Today’s Lecture

Motivation

- IP provides a weak, but efficient service model (best-effort)
  - Packets can be delayed, dropped, reordered, duplicated
- IP packets are addressed to a host
  - How to decide which application gets which packets?
- How should hosts send packets into the network?
  - Too fast may overwhelm the network
  - Too slow is not efficient

Outline

- Motivation
- Transport layer
  - TCP
  - UDP
- Provide a way to decide which packets go to which applications (multiplexing-demultiplexing)
- Can
  - Provide reliability, in order delivery, at most once delivery
  - Support messages of arbitrary length
  - Govern when hosts should send data → can implement congestion and flow control

Transport Layer
Congestion vs. Flow Control

- Flow Control – avoid overflowing the receiver
- Congestion Control – avoid congesting the network
- What is network congestion?

Flow Control
- avoid overflowing the receiver

Congestion Control
- avoid congesting the network

What is network congestion?

Transport Layer (cont’d)

UDP: Not reliable
TCP: Ordered, reliable, well-paced

Ports

- Need to decide which application gets which packets
- Solution: map each socket to a port
- Client must know server’s port
- Separate 16-bit port address space for UDP and TCP
  - (src_IP, src_port, dst_IP, dst_port) uniquely identifies TCP connection
- Well-known ports (0-1023): everyone agrees which services run on these ports
  - e.g., ssh:22, http:80
  - On UNIX, must be root to gain access to these ports (why?)
- Ephemeral ports (most 1024-65535): given to clients
  - e.g. chat clients, p2p networks

UDP User (Unreliable) Data Protocol

- Minimalist transport protocol
- Same best-effort service model as IP
- Messages up to 64KB
- Provides multiplexing/demultiplexing to IP
- Does not provide flow and congestion control
- Application examples: video/audio streaming

Outline

- Motivation
- Transport Layer
  - UDP
  - TCP

Headers

- IP header – used for IP routing, fragmentation, error detection
- UDP header – used for multiplexing/demultiplexing, error detection
- TCP header – used for multiplexing/demultiplexing, flow and congestion control
**UDP Service & Header**

- **Service:**
  - Send datagram from (IPa, Port1) to (IPb, Port2)
  - Service is unreliable, but error detection possible

- **Header:**
  - Source port: 0
  - Destination port: 16
  - UDP length: 31
  - UDP checksum: (variable)

**TCP Service**

1) Open connection: 3-way handshaking
2) Reliable byte stream transfer from (IPa, TCP_Port1) to (IPb, TCP_Port2)
   - Indication if connection fails: Reset
3) Close (tear-down) connection

**Outline**

- **Motivation**
- **Transport Layer**
- **UDP**
  - TCP

**TCP: Transport Control Protocol**

- Reliable, in-order, and at most once delivery
- Stream oriented: messages can be of arbitrary length
- Provides multiplexing/demultiplexing to IP
- Provides congestion control and avoidance
- Application examples: file transfer, chat

**Open Connection: 3-Way Handshaking**

- Goal: agree on a set of parameters: the start sequence number for each side
  - Starting sequence numbers are random

**3-Way Handshaking (cont’d)**

- Three-way handshake adds 1 RTT delay
- Why?
  - Congestion control: SYN (40 byte) acts as cheap probe
  - Protects against delayed packets from other connection (would confuse receiver)
Close Connection (Two-Army Problem)

- Goal: both sides agree to close the connection
- Two-army problem:
  - “Two blue armies need to simultaneously attack the white army to win; otherwise they will be defeated. The blue army can communicate only across the area controlled by the white army which can intercept the messengers.”

  - What is the solution?

Stop & Wait

- Send; wait for ack
- If timeout, retransmit; else repeat

Sliding Window

- \( \text{window} \) = set of adjacent sequence numbers
- The size of the set is the \( \text{window size} \)
- Assume window size is \( n \)
- Let \( A \) be the last ack'd packet of sender without gap; then window of sender = \( \{A+1, A+2, \ldots, A+n\} \)
- Sender can send packets in its window
- Let \( B \) be the last received packet without gap by receiver, then window of receiver = \( \{B+1, \ldots, B+n\} \)
- Receiver can accept out of sequence, if in window

Reliable Transfer

- Retransmit missing packets
  - Numbering of packets and ACKs
- Do this efficiently
  - Keep transmitting whenever possible
  - Detect missing ACKs and retransmit quickly
- Two schemes
  - Stop & Wait
  - Sliding Window (Go-back-n and Selective Repeat)

Go-Back-n (GBN)

- Transmit up to \( n \) unacknowledged packets
- If timeout for ACK(\( A \)), retransmit \( k, k+1, \ldots \)
**GBN Example w/o Errors**

Sender Window: 
1. `{1}`
2. `{1, 2}`
3. `{1, 2, 3}`
4. `{2, 3, 4}`
5. `{3, 4, 5}`
6. `{4, 5, 6}`

Receiver Window: 
- `{}`
- `{1}`
- `{1, 2}`
- `{1, 2, 3}`
- `{2, 3, 4}`
- `{3, 4, 5}`
- `{4, 5, 6}`

**SR Example with Errors**

Sender Window: 
1. `{1}`
2. `{1, 2}`
3. `{1, 2, 3}`
4. `{2, 3, 4}`
5. `{3, 4, 5}`
6. `{4, 5, 6}`

Receiver Window: 
- `{1}`
- `{1, 2}`
- `{1, 2, 3}`
- `{2, 3, 4}`
- `{3, 4, 5}`
- `{4, 5, 6}`

**GBN Example with Errors**

Sender Window: 
1. `{1}`
2. `{1, 2}`
3. `{1, 2, 3}`
4. `{2, 3, 4}`
5. `{3, 4, 5}`
6. `{4, 5, 6}`

Receiver Window: 
- `{1, 2}`
- `{1, 2, 3}`
- `{2, 3, 4}`
- `{3, 4, 5}`
- `{4, 5, 6}`

**Selective Repeat (SR)**

- Sender: transmit up to $n$ unacknowledged packets; assume packet $k$ is lost
- Receiver: indicate packet $k$ is missing
- Sender: retransmit packet $k$

**TCP Flow Control**

- Each byte has a sequence number
- Initial sequence numbers negotiated via SYN/SYN-ACK packets
- ACK contains the sequence number of the next byte expected by the receiver

**Observations**

- With sliding windows, it is possible to fully utilize a link, provided the window size is large enough. Throughput is $\approx \frac{n}{RTT}$
- Stop & Wait is like $n = 1$.
- Sender has to buffer all unacknowledged packets, because they may require retransmission
- Receiver may be able to accept out-of-order packets, but only up to its buffer limits
### TCP Flow Control

- Receiver window (MaxRcvBuf – maximum buffer size at receiver)

  \[ \text{AdvertisedWindow} = \text{MaxRcvBuf} - (\text{LastByteRcvd} - \text{LastByteRead}) \]

- Sender window (MaxSendBuf – maximum buffer size at sender)

  \[ \text{SenderWindow} = \text{AdvertisedWindow} - (\text{LastByteSent} - \text{LastByteAcked}) \]

  \[ \text{MaxSendBuffer} \geq \text{LastByteWritten} - \text{LastByteAcked} \]

### Retransmission Timeout (cont’d)

- If haven’t received ack by timeout, retransmit packet after last acked packet

- How to set timeout?
  - Too long: connection has low throughput
  - Too short: retransmit packet that was just delayed
    - Packet was probably delayed because of congestion
    - Sending another packet too soon just makes congestion worse
  - Solution: make timeout proportional to RTT

### Retransmission Timeout

- If haven’t received ack by timeout, retransmit packet after last acked packet

- How to set timeout?
  - Too long: connection has low throughput
  - Too short: retransmit packet that was just delayed
    - Packet was probably delayed because of congestion
    - Sending another packet too soon just makes congestion worse
  - Solution: make timeout proportional to RTT

### RTT Estimation

- Use exponential averaging:

  \[ \text{SampleRTT} = \text{AckRcvdTime} - \text{SendPacket Time} \]

  \[ \text{EstimatedRTT} = \alpha \times \text{EstimatedRTT} + (1 - \alpha) \times \text{SampleRTT} \]

  \[ \text{TimeOut} = 2 \times \text{EstimatedRTT} \]

  \[ 0 < \alpha \leq 1 \]

### Exponential Averaging Example

- \( \text{EstimatedRTT} = g^t \times \text{EstimatedRTT} + (1 - g^t) \times \text{SampleRTT} \)

  Assume RTT is constant \( \Rightarrow \text{SampleRTT} = \text{RTT} \)

- \( \text{EstimatedRTT} (g=0.5) \)

- \( \text{EstimatedRTT} (g=0.8) \)
Problem

• How to differentiate between the real ACK, and ACK of the retransmitted packet?

TCP Header

• Sequence number, acknowledgement, and advertised window – used by sliding-window based flow control
• Flags:
  - SYN, FIN – establishing/terminating a TCP connection
  - ACK – set when Acknowledgement field is valid
  - URG – urgent data; Urgent Pointer says where non-urgent data starts
  - PUSH – don’t wait to fill segment
  - RESET – abort connection

Karn/Partridge Algorithm

• Measure SampleRTT only for original transmissions
• Exponential backoff → for each retransmission, double EstimatedRTT

Jacobson/Karels Algorithm

• Problem: exponential average is not enough
  - One solution: use standard deviation (requires expensive square root computation)
  - Use mean deviation instead

\[
\text{Difference} = \text{SampleRTT} - \text{EstimatedRTT} \\
\text{EstimatedRTT} = \text{EstimatedRTT} + \delta \times \text{Difference} \\
\text{Deviation} = \text{Deviation} + \delta \times \left| \text{Difference} - \text{Deviation} \right| \\
\text{Timeout} = \mu \times \text{EstimatedRTT} + \phi \times \text{Deviation} \\
0 < \delta \leq 1 \\
\mu = 1 \\
\phi = 4
\]

Summary

• UDP: Multiplex, detect errors
• TCP: Reliable Byte Stream
  - 3-way handshaking
  - Reliable transmissions: ACKs…
  - S&W not efficient → Go-Back-n
  - What to ACK? (cumulative, …)
  - Timer Value: based on measured RTT