Reliable Byte-Stream (TCP)

Outline
  Connection Establishment/Termination
  Sliding Window Revisited
  Flow Control
  Adaptive Timeout

End-to-End Protocols

- Underlying best-effort network
  - drop messages
  - re-orders messages
  - delivers duplicate copies of a given message
  - limits messages to some finite size
  - delivers messages after an arbitrarily long delay

- Common end-to-end services
  - guarantee message delivery
  - deliver messages in the same order they are sent
  - deliver at most one copy of each message
  - support arbitrarily large messages
  - support synchronization
  - allow the receiver to flow control the sender
  - support multiple application processes on each host
Simple Demultiplexor (UDP)

- Unreliable and unordered datagram service
- Adds multiplexing
- No flow control
- Endpoints identified by ports
  - servers have well-known ports
  - see /etc/services on Unix
- Header format
  - Optional checksum
    - pseudo header + UDP header + data

![Header Format Diagram]

UDP

![UDP Diagram]
TCP Overview

- Connection-oriented
- Byte-stream
  - app writes bytes
  - TCP sends segments
  - app reads bytes
- Full duplex
- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network

Segment Format

- SrcPort
- DstPort
- SequenceNum
- Acknowledgment
- HdrLen
- Flags
- AdvertisedWindow
- Checksum
- UrgPtr
- Options (variable)
- Data
Segment Format (cont)

- Each connection identified with 4-tuple:
  - (SrcPort, SrcIPAddr, DsrPort, DstIPAddr)
- Sliding window + flow control
  - ACK, SequenceNum, AdvertisedWindow

- Flags
  - SYN, FIN, RESET, PUSH, URG, ACK
- Checksum
  - pseudo header + TCP header + data

Connection Establishment

Active participant (client)  Passive participant (server)

- SYN, SequenceNum = x
- SYN+ACK, SequenceNum = y
- ACK, Acknowledgment = x+1
- Acknowledgment = y+1
Connection Termination

First participant

FIN, SequenceNum = x

ACK, Acknowledgment = x + 1

Second participant

FIN, SequenceNum = y,

Acknowledgment = y + 1

ACK, Acknowledgment = x + 1

State Transition Diagram

CLOSED

LISTEN

SYN_RCVD

SYN_SENT

ESTABLISHED

CLOSE_WAIT

LAST_ACK

CLOSING

TIME_WAIT

FIN_WAIT_2

FIN_WAIT_1

Passive open /SYN

Active open /SYN

Send.SYN

SYN + ACK

FIN/ACK

Timeout after two segment lifetimes
Sliding Window Revisited

- **Sending side**
  - $\text{LastByteAcked} \leq \text{LastByteSent}$
  - $\text{LastByteSent} \leq \text{LastByteWritten}$
  - buffer bytes between $\text{LastByteAcked}$ and $\text{LastByteWritten}$

- **Receiving side**
  - $\text{LastByteRead} < \text{NextByteExpected}$
  - $\text{NextByteExpected} \leq \text{LastByteRcvd} + 1$
  - buffer bytes between $\text{LastByteRead}$ and $\text{LastByteRcvd}$

Flow Control

- Send buffer size: $\text{MaxSendBuffer}$
- Receive buffer size: $\text{MaxRcvBuffer}$
- Receiving side
  - $\text{LastByteRcvd} - \text{LastByteRead} \leq \text{MaxRcvBuffer}$
  - $\text{AdvertisedWindow} = \text{MaxRcvBuffer} - ((\text{NextByteExpected} - 1) - \text{LastByteRead})$
- Sending side
  - $\text{LastByteSent} - \text{LastByteAcked} \leq \text{AdvertisementWindow}$
  - $\text{EffectiveWindow} = \text{AdvertisementWindow} - (\text{LastByteSent} - \text{LastByteAcked})$
  - $\text{LastByteWritten} - \text{LastByteAcked} \leq \text{MaxSendBuffer}$
  - block sender if $(\text{LastByteWritten} - \text{LastByteAcked}) + y > \text{MaxSenderBuffer}$

- Always send ACK in response to arriving data segment
- Persist when $\text{AdvertisementWindow} = 0$
Protection Against Wrap Around

- 32-bit SequenceNum

<table>
<thead>
<tr>
<th>Bandwidth</th>
<th>Time Until Wrap Around</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (1.5 Mbps)</td>
<td>6.4 hours</td>
</tr>
<tr>
<td>Ethernet (10 Mbps)</td>
<td>57 minutes</td>
</tr>
<tr>
<td>T3 (45 Mbps)</td>
<td>13 minutes</td>
</tr>
<tr>
<td>FDDI (100 Mbps)</td>
<td>6 minutes</td>
</tr>
<tr>
<td>STS-3 (155 Mbps)</td>
<td>4 minutes</td>
</tr>
<tr>
<td>STS-12 (622 Mbps)</td>
<td>55 seconds</td>
</tr>
<tr>
<td>STS-24 (1.2 Gbps)</td>
<td>28 seconds</td>
</tr>
</tbody>
</table>

Silly Window Syndrome

- How aggressively does sender exploit open window?

- Receiver-side solutions
  - after advertising zero window, wait for space equal to a maximum segment size (MSS)
  - delayed acknowledgements
Nagle’s Algorithm

- How long does sender delay sending data?
  - too long: hurts interactive applications
  - too short: poor network utilization
  - strategies: timer-based vs self-clocking

```java
when application produces data to send
if both the available data and the window >= MSS
  send a full segment
else
  if there is unACKed data in flight
    buffer the new data until an ACK arrives
  else
    send all the new data now
```

Adaptive Retransmission

- Round-Trip Time Estimation:
  - wait at least one RTT before retransmitting
  - importance of accurate RTT estimators:
    - Low RTT -> unneeded retransmissions
    - High RTT -> poor throughput
  - RTT estimator must adapt to change in RTT
    - But not too fast, or too slow!

- problem: If the instantaneously calculated RTT is 10, 20, 5, 12, 3, 5, 6; what RTT should we use for calculations?
  - $\text{EstimatedRTT} = \alpha \times \text{EstimatedRTT} + (1 - \alpha) \times \text{SampleRTT}$
  - recommended value for $\alpha$: 0.8 - 0.9
  - retransmit timer set to $\beta$ RTT, where $\beta = 2$
Retransmission Ambiguity

Karn/Partridge Algorithm

• Accounts for retransmission ambiguity
• If a segment has been retransmitted:
  – don’t count RTT sample on ACKs for this segment
  – reuse RTT estimate only after one successful transmission
  – double timeout after each retransmission
Jacobson/Karels Algorithm

• Key observation:
  – using $\beta$ RTT for timeout doesn’t work
  – at high loads round trip variance is high
• Solution:
  – if $D$ denotes mean variation
  – $\text{timeout} = \text{RTT} + 4D$

Jacobson/Karels Algorithm

• New Calculations for average RTT
• $\text{Diff} = \text{SampleRTT} - \text{EstimatedRTT}$
• $\text{EstimatedRTT} = \text{EstimatedRTT} + (d \times \text{Diff})$
• $\text{Dev} = \text{Dev} + d \times (|\text{Diff}| - \text{Dev})$
  – where $d$ is a factor between 0 and 1
• Consider variance when setting timeout value
• $\text{TimeOut} = m \times \text{EstimatedRTT} + f \times \text{Dev}$
  – where $m = 1$ and $f = 4$
Record Boundaries

- Byte-stream protocol: write 8+2+20 bytes and read 5+5+5+5+5+5 (loop).
- TCP offers two features to insert record boundaries:
  - URG flag
  - push operation

TCP Extensions

- Implemented as header options
- Better way to measure RTT (use actual system clock for sending time and add timestamp to segment).
- 64-bit sequence numbers: 32-bit sequence number in low-order 32 bits, timestamp in high-order 32 bits.
- Shift (scale) advertised window.