{ secs}$

Caching - Conditional GET

- Goal:** don't send object if cache has up-to-date cached version
- 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

Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Socket programming with UDP
- 2.8 Socket programming with TCP

Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Socket programming with UDP
- 2.8 Socket programming with TCP

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: map between IP addresses and name?

Domain Name System:

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

DNS

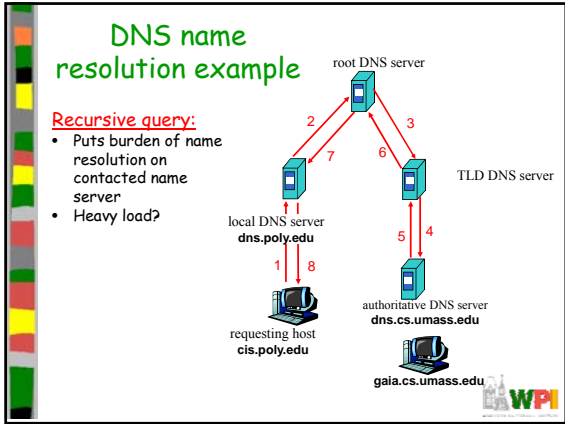
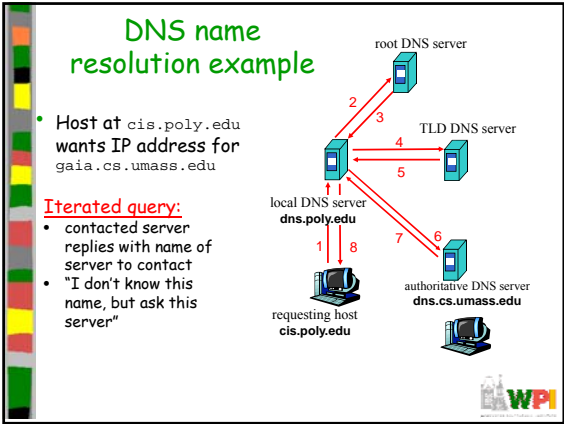
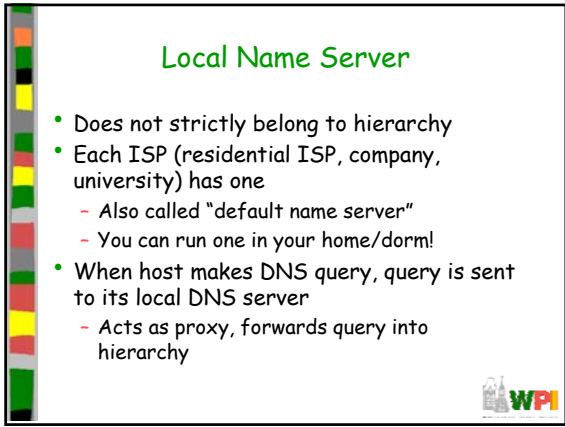
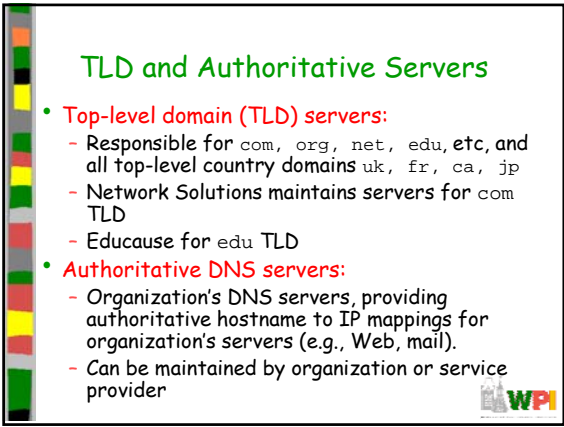
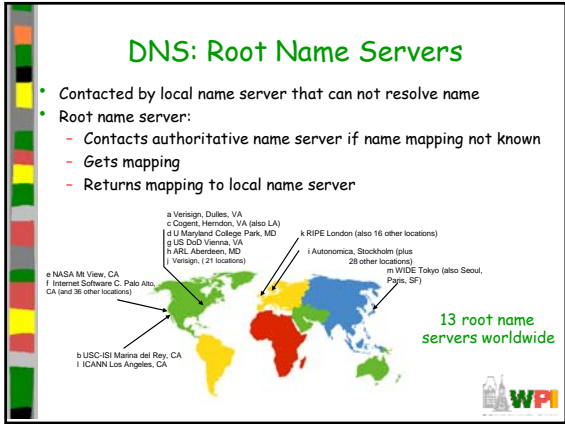
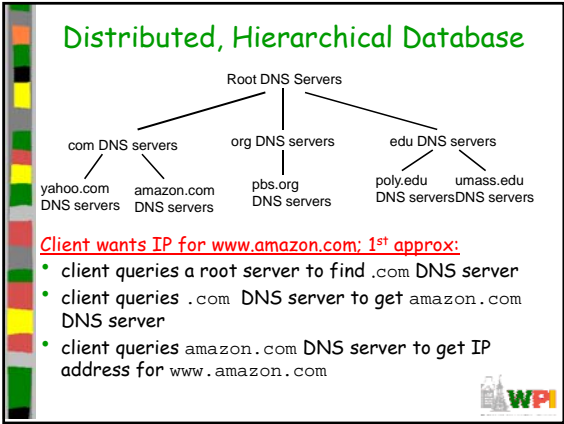
DNS services

- hostname to IP address translation
- host aliasing
 - Aliases, where canonical name is "real" name
- mail server aliasing
- load distribution
 - replicated Web servers: set of IP addresses for one name

Why not centralize DNS?

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

→ doesn't *scale!*



DNS: caching and updating records

- Once (any) name server learns mapping, it *caches* mapping
 - Cache entries timeout (disappear) after some time
 - TLD servers typically cached in local name servers
 - Thus root name servers not visited often
- Originally thought DNS names quite static, but increasingly not so → update/notify mechanisms under design by IETF
 - RFC 2136: <http://www.ietf.org/rfc/rfc2136.txt>



DNS Records

DNS: distributed db storing resource records (RR)

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

- Type=A
 - name is hostname
 - value is IP address
- 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=NS
 - name is domain (e.g. foo.com)
 - value is hostname of authoritative name server for this domain
- Type=MX
 - value is name of mailserver associated with name

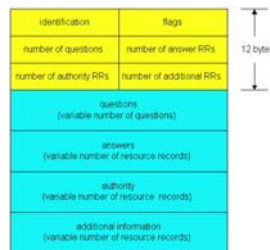


DNS protocol, messages

DNS protocol: *query* and *reply* messages, both with same *message format*

msg header

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



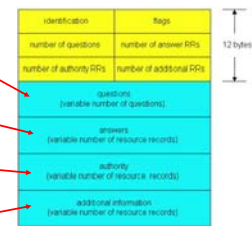
DNS protocol, messages

Name, type fields for a query

Resource records in response to query

Records for authoritative servers

Additional "helpful" info that may be used



Inserting records into DNS

- Example: new startup "Network Utopia"
 - How do people get IP address of your Web site?
 - How do they send you email?
- Register name networkutopia.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.networkutopia.com; Type MX record for networkutopia.com for mail



Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Socket programming with UDP
- 2.8 Socket programming with TCP



Pure P2P Architecture

- no always-on server
- Arbitrary end systems directly communicate
- Peers are intermittently connected and change IP addresses

peer-peer

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

WPI

File Distribution: Client-Server vs P2P

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

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

WPI

File Distribution Time: Client-Server

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

Time to distribute F to N clients using client-server approach

$$d_{cs} = \max \{ NF/u_s, F/\min(d_i) \}$$

increases linearly in N (for large N)

WPI

File Distribution Time: P2P

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

$$d_{p2p} = \max \{ F/u_s, F/\min(d_i), NF/(u_s + \sum u_i) \}$$

WPI

Client-Server vs P2P: Example

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

N	Client-Server Time (hours)	P2P Time (hours)
0	0	0.5
5	0.5	0.5
10	1.0	0.5
15	1.5	0.5
20	2.0	0.5
25	2.5	0.5
30	3.0	0.5
35	3.5	0.5

WPI

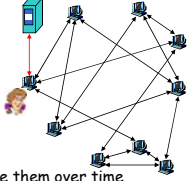
File Distribution: BitTorrent

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file

WPI

BitTorrent (1)



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

WPI

BitTorrent (2)

Pulling Chunks

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

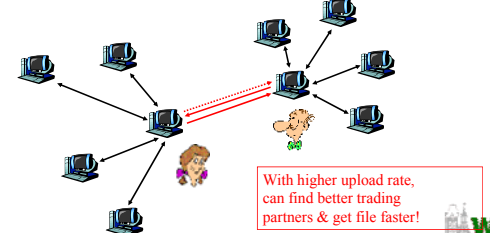
Sending Chunks: tit-for-tat

- Alice sends chunks to four neighbors currently sending her chunks *at the highest rate*
 - Re-evaluate top 4 every 10 secs
- Every 30 secs: randomly select another peer, starts sending chunks
 - Newly chosen peer may join top 4 (5 total)
 - "optimistically unchoke"

WPI

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



With higher upload rate, can find better trading partners & get file faster!

WPI

Distributed Hash Table (DHT)

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

WPI

DHT Identifiers

- Assign integer identifier to each peer in range $[0, 2^n - 1]$
 - Each identifier can be represented by n bits
- Require each key to be an integer in **same range**
- To get integer keys, hash original key
 - e.g., key = $h(\text{"Led Zeppelin IV"})$
 - This is why they call it a distributed "hash" table

WPI

How to Assign Keys to Peers?

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

WPI

Circular DHT (1)

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

WPI

Circle DHT (2)

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

Define closest as closest successor

Who's resp for key 1110?

WPI

Circular DHT with Shortcuts

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

WPI

Peer Churn

- To handle peer churn, require each peer to know IP address of its two successors.
- Each peer periodically pings its two successors to see if still alive
- Peer 5 abruptly leaves
- Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor.
- What if peer 13 wants to join?

WPI

P2P Case study: Skype

- Inherently P2P: pairs of users communicate
- Proprietary application-layer protocol (inferred via reverse engineering)
- Hierarchical overlay with Super Nodes (SNs)
- Index maps usernames to IP addresses; distributed over SNs

WPI


Peers as Relays

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

WPI

Chapter 2: Application layer


- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 **Socket programming with UDP**
- 2.8 **Socket programming with TCP**
- (See Sockets slide deck)



Chapter 2: Summary

Study of network apps now complete!

- Application architectures
 - client-server
 - P2P
 - hybrid
- Application service requirements:
 - reliability, bandwidth, delay
- Internet transport service model
 - connection-oriented, reliable: TCP
 - unreliable, datagrams: UDP
- specific protocols:
 - HTTP
 - DNS
 - P2P: BitTorrent, Skype
- socket programming



Chapter 2: Summary

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 being communicated

Important themes:

- control vs data msgs
 - in-band, out-of-band
- centralized vs decentralized
- stateless vs stateful
- reliable vs unreliable msg transfer
- "complexity at network edge"

