CS118 Discussion 1D, Week 3

Zengwen Yuan
Boelter Hall 2760, Friday 4:00—5:50 p.m.
Outline

• Application Layer Protocol: DNS, CDN, P2P

• Transport Layer Protocol: UDP, principles of reliable transport protocol

• HW2 and Project 1 clarification
Application Layer: protocols

• DNS:
  • What is the transport layer protocol?
  • How the scalability is achieved?
  • Who will use iterative/recursive query?
  • Why is DNS resolver needed?
Application Layer: protocols

- DNS: convert hostname to IP address (and more)
- A distributed and hierarchical database
  - Root DNS servers (a—m)
  - Top-level domain (TLD) servers
  - Authoritative DNS servers
  - local DNS server (caching resolver, stub resolver)
DNS protocol: exercise

• Assume the caching resolver’s cache is empty initially

• Host A queries www.ucla.edu, how many queries should the caching resolver issue?

• After A’s DNS query, host B queries www.mit.edu, how many queries should the caching resolver issue?
DNS protocol: exercise

• Assume the caching resolver’s cache is empty initially

• Host A queries www.ucla.edu; how many queries should the caching resolver issue?

• After A’s DNS query, host B queries www.mit.edu; how many queries should the caching resolver issue?
DNS protocol: exercise

• Assume the caching resolver’s cache is empty initially

• Host A queries www.ucla.edu, how many queries should the caching resolver issue?

• After A’s DNS query, host B queries www.mit.edu, how many queries should the caching resolver issue?

www.ucla.edu
$ dig google.com
; <<>> DiG 9.8.3-P1 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44777
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 4, ADDITIONAL: 4

;; QUESTION SECTION:
google.com.            IN  A

;; ANSWER SECTION:
google.com. 76 IN A 172.217.2.14

;; AUTHORITY SECTION:
google.com. 85950 IN NS ns1.google.com.
google.com. 85950 IN NS ns2.google.com.

;; ADDITIONAL SECTION:
ns1.google.com. 59591 IN A 216.239.32.10
ns2.google.com. 50756 IN A 216.239.34.10
ns3.google.com. 40354 IN A 216.239.36.10
ns4.google.com. 36005 IN A 216.239.38.10

;; Query time: 84 msec
;; SERVER: 158.69.209.100#53(158.69.209.100)
;; WHEN: Thu Jan 19 20:37:48 2017
;; MSG SIZE  rcvd: 180
$ dig any mit.edu
$ dig 206.5.217.172.in-addr.arpa
Application Layer: protocols

- P2P: no always-on server, peers are intermittently connected

- Calculate content distribution time

\[ D_{cs} = \max\left(\frac{NF}{\mu_s}, \frac{F}{d_{min}}\right) \]

\[ D_{p2p} = \max\left(\frac{F}{\mu_s}, \frac{F}{d_{min}}, \frac{NF}{\mu_s + \sum_i \mu_i}\right) \]
Transport Layer
Transport Layer V.S. Network Layer

- Network layer: logical communication between **hosts**
  - **IP address** is used for identifying a host

- Transport layer: logical communication between **processes**
  - **IP address and port number** are used for identifying a process
Multiplexing and De-multiplexing

• Multiplexing at send host: gather data from multiple sockets

• De-multiplexing at receiving host: deliver received segments to the right socket

• **Five tuples** (src_ip, src_port, dst_ip, dst_port, protocol) are used for multiplexing/demultiplexing

  • How to identify a TCP/UDP socket? `lsof -i`

  • Can TCP and UDP share the same port numbers? **Yes!** e.g. DNS
UDP

- No connection establishment
- No connection state
- Small packet overhead (8 byte)
- How to calculate checksum?
  - **Pseudo header** + **UDP header** + data
  - Also applicable to TCP
  - Why pseudo header?
Principles of Reliable Data Transfer

- How to deal with bit errors?
  - Error detection (e.g. checksum)
  - Receiver feedback
  - Retransmission
  - Why not error correction?

- How to deal with duplicate packets due to retransmission? **Sequence number**

- How can the sender detect that ACK or data is lost? **Timer**
Serve Multiple TCP Connections Simultaneously

• Problem: `accept()` works under **blocking mode** by default
  • Unless a new connection request arrives, `accept()` will not return
• Perquisite: `listen()` allows multiple TCP connection
• Three approaches
  • `fork()`: each connection is served by a new process
    • Easy to write, but expensive and hard to share data between processes
  • POSIX pthread: each connection is served by a new thread
    • Can be hard to maintain
• **Non-blocking mode**: use `select()`
What is `select()`?

- A monitor for multiple sockets (or fd, file descriptors)
  - Given a set of sockets, if any of them were ready to receive/send, `select()` would exit

- `int select (int numfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);`
  - `numfds`: the highest file descriptor plus one
  - `readfds, writefds, exceptfds`: set of sockets
  - `timeout`: timer for `select` to exit without any changes
  - return when some sockets are ready, or timeout
  - return value: the number of sockets that are ready
What is `fd_set`?

- A set of sockets (or file descriptors) that will be monitored by `select()`

- Macros of set operation
  - `FD_SET(int fd, fd_set *set);` // add fd to the set
  - `FD_CLR(int fd, fd_set *set);` // remove fd from the set
  - `FD_ISSET(int fd, fd_set *set);` // return if fd is in the set
  - `FD_ZERO(fd_set *set);` // clear all entries in the set
How to use `select()`?

- Assume the server has created a socket `sock`, which is bound with server’s IP address and port number.

```c
fd_set active_fd_set; // set for holding sockets
int new_sock; // socket representing client
/* Initialize the set of active sockets */
FD_ZERO(&active_fd_set);
/* put sock to the set, s.t. we can monitor whether a new connection request arrives */
FD_SET(sock, &active_fd_set);

while (1) {
    /* Block until some sockets are active. Let N is #sockets+1 in active_fd_set */
    if (select(sock + 1, &active_fd_set, NULL, NULL, NULL) < 0) {
        exit(-1); // error
    }
    /* Now some sockets are active */
    if (FD_ISSET(sock, &active_fd_set)) { // new connection request
        new_sock = accept(sock, (struct sockaddr*) &client_addr, sizeof(client_addr));
        FD_SET(new_sock, &active_fd_set);
    }
    /* Decide whether client has sent data */
    if (FD_ISSET(new_sock, &active_fd_set)) {
        /* receive and process data here */
    }
} // end of while
```