Chapter 17 Transport Protocols

• Connection-Oriented Transport Protocol
  ➢ Reliable Network Service: Design Issues
  ➢ Unreliable Network Service: Design Issues
  ➢ TCP Congestion Control
    ➢ AIMD (Additive Increase/Multiplicative Decrease)
    ➢ Slow Start
• Connectionless Transport Protocol: UDP

Design Issues under Reliable Network Service: Addressing & Multiplexing

• Addressing: Target user specified by
  ➢ User identification
    ➢ Usually (host, port), called a socket in TCP
    ➢ Port represents a particular transport service (TS) user: e.g. “DNS” port 53, “Unix talk” port 517,
    ➢ see /etc/services on Unix
    ➢ Host address and Network number
      ➢ In an internet, a global internet address
• Multiplexing
  ➢ Multiple users employ same transport protocol
  ➢ User identified by port number or service access point (SAP)

Connection-Oriented Transport Protocol Mechanisms

• Logical connection between TS: end-to-end transport of data, shield TS users from details of underlying systems
• Reliable Sequencing Network Service
  ➢ virtually 100% reliable delivery by network service: e.g. reliable packet switched network using X.25
  ➢ Transport service is end-to-end protocol between two systems on same network
• Design Issues
  ➢ Addressing
  ➢ Multiplexing
  ➢ Flow Control
  ➢ Connection Establishment & Termination: Negotiation of optional parameters (e.g., QoS); Triggers allocation of transport entity resources (e.g., buffer space)
• Unreliable Network Service
  ➢ E.g., Internet
  ➢ More Design Issues

Design Issues: Flow Control in Transport Layer

• Challenges:
  ➢ Longer transmission delay between transport entities (end-to-end) compared with actual transmission time
  ➢ Delay in communication of flow control info
  ➢ Variable transmission delay: Difficult to use timeouts
• Mechanisms under Reliable Network Services
  ➢ Do nothing: Segments that overflow are discarded
  ➢ Refuse further segments
    ➢ Backpressure type of Mechanism: clumsy & coarse grained
  ➢ Use fixed sliding window protocol (sequence number /window size/ACK)
    ➢ Works well on reliable network: failure on ACK is taken as flow control indication
    ➢ Does not work well on unreliable network
      ➢ Can not distinguish between lost segment and flow control tactic
    ➢ Use credit scheme
Credit Allocation Scheme

- Use credit scheme
  - Greater control on reliable network
  - More effective on unreliable network
  - Decouples flow control from ACK
    - May ACK without granting credit and vice versa
  - Each octet has sequence number
  - Each transport segment has Seq number, Ack number and window size in header

Transport Protocol within Internetworking

- Unreliable Network Service: e.g. Internet
  - Segments may get lost
  - Segments may arrive out of order
- Design Issues of Transport Protocol
  - Ordered Delivery
  - Retransmission strategy
  - Duplication detection
  - Flow control (Credit Allocation Scheme)
  - Connection establishment/termination
  - Crash recovery

Design Issues under Unreliable Service

- Ordered Delivery
  - Segments may arrive out of order
    - Number segments sequentially: TCP numbers each byte sequentially
    - Segments are numbered by the first octet number in the segment
- Retransmission Strategy
  - Segment damaged/failed in transit
    - Transmitter does not know of failure
    - Receiver must acknowledge successful receipt
  - Use cumulative acknowledgement for efficiency
  - Time out waiting for ACK triggers re-transmission
    - Fixed timer: Based on understanding of network behavior
    - Too small leads to unnecessary re-transmissions
    - Too large and response to lost segments is slow
    - Should be a bit longer than round trip time
    - Can not adapt to changing network conditions
    - Adaptive scheme

Duplication Detection

- If ACK lost, segment is re-transmitted -> duplicate
- Receiver must recognize duplicates
- Duplicate received prior to closing connection
  - Receiver assumes ACK lost and ACKs duplicate
  - Sender must not get confused with multiple ACKs
  - Sequence number space large enough to not cycle within maximum life of segment
- Duplicate received after closing connection
Three Way Handshake: Connection Establishment in TCP

- Entity in CLOSE WAIT state sends last data segment, followed by FIN
  - FIN could arrive before last data segment
- Receiver accepts FIN
  - Closes connection
  - Loses last data segment
- Associate sequence number with FIN
- Receiver waits for all segments before FIN sequence number

TCP Services & Mechanisms

- Reliable communication service between pairs of processes
  - Across variety of reliable and unreliable networks and internets
- Connection establishment
  - Three way handshake
  - Between pairs of ports
- Data transfer
  - Logical stream of octets
  - Flow control by credit allocation of number of octets
  - Data buffered at transmitter and receiver
- Connection termination
  - TCP users issues CLOSE primitive
  - Transport entity sets FIN flag on last segment sent

TCP Congestion Control

- Window management
  - AIMD
  - Slow Start
- Retransmission timer management
  - Estimate round trip delay by observing pattern of delay
  - Set time to value somewhat greater than estimate
  - Simple average
  - Exponential average
  - RTT Variance Estimation (Jacobson’s algorithm)
Additive Increase/Multiplicative Decrease

- Objective: adjust to changes in the available window capacity
  - Instead of a fixed size window for a TCP connection
- New state variable per connection: CongestionWindow
  - limits how much data source has in transit
  
  Note: CongestionWindow is set by TCP source based on the observed congestion level
- Idea:
  - increase CongestionWindow when congestion goes down
  - decrease CongestionWindow when congestion goes up
- Question: how does the source determine whether or not the network is congested?
  - Answer: a timeout occurs
    - timeout signals that a packet was lost
    - packets are seldom lost due to transmission error
    - lost packet implies congestion

AIMD: Additive Increase/Multiplicative Decrease

- Algorithm
  - increment CongestionWindow by one packet per RTT (linear increase)
  - divide CongestionWindow by two whenever a timeout occurs (multiplicative decrease)

AIMD (cont)

- Trace: sawtooth behavior
- Reason of being conservative: consequences of having too large a window are much worse than those of it being too small
  - Note: the accurate timeout mechanism is desirable
- Problem: it takes too long to ramp up a connection when it is starting from scratch (cold start)

Slow Start

- Slow start increase the congestion window exponentially rather than linearly
- Idea:
  - begin with CongestionWindow = 1 packet
  - double CongestionWindow each RTT (increment by 1 packet for each ACK)
- Why “slow”?
  - compare with the original behavior of TCP (advertised window), not compare with “linear growth” of AIMD
  - Exponential growth, but slower than all at once (original TCP)
- Used:
  - when first starting connection
  - when connection goes dead waiting for timeout (more knowledge)
- In Reality
  - interplay of “slow start” & “AIMD”
UDP

- User datagram protocol: RFC 768
- **Connectionless** service for application level procedures
  - Unreliable
  - Delivery and duplication control **not guaranteed**
  - **Adds multiplexing**
- Advantages: reduced overhead
  - E.g., real time applications such as voice and telemetry

<table>
<thead>
<tr>
<th>Bit:</th>
<th>0</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>Destination Port</td>
<td>Length</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

Spring, 2003 EE 4272