CS 640 Introduction to Computer Networks

Lecture 15

Today’s lecture

• Transport layer – TCP

Congestion in the Internet

• Checksums are effective for detecting bit errors but they are not the only problem…
• We know that traffic is bursty
  – Statistical multiplexing of ON/OFF sources
  – Heavy-tailed file sizes
  – Routers have limited buffer capacity
  – Packets dropped when buffers full
    • Buffers do protect from short bursts
• Congestion lengthens delays and lowers throughput
  – Standard throughput/load curve
How can we deal with congestion?

- Over-provision networks
  - Very expensive
  - Commonly done
    - Networks designed to normally operate at 5-50% capacity
- Call admission control (phone networks)
- Develop protocols to respond to congestion
  - Route away from congestion
    - Good idea – how can we do it?
  - Retransmit in the face of loss
    - This is the state of the art

Congestion Control Basics

- UDP will send packets at any specified rate
  - Does not have mechanisms to handle congestion
- Issues:
  - Detecting congestion
  - Reacting to congestion
  - Avoiding congestion
    - Shaping traffic
    - QoS mechanisms
- Transport protocol will deal with congestion…

Congestion control in the Internet

- TCP implements congestion control
  - Detects congestion through packet losses
  - Reduces rate aggressively in response to congestion
  - Increases rate cautiously to use up available bandwidth
  - Works well for large flows
- Why the Internet doesn’t experience congestion collapse
  - Backbones overprovisioned
  - TCP congestion control
  - Sources’ rate limited by nearest bottleneck link
Next two lectures

- TCP
  - Introduction
  - Header format
  - Connection establishment and termination
  - Reliability
  - Roundtrip estimation
  - Congestion control (not today)

TCP Overview

- TCP is the most widely used Internet protocol
  - Web, Peer-to-peer, FTP, telnet, …
  - A focus of intense study for many years
- A two way, reliable, byte stream oriented end-to-end protocol
- Closely tied to the Internet Protocol (IP)
- Our goal is to understand the RENO version of TCP (most widely used TCP today)
  - mainly specifies mechanisms for dealing with congestion

TCP Features

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcPort</td>
<td>0</td>
</tr>
<tr>
<td>DstPort</td>
<td>4</td>
</tr>
<tr>
<td>SequenceNum</td>
<td>8</td>
</tr>
<tr>
<td>Acknowledgment</td>
<td>12</td>
</tr>
<tr>
<td>HdrLen</td>
<td>16</td>
</tr>
<tr>
<td>Flags</td>
<td>17</td>
</tr>
<tr>
<td>AdvertisedWindow</td>
<td>18</td>
</tr>
<tr>
<td>Checksum</td>
<td>20</td>
</tr>
<tr>
<td>UrgPtr</td>
<td>21</td>
</tr>
<tr>
<td>Options (variable)</td>
<td>22</td>
</tr>
<tr>
<td>Data</td>
<td>32</td>
</tr>
</tbody>
</table>

- 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 is the same as UDP
  - pseudo header + TCP header + data

**Sequence Numbers**

- 32 bit sequence numbers
  - Wrap around supported
- TCP breaks byte stream from application into packets (limited by Max. Segment Size)
- Each byte in the data stream is considered
- Each packet has a sequence number
  - Initial number selected at connection time
  - Subsequent numbers give first data byte in packet
- ACKs indicate *next byte expected*
### Sequence Number Wrap Around

<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>

*Protect against this by adding a 32-bit timestamp to TCP header*

---

### Connection Establishment

Active participant (client)  
SYN, SequenceNum = x
SYN + ACK, SequenceNum = y,  
ACK, Acknowledgment = y + 1
ACKnowledgment = x + 1

Passive participant (server)

---

### Connection Termination

Active participant (server)  
FIN, SequenceNum = x

Passive participant (client)  
Acknowledgment = y + 1
FIN, SequenceNum = y
Acknowledgment = x + 1
Reliability in TCP

- Checksum used to detect bit level errors
- Sequence numbers help detect sequencing errors
  - Duplicates are ignored
  - Out of order packets are reordered (or dropped)
  - Lost packets are retransmitted
- Timeouts used to detect lost packets
  - Requires RTO calculation
  - Requires sender to maintain data until it is ACKed

Sliding Window Revisited

- Sending side
  - LastByteAcked ≤ LastByteSent
  - LastByteSent ≤ LastByteWritten
  - buffer bytes between LastByteAcked and LastByteWritten

- Receiving side
  - LastByteRead < NextByteExpected
  - NextByteExpected < LastByteRcvd + 1
  - buffer bytes between LastByteRead and LastByteRcvd
Flow Control in TCP

- Send buffer size: MaxSendBuffer
- Receive buffer size: MaxRcvBuffer
- Receiving side
  - LastByteRcvd - LastByteRead ≤ MaxRcvBuffer
  - AdvertisedWindow = MaxRcvBuffer - (NextByteExpected - 1)
- Sending side
  - LastBytesWritten - LastByteAcked ≤ MaxSendBuffer
    - block sender if (LastBytesWritten - LastByteAcked) + y > MaxSendBuffer
  - EffectiveWindow = AdvertisedWindow - (LastBytesSent - LastByteAcked)
- Always send ACK in response to arriving data segment
- Persist sending one byte seg. when AdvertisedWindow = 0

Keeping the Pipe Full

- 16-bit AdvertisedWindow controls amount of pipelining
- Assume RTT of 100ms
- Add scaling factor extension to header to enable larger windows

<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>OC-3 (155 Mbps)</td>
<td>1.8MB</td>
</tr>
<tr>
<td>OC-12 (622 Mbps)</td>
<td>7.4MB</td>
</tr>
<tr>
<td>OC-24 (1.2 Gbps)</td>
<td>14.8MB</td>
</tr>
</tbody>
</table>

Making TCP More Efficient

- Delayed acknowledgements
  - Try to piggyback ACKs with data
  - Try not to send small packets, sender sends only when it has enough data to fill MSS
    - See Nagle’s algorithm
- Acknowledge every other packet
  - Many instances in transmission sequence which require an ACK
Karn/Partridge Algorithm for RTO

- Degenerate cases with for RTT measurements
  - Solution: Do not sample RTT when retransmitting
- After each retransmission, set next RTO to be double the value of the last
  - Exponential backoff is well known control theory method
  - Loss is most likely caused by congestion so be careful

Jacobson/ Karels Algorithm

- In late ’80s, Internet was suffering from congestion collapse
- New Calculations for average RTT – Jacobson ’88
- Variance is not considered when setting timeout value
  - If variance is small, we could set RTO = EstRTT
  - If variance is large, we may need to set RTO > 2 x EstRTT
- New algorithm calculates both variance and mean for RTT
  \[ \text{Diff} = \text{sampleRTT} - \text{EstRTT} \]
  \[ \text{EstRTT} = \text{EstRTT} + \delta \times \text{Diff} \]
  \[ \text{Dev} = \text{Dev} + \delta \times (|\text{Diff}| - \text{Dev}) \]
  - Initially settings for \text{EstRTT} and \text{Dev} given
  - \( \delta \) is a factor between 0 and 1 (typical value is 0.125)
  \[ \text{TimeOut} = \mu \times \text{EstRTT} + \phi \times \text{Dev} \]
  - where \( \mu = 1 \) and \( \phi = 4 \n- When variance is small, TimeOut is close to EstRTT
- When variance is large Dev dominates the calculation
- Another benefit of this mechanism is that it is very efficient to implement in code (does not require floating point)
- Notes
  - algorithm only as good as granularity of clock (500ms on Unix)
  - accurate timeout mechanism important to congestion control (later)
- These issues have been studied and dealt with in new RFC’s for RTO calculation.
- TCP RENO uses Jacobson/Karels

Jacobson/ Karels contd.