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
  - Source Port (16)
  - Destination Port (16)
  - Checksum (16)
  - Length (2)
  - Data (variable)
- Optional checksum
  - pseudo header + UDP header + data
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

Data Link Versus Transport

- Potentially connects many different hosts
  - need explicit connection establishment and termination
- Potentially different RTT
  - need adaptive timeout mechanism
- Potentially long delay in network
  - need to be prepared for arrival of very old packets
- Potentially different capacity at destination
  - need to accommodate different node capacity
- Potentially different network capacity
  - need to be prepared for network congestion

Segment Format
Segment Format (cont)

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

```
        Data(SequenceNum)
           |
           v
 Sender --------+-------- Receiver
          |      |
          |      v
          Acknowledgment + AdvertisedWindow
```

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

Connection Establishment and Termination

```
Active participant  Passive participant
(client)             (server)

SYN, SequenceNum = x

SYN + ACK, SequenceNum = x + 1

ACK, Acknowledgment = y + 1
```

State Transition Diagram

```
```

Spring 2002
CS 461
Sliding Window Revisited

- Sending side
  - LastByteAcked <= LastByteSent
  - LastByteSent <= LastByteWritten
  - buffer bytes between LastByteAcked and LastByteWritten

- Receiving side
  - LastByteSent < NextByteExpected
  - NextByteExpected <= LastByteRcvd + 1
  - buffer bytes between NextByteRead and LastByteRcvd

Flow Control

- Send buffer size: MaxSendBuffer
- Receive buffer size: MaxRcvBuffer

- Receiving side
  - LastByteRcvd - LastByteRead <= MaxRcvBuffer
  - AdvertisedWindow - MaxRcvBuffer - NextByteRead

- Sending side
  - LastByteSent - LastByteAcked <= AdvertisedWindow
  - EffectiveWindow - AdvertisedWindow - (LastByteSent - LastByteAcked)
  - LastByteWritten - LastByteAcked <= MaxSendBuffer
  - block sender if (LastByteWritten - LastByteAcked + y) > MaxSenderBuffer

- Always send ACK in response to arriving data segment
- Persist when AdvertisedWindow = 0

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

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>569KB</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
TCP Extensions

- Implemented as header options
- Store timestamp in outgoing segments
- Extend sequence space with 32-bit timestamp (PAWS)
- Shift (scale) advertised window

Adaptive Retransmission
(Original Algorithm)

- Measure SampleRTT for each segment / ACK pair
- Compute weighted average of RTT
  - \( \text{EstRTT} = \alpha \times \text{EstRTT} + \beta \times \text{SampleRTT} \)
  - where \( \alpha + \beta = 1 \)
  - \( \alpha \) between 0.8 and 0.9
  - \( \beta \) between 0.1 and 0.2
- Set timeout based on EstRTT
  - \( \text{Timeout} = 2 \times \text{EstRTT} \)

Karn/Partridge Algorithm

- Do not sample RTT when retransmitting
- Double timeout after each retransmission
Jacobson/ Karels Algorithm

- New Calculations for average RTT
- \( \text{Diff} = \text{SampleRTT} - \text{EstRTT} \)
- \( \text{EstRTT} = \text{EstRTT} + (\delta \times \text{Diff}) \)
- \( \text{Dev} = \text{Dev} + \delta (|\text{Diff}| - \text{Dev}) \)
  - where \( \delta \) is a factor between 0 and 1
- Consider variance when setting timeout value
- \( \text{Timeout} = \mu \times \text{EstRTT} + \phi \times \text{Dev} \)
  - where \( \mu = 1 \) and \( \phi = 4 \)

Notes
- Algorithm only as good as granularity of clock (500ms on Unix)
- Accurate timeout mechanism important to congestion control (later)