Outline

- TCP Connection Management
- Sliding Window
- ACK Strategy
- Nagle’s algorithm
- Timeout estimation
- Flow Control
TCP Connection Management

TCP client lifecycle

TCP server lifecycle
TCP state-transition diagram

A

I-finished(M)

FIN(M)

timer on FIN

FIN (N)

ACK (M+1)

Wait for B to finish

ack(N+1)

ACK(N+1)

I-finished

may send more data

wait for 2MSL before deleting the conn state

Done, delete state

B

CLOSED

LISTEN

SYN_SENT

SYN_RCVD

ESTABLISHED

FIN_WAIT_1

FIN_WAIT_2

CLOSING

TIME_WAIT

CLOSE_WAIT

LAST_ACK

CLOSED

Active open/SYN

Passive open

Close

Send/SYN

FIN/ACK

ACK

FIN/ACK

ACK

FIN/ACK

ACK

Close/FIN

Close/FIN

Close/FIN

Timeout after two segment lifetimes

CS 349/Fall05
Connection Reset

• RST Segments
  - When connection request arrives to non-existing server, OS sends back RST signal
  - Can be used to do *abortive release* of established connection
    • Receiver throws away queued data
Sliding Window Revisited

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

- **Receiving side**
  - $\text{LastByteRead} < \text{NextByteExpected}$
  - $\text{NextByteExpected} \leq \text{LastByteRcvd} + 1$
  - buffer bytes between $\text{NextByteRead}$ and $\text{LastByteRcvd}$
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>
Keeping the Pipe Full

- **16-bit AdvertisedWindow**

<table>
<thead>
<tr>
<th>Bandwidth</th>
<th>Delay x Bandwidth Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (1.5 Mbps)</td>
<td>18KB</td>
</tr>
<tr>
<td>Ethernet (10 Mbps)</td>
<td>122KB</td>
</tr>
<tr>
<td>T3 (45 Mbps)</td>
<td>549KB</td>
</tr>
<tr>
<td>FDDI (100 Mbps)</td>
<td>1.2MB</td>
</tr>
<tr>
<td>STS-3 (155 Mbps)</td>
<td>1.8MB</td>
</tr>
<tr>
<td>STS-12 (622 Mbps)</td>
<td>7.4MB</td>
</tr>
<tr>
<td>STS-24 (1.2 Gbps)</td>
<td>14.8MB</td>
</tr>
</tbody>
</table>

assuming 100ms RTT
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
## TCP Recvr: when to send ACK?

<table>
<thead>
<tr>
<th>Event</th>
<th>TCP Receiver action</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in-order segment arrival, no gaps, everything earlier already ACKed</strong></td>
<td>delayed ACK: wait up to 200ms, If nothing arrived, send ACK</td>
</tr>
<tr>
<td><strong>in-order segment arrival, no gaps, one delayed ACK pending</strong></td>
<td>immediately send one cumulative ACK</td>
</tr>
<tr>
<td><strong>out-of-order arrival: higher-than-expect seq. #, gap detected</strong></td>
<td>send duplicate ACK, indicating seq. # of next expected byte</td>
</tr>
<tr>
<td><strong>arrival of segment that partially or completely fills a gap</strong></td>
<td>immediate ACK if segment starts at the lower end of gap</td>
</tr>
</tbody>
</table>
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

- When application generates additional data
  - if fills a max segment (and window open): send it
  - else
    - if there is unack’ed data in transit: buffer it until ACK arrives
    - else: send it
TCP Flow Control

- receive side of TCP connection has a receive buffer:

- app process may be slow at reading from buffer

- flow control
  sender won’t overflow receiver’s buffer by transmitting too much, too fast

- speed-matching service: matching the send rate to the receiving app’s drain rate
TCP Flow control: how it works

(Suppose TCP receiver discards out-of-order segments)

- spare room in buffer
  = RcvWindow
  = RcvBuffer - [LastByteRcvd - LastByteRead]

- Rcvr advertises spare room by including value of RcvWindow in segments
- Sender limits unACKed data to RcvWindow
  - guarantees receive buffer doesn’t overflow
Flow Control (cont.)

• If the receiver indicated size of zero in last ACK how can the server send more data?
  - Sender periodically sends probes
  - Receiver sends *window update* ACK
Setting Timers

• The sender needs to set retransmission timers in order to know when to retransmit a packet the may have been lost

• How long to set the timer for?
  – Too short: may retransmit before data or ACK has arrived, creating duplicates
  – Too long: if a packet is lost, will take a long time to recover (inefficient)
Timing Illustration

Timeout too long $\rightarrow$ inefficiency

Timeout too short $\rightarrow$ duplicate packets
Adaptive Timers

- The amount of time the sender should wait is about the round-trip time (RTT) between the sender and receiver
  - For link-layer networks (LANs), this value is essentially known
  - For multi-hop WANS, rarely known
- Must work in both environments, so protocol should adapt to the path behavior
- Measure successive ack delays $T(n)$
  Set timeout = average + 4 deviations
RTT measurement and RTO

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

\[ \text{rttvar} = \text{rttvar} + \beta \times (|\text{diff}| - \text{rttvar}) \]

Where \( \text{diff} = \text{SampleRTT} - \text{SRTT} \)

Assume SRTT = 500msec, rttvar = 120, \( \alpha = 1/8 \), \( \beta = 1/4 \):

\( \text{diff} = \text{SampleRTT} - \text{SRTT} = 80\text{ms} \)

SRTT = SRTT + \( \alpha \times \text{diff} = 510\text{ms} \)

rttvar = rttvar + \( \beta \times (|\text{diff}| - \text{rttvar}) = 110 \)

RTO = SRTT + 4 \times rttvar = 510 + 440 = 950
How to measure RTT in case of retransmission?

- Karn’s algorithm
  - On retx, don’t update estimated RTT (and double RTO)
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?
Close Connection

- 4-ways tear down connection

- Avoid reincarnation
- Can retransmit FIN ACK if it is lost