10/4 Scribe Notes

Transport Layer:
UDP: Connectionless protocol
TDP: Connection-oriented protocol

- Transport services provide logical communication between applications processes running on different hosts

Transport protocols actions (end to end system communication):

1. Sender: breaks application message into segments, passes to network layer
2. Receiver: reassembles segments into messages, passes to application layer

- Two main transport layer protocols: TCP, UDP

Network layer: logical communication between hosts
Transport layer: logical communication between processes (enhances network layer services)

During Transport layer:

- Transport layer functionalities are primarily implemented at the edge of the network

Sender:

1. Is passed an application layer message
2. Determines segment header fields values
3. Creates segment
4. Passes segment to IP

Receiver:

1. Receives segment from IP
2. Checks header values
3. Extract application-layer message
4. Demultiplexes message up to application via socket

TCP (Transmission Control Protocol):

- Reliable, in-order delivery (TCP will reorder packets and make sure every packet is sent properly)
- Congestion control (ensure multiple senders do not overload connection links)
- Flow control (make sure receiver is not overwhelmed)
    - Decide how much information to send at each node
- Connection setup (TCP creates a connection then sends packets)

UDP (User Datagram Protocol):

- Unreliable unordered delivery (packets could be lost and sent out of order for the sake of speed, most commonly in games)
- No-frills extension of “best-effort” IP
- Services not provided:

    - Delay guarantees
    - Bandwidth guarantees

Demultiplexing: Going up the transport to application layer
Multiplexing: Going from application layer down to transport layer

- Usually when packets are sent, these packets are nested packets which each have their own layer involved. (e.g. an ipv4 packet containing an ipv6 packet which contains a TCP packet which contains an HTTP packet, it demultiplexes as it moves from ip -> TCP -> HTTP)

Socket:

- Process sends/receives messages to/from its socket
- TCP and UDP send messages from sockets
- Analogous to a door
    - Sending process shoves message out door
    - Sending process relies on transport infrastructure on other door to deliver message
    - Receiving message is incoming packet from door

- To receive a message, the process must have an identifier.
- Host device has a unique 32-bit IP address (source and destination port number)
- IP address of host is not enough to identify the packet because many processes can run on same host
- Identifier includes both IP and port associated with process

Common Ports:

- HTTP server: 80
- Mail server: 25
- SSH: 22
- HTTPS: 443

Multiplexing at sender:

- Handle data from multiple sockets, add transport header.

Demultiplexing at receiver:

- Host receives IP datagrams and each datagram has source and destination
- Each datagram contains one transport-layer segment
- Each segment has source, destination port number
- Host uses ip and port to direct segment to appropriate socket
- Use demultiplexing for the sake of scalability to run multiple processes at a time

Connection-oriented demultiplexing:

- One socket per connection
- TCP socket identified by 4-tuple
    - Source IP address
    - Source port number
    - Dest IP address
    - Dest port number
- Demux: receiver uses all four values to direct segment to appropriate socket
- Server may support many simultaneous TCP sockets and each socket is identified by this 4-tuple
- Multiplexing and demultiplexing is based on the segment, datagram header field values

UDP: demultiplexing using destination port number

- When creating datagram to send into UDP socket, must specific destination ip and port #
- When receiving host gets UDP segment, checks destination port and directs packet to process with that port number
- Same destination port number, but different source IP address/port number, it goes to same socket

TCP: demultiplexing using 4-tuple: source ip/port, destination ip/port

- same destination IP address/port number, but different source IP address/port number, each connection has its own socket

Graham Quotes:
“Its ok to die”