Application Layer

Abusayeed Saifullah

CS 5600 Computer Networks

These slides are adapted from Kurose and Ross
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
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)
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:**
- LAN utilization: 0.15%  
  - problem!
- access link utilization = 99%
- total delay = Internet delay + access delay + LAN delay  
  = 2 sec + minutes + usecs
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

consequences:
- LAN utilization: 0.15%
- access link utilization = 99%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs

Cost: increased access link speed (not cheap!)
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: 0.15%
- access link utilization = ?
- total delay = ?

*How to compute link utilization, delay?*

**Cost:** web cache (cheap!)
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 = 0.9 Mbps
  - Utilization = 0.9 / 1.54 = 0.58

- 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!)
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

  ![Diagram showing the flow of request and response with conditional GET](image-url)
Roadmap

1 principles of network applications
2 Web and HTTP
3 FTP
4 electronic mail
   - SMTP, POP3, IMAP
5 DNS
6 P2P applications
FTP: the file transfer protocol

- transfer file to/from remote host
- client/server model
  - client: side that initiates transfer (either to/from remote)
  - server: remote host
- ftp: RFC 959
- ftp server: port 21
FTP: separate control, data connections

- FTP client contacts FTP server at port 21, using TCP
- client authorized over control connection
- client browses remote directory, sends commands over control connection
- when server receives file transfer command, server opens 2nd TCP data connection (for file) to client
- after transferring one file, server closes data connection

FTP vs HTTP:
- 2 parallel TCP in FTP → control conn. is “out of band”
- FTP server maintains “state”: current directory, earlier authentication
FTP commands, responses

sample commands:
- sent as ASCII text over control channel
- USER username
- PASS password
- LIST return list of file in current directory
- RETR filename retrieves (gets) file
- STOR filename stores (puts) file onto remote host

sample return codes
- status code and phrase (as in HTTP)
- 331 Username OK, password required
- 125 data connection already open; transfer starting
- 425 Can’t open data connection
- 452 Error writing file
Roadmap

1 principles of network applications
2 Web and HTTP
3 FTP
4 electronic mail
   - SMTP, POP3, IMAP
5 DNS
6 P2P applications
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, FTP)
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCII
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
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
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)
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 msg
- SMTP: multiple objects sent in one msg
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 msgs on server
- **HTTP**: gmail, Hotmail, Yahoo! Mail, etc.
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**

```
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
```
**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
Roadmap

1 principles of network applications
2 Web and HTTP
3 FTP
4 electronic mail
   - SMTP, POP3, IMAP
5 DNS
6 P2P applications
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
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!*
**DNS: a distributed, hierarchical database**

Client wants IP for `www.amazon.com`; 1st time:

- 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`.

---

![Diagram of DNS servers hierarchy](image)
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 root name "servers" worldwide

a. Verisign, Los Angeles CA (5 other sites)
b. USC-ISI Marina del Rey, CA
c. Cogent, Herndon, VA (5 other sites)
d. U Maryland College Park, MD
e. NASA Mt View, CA
f. Internet Software C. Palo Alto, CA (and 48 other sites)
g. US DoD Columbus, OH (5 other sites)
h. ARL Aberdeen, MD
i. Netnod, Stockholm (37 other sites)
j. Verisign, Dulles VA (69 other sites)
k. RIPE London (17 other sites)
l. ICANN Los Angeles, CA (41 other sites)
m. WIDE Tokyo (5 other sites)
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
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
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”
DNS name resolution example

recursive query:
- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?
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
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=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
DNS protocol, messages

- *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

<table>
<thead>
<tr>
<th>identification</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td># questions</td>
<td># answer RRs</td>
</tr>
<tr>
<td># authority RRs</td>
<td># additional RRs</td>
</tr>
<tr>
<td>questions (variable # of questions)</td>
<td></td>
</tr>
<tr>
<td>answers (variable # of RRs)</td>
<td></td>
</tr>
<tr>
<td>authority (variable # of RRs)</td>
<td></td>
</tr>
<tr>
<td>additional info (variable # of RRs)</td>
<td></td>
</tr>
</tbody>
</table>
**DNS protocol, messages**

<table>
<thead>
<tr>
<th>identification</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td># questions</td>
<td># answer RRs</td>
</tr>
<tr>
<td># authority RRs</td>
<td># additional RRs</td>
</tr>
</tbody>
</table>

- **Identification**: 2 bytes
- **Flags**: 2 bytes
- **Questions (variable # of questions)**: # questions
- **Answers (variable # of RRs)**: # answer RRs
- **Authority (variable # of RRs)**: # authority RRs
- **Additional info (variable # of RRs)**: # additional RRs

**Details**:
- **Name, type fields** for a query
- **RRs in response to query**
- **Records for authoritative servers**
- **Additional “helpful” info that may be used**
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
  - 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
  - (networkutopia.com, 212.212.77.4, A)
  - (networkutopia.com, mail.networkutopia.com, MX)
Roadmap

1 principles of network applications
2 Web and HTTP
3 FTP
4 electronic mail
   - SMTP, POP3, IMAP
5 DNS
6 P2P applications
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)
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
**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
  - $d_{\text{min}} = \min$ client download rate
  - min client download time at least: $F/d_{\text{min}}$

\[
D_{c-s} \geq \max\{NF/u_s, F/d_{\text{min}}\}
\]

Increases linearly in $N$.
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: $F/d_{\text{min}}$
- **clients**: as aggregate must download $NF$ bits
  - max upload rate (limiting max download rate) is $u_s + \sum u_i$

Time to distribute $F$ to $N$ clients using P2P approach

$$D_{P2P} \geq \max\{F/u_s, F/d_{\text{min}}, NF/(u_s + \sum u_i)\}$$

... increases linearly in $N$ ...

... but so does this, as each peer brings service capacity
Client-server vs. P2P: example

client upload rate = $u$, $F/u = 1$ hour, $u_s = 10u$, $d_{min} \geq u_s$
P2P file distribution: BitTorrent

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

**tracker**: tracks peers participating in torrent

**torrent**: group of peers exchanging chunks of a file

Alice arrives ...
... obtains list of peers from tracker
... and begins exchanging file chunks with peers in torrent
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
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 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - “optimistically unchoke” this peer
  - newly chosen peer may join top 4
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!