Introduction to Networking
CSE 333 Winter 2021

Instructor: John Zahorjan

Teaching Assistants:
Matthew Arnold
Elizabeth Haker
Leo Liao
Guramrit Singh

Nonthakit Chaiwong
Henry Hung
Tim Mandzyuk

Jacob Cohen
Chase Lee
Benjamin Shmidt
Intro to Networking

❖ How the Internet is Designed / Works
  ▪ CSE 461

❖ Problems in Writing Distributed Programs and Their Solutions
  ▪ CSE 452

❖ This Course
  ▪ Experience with the simplest, most straightforward distributed application
  ▪ “Client/Server” when not much can go wrong

❖ I’m going to take a lot of liberty simplifying things here
First, Some Demos (Then, Some Explanation)

- `ifconfig` (Linux) to list network addresses
- `nc` utility launched as server and as client
- `nc` server and browser as client
- HTTP
- Persistent connection
IP (Internet Protocol)

- IP carries data from one IP address (network adapter/machine) to another
  - From one OS to another

```
  Apps   os      Internet      os   Apps
     |       |              |       |
     IP Address |              | IP Address|
```
Sockets / Ports

- Internet packets carry IP address and port number
- IP address is used to name host to which packet is delivered
- Ports are used by destination OS to determine to which socket’s buffer to add incoming data
  - Sockets are bound to ports
TCP (Stream) Sockets

- TCP is a reliable, stream protocol
- stream: it’s a linear sequence of bytes, just like a file
- reliable: bytes read are in the same order as they were sent
  - IF they arrive
Client / Server Architecture

- The server is always running and has bound a socket to a “well known port”
  - Example: web server sockets are usually bound to port 80

- The client comes up, establishes a TCP connection to the server’s socket, and sends requests
Server Socket Setup

1. Server creates a socket

2. Server binds socket to a port

3. Server invokes listen()

4. Server thread blocks on accept()
Server Socket Setup

5. Client connects

6. OS creates new socket, connects remote client streams to it, and returns it (from accept())

7. The new socket is “connected” to the client’s socket on their machine. Writing to the socket sends data that can be read by the client. Reading from the socket reads data the client has sent, or blocks.

The server ends up with a distinct socket for each client connection. More than one client can be connected at the same time.
Request / Response Application Protocol

- The applications send messages to each other
- The rules about what messages can be/must be sent, how to format them, are set by the application level protocol

- **Request/Response Protocol**
  - Simple two message exchange
    - Client sends a request to the server
    - Server responds with normal response or an error indication
  - *Similar* to procedure call
Let’s See All This in Action

- We’ll do this live, but what we’re doing is on the following slides
ifconfig: attu2

attu2> ifconfig
eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
   inet 128.208.1.138 netmask 255.255.255.0 broadcast 128.208.1.255
   inet6 fe80::46a8:42ff:fe4a:76a8 prefixlen 64 scopeid 0x20<link>
   inet6 2607:4000:200:10::8a prefixlen 64 scopeid 0x0<global>
   ether 44:a8:42:4a:76:a8 txqueuelen 1000 (Ethernet)
RX packets 9787460872 bytes 11437771351898 (10.4 TiB)
RX errors 0 dropped 41 overruns 0 frame 0
TX packets 5065166238 bytes 3366916108204 (3.0 TiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
device interrupt 18

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
   inet 127.0.0.1 netmask 255.0.0.0
   inet6 ::1 prefixlen 128 scopeid 0x10<host>
   loop txqueuelen 1000 (Local Loopback)
RX packets 115725209 bytes 64235696673 (59.8 GiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 115725209 bytes 64235696673 (59.8 GiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
nc utility as client and server

1. start server
   attu7> nc -l 5566
   one
two
three
attu7>

2. start client
   attu2> nc attu7.cs.washington.edu 5566
   one
two
three
attu2>

3. send stream of data

4. server reads data
nc as server, browser as client

```
attu7> nc -l 5566
```

GET /one/two/three/ HTTP/1.1
Host: attu7.cs.washington.edu:5566
Connection: keep-alive
DNT: 1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.190 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

^C
attu7>
HTTP Protocol

**Encoding**: text

**Framing**: \r\n to end each line
empty line (\r\n\r\n) to end request

GET /one/two/three/ HTTP/1.1
Host: attu7.cs.washington.edu:5566
Connection: keep-alive
DNT: 1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.190 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
HTTP Request (Sent by Chrome)

The first line is the request line.
All following lines are key: value

GET /one/two/three/ HTTP/1.1
Host: attu7.cs.washington.edu:5566
Connection: keep-alive
DNT: 1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.190 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
HTTP Response (Sent by courses.cs)

HTTP/1.1 302 Found
Date: Sun, 28 Feb 2021 22:16:05 GMT
Server: Apache/2.4.6 (CentOS)
Expires: Wed, 01 Jan 1997 12:00:00 GMT
Cache-Control: private, no-store, no-cache, max-age=0
Location: https://courses.cs.washington.edu/courses/cse333/21wi/
Content-Length: 328
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="https://courses.cs.washington.edu/courses/cse333/21wi/">here</a>.</p>
<hr>
<address>Apache/2.4.6 (CentOS) Server at courses.cs.washington.edu Port 80</address>
</body></html>
Persistent Connections

- The client can request that the server not close the TCP connection when it responds to the request
  - Because the client anticipates sending another request to the same server
- **Connection: keep-alive**
  - in HTTP request and response headers

- Pipelined request example:

  ```
  GET / HTTP/1.1
  Host: www.cs.washington.edu:80
  Connection: keep-alive
  
  GET / HTTP/1.1
  Host: www.cs.washington.edu:80
  Connection: keep-alive
  ```

  End of request sentinels
Okay, Now Some Networking Background

- This is not CSE 461...
Network Concepts: Local Area Network

- **Hosts**
  - Computers, each running an OS

- **Network Adapters**
  - Ethernet card, Wifi card, cellular interface
  - Converts bits in memory into analog signals on the local area network when transmitting, and analog signals into bits when receiving

- **Local Area Network**
  - Wire or fiber or radio waves
  - *Data sent by any the adapter of any host is received by every adapter on the local network*
Network Concepts: LAN Address

• Suppose one host wants to tell another host to power down
  • All the hosts have previously agreed that if they hear the bit pattern 0101010101010101 it represents a request to power down
• How can the green host ask only the orange host to power down?
• The bits of the message are preceded by bits representing the name of the intended recipient
  • Orange 0101010101010101
  • If you’re name isn’t orange, you ignore the message
Network Concepts: A Network

A LAN is sometimes referred to as “a network.”
Network Concepts: Internet
Internet Data Delivery

- When the sender of some data puts that data on its LAN, the receiver won’t hear it (unless it’s on the same LAN)
- The Internet is responsible for routing the bits from the source network to the destination network
  - From the source LAN to the LAN the receiver is on

- Two possible approaches to routing:
  1. Send every message to every LAN, thus ensuring it gets on receiver’s LAN and is heard by receiver
  2. Send it to just one LAN, the receiver’s LAN, along some directed path
Network Concepts: Internet

- Sender needs to name receiver
- MAC addresses aren’t very helpful because they’re essentially random
- Create a new namespace with the property that “similar names are in the same place”
- IP addresses
IP Addresses

- IP addresses are (usually) global in scope
- They name adapters, which means they name hosts
- IPv4 addresses are 32 bits
  - 4G names
- IPv6 addresses are 128 bits
  - \( \infty \) names

128.208.1.137

128.208.1.138

128.208.1.139

Internet
Network Connections

- IP addresses name hosts, but we want to communicate from one host to another
  - We want to communicate from an application running on one host to an application running on another host

- Ports / Sockets
  - Internet messages carry an IP address: host identifier
  - They also carry a port number (a small int)
  - The operating system maintains a map from port numbers at its IP address and running applications on its system
    - Sockets
Connections and Demultiplexing

- Demultiplexing is taking data arriving from one source and moving it forward to one of several next steps.

- TCP demultiplexes using (TCP, source IP, source port, dest IP, dest port) as the key.
A Bit of Context Without Context: SO_REUSEADDR

- When a TCP socket is closed, it goes into a TIMED_WAIT state
  - It is still there for a little while related to how long undelivered packets might still be “in flight”
  - By being there, it prevents a new socket from being created and binding to the same address:port
  - That prevents confusion where data sent to the now closed connection arrives late and is demultiplexed to the new socket’s incoming buffer

- This can mean that when you terminate your server and then “immediately” restart it, it fails
  - It fails because it tries to acquire the same IP:port as it just had, and that address is being kept busy exactly so you can’t acquire it again for a while 😞
A Bit of Context Without Context: SO_REUSEADDR

/* Set a socket option to enable re-use of the port number as soon as the server exits. ("man setsockopt" and "man 7 socket") */

int optval = 1;
if (setsockopt(listen_sock_fd_, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval))
{
    perror("setsockopt(SO_REUSEADDR) failed");
    exit(1);
}
Next...

What’s wrong with this pseudo-code?

- server_sock = createServerSock();
  while(1)
  {
    client_sock = server_sock.accept();
    while ( request = read_request(client_sock) )
    {
      process_request(request);
    }
  }