Chapter 17  Transport Protocols

• Connection-Oriented Transport Protocol
  ➢ Under Reliable Network Service
  ➢ Design Issues
  ➢ Under Unreliable Network Service
  ➢ Design Issues
  ➢ TCP & Congestion Control
    ➢ AIMD
    ➢ Slow Start

• Connectionless Transport Protocol: UDP

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
  ➢ Assume arbitrary length message, reliable
  ➢ 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

• Unreliable Network Service
  ➢ E.g., Internet
  ➢ More Design Issues

Design Issue under Reliable Network Service: Addressing

• Target user specified by:
  ➢ User identification
    ➢ Usually (host, port), called a socket in TCP
    ➢ Port represents a particular transport service (TS) user
  ➢ Transport entity identification
    ➢ Generally only one per host, not needed
    ➢ If more than one, then usually one of each type
      ➢ Specify type of transport protocol (TCP, UDP)
  ➢ Host address and Network number
    ➢ In an internet, a global internet address

Design Issue under Reliable Network Service (Con’t)

• Multiplexing
  ➢ Multiple users employ same transport protocol
  ➢ User identified by port number or service access point (SAP)

• Flow Control
  ➢ Flow need to be controlled
    ➢ The receiving user can not keep up
    ➢ The receiving transport entity can not keep up
  ➢ 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
Coping with Flow Control Requirements

- Do nothing: Segments that overflow are discarded
  - Sending transport entity will fail to get ACK and will retransmit
  - Thus further adding to incoming data
- 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
  - 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

Credit Allocation Scheme

- Design Issue under Reliable Network Service: Establishment/Termination
  - By mutual agreement
    - Allow each end to know the other end exists
    - Negotiation of optional parameters (e.g., QoS)
    - Triggers allocation of transport entity resources (e.g., buffer space)

Internetworking: Unreliable Network Service

- 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
  - 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

**Connection Establishment**

- Two way handshake
  - A send SYN, B replies with SYN
  - Lost SYN handled by re-transmission
  - Can lead to duplicate SYNs
  - Ignore duplicate SYNs once connected
  - Lost or delayed data segments can cause connection problems
    - Segment from old connections
    - Start segment numbers far removed from previous connection
    - Use SYN i
    - Need ACK to include i

- Three Way Handshake

**Three Way Handshake**

- A initiates a connection
- B accepts and acknowledges
- A acknowledges and begins transmission

- (a) Normal operation

- Obsolete SYN arrives
- B accepts and acknowledges
- A rejects B’s connection

- (b) Delayed SYN

- A initiates a connection
- OLD SYN arrive at A; A rejects
- B accepts and acknowledges

- (c) Delayed SYN, ACK

- A acknowledges and begins transmission
Connection Termination

- Entity in CLOSE WAIT state sends last data segment, followed by FIN
- FIN arrives before last data segment ?? Remember: Internet
- 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

- Reliable communication between pairs of processes
- Across variety of reliable and unreliable networks and internets
- Two labeling facilities
  - Data stream push (sender side)
    - TCP user can request transmission of all data with push flag
    - Avoids waiting for full buffers
  - Urgent data signal (receiver side)
    - Indicates urgent data is upcoming in stream
    - User decides how to handle it

TCP Mechanisms

- 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 capacity
- 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 (cont)

- Algorithm
  - increment CongestionWindow by one packet per RTT (linear increase)
  - divide CongestionWindow by two whenever a timeout occurs (multiplicative decrease)
  - Note: In practice: increment a little for each ACK
  - Increment = (MSS * MSS)/CongestionWindow
  - CongestionWindow += Increment

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
**Slow Start (cont)**

- Exponential growth, but slower than all at once (original TCP)
- Used…
  - when first starting connection
  - when connection goes dead waiting for timeout (more knowledge)
- Trace of TCP CongestionWindow: interplay of “slow start” & “AIMD”

- Problem: lose up to half a CongestionWindow’s worth of data
- Idea of so-called “packet pair”: gap between the ACKs

---

**UDP**

- User datagram protocol: RFC 768
- Connectionless service for application level procedures
  - Unreliable
  - Delivery and duplication control not guaranteed
  - Adds multiplexing

- Reduced overhead

![UDP Diagram]