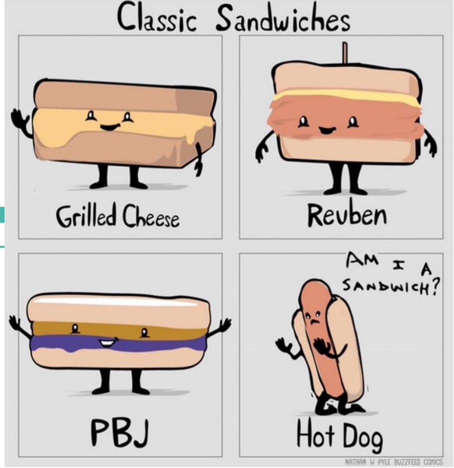


Computer Networks

Socket API, HW 1 fundamentals
Spring 2022
With Monty, Edan, Jason, and Mark!

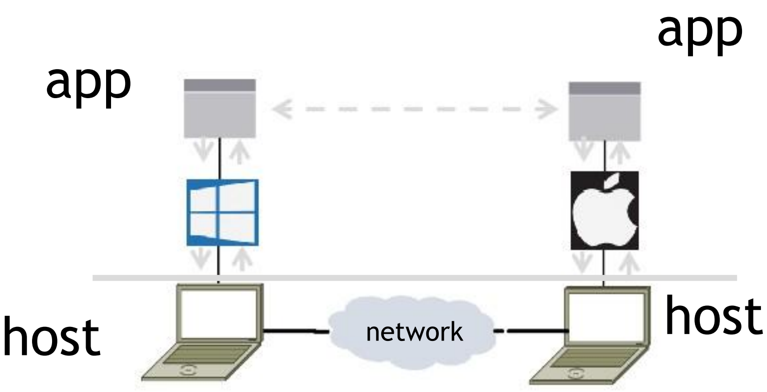
Administrivia

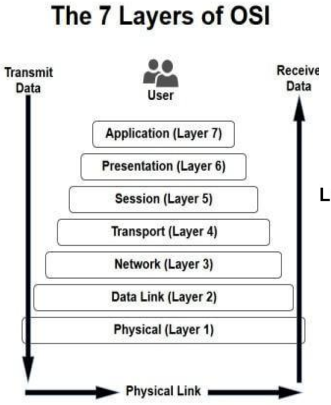
- Project 1 is out! Due April 18th at 11:00pm
  - Can be done in groups of 2-3
  - Can be done in any language (recommend Java / Python)
    - Future labs will be in Python
    - Intent is to allow you to become familiar with some languages Socket API!
- Homework 1 is out! Due April 11th at 11:00pm
  - Read Chapter 1, specifically section 1.5 and beyond
- Quiz 1 in class tomorrow

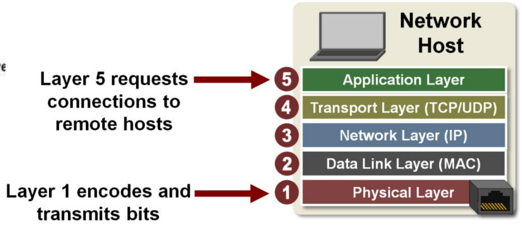
Socket API & Project 1

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

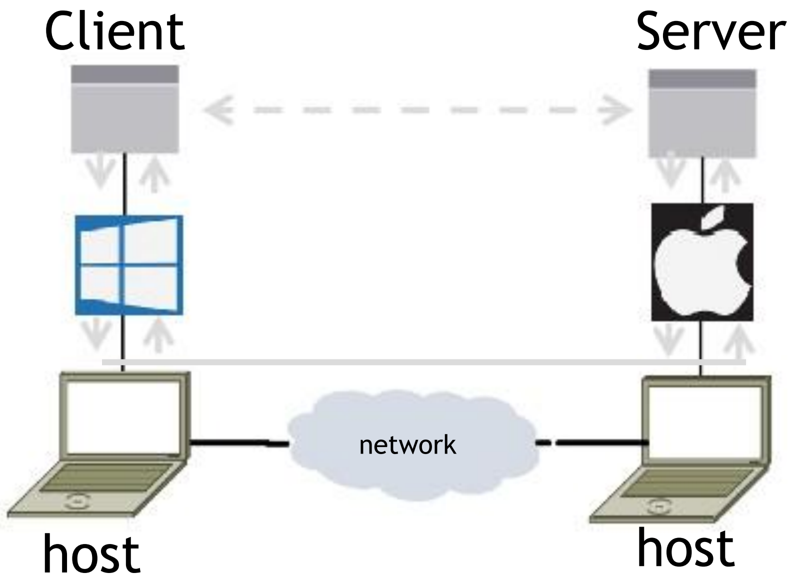





Network Host
Layer 5 requests connections to remote hosts
Layer 1 encodes and transmits bits
5 Application Layer
4 Transport Layer (TCP/UDP)
3 Network Layer (IP)
2 Data Link Layer (MAC)
1 Physical Layer

Project 1

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

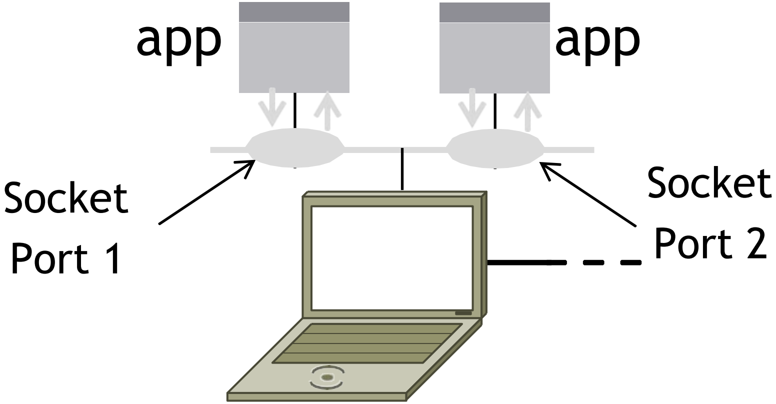


Socket API

- 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

Ports

- 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)

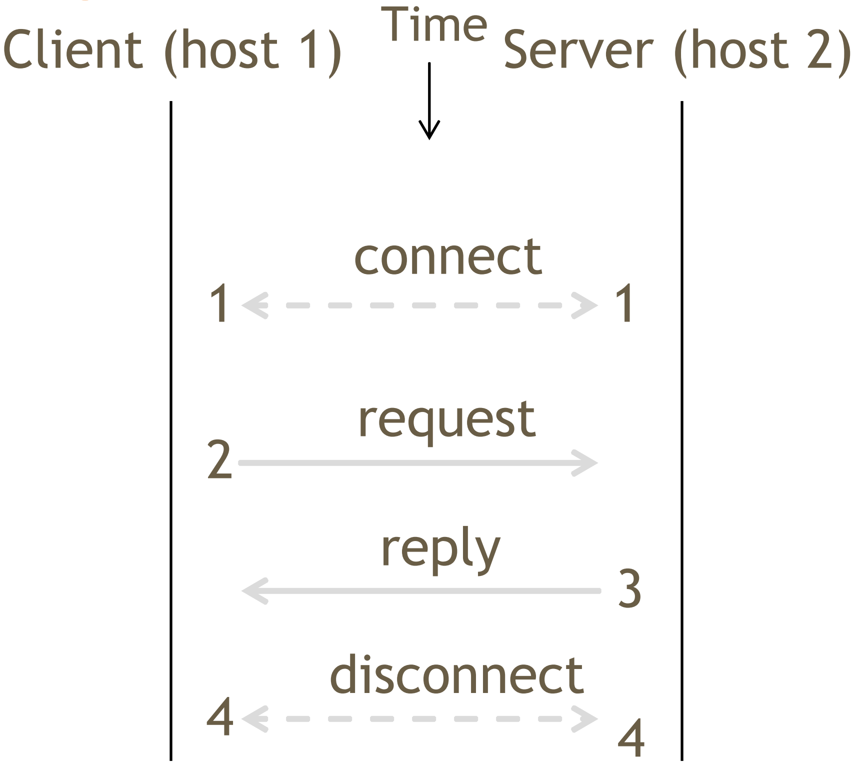


Socket API Operations

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

https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html
https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html

Using TCP Sockets



Client (host 1)

Time

Server (host 2)

connect

1

1

request

2

reply

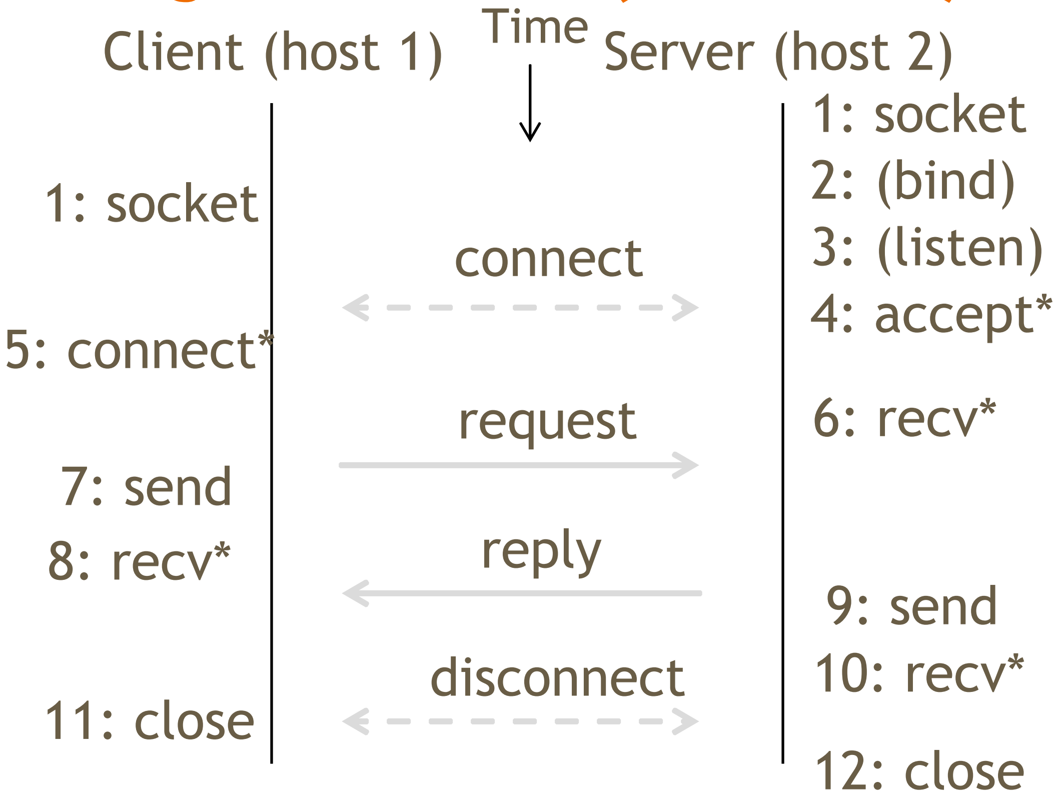
3

disconnect

4

4

Using TCP Sockets (Continued)



Client (host 1)

Time

Server (host 2)

1: socket

2: (bind)

3: (listen)

4: accept*

1: socket

connect

5: connect*

request

6: recv*

7: send

8: recv*

reply

9: send

10: recv*

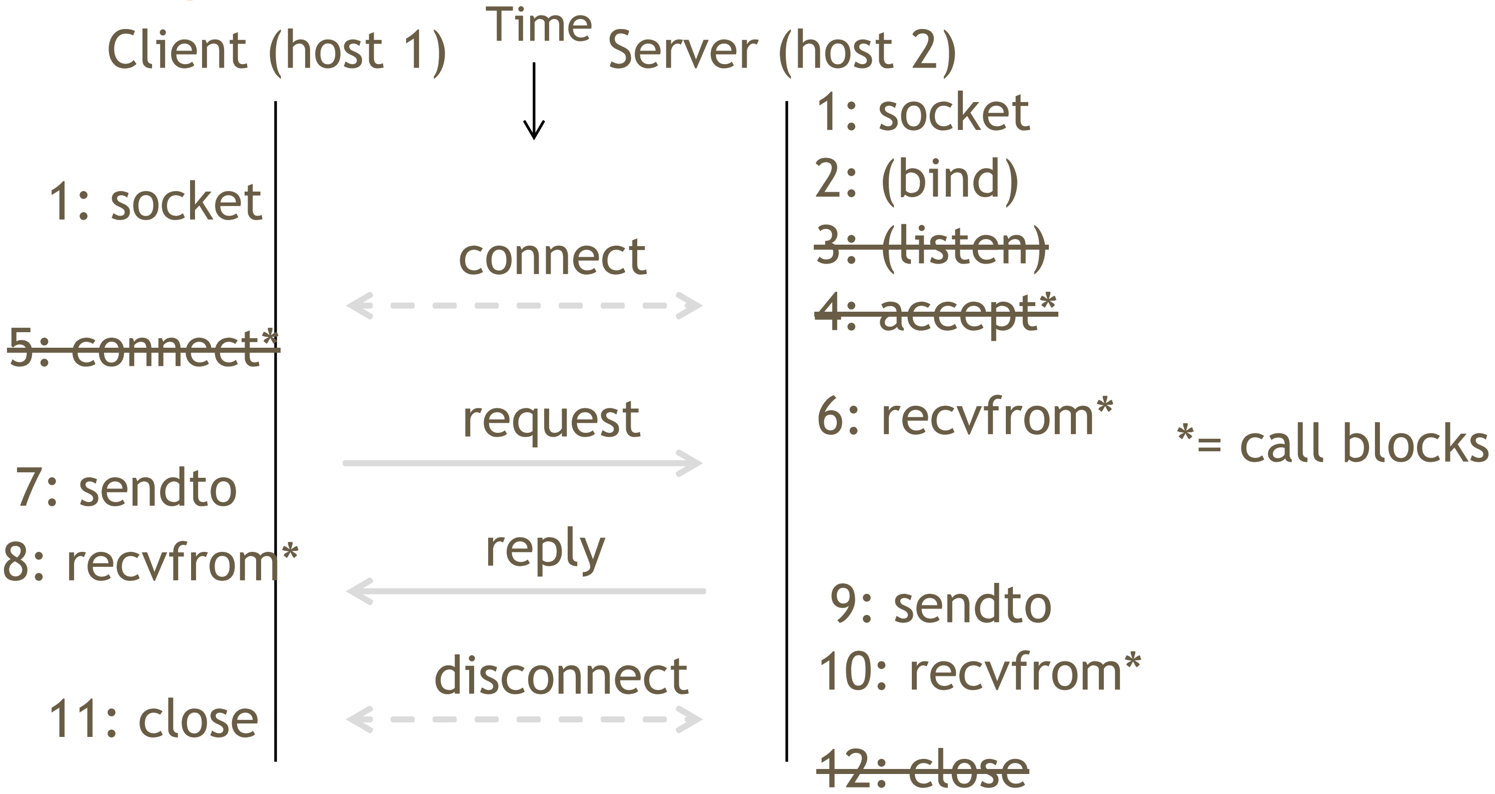
disconnect

11: close

12: close

*= call blocks

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*

9: sendto

10: recvfrom*

~~12: close~~

*= call blocks

Client Program Outline

```
socket()      // make socket
getaddrinfo()  // server and port name
               // www.example.com:80
connect()      // connect to server

send()       // send request
recv()       // await reply [block]
…             // do something with
close()               data!
              // 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
```

Python Examples with socket

- Server

```
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

```
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

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

- Client

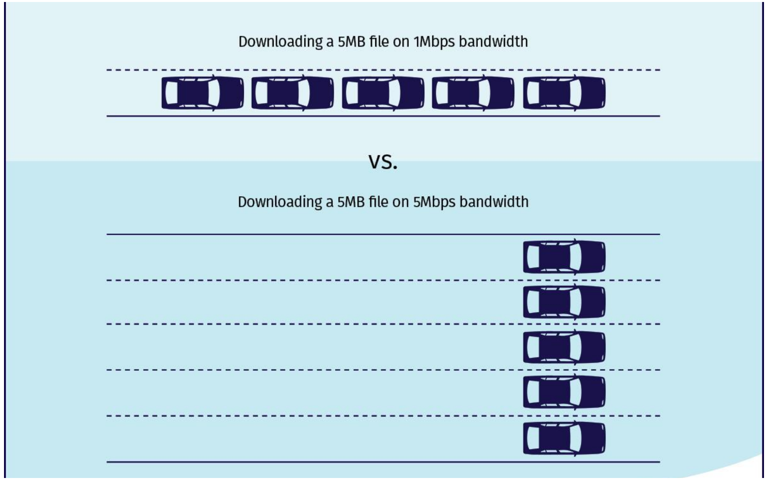
```
Socket socket = new Socket(server, 9090);
out =
        new PrintWriter(socket.getOutputStream(), true);
socket.close();
```

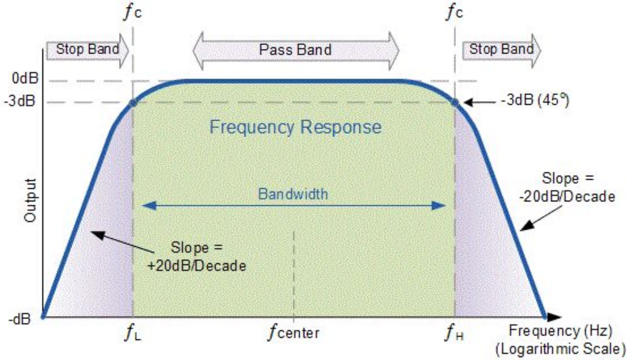
- http://cs.lmu.edu/~ray/notes/javanetexamples/
- https://docs.oracle.com/javase/tutorial/networking/datagrams/clientServer.html
- https://docs.oracle.com/javase/tutorial/networking/sockets/index.html

HW1 Fundamentals

Bandwidth

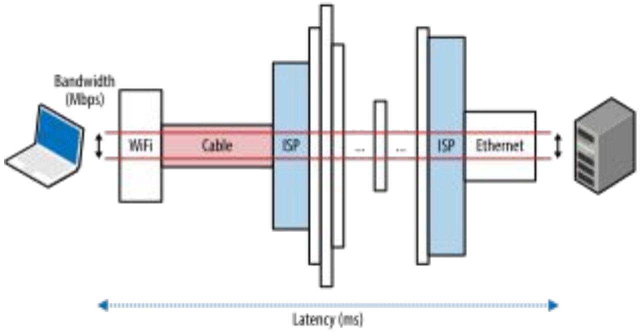
- Bandwidth (data rate): The number of bits that can be transmitted over a period of time
  - Units of bits per second (bps)
  - Confusingly also used to refer to the frequency range of a signal
    - In this case the units are given as hertz (Hz)
- Throughput: The measured performance of a system
 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


Downloading a 5MB file on 1Mbps bandwidth
vs.
Downloading a 5MB file on 5Mbps bandwidth



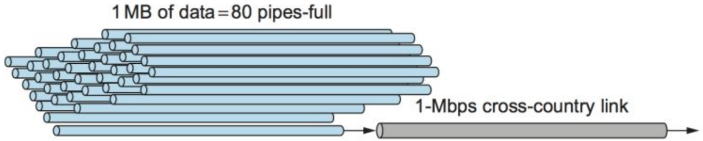
Latency

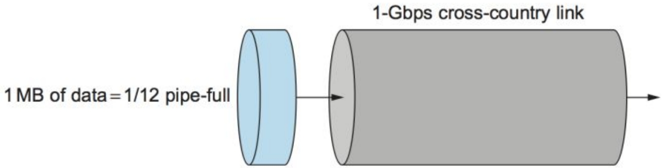
- Latency: How long it takes for a message to travel from one point in the network to another
  - Units of seconds
  - Round trip time (RTT) defined as latency for message to travel from one point in the network to another, then back to the starting point
- Latency can be calculated as:
  - Latency = Propagation + Transmit + Queue
  - Propagation = Distance / Speed Of Light (varies by medium)
  - Transmit = Size / Bandwidth
- **Important**: Talking about bit or message?



Bandwidth x Delay Product

- Product between bandwidth and delay
    - Units in bits (bps * s = b)
    - Delay generally measured as either one way latency, or RTT
        - Propagation Delay
    - Conceptually defines the maximum amount of data that can be "in-flight" at a given time
        - think the amount of water in a pipe





Example

- Consider a point to point link 50 km in length. At what bandwidth would propagation delay (at a speed of 2 * 10^8 m/s) equal transmit delay for 100 byte packets?
- What about 512 byte packets?

Example

- Consider a point to point link 50 km in length. At what bandwidth would propagation delay (at a speed of 2 * 10^8 m/s) equal transmit delay for 100 byte packets?
    - Propagation = Distance / Speed Of Light (varies by medium)
    - Transmit = Size / Bandwidth
    - Propagation delay = 50 * 10^3 m / (2 * 10^8 m/sec) = 250 μs
    - 100 * 8 = 800 bits -> 800 bits / 250 μs = 3.2 Mbps
- What about 512 byte packets?
    - 512 * 8 / 250 μs = 16.4 Mbps

Exercise

- 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 travel 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 the 5 Mbit picture to you on Earth. How quickly after the picture is taken can you receive the image from Betty?

Exercise

- 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 travel 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 = 184 * 128 * 10^3 = 2.81 MB
  - Say your aunt Betty takes a selfie on Olympus Mons, and sends the 5 Mbit picture to you on Earth. How quickly after the picture is taken can you receive the image from Betty?
    - Transmit delay for 5 MB = 41943040 bits / (128 * 10^3 bps) = 328 seconds. Total time = transmit delay + propagation delay = 328 + 184 = 512 seconds.

Thanks for coming!