CS118 Discussion 1B

Week 1

Zhiyi Zhang (zhiyi@cs.ucla.edu)

About me

- 4th year Ph.D. candidate in Internet Research Lab (IRL)
- How to reach me: Email to zhiyi@cs.ucla.edu
- Welcome to visit my personal website: https://zhiyi-zhang.com
- Our lab is recruiting students
  - Who have spare time
  - Who are interested in reference letters, capstone projects, research
  - Who are interested in networking and/or network security

Contents

- Logistics
- Computer Networking: Why bother?
- Important Concepts Review
- Network programming: A quick start

Logistics

Academic Integrity Agreement and Time Survey

- Submit your signed Academic Integrity Agreement to GradeScope
  - 1. Go to https://www.gradescope.com/.
  - 2. Choose to sign up as a "Student". For the course entry code, fill in: **932KV3**.
  - 3. Fill in your name (Official name on CCLE preferred), email, and UID. For the UID, just put in the 9-digit string WITHOUT SPACE. Please make sure the UID is correct. Your final grades will be uploaded from Gradescope by the UID.

- Fulfill the time slot survey so as to find common available time slots for Quizzes

Grading Breakdown

- Homework 20%
- Projects 20%: All in C/C++
    - Project 1: web server
    - Project 2: implement TCP-like reliable transmission over unreliable UDP
        - Phase I: TCP connection setup/teardown
        - Phase II: Reliable transmission
        - Bonus credit 3% if TCP's selective repeat is implemented
- Four Quiz 60%

Homework and Quizzes Requirement

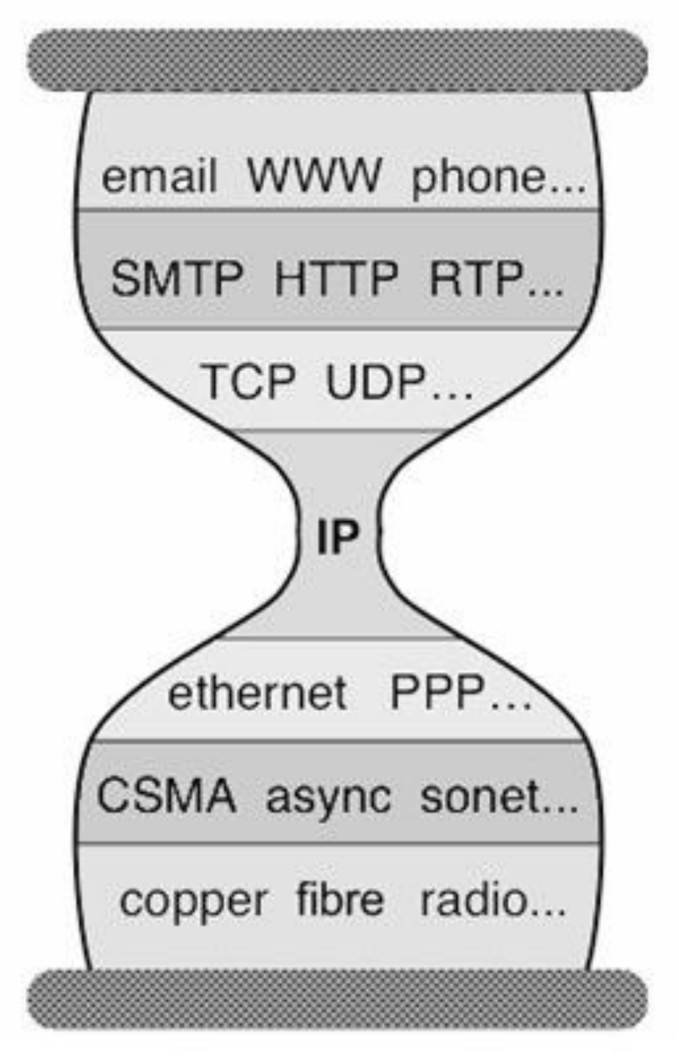
- Hard deadline
  - Questions posted on every Wednesday
  - Due by 6PM next Thursday
  - Solution posted one hour after due time
- We will use GradeScope for all online homework/quiz submission
- Format depends on each homework assignment and quiz
- Quiz time will be decided later: because of students from different time zones

Course Projects Requirement

- Hard deadline
  - (tentative) April 24, Friday
  - (tentative) June 5, Friday
- No team this quarter because of COVID-19
- Environment: Ubuntu Linux
- Set up your own development environment ASAP
  - If you have native ubuntu laptop/server, you don’t need extra preparation
  - If you have windows/mac OS, you can install a Ubuntu virtual machine (already covered by CS35L)
  - You can also use UCLA servers by SSH (already covered by CS35L)

Schedule

- Part 1. Introduction
- A top-down approach: network architecture
    - Part 2. Application layer
        - Quiz 1
    - Part 3. Transport layer
        - Quiz 2
    - Part 4. Network layer
        - Quiz 3
    - Part 5. Link layer & LAN
- Part 6. Wireless and mobile networks
- Part 7. Network Security
    - Quiz 4



Computer Networks: Why bother?

Computer Networking

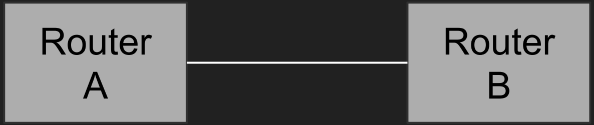
- It matters a lot
  - It is being used by us all: we live in a connected world, e.g., we are using zoom now :D
  - As a computer science student/engineer/researcher, we need to know **how** machines are connected, **why** they are connected in the current way, **what** can we do to further develop computer networking technologies
- Cutting edges in computer networking
  - Super high-throughput wireless technologies like mmWave used in 5G
  - Internet of Things: vehicular networking, smart home, smart city
  - Network security: authenticity, availability, integrity, privacy
  - Ethereum, BTC, etc. require privacy-preserving, distributed network
  - Data centric networking like Named Data Networking (NDN)

Review of Important Concepts

Delay

- Transmission Delay = L/R
- Propagation Delay = d/s
- Queuing Delay
- Processing Delay

Link length: 100km
Bandwidth: 1Mbps
Packet Size: 1000bits
Queuing: 2 packets
Processing time: ignored



Example Question:

- Propagation Delay = 100km/(2*10^8m/s) = 0.5ms
- Transmission Delay = 1000bits/1Mbps = 1ms
- Queuing Delay = 2*(1ms +T_p) = 2ms
- Total Delay = 3.5ms

Network Programming

Basics about network programming

- Asynchronous programming v.s. Synchronous programming
  - Async programming is common for network programming because packets does not come back immediately -- wait for your next packet (event) and react (callback)
  - There are also sync programming, e.g., busy waiting for network response or timeout
- A typical communication model: Client-Server model
  - **Client**: request data
    - Init communication
    - Send request and wait for response from the server
    - E.g., your web browser
  - **Server**: respond data
    - Reachable by clients
    - Wait for requests from clients and process requests
    - E.g., Youtube server
- There are other communication models, e.g., peer to peer

Network Byte Order

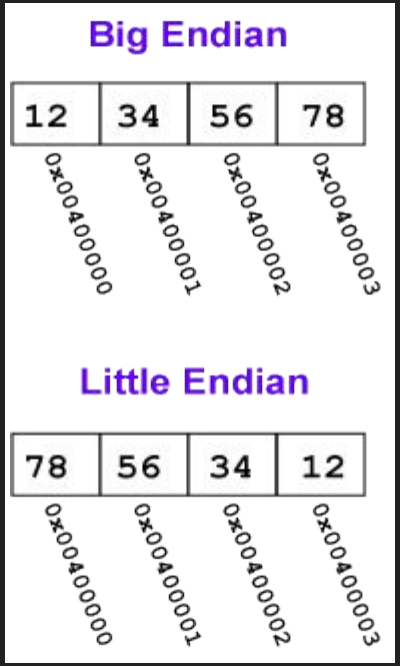
Two ways to store bytes in memory, e.g., 0x12345678

- Starts with least significant byte (little endian)
- Starts with most significant byte (big endian)

Internet protocols use big-endian ordering: how to translate?

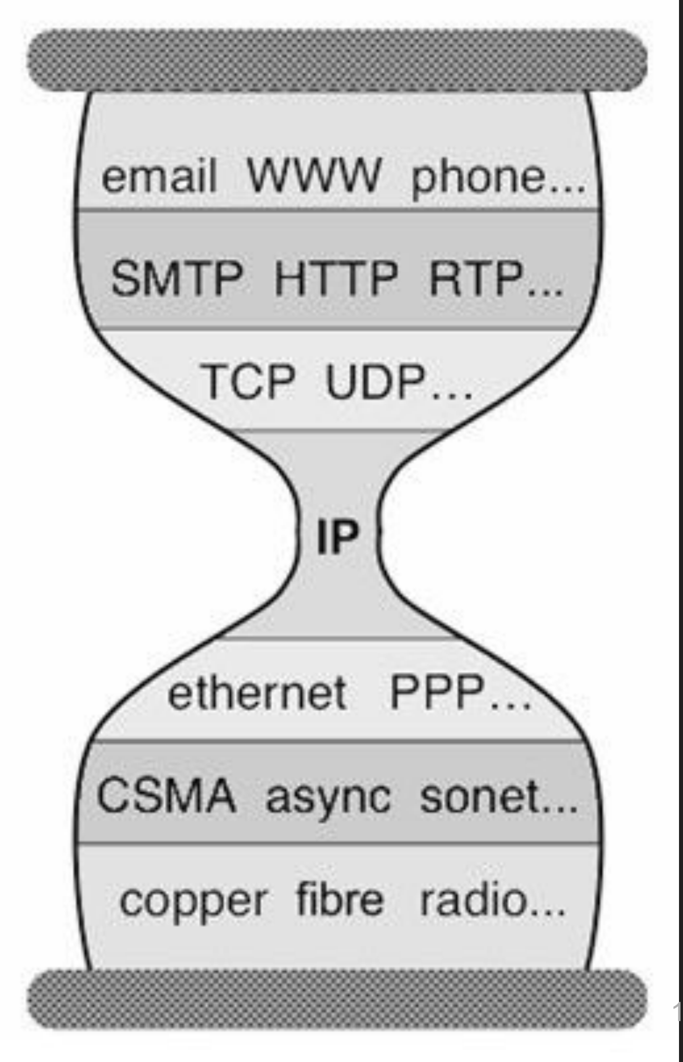
```
#include <netinet/in.h>

uint16_t htons(uint16_t host16bitvalue);
uint32_t htonl(uint32_t host32bitvalue);
uint16_t ntohs(uint16_t net16bitvalue);
uint32_t ntohl(uint32_t net32bitvalue);
```



Where we start

- Application layer
  - What network functions can we use?
- Transport layer abstracts the network for applications
  - TCP, UDP
- What are the interfaces for programmers
  - **Socket**



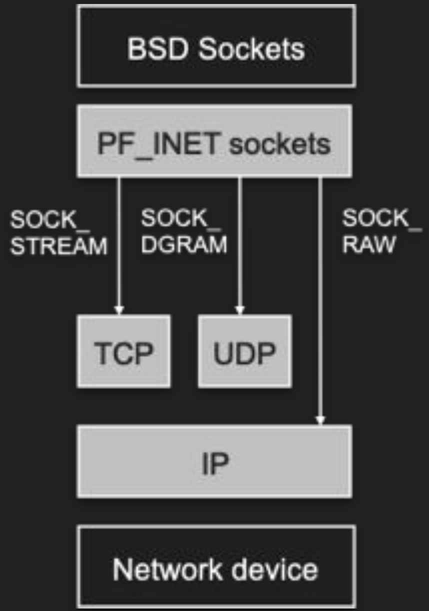
Before we dive in to socket, brief intro to TCP/UDP

- TCP: Transmission Control Protocol
    - Reliable data transfer
        - Reliable delivery
        - In order
    - With congestion control: when network is congested, send less packets
- UDP: User Data protocol
    - Best effort
        - No guarantee on reliable, in-order packet delivery
        - No congestion control

More details will covered by later lectures

Socket Programming

- Socket: <IP Address, Port Number>
  - IP identifies a machine, port identifies a process
  - Therefore, socket identifies a process on a machine
    - e.g., youtube web server process on a Google’s cloud server, the web browser on your laptop
- IP address: assigned by your local network, organization, company, etc.
  - A laptop at home: 192.168.0.10
  - A google server: 172.217.5.206
- Port number: used by processes with conventions
  - HTTP: 80, SSH: 22, HTTPS: 443
  - Web development: 8080? (😆actually you can use whatever port number >= 2014)



Socket Programming: Socket Structs

- socketaddr_in: socket address structure for Internet
- socketaddr: generic socket address structure
    - Why generic?
        - socketaddr_in: ipv4
        - socketaddr_in6: ipv6
        - socketaddr_un: unix socket

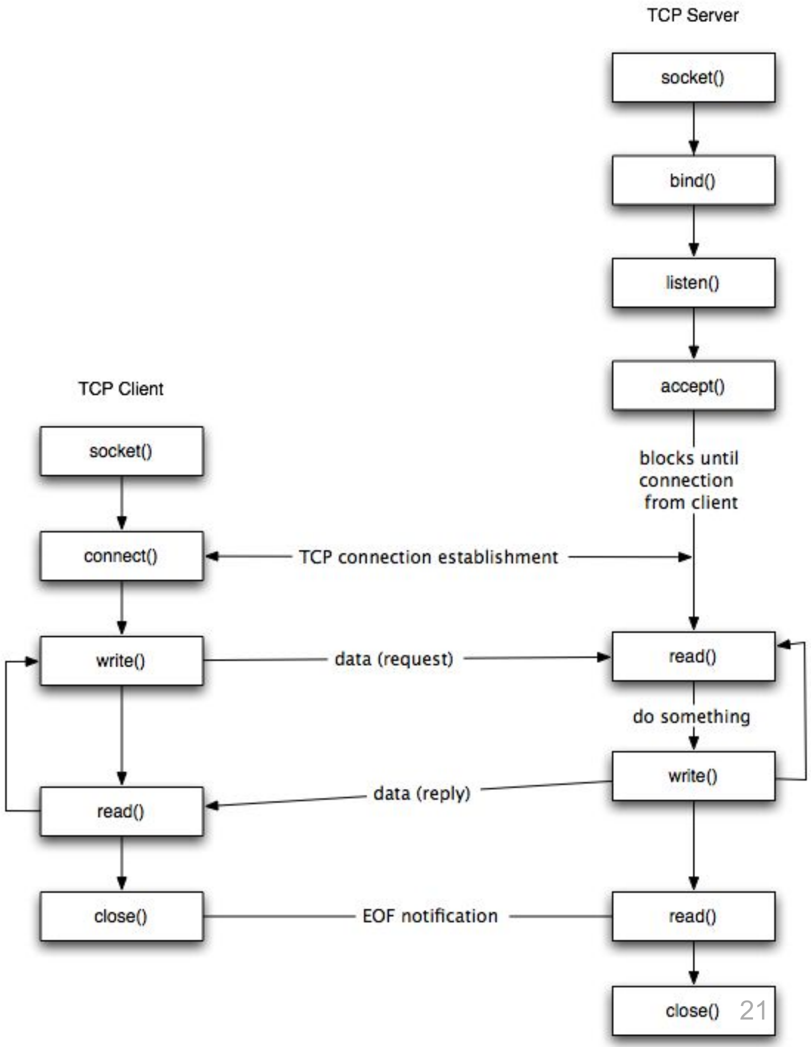
```
struct in_addr{
in_addr_t s_addr; /*32 bit IPv4 network byte ordered address*/
};


struct sockaddr_in {
   uint8_t sin_len; /* length of structure (16)*/
   sa_family_t sin_family; /* AF_INET*/
   in_port_t sin_port; /* 16 bit TCP or UDP port number */
   struct in_addr sin_addr; /* 32 bit IPv4 address*/
   char sin_zero[8]; /* not used but always set to zero */
 };

 struct sockaddr {
    uint8_t sa_len;
    sa_family_t sa_family; /* address family: AD_xxx value */
    char sa_data[14];
 };
```

TCP Socket Workflow

- Server waiting for connection
- Connection setup
- Data transmission
- Connection tear down



TCP Socket Programming: APIs

Client:

1. Create a socket using the **socket()** function;
2. Connect the socket to the address of the server using the **connect()** function;
3. Send and receive data by means of the **read()** and **write()** functions.
4. Close the connection by means of the **close()** function.

Server:

1. Create a socket with the **socket()** function;
2. Bind the socket to an address using the **bind()** function;
3. Listen for connections with the **listen()** function;
4. Accept a connection with the **accept()** function system call. This call typically blocks until a client connects with the server.
5. Send and receive data by means of **write()** and **read()**. (send(), receive())
6. Close the connection by means of the **close()** function.

Socket

```
#include <sys/socket.h>

 int socket (int family, int type, int protocol);
```

- Used by a server or a client to create the socket handler
  - family: protocol family AF_INET for IPv4
  - type: constant describing the socket type
    - SOCK_STREAM for stream sockets (TCP)
    - SOCK_DGRAM for datagram sockets (UDP)
  - protocol: protocol used with socket. Normally only a single protocol exists to support particular socket type within a given protocol family, in which case protocol can be specified as 0

Bind

```
#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);
```

- Used by a server to assign a local protocol address to a socket
    - sockfd: socket handler
    - servaddr: socket address
    - addrlen: socket address struct length

Listen

```
#include <sys/socket.h>

 int listen(int sockfd, int backlog);
```

- Used by a server to convert the socket into passive state and wait for connections (rather than initiating a connection)
    - sockfd: socket handler
    - backlog: the maximum number of connections this program can serve simultaneously

Accept

```
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);
```

- Used by a server to accept a new connection
  - sockfd: socket handler
  - client_addr: socket address of the client returned from the call
  - addrlen: client socket address struct length

Connect

```
#include <sys/socket.h>

int connect (int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);
```

- Used by a client to establish a connection with a TCP server
  - sockfd: the socket handler
  - servaddr: socket address
  - addrlen: socket address struct length

Read and Write

```
int write(int sockfd, char* buf, size_t nbytes);
int read(int sockfd, char* buf, size_t nbytes);
```

- Used by a server or a client to read/write data to the socket
    - sockfd: socket handler
    - buf: payload
    - nbytes: payload size in bytes
- Return the # bytes read/write from/to the socket

Close

```
#include <unistd.h>

int close(int sockfd);
```

- Used by a server or a client to close the connection
  - sockfd: socket handler
  - servaddr: socket address
  - addrlen: socket address struct length

To learn more of these APIs

- Use Linux manual as your reference when coding
  - E.g., type in **man socket** in your Linux/MacOS terminal
- Online tutorial
  - Google
  - https://www.cs.dartmouth.edu/~campbell/cs60/socketprogramming.html

Some useful tools

- Translate a host name (e.g., www.google.com) to an IP address with **gethostbyname()**
- Translate an IP address to a host name with **gethostbyaddr()**
- Translate an IP address to a string (char*) in the format of “192.168.0.1” with **inet_ntoa()**
- Translate a string (char*) in the format of “xxx.xxx.xxx.xxx” to an IP address by **inet_addr()** or **inet_aton()**

Use man to learn the details of these functions and use them to simplify your program!

Let’s get our hands dirty