Chapter 2 Application Layer

2.1 Principle of network applications

Application architecture

1. client - server

- server
  - always on host
  - permanent IP
  - data center for scaling
- client

  - communicate with server
  - dynamic IP
  - do not communicate directly with each other

  Process communication

  client process: initiates communication
  server process: listen and wait to be contacted

  To recieve message we need identifiers, which right here is IP addess and port numbers.

1. peer - to - peer (P2P)

- server: NO SERVER!!!!!!
- self scalability: new peers bring new service capacity, as well as new service demands
- complex management among peers inorder to maintain service

Transport services

What a transport service needs:

- data integrity: 100% reliable data transfer
- timing: some require low delay
- throughtput: some require minimum amount of throughput

1. TCP: reliable transport, flow control, congestion control, connection - oriented

2. UDP: unreliable data transfer, useful for streaming and internet telephone
3. SSL: encrypted TCP connection

2.2 Web and HTTP

Web page consists of objects. Object can be HTML file, JPEG image, JAVA applet, audio file ... Each object is addressable by URL, which consists of hostname and path name.

HTTP: HyperText Transfer Protocol

- client / server model
  - client: browser that requests and recieves
  - server: response to request

- Uses TCP, at port 80, flow as below:
  - client initiates TCP
  - server accepts TCP
  - HTTP holds exchanges between them
  - TCP close (continue discuss in persistent vs. non - persistent connection)

HTTP is STATELESS, which means server maintain no information about past client request.

Persistent connection / Non - persistent connection

Non - persistent HTTP: one object sent over TCP connection. Multiple objects requires multiple connection. Persistent HTTP: Multiple opjects can be sent over a single TCP connection.

In a perspective of RTT, non - persistent connection requires 2 RTT per object, while persistent connection will only need 1 RTT per object when client - server maintains connected.

HTTP message format

- request line: method, URL, version, respectively
  - method POST: input is uploaded to server
  - method GET: request in URL
  - method HEAD: ask server to leave requested object out of response
  - method PUT (HTTP / 1.1 add) : upload files in entity
  - method DELETE (HTTP / 1.1 add) : request to delete file in URL

- response line: version, status, status phrase
  - status 200: OK
  - status 301: Moved permanently, new location specified in this msg
  - status 400: Bad request, request msg not understood

- status 404: Not found, requested document not found
- status 505: HTTP version not supported

- header lines: header fields and its values
  - Connection: Non - persistent connection
  - Keep-Alive (seconds): Persistent connection

- blank line: literally blank
- entity body: what to transfer

User - server interaction: cookies

It is often desirable for websites to recognize users.

Cookie component (respectively)

- cookie header in HTTP response, server assigns a cookie ID for client
- cookie header in HTTP request, for clients to request cookie data from server backend
- cookie file kept by host
- back - end database for server

Usage:

- authorization
- shopping cart
- recommendation
- user - session state

Web caching (proxy server)

Goal: satisfy client request without involving origin server.

Flow:

- client initiates a TCP connection and HTTP request to web cache
- web cache check for locally cached data, returns in HTTP format
- else web cache TCP connects origin server for the request
- web cache recieves and copy to local for future use and send it back to user

Cache is both a server and a client at the same time.

Why web - cache

- 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)

2.3 SMTP, POP3, IMAP

Email components :

- user agent
- mail server
- simple mail transfer protocol (SMTP)

SMTP (delivery to server)

- TCP, port 25
- three phase:
    - handshake
    - tranfer
    - closure

- persistent connection
- requires message (header & body) to be in 7-bit ASCII

POP, IMAP, HTTP (retrieval from server)

- Post Office Protocol (POP) : authorization
- Internet Mail Access Protocol (IMAP)
- HTTP

2.4 DNS

The domain name system (DNS) implemented by the hierarchy of DNS servers (name servers).

DNS services: host to IP address translation.

- host aliasing
- mail server aliasing:
- load distribution: a set of IP addresses is thus associated with one canonical hostname. Server responds with the entire set of IP addresses, but rotates the ordering of the addresses within each reply.

DNS database

- hierarchical: (three main level)
    - Root DNS server

    - top - level DNS server: (com / org / edu) DNS server
    - authorative DNS server:(yahoo.com / pbs.org / ntu.edu) DNS server

- client wants IP for www.amazon.com

    - queries Root server for .com DNS server
    - then queries .com for amazon.com DNS server
    - then queries amazon.com for IP of www.amazon.com

- local DNS name server
    - each ISP has one
    - when host makes DNS query, query is sent to its local DNS server

DNS records

These are resource records (RR).

1. type = A
    - name: hostname
    - value: IP address

2. type = NS
    - name: domain
    - value: hostname of authoritative name server for this domain

3. type = CNAME
    - name: alias for "canonical" (real) name
    - value: canonical name

4. type = MX
    - name: some name
    - value: name of mailserver associated with name

2.5 P2P applications

Pure P2P architecture:

- No always - on server
- arbitrary end systems directly communicate

client - server vs. P2P

How much time to distribute file (size F) from one server to N peers?

server:

- upload capacity: u_s

client i :

- upload capacity: u_i
- download capacity: d_i

client - server

1. Server: must sequentially send (upload) N, F file copies:
   - time to send one copy: F/us
   - time to send N copies: NF/us

2. Client: each client must download file copy
   - d_{min} = min client download rate
   - min client download time: F/d_{min}

Distribution time: $D_{c-s} \geq max\{NF/u_s, F/d_{min}\}$

P2P

1. Server: must upload at least one copy
   - time to send one copy: F/us

2. Client: each client must download file copy
   - min client download time: F/d_{min}

3. Clients as aggregate must download $N * F$ bits.

   - max upload rate $u_s + \Sigma u_i$

Distribution time: $D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$

BitTorrent

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

- tracker: tracks peers participating in torrent (role of manager)

- torrent: group of peers exchanging chunks of a file

Address problems for BitTorrent: (考古出現)

- For upload between peers to be interchangeable they need to have the same data at the same time. BitTorrent solves this by making swarms consisting of all peers trying to download the same file.

- For a peer to provide data which it isn't interested in then the transfer of data to it in the first place is in some sense wasted. BitTorrent solves this by not having peers download anything which they themselves don't want.
- Peer transfer rate capacities are unknown and changing in real time. This is handled by using TCP-style rate limiting and transferring over several connections at once.
- Peers are untrusted. This is fixed by using secure hashes to authenticate data before accepting it.
- Latency between peers is fairly high. This is mostly solved by copping out and doing complete downloads before starting playback. Playing while streaming is a more interesting problem, and further improvement is possible.

2.6 2.7 Not in exam, so no note ^_^

Have learned client / server socket programming through project 1.