Section 1:
Socket API & Traceroute

(§1.3.4, 6.1.2-6.1.4)

Originally by David Wetherall (djw@)

Outline

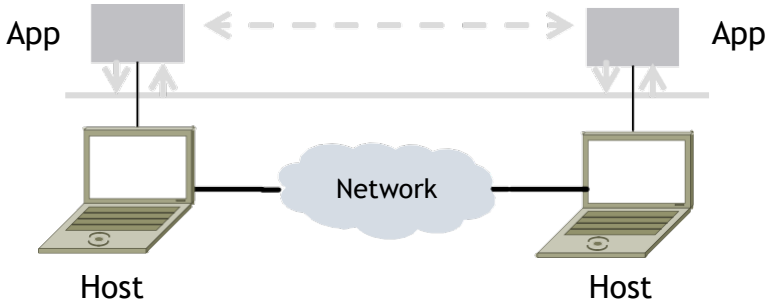
- Administrivia
- Project 1: Socket API
- Traceroute

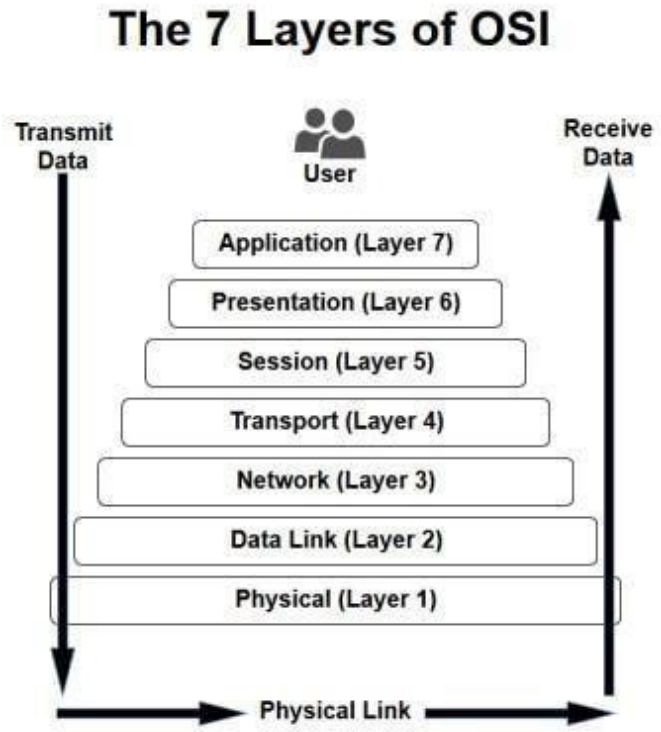
Administrivia

- Sections will be recorded
- Different weeks will be led by different TA's
- HW1 due Monday Apr 12
- Project 1 due Monday Apr 19

Network-Application Interface

- Application Layer APIs
  - Defines how apps use the network
  - Lets apps talk to each other
  - Hides the other layers of the network





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

Project 1

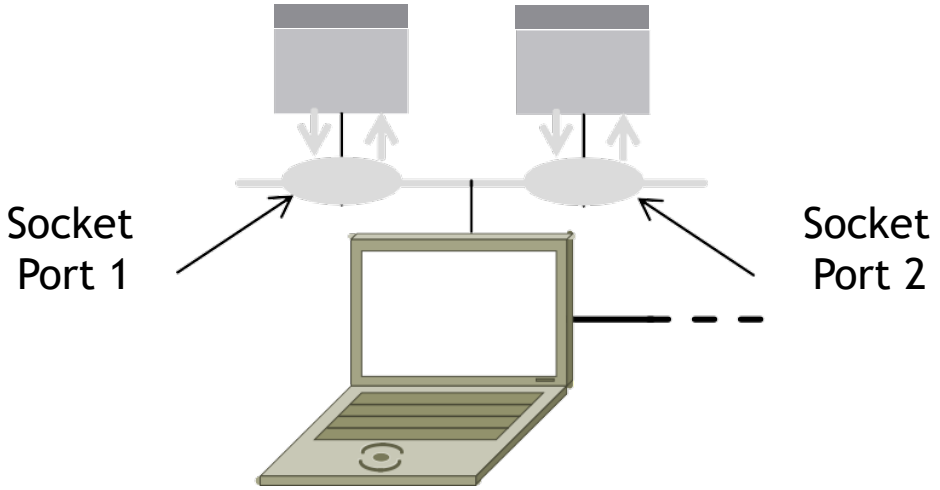
- This is the basis for many apps!
  - File transfer: send name, get file (§6.1.4)
  - Web browsing: send URL, get page
  - Echo: send message, get it back
- Let's see how to write this app …

Socket API (Generalized)

- 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
  - Datagrams (UDP): unreliably send separate messages

Socket API (2)

- **Sockets** let apps attach to the local network at different ports
- **Ports** are used by OS to distinguish services/apps using internet

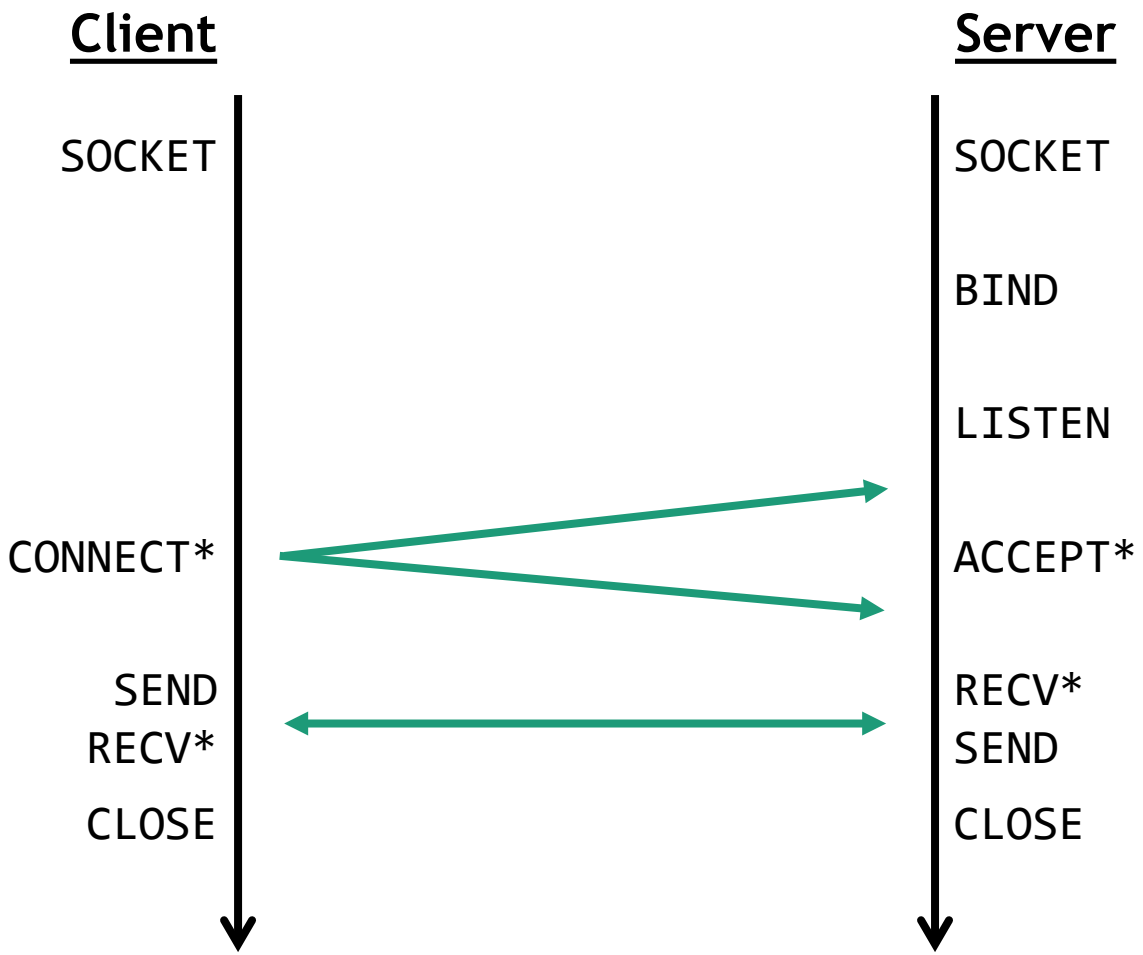


Socket API (3)

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 Sockets



* Denotes a blocking call
Use threads to avoid blocking

Client Program (Outline)

```
socket();         // create socket
getaddrinfo();    // server and port name
                  // www.example.com:80
connect();        // connect to a server [blocking]
…
send();           // send data
recv();           // await reply [blocking]
…                 // process reply
close()           // done, disconnect
```

Server Program (Outline)

```
socket();           // create socket
getaddrinfo();      // get info for port on this host
bind();             // associate port with socket
listen();           // start accepting connections
while (true) {
  accept();         // wait for a connection [blocking]
                    // returns a new socket
  {                 // spawn a new thread for each connection
    recv();         // wait for request [blocking]
    …               // process reply
    send();         // send reply
    close();        // close connection with client
  }
}
close();            // close the server socket
```

Java Tips

- `ServerSocket` for TCP server socket
- `Socket` for TCP client socket
- `DatagramSocket` for UDP server/client socket

Some other useful utils:

- `ByteBuffer` to manipulate bytes

Python Tips

- `socket.socket(socket.AF_INET, socket.SOCK_DGRAM)` for UDP
- `socket.socket(socket.AF_INET, socket.SOCK_STREAM)` for TCP

Might be useful:

- `socketserver`
- `struct.pack()` and `struct.unpack()` to manipulate bytes

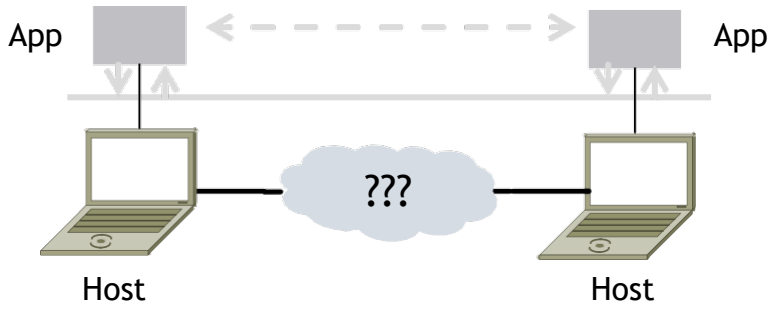
Some guidelines

- Make sure your code runs on **`attu`**.
  - Python users can only use packages that are available on **`attu`**

    (no **`pip`** unfortunately)
- Small portions of the grade will be awarded to robustness of your server
  - Your server should accept clients outside localhost
  - Close connection when client sends faulty packets or timeout.
    - Padding and payload length; Number of packets; Correct content; etc.
  - Multithreaded?

Traceroute

- Apps talk to other apps with no real idea of what is inside the network
  - This is good! But you may be curious …
- Peeking inside the Network with Traceroute



Traceroute

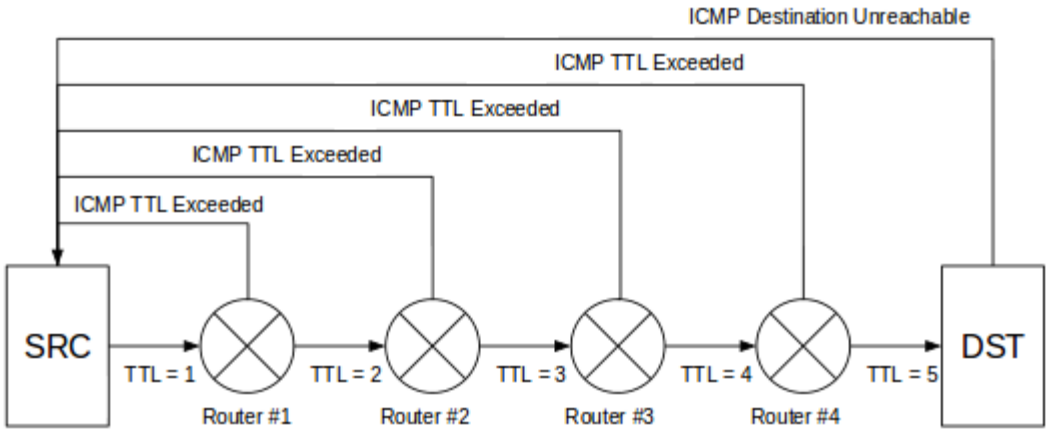
- Widely used command-line tool to let hosts peek inside the network
- On all OSes (`tracert` on Windows)
- Developed by Van Jacobson ~1987
- Uses a network-network interface (IP) in ways we will explain later

Van Jacobson

Credit: Wikipedia (public domain)

Traceroute

- Probes successive hops to find network path
- Takes advantage of ICMP error messages
- Packets keep track of a Time To Live (TTL)
  - Reduced by 1 at every hop
  - Packet is not forwarded if this value reaches 0; returns ICMP error message
  - Determines the number of hops a packet can make
  - Prevents circular routing



Using Traceroute

Hop	RTT 1	RTT 2	RTT 3	Reverse DNS [IP]

* = No Response within a certain timeout (Not all routers/servers respond to ICMP traffic)

Using Traceroute (2)

- ISP names and places are educated guesses