TCP Services

- Reliability mechanisms: soft state, sequence numbers, retransmissions, timeouts, 3-way handshake
- Congestion Control: dynamic windows, robust stability issues.
TCP: Key features

- Connection-oriented
- Point-to-point: 2 end-points (no broadcast or multicast)
- Reliable transfer: Data is delivered in-order
- Full duplex communication
- Byte-stream interface: sequence of octets
- Reliable startup: Data on old connection does not confuse new connections
- Graceful shutdown: Data sent before closing a connection is not lost. Reset or immediate shutdown also possible.

Reliability

- Reliability provided by:
  - Reliable connection startup: Data on old connection does not confuse new connections
  - Graceful connection shutdown: Data sent before closing a connection is not lost.
  - Data segmented for transmission and acknowledged by destination. Timeout + Retransmission provided if data unacknowledged. This is formally called "soft-state"
  - Checksum provided to catch errors.
  - Re-sequencing of out-of-order data; discarding of duplicate data.
  - Window flow control => sender cannot overrun receiver buffers
TCP Header Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Size in bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>16</td>
<td>Identifies source user process</td>
</tr>
<tr>
<td>20 = FTP, 23 = Telnet, 53 = DNS, 80 = HTTP, ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination Port</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Sequence Number</td>
<td>32</td>
<td>Sequence number of the first byte in the segment. If SYN is present, this is the initial sequence number (ISN) and the first data byte is ISN+1.</td>
</tr>
<tr>
<td>Ack number</td>
<td>32</td>
<td>Next byte expected</td>
</tr>
</tbody>
</table>

TCP Header Details

- Source Port
- Destination Port
- Sequence Number
- Ack number
Three-Way Handshake

- Necessary and sufficient for unambiguous setup/teardown even under conditions of loss, duplication, and delay.

TCP Interactive Data Flow

- Problems:
  - Overhead: 40 bytes header + 1 byte data
  - To batch or not to batch: response time important

- Batching acks:
  - Delay-ack timer: piggyback ack on echo
    - 200 ms timer (fig 19.3)

- Batching data:
  - Nagle's algo: Don't send packet until next ack is received.
  - Developed because of congestion in WANs.
TCP Bulk Data Flow

- Sliding window: Send multiple packets while waiting for acks (fig 20.1) up to a limit (W).
- Receiver need not ack every packet.
- Acks are cumulative. Ack # = Largest consecutive sequence number received + 1.
- Two transfers of the data can have different dynamics (eg: fig 20.1 vs fig 20.2).

TCP Bulk Data Flow (Contd)

- End-to-end flow control.
- Window update acks: receiver ready.
- Default buffer sizes: 4096 to 16384 bytes.
- Ideal: window and receiver buffer = bandwidth-delay product.
- Right edge, Left edge, usable window.
- "Closes" => left edge (snd_una) advances.
- "Opens" => right edge advances (receiver buffer freed => receiver window increases).
- "Shrinks" => right edge moves to left (rare).
The Congestion Problem

Problem: demand outstrips available capacity ...

Q: Will the "congestion" problem be solved when:

a) Memory becomes cheap (infinite memory)?
   No buffer
   Too late
   All links 19.2 kb/s
   Replace with 1 Mb/s

File Transfer Time = 7 hours

File Transfer time = 5 mins

Q: b) Links become cheap (high speed links)?

Ans: None of the above solves congestion!

Congestion: Demand > Capacity

It is a dynamic problem => Static solutions are not sufficient

TCP provides a dynamic solution

Scenario: All links 1 Gb/s. A & B send to C.

c) Processors become cheap (fast routers switches)?
TCP Congestion Control

Window flow control
- avoid receiver overrun

Dynamic window congestion control:
- avoid/control network overrun

Observation:
- Not a good idea to start with a large window and dump packets into network
- Treat network like a black box and start from a window of 1 segment ("slow start")
- Increase window size exponentially ("exponential increase") over successive RTTs => quickly grow to claim available capacity.
- Technique: Every ack: increase cwnd (new window variable) by 1 segment.
- Effective window = Min(cwnd, Wrcvr)

Dynamics
- Rate of acks = rate of packets at the bottleneck: "Self-clocking" property.
- 100 Mbps
- 10 Mbps
- Router Q
- 1st RTT
- 2nd RTT
- 3rd RTT
- 4th RTT
Packet loss as an indicator of congestion.

Set slow start threshold ($ssthresh$) to $\min(cwnd, W_{rcvr})/2$.

Retransmit pkt, set $cwnd$ to 1 (reenter slow start).

**Congestion Window ($cwnd$)**

- **Receiver Window**
- **Idle Interval**
- **Timeout**

$1$ ssthresh

Shivkumar Kalyanaraman
Rensselaer Polytechnic Institute

**Congestion avoidance**

Increment $cwnd$ by 1 per ack until $ssthresh$.

Increment by $1/cwnd$ per ack afterwards ("Congestion avoidance" or "linear increase").

**Idea:**
$ssthresh$ estimates the bandwidth-delay product for the connection.

**Initialization:**
$ssthresh$ = Receiver window or default $65535$ bytes. Larger values thru options.

If source is idle for a long time, $cwnd$ is reset to one.
Timeout and RTT Estimation

- **Timeout:** for robust detection of packet loss

- **Problem:** How long should timeout be?
  - Too long => underutilization; too short => wasteful retransmissions

- **Solution:** adaptive timeout: based on RTT

- **RTT estimation:**
  - **Early method:** exponential averaging:
    \[
    R ← αR + (1 - α)M \quad \{M = \text{measured RTT}\}
    \]
  - \[RTO = βR\quad \{β = \text{delay variance factor}\}\]

- **Suggested values:**
  - \[α = 0.9, \quad β = 2\]

Timer Backoff / Karn's Algorithm

- **Timer backoff:** If timeout, \[RTO = 2 \times RTO\] \{exponential backoff\}

- **Retransmission ambiguity problem:**
  - During retransmission, it is unclear whether an ack refers to a packet or its retransmission.

- **Problem for RTT estimation:**
  - Karn/Partridge: don't update RTT estimators during retransmission.
  - Restart RTO only after an ack received for a segment that is not retransmitted
TCP Performance Optimization

- SACK: selective acknowledgments specify blocks of packets received at destination.
- Random early drop (RED) scheme spreads the dropping of packets more uniformly and reduces average queue length and packet loss rate.
- Scheduling mechanisms protect well-behaved flows from rogue flows.

Congestion control summary

- Sliding window limited by receiver window.
- Dynamic windows: slow start (exponential rise), congestion avoidance (linear rise), multiplicative decrease.
- Adaptive timeout: need mean RTT & deviation
- Timer back off and Karn's algo during retransmission
- Go-back-N or Selective retransmission
- Cumulative and Selective acknowledgements
- Timeout avoidance: FRR
- Drop policies, scheduling and ECN
Summary

- TCP reliability model and mechanisms
- Congestion control mechanisms