CS118 Discussion 1B, Week 6

Boyan Ding

Outline

- Network Layer
  - Overview: data v.s. control plane
  - IPv4/IPv6, DHCP
- Project 2 overview

Network layer: overview

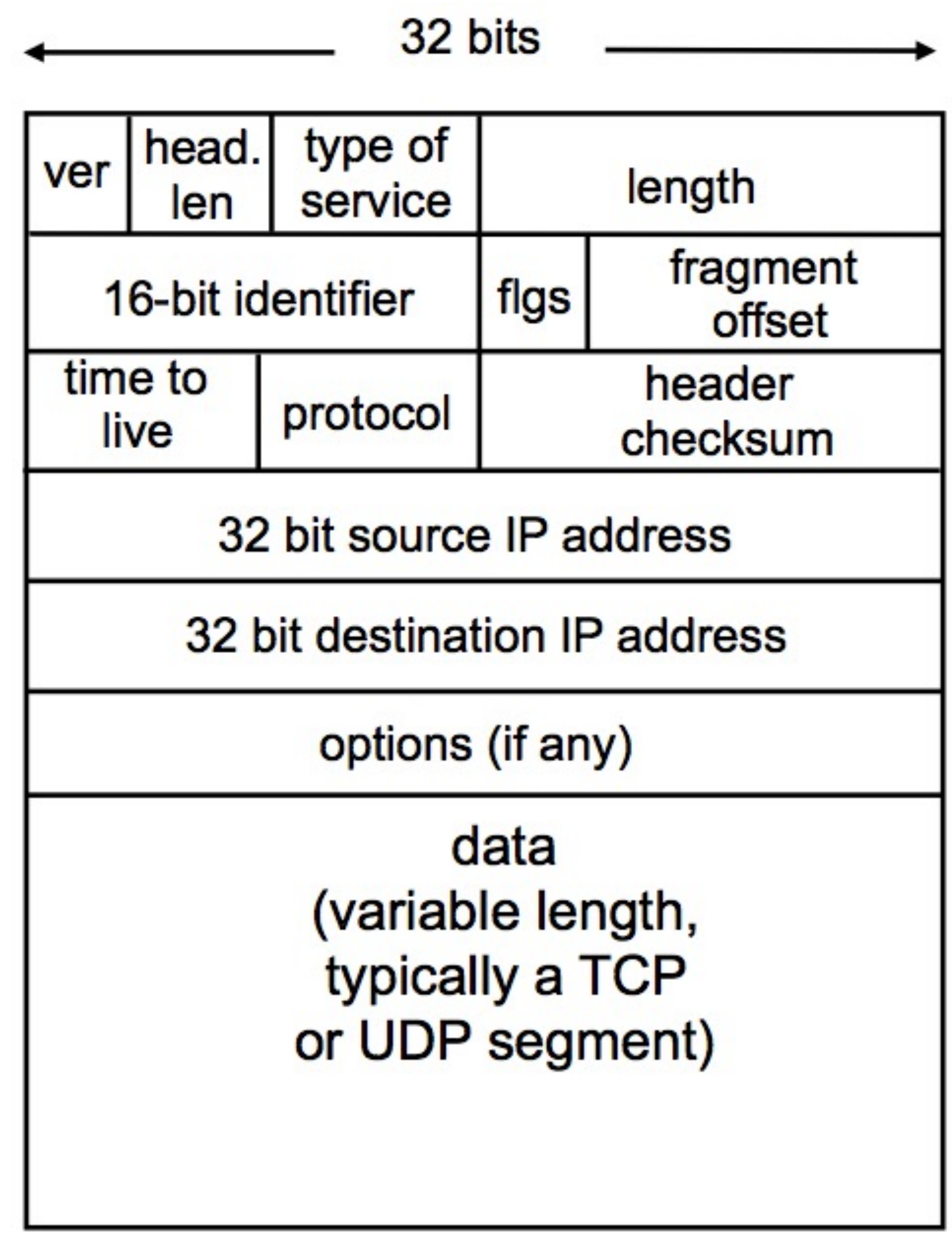
- Basic functions for network layer
  - Forwarding/Routing
- Network service model
  - Guaranteed delivery
  - Guaranteed delivery w/ bounded delay
  - In-order packet delivery
  - Guaranteed minimal bandwidth

Network layer: overview

- Connection v.s. connection-less delivery
  - circuit switch/packet switch
- Network layer protocols
  - Addressing and fragmentation: IPv4, IPv6
  - Routing: RIP, OSPF, BGP, DVMRP, PIM
  - Others: DHCP, ICMP, NAT

IPv4 Header

- **Header length**: 4-byte unit
- **Length**: 1-byte unit
- **Fragmentation**: id + MF/DF + offset (8-byte unit)
- **TTL**: time to live
- **Checksum**
  - Is it redundant?
  - Why is it just checksum for header?
- **Protocol**: identifies the upper layer protocol
- **Source and destination IP addresses**



IP address

- Globally recognizable identifier
- IPv4: 0.0.0.0~255.255.255.255
  - Most IP addresses are globally unique
  - Exception — why?
- Network id, host id
- CIDR address

IP address classes

- http://www.vlsm-calc.net/ipclasses.php

Class	1st Octet Decimal Range	1st Octet High Order Bits	Network/Host ID (N=Network, H=Host)	Default Subnet Mask	Number of Networks	Hosts per Network (Usable Addresses)
A	1 – 126*	0	N.H.H.H	255.0.0.0	126 (2^7 – 2)	16,777,214 (2^{24} – 2)
B	128 – 191	10	N.N.H.H	255.255.0.0	16,382 (2^{14} – 2)	65,534 (2^{16} – 2)
C	192 – 223	110	N.N.N.H	255.255.255.0	2,097,150 (2^{21} – 2)	254 (2^8 – 2)
D	224 – 239	1110	Reserved for Multicasting			
E	240 – 254	1111	Experimental; used for research			

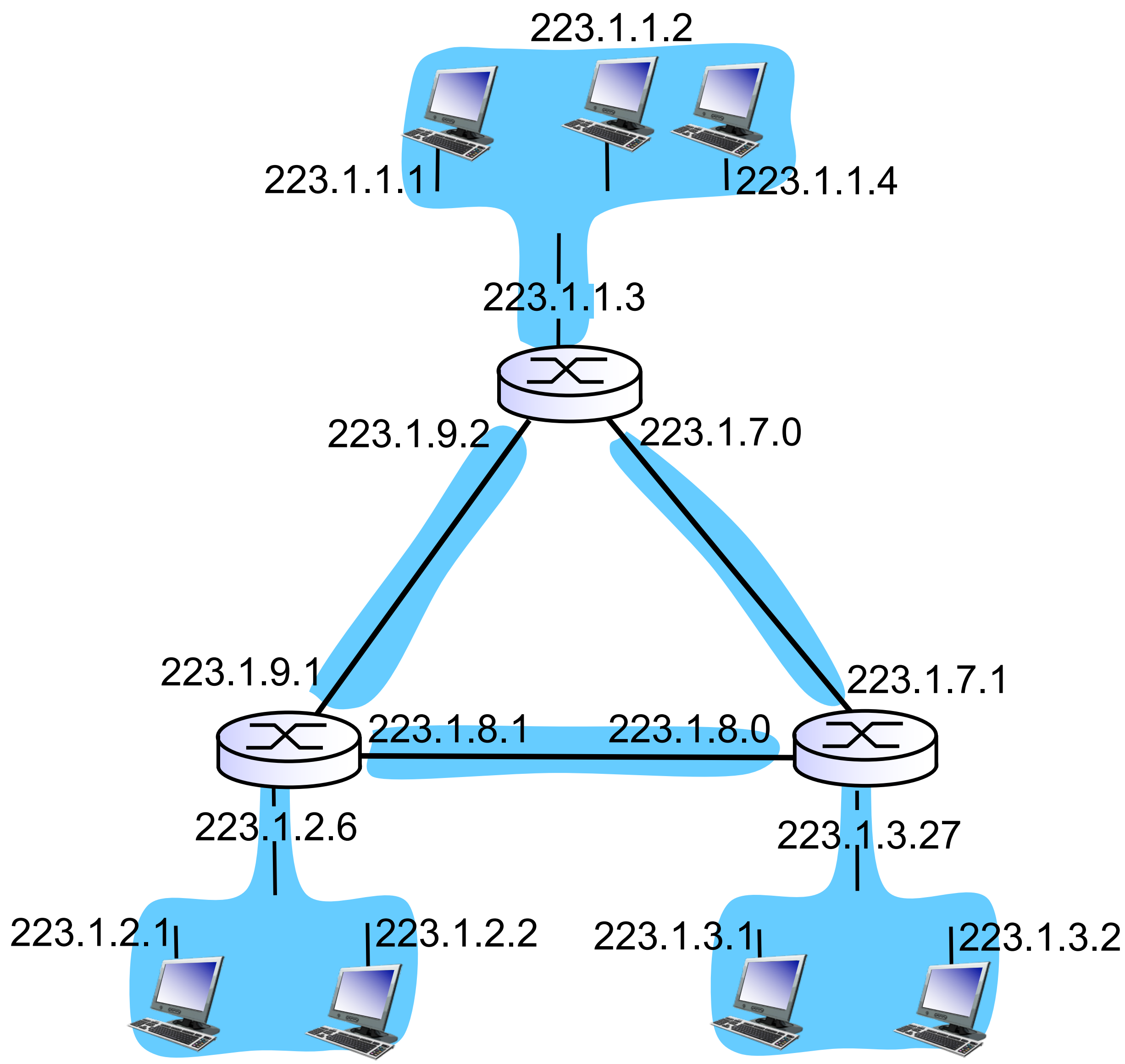
Class	Private Networks	Subnet Mask	Address Range
A	10.0.0.0	255.0.0.0	10.0.0.0 - 10.255.255.255
B	172.16.0.0 - 172.31.0.0	255.240.0.0	172.16.0.0 - 172.31.255.255
C	192.168.0.0	255.255.0.0	192.168.0.0 - 192.168.255.255

Hierarchical addressing

- subnet: a portion of addressing space
  - extend bits from the network id
  - <network address>/<subnet mask>
- route aggregation

Quick question

- How many subnets



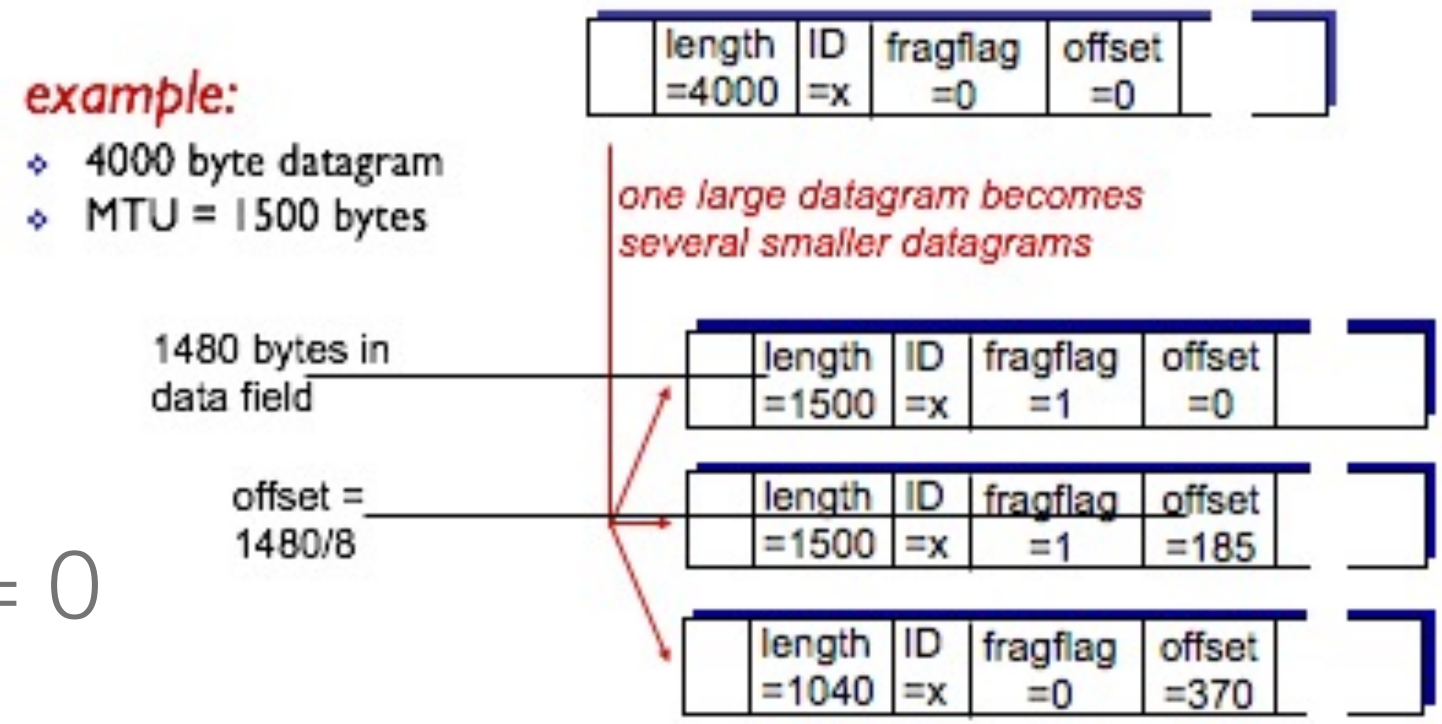
CIDR address

- a.b.c.d/x
  - x: # bits in network ID portion of the address
  - address: a.b.c.d, network mask: 2^32 — 2^(32-x)

CIDR	`11001000 00010111 00010000 00000000`
IP prefix	`200.23.16.0/23`
netmask	`11111111 11111111 11111110 00000000`
	`255.255.254.0`

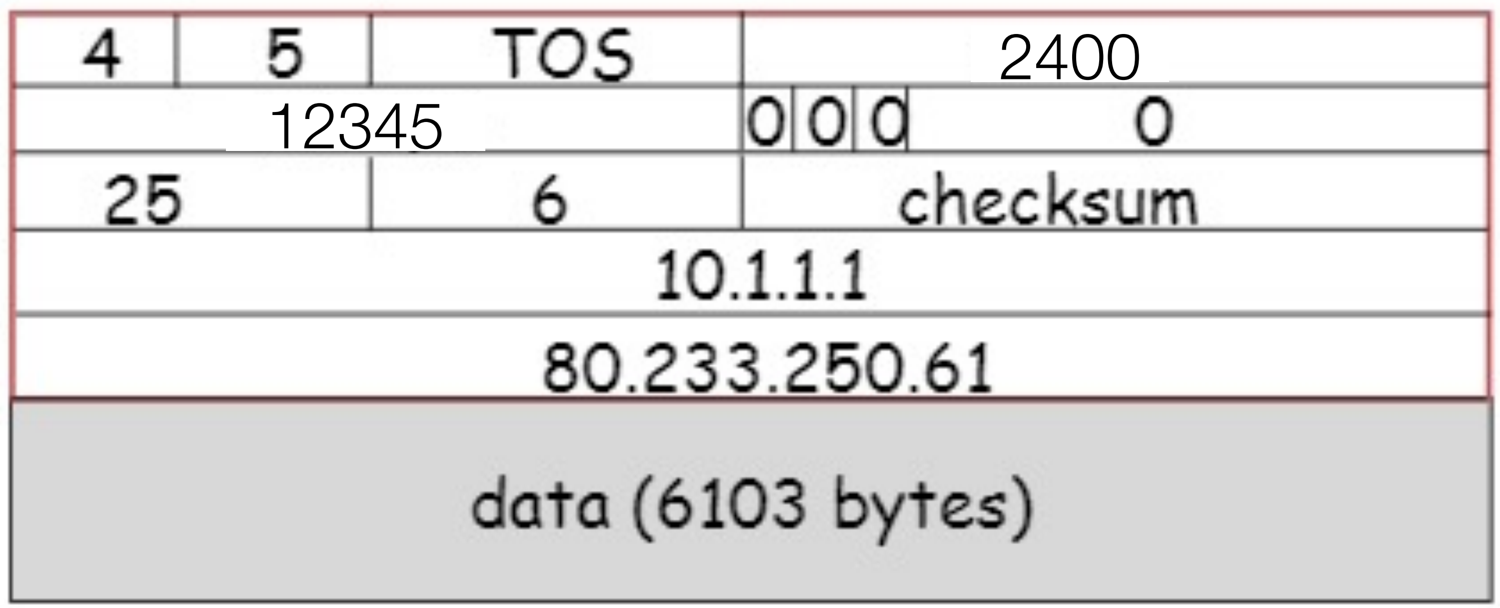
IP fragmentation and reassembly

- MTU: maximum transmission unit
- identifier
- flag bit: three bit
    - DF (Do not Fragment) = 0
    - MF (More Fragments) = 0?
- offset



Quick question

- Consider following IP packet



- Assume MTU = 1450 Bytes. Show the header length, total length, identification, flags, fragment offset, TTL, and IP payload size.

Quick question

- Consider following IP packet

<table>
<tr><td>4</td><td>5</td><td>TOS</td><td colspan="4">2400</td></tr>
<tr><td colspan="3">12345</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="2">25</td><td>6</td><td colspan="4">checksum</td></tr>
<tr><td colspan="7">10.1.1.1</td></tr>
<tr><td colspan="7">80.233.250.61</td></tr>
</table>

- Assume MTU = 1450 Bytes. Show the header length, total length, identification, flags, fragment offset, TTL, and IP payload size.

For the first packet: 20 bytes, 1444 bytes, ID = 12345, 01, Offset = 0, TTL = 25, 1424 bytes.
For the second packet: 20 bytes, 976 bytes, ID = 12345, 00, Offset = 178, TTL = 25, 956 bytes.

Switching

- Longest prefix matching

Destination Address Range	Link interface
11001000 00010111 00011000 *********	0
11001000 00010111 00010*** *********	1
11001000 00010111 0001**** *********	2
******** ******** ******** ********	3

- Linear lookup

DHCP: Dynamic Host Configuration Protocol

- Dynamically allocates the following info to a host
  - IP address for the host
  - IP address for default router
  - Subnet mask
  - IP address for DNS caching resolver
- Allows address reuse

DHCP: operations

- Host broadcasts “DHCP discovery” msg [optional]
- DHCP server responds with “DHCP offer” msg [optional]
- Host requests IP address: “DHCP request” msg
- DHCP server sends address: “DHCP ack” msg

NAT (network address translation)

- Depletion of IPv4 addresses — short-term solution
- Use private IP addresses
- Side-benefit: security
- How to achieve?
    - <public IP:port> — <private IP:port> mapping

NAT: detail

- outgoing packets:
    - replace (source IP address, source port #) of every outgoing packet to (NAT IP address, new port #)
- remote clients/servers will respond using (NAT IP address, new port #) as destination address
- remember (in NAT translation table) every (source IP address, port #) to (NAT IP address, new port #) translation pair
- incoming packets:
    - replace (destination NAT IP address, destination port #) of every incoming packet with corresponding (source IP address, port #) stored in NAT table

NAT: downside

- Increased complexity
- Single point of failure
- Cannot run services inside a NAT box

Project 2 overview

The best way to approach this project is in incremental steps. Do not try to implement all of the functionality at once.

- First, assume there is no packet loss, implement the header fields and connection control functions (initialization with 3-way handshake and termination). Just have the client initiate the connection with 3-way handshake, send a small file (200 Bytes) as a packet, and the server respond with an ACK, and then the server use FIN procedure to close the connection.
- Second, introduce a large file transmission and pipe-lining. This means you must divide the file into multiple packets and transmit the packets based on the specified window size.
- Third, introduce packet loss. Now you have to add a timer for last sent packet (Go-Back-N) or several timers for each unacked packets (Selective repeat). If a timer times out, the corresponding (lost) packet should be retransmitted for the successful file transmission.

Stage 0 : Small file transmission

- Small file transmission
  - A client initiate file transmission
  - A server accept connection requests, receive the file and save it with x.file
    - X indicated the counter of connection (starts with 1)
- Test
  - ./server 5000
  - ./client localhost 5000 testfile
  - In the server folder, check whether 1.file is saved and compare two files with diff command.

Stage 1: connection management

- Connection management
  - Setup and teardown already provided

Random #

```
    Client1                                     Server
(A 500-byte file to send)
      |                                            |
      |         seq=12345, ack=0, SYN              |
      | -------------------------------------->    |
      |         seq=221, ack=12346, SYN, ACK       |
      | <--------------------------------------    |
      |                                            |
      |         seq=12346, ack=222, ACK            |
      | -------------------------------------->    |
      |      (if include 500-byte payload)         |
      |         seq=222, ack=12846, ACK            |
      | <--------------------------------------    |
      |                                            |
      |                                            |
      |   FIN procedures to close connection       |
      | <------------------------------------->    |
      |         (saves file 1.file)                |
      |                                            |
    close                                          |
  connection                                       |
```

```
            Client                                     Server
      (A 10000-byte file to send)
               |                                          |
              ...                                        ...
               |      (10000 bytes transferred)           |
               |                                          |
               |      seq=22346, ack=0, FIN               |
               | ------------------------------------->   |
               |      (no payload in FIN)                 |
               |                                          |
               |      seq=4322, ack=22347, ACK            |
               | <-------------------------------------   |
               |                                          |
               |                                          |
               |      seq=4322, ack=0, FIN                |
         +-->  | <-------------------------------------   |
     2   |     |      seq=22347, ack=4323, ACK            |
     s   |     | ------------------------------------->   |
     e   |     |                                          |
     c   |    ...     (if ACK is lost)                   ...
     o   |     |                                          |
     n   |     |      seq=4322, ack=0, FIN                |
     d   |     | <-------------------------------------   |
     s   |     |      seq=22347, ack=4323, ACK            |
         |     | ------------------------------------->   |
     w   |     |                                          |
     a   |    ...                                       close
     i   |     |                                     connection
     t   |     |
         |     |
         +-->  |
             close
           connection
```

Stage 1: connection management

- Packet header struct (12 bytes)
  - Needed fields: a Sequence Number field, an Acknowledgment Number field, and ACK, SYN , and FIN flags.
  - Example :
    - uint16_t to represent each field.
    - In total, 5*2 bytes are used. Then pad 2 byte of zeros.
  - Functions: printPacket(), htonHeader(), ntohHeader().

Stage 1: connection management

- Packet header struct (12 bytes)
- Example to construct a SYN packet
  - Header h1, then memset the struct
  - Set sequence number fields of h1 with a random number
  - Set SYN flag
  - Print header “SEND 12345 0 SYN”
- Example to parse a packet
  - Print header “RECV 4321 12346 SYN ACK”

Stage 1: connection management

- Client side logic
  - Send a packet with SYN to initiate the connection.
  - After receive packet with ACK, start send packets with data.
  - After transmitting the entire file, send FIN packet and wait for ACK.
  - After receive server FIN, send ACK and wait for 2 seconds to close the connection.

Note: always need to print out the header

Stage 1: connection management

- Server side logic
  - If a SYN packet, reply with packet with SYN flag and ACK flag, set ACK number field and sequence number field
  - If a data packet, write data field to file
  - If a FIN packet, reply with packet with ACK flag. Then send a packet with FIN flag. After receive ACK from client, close the connection.

Note: always need to print out the header

Stage 2: large file transmission and pipelining

- Pipelining
  - For client side, send 10 packets at the same time.
    - For every received ACK, send a new packet out. Keep the window at 10.
  - For server side, no much difference
- Large file transmission
  - Pay attention to sequence number (max = 25600)

Stage 3: reliable data transfer with packet loss

- Go-back-N is recommended
- For client side
  - Keep a timer, restart the timer for every sent packet.
  - If timeout, resend all packets in the window.
- For server side
  - Keep expected sequence number
  - Every time a data packet is received, check whether the sequence number is expected. If expected, write data field, otherwise drop it.