Section 1: Sockets API + HW 1

CSE 461 Winter 2024
Your TAs :) 

- BS/MS student!
- First-time 461 TA
- Hobbies: learning Italian

- BS/MS student–just started the MS!
- Research assistant in syslab (I do OS things)
- Hobbies: trivia, violin, hiking/travel
Administrivia - Course Structure

● Assignments
  ○ 3 group projects
    ■ P1: Building a client and server application
    ■ P2: Practicing with Software-Defined Networking (SDN)
    ■ P3: Experimenting to learn about latency in real-world networks (Bufferbloat)
  ○ About 5 homework assignments (Gradescope)
    ■ Detailed practice with the concepts discussed in textbook & lecture
    ■ Conceptual overview
  ○ In-person Midterm & Final Exam
  ○ Occasional “surprise” quizzes

● Quiz Sections
  ○ Intro to labs (helpful hints!) + networking software
  ○ Reviewing and clarifying conceptual topics (e.g. various protocols)
  ○ More practices with mechanics (e.g. calculations, algorithms, etc.)
**Administrivia**

- Project 1 is out, **due Jan. 25**
  - Can be done in groups of 2-3
  - Can be done in any language (recommend Python/Java)
    - But future labs will be fully in Python
  - Goal is to help you get familiar with some language's Socket API
  - **NEW THIS QUARTER:** 10% of points on style
    - modularity, readability, consistent naming scheme..... just good programming practices in general
    - we'll provide some guidelines on Ed/spec
Socket API & Project 1
Network Stack - OSI Model vs TCP/IP Model

OSI Basic Reference Model

- APPLICATION
- PRESENTATION
- SESSION
- TRANSPORT
- NETWORK
- DATA LINK
- PHYSICAL

Protocols in Each Layer

- Application: Modbus, SEP2, DNP3, HTTP, IEC 61850, CIM, ICCP, BACnet, OpenADR, GOOSE
- Presentation: Compression, encryption protocols
- Session: NFS, SQL, SMB, RPC, P2P tunneling, SCP, SDP, SIP, H.323
- Transport: TCP, UDP
- Network: IPv4/IPv6, ARP, IGMP, ICMP
- Data Link: Ethernet
- Physical: RS 232, UTP cables (CAT 5, 6), DSL, Optic fiber

TCP/IP Model

- APPLICATION
- TRANSPORT
- INTERNET
- NETWORK INTERFACE
- Bits and Frames

7 layers vs. 5 layers
Network Stack - Packet Encapsulation

Network Host

1. Physical Layer
2. Data Link Layer (MAC)
3. Network Layer (IP)
4. Transport Layer (TCP/UDP)
5. Application Layer

Application
Transport
Internet
Link

Frame header → Frame data → Frame footer

IP header → IP data

UDP header → UDP data
Network-Application Interface

- Defines the operations that programs (apps) call to use the network
  - Application Layer API
  - Defined by the Operating System
    - These operations are then exposed through a particular programming language
    - All major Operating Systems support the Socket API
  - Allows two computer programs potentially running on different machines to talk
  - Hides the other layers of the network
Project 1 - Overview

- Part 1: Simple Client
  - Send requests to attu server
  - Wait for a reply
  - Extract the information from the reply
  - Continue...

- Part 2: Simple Server
  - Server handles the Client requests
  - Multi-threaded

- This is the basis for many apps!
  - File transfer: send name, get file
  - Web browsing: send URL, get page
  - Echo: send message, get it back
Socket API

- Simple application-layer abstractions (APIs) to use the network
  - The network service API used to write all Internet applications
  - Part of all major OSes and languages; originally Berkeley (Unix) ~1983
- Two kinds of sockets
  - Streams (TCP): reliably send a stream of bytes
    - Detects packet loss with timeouts (uses adaptive timeout protocol)
    - Uses flow control: similar to selective repeat
  - Datagrams (UDP): unreliably send separate messages
Sockets let apps attach to the local network at different ports

- Ports are used by OS to distinguish services/apps all using the same physical connection to the internet
- Think of ports like apartment numbers, allowing mail sent to a shared building address (IP) to be sorted into the correct destination unit (application)
## Socket API Operations

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOCKET</td>
<td>Create a new communication endpoint</td>
</tr>
<tr>
<td>BIND</td>
<td>Associate a local address (port) with a socket</td>
</tr>
<tr>
<td>LISTEN</td>
<td>Announce willingness to accept connections; (give queue size)</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>Passively establish an incoming connection</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Actively attempt to establish a connection</td>
</tr>
<tr>
<td>SEND</td>
<td>Send some data over the connection</td>
</tr>
<tr>
<td>RECEIVE</td>
<td>Receive some data from the connection</td>
</tr>
<tr>
<td>CLOSE</td>
<td>Release the connection</td>
</tr>
</tbody>
</table>

[https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html](https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html)
[https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html](https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html)
Using TCP Sockets

Client (host 1) -> Server (host 2) -> Time

1 ← connect
2 request
3 reply
4 disconnect
Using TCP Sockets (cont.)

Client (host 1)  Time  Server (host 2)

1: socket
5: connect*

connect

request

reply

disconnect

1: socket
2: (bind)
3: (listen)
4: accept*
6: recv*

* = call blocks

7: send
8: recv*

9: send
10: recv*
12: close
Using UDP Sockets

Client (host 1)  Time  Server (host 2)

1: socket
5: connect*
7: sendto
8: recvfrom*
11: close

connect
request
reply
disconnect

1: socket
2: (bind)
3: (listen)
4: accept*
6: recvfrom*  *= call blocks
9: sendto
10: recvfrom*
12: close

12: close
Client Program Outline

socket() // make socket
getaddrinfo() // server and port name
    // www.example.com:80
connect() // connect to server
send() // await reply [block]
recv() // do something with data!
...
close() // done, disconnect
Server Program Outline

socket()    // make socket
getaddrinfo() // for port on this host
bind()      // associate port with socket
listen()    // prepare to accept connections
accept()    // wait for a connection [block]
...
recv()      // wait for request [block]
...
send()      // send the reply
close()     // eventually disconnect

create a new thread for new client connection!
Server

```python
listener = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
listener.bind(server_address)

while True:
    try:
        connection, client_addr = listener.accept()
        try:
            connection.recv(n_bytes)
        finally:
            connection.close()
    except:
        listener.close()
```

Client

```python
socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket.connect(server_address)
socket.sendto(message, server_address)
socket.close();
```

- Python socket documentation
- UDP socket example
- socketserver (a little overkill)
Java Examples with Socket & ServerSocket

Server:

```java
ServerSocket listener = new ServerSocket(9090);
try {
    while (true) {
        Socket socket = listener.accept();
        try {
            socket.getInputStream();
        } finally {
            socket.close();
        }
    }
} finally {
    listener.close();
}
```

Client:

```java
Socket socket = new Socket(server, 9090);
out = new PrintWriter(socket.getOutputStream(), true);
socket.close();
```

- [http://cs.lmu.edu/~ray/notes/javanetexamples/](http://cs.lmu.edu/~ray/notes/javanetexamples/)
- [https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html](https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html)
HW1 Fundamentals
Traceroute

- Goal: find network path from our system to a given remote host
- Core mechanism: **Time-To-Live** (TTL)
  - TTL defines the number of hops a packet will travel through until it is dropped
    - TTL is decremented every hop
    - Once TTL is 0 then the packet is dropped and a report is sent to the source

**Resources:**
Traceroute

- Traceroute sends out three packets per TTL increment
  - To have 3 trials of data for each hop distance
- Each data point corresponds to the total RTT time
Using Traceroute

dominickta@Prota ~

```
traceroute edstem.org
traceroute: Warning: edstem.org has multiple addresses; using 172.66.40.189
traceroute to edstem.org (172.66.40.189), 64 hops max, 52 byte packets
1  10.18.0.2 (10.18.0.2)  6.327 ms  6.836 ms  8.570 ms
2  lo0--5.uwcr-ads-1.infra.washington.edu (198.48.65.5)  6.998 ms  4.229 ms  11.492 ms
3  10.132.5.66 (10.132.5.66)  11.876 ms  9.706 ms  5.816 ms
4  10.132.255.17 (10.132.255.17)  11.205 ms  4.184 ms  3.535 ms
5  10.132.255.18 (10.132.255.18)  15.567 ms  5.217 ms  2.527 ms
6  ae20--4000.icar-sttl1-2.infra.pnw-gigapop.net (209.124.188.132)  7.411 ms  5.022 ms  20.059 ms
7  six.as13335.com (206.81.81.10)  29.820 ms  105.949 ms  11.541 ms
8  172.71.140.3 (172.71.140.3)  25.781 ms
8  172.71.144.3 (172.71.144.3)  10.867 ms
8  172.71.140.3 (172.71.140.3)  9.835 ms
9  172.66.40.189 (172.66.40.189)  17.161 ms  9.327 ms  7.720 ms
```
Latency & Bandwidth
Latency

- Latency: Total time for a message to arrive on a network
  - Round trip time (RTT) is the latency for travel from source to destination to source
- **Latency = Propagation + Transmit + Queue**
  - **Propagation = Distance / “Speed Of Light”**
    - How long it takes for information to travel a distance from source to destination
    - Speed varies by medium
  - **Transmit = Size / Bandwidth**
    - How long it takes for information to be put onto the wire before travelling
  - **Queue time**
    - How long data has to wait until it’s their turn to be transmitted
Bandwidth

- **Bandwidth** (data rate): The number of bits that can be transmitted over a period of time
  - Units of bits per second (bps)
  - Confusingly also used to refer to the frequency range of a signal
    - In this case the units are given as hertz (Hz)

- **Throughput**: The measured performance of a system
  - Units of bits per second (bps)

- Analogy: bandwidth is a pipe and throughput is the water
Bandwidth & Transmission Time

Transmission time = Size of data / Bandwidth

- Transmission time of 1 bit of data at a bandwidth of 1 Mbps?
  - 1 bit / 1,000,000 bps = 1/1,000,000 seconds = 1 microsecond

- Transmission time of 1 bit of data at a bandwidth of 2 Mbps?
  - 1 bit / 2,000,000 bps = 1/2,000,000 seconds = 0.5 microseconds

Figure 1.16.: Bits transmitted at a particular bandwidth can be regarded as having some width: (a) bits transmitted at 1 Mbps (each bit is 1 microsecond wide); (b) bits transmitted at 2 Mbps (each bit is 0.5 microseconds wide).
Bandwidth-Delay Product

- Product between bandwidth and propagation delay
  - Units in bits \( \text{bps} \times \text{s} = \text{b} \)
- Propagation delay is either one way latency or RTT
  - Usually RTT
- Conceptually defines the maximum amount of data that can be “in-flight” at a given time
  - think the amount of water in a pipe
Exercises!
Exercise 1

Suppose we have a network link with a **bandwidth of 10 Mbps**. We want to send a **100 KB file** to a friend somewhere else in the network. The RTT from us to our friend is **20 ms**. How long does it take for the entire file to be delivered?

- **Transmit time** = $\frac{100 \text{ KB}}{10 \text{ Mbps}} = \frac{100,000 \text{ B}}{10 \text{ Mbps}}$
  
  $= \frac{800,000 \text{ b}}{10,000,000 \text{ bps}} = 0.08 \text{ seconds} = 80 \text{ ms}$

- At $t=80\text{ms}$, the final bit of data is transmitted onto the wire.
  - This bit still needs to actually travel to the destination (**propagation delay**)

- At $t=90\text{ms}$, the final bit of data arrives at the destination
  - Note that we added $\frac{1}{2}$ of the RTT!
Consider a point to point link **50 km in length**. Suppose the propagation speed is **2 * 10^8 m/s**. At what bandwidth in **Mbps** would the propagation delay equal the transmit delay for **100 B packets**?

- Propagation delay = Distance / Speed Of Light (varies by medium)
  - \( \frac{50 \times 10^3 \text{ m}}{2 \times 10^8 \text{ m/s}} = 0.00025 \text{ seconds} = 250 \text{ microseconds} \)

- Transmit = Size / Bandwidth
  - \( 250 \text{ microseconds} = \frac{100 \text{ B}}{x \text{ Mbps}} \) (solve for X)
  - \( 100 \times 8 = 800 \text{ bits} \rightarrow 800 \text{ bits} / 250 \mu \text{s} = 3.2 \text{ Mbps} \)

What about for **512 byte packets**?

- \( 512 \times 8 \text{ bits} / 250 \mu \text{s} = 16.4 \text{ Mbps} \)
Exercise 3

Suppose a **128-kbps** point-to-point link is set up between Earth and a SpaceX colony on Mars. The distance from Earth to Mars (when they are closest together) is approximately **55 Gm**, and data travels over the link at the speed of light (**3 * 10^8 m/s**)

- Calculate the minimum RTT for the link.
- Calculate the delay x bandwidth product for the link.
- Say your aunt Betty takes a selfie on Olympus Mons, and sends a 5 MB picture to you on Earth. How quickly after the picture is taken can you receive the image from Betty?
Exercise 3

Suppose a 128-kbps point-to-point link is set up between Earth and a SpaceX colony on Mars. The distance from Earth to Mars (when they are closest together) is approximately 55 Gm, and data travels over the link at the speed of light (3 * 10^8 m/s)

- Calculate the minimum RTT for the link.
  - RTT = 2 * Propagation delay = 2 * 55 * 10^9 m / (3 * 10^8 m/s) = 2 * 184 = 368 seconds
- Calculate the delay x bandwidth product for the link.
  - delay x bandwidth = 368 seconds * (128 * 10^3 bps) = 5.888 MB
- Say your aunt Betty takes a selfie on Olympus Mons, and sends a 5 MB picture to you on Earth. How quickly after the picture is taken can you receive the image from Betty?
  - Transmit delay for 5 MB = 40,000,000 bits / (128 * 10^3 bps) = 312.5 seconds
  - Total time = transmit delay + propagation delay = 312.5 + 184 = 496.5 seconds = about 9 minutes
That’s it!