CS 352
Reliability: Sliding Windows

CS 352, Lecture 10.1
http://www.cs.rutgers.edu/~sn624/352

Srinivas Narayana
Transport

Application

Transport

Network

Host-to-Net

HTTP

HTTPS

FTP

SMTP

DNS

TCP

UDP

IP

802.11

X.25

ATM

...
Modularity through layering

Apps: useful user-level functions

Transport: provide guarantees to apps

Network: best-effort global pkt delivery

Link: best-effort local pkt delivery
How do apps get perf guarantees?

- The network core provides no guarantees on packet delivery
- Transport software on the endpoint oversees implementing guarantees on top of a best-effort network
- Three important kinds of guarantees
  - Reliability
  - Ordered delivery
  - Resource sharing in the network core

Transmission Control Protocol (TCP)
Review: Stop-and-Wait Reliability

- Stop and wait: sender waits for an ACK/RTO before sending another packet

- Suppose no packets are dropped
  - $\text{RTT} = \text{RTO} = 100$ milliseconds
  - Packet size: $12$ Kbit ($1\ K = 10^3$)
  - Link rate: $12$ Mbit/s ($1\ M = 10^6$)

- Rate of data transmission?
  - one packet per $\text{RTT} = 12$Kbits / $100$ms 
    \[ = 120\ \text{Kbit/s} \]

\[120\ \text{Kilobit/s} \approx 1\%\ of\ link\ rate\]
Making reliable transmissions efficient

• Terminology: unACKed data / packets in flight
  • Data that has been sent, but not known (by the sender) to be received

• With just one packet in flight, the data rate is limited by the packet delay (RTT) rather than available bandwidth (link rate)
  • Larger the delay, slower the data rate, regardless of link rate

• Idea: Keep many packets in flight!
  • More packets in flight improves throughput

• We say such protocols implement pipelined reliability
Why does pipelined reliability help?

Suppose sender has multiple, in-flight (yet-to-be-acknowledged) packets. New packets transmitted \textit{concurrently} with in-flight packets. Packets and ACKs (of prior packets) are concurrently transmitted.

\textbf{➜ More data and ACKs transmitted within the same duration}
Tracking packets in flight

• If there are $N$ packets in flight, throughput improves by $N$ times relative to stop-and-wait.
  • Stop and wait: send 1 packet per RTT
  • Pipelined: send $N$ packets per RTT

• We term the in-flight data the **window**

• We term the amount of in-flight data the **window size**
Sliding Windows
Window

- Window: Sequence numbers of in-flight data
- Window size: The amount of in-flight data (unACKed)
Sliding window (sender side)

- Suppose sequence number 2 is acknowledged by the receiver
  - Sender can transmit sequence # 5
  - The window “slides” forward

Window size = 3

Sender’s point of view:
- Last seq # known to be received at receiver (ACK’ed)
- Last sequence # sent

[Diagram showing sliding window with sequence numbers and window size]
Sliding window (sender side)

• Suppose sequence number 2 is acknowledged by the receiver
  • Sender can transmit sequence # 5
  • The window “slides” forward

Window size = 3

Sender’s point of view:

Last seq # known to be received at receiver (ACK’ed)

Last sequence # sent
Sliding window (receiver side)

• Window of in-flight packets can look different between sender and the receiver: receiver has more recent info of in-flight packets.

• Receiver only accepts sequence #s as allowed by the current receiver window.

Receiver’s point of view:

0 1 2 3 4 5 6 7 0 1

Last seq # received and ACK’ed by receiver

Window size = 3

Highest sequence # accepted

Receiver will not accept this seq #.
Packet dropped

Sender:

0 1 2 3 4 5 6 7 0 1

Window = 3
Summary of sliding windows

• Sender and receiver can keep several packets of in-flight data
  • Book-keep the sequence numbers using the window

• Windows slide forward as packets are ACKed (at receiver) and ACKs are received (at sender)

• Common case: Improve throughput by sending and ACKing more packets in the same duration

• Key challenge: how should the sender and receiver collaboratively track the packets that must be retransmitted?
Making Retransmissions Efficient

CS 352, Lecture 10.2
http://www.cs.rutgers.edu/~sn624/352

Srinivas Narayana
Pipelined Reliability

- If there are $N$ packets in flight, throughput improves by $N$ times relative to stop-and-wait.
  - Stop and wait: send 1 packet per RTT
  - Pipelined: send $N$ packets per RTT

- Q1: how should sender efficiently identify which pkts were dropped and (hence) retransmitted?

- Q2: how much data to keep in flight (i.e., what is $N$?) to reduce drops/retransmits?
Q1. Identifying the Dropped Packets
Q1: Identifying dropped packets

• Suppose 4 packets were sent, but one was dropped. How does sender know which one(s) were dropped?

• Recall: Receiver writes sequence numbers on the ACK
  • Sender infers which bytes were received successfully using the ACK #s

• Q1.1: Should receivers ACK subsequent packets upon detecting data loss?
• Q1.2: If so, what sequence number should receiver put on the ACK?
Receiver strategies upon packet loss

ACK subsequent pkts?

No

Go-back-N

Yes

Selective Repeat

What seq # on ACK?

Last pkt in order

Cumulative ACK

Seq # ranges received so far

Selective ACK

TCP's default

Sender

1

2

3

4

5

Receiver
Sliding Window with Go Back N

• When the receiver notices a missing or erroneous frame:

• It simply discards all frames with greater sequence numbers
  • The receiver will send no ACK

• The sender will eventually time out and retransmit all the frames in its sending window
Go back N

Sender
Maximum window size = 8

Receiver
Maximum window size = 8

Frame with error

Discarded by receiver

RTO

Time
Go back N

• Go Back N can recover from erroneous or missing frames.

• But it is wasteful.

• If there are errors, the sender will spend time and network bandwidth retransmitting data the receiver has already seen.
Selective repeat with cumulative ACK

Idea: sender should only retransmit dropped/corrupted segments.

• The receiver stores all the correct frames that arrive following the bad one. (Note that the receiver requires a memory buffer for each sequence number in its receiver window.)

• When the receiver notices a skipped sequence number, it keeps acknowledging the first in-order sequence number it wants to receive. This is what we mean by cumulative ACK.

• When the sender times out waiting for an acknowledgement, it just retransmits the first unacknowledged packet, not all its successors.

• Note that the RTO applies independently to each sequence #
Selective repeat with cumulative ACK

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

Frame with error

Buffered by receiver

Maximum window size = 8

RTO

Time
Selective repeat with selective ACK

Sender
Maximum window size = 8

Receiver
Maximum window size = 8

Frame with error

Buffered by receiver

Time

RTO
TCP: Cumulative & Selective ACKs

• Sender retransmits the seq #s it thinks aren’t received successfully yet

• Pros & cons: selective vs. cumulative ACKs
  • Precision of info available to sender
  • Redundancy of retransmissions
  • Packet header space
  • Complexity (and bugs) in transport software

• On modern OSes, TCP uses selective ACKs by default
Memory Buffers in the Transport Layer
Receive-side sockets need memory buffers

• Since TCP uses selective repeat, the receiver must buffer data that is received out of order:
  • e.g., hold packets so that only the “holes” (due to drops) need to be filled in later, without having to retransmit packets that were received successfully

• Apps read from the receive-side socket buffer when you do a recv() call.

• Even if data reliably received in order, applications may not always read the data immediately
  • What if you invoked recv() in your socket program infrequently (or never)?
  • For the same reason, UDP sockets also have buffers
Sender-side sockets need memory buffers

• The possibility of packet retransmission in the future means that data can’t be immediately discarded from the sender once transmitted.

• Transport layer must wait for ACK of a piece of data before reclaiming the memory for that data.
Q2. How much data to keep in flight?
Q2: How much data to keep in flight?

- Challenging question! We want to increase throughput. But:
  - The receiving app must keep up: otherwise, receiver socket buffer will fill up. Once full, subsequent packets are dropped.
  - Even if receiving app is fast, there must be sufficient buffering for selective repeat, if some data is dropped/corrupted.
  - The network path must be able to keep up.
  - We don’t want window to be so large that pkts dropped anyway.
- Challenge: The sender must figure out where the bottleneck is: receiving app? Socket buffer? A link along the network path?
- Flow control and congestion control
Inspecting TCP stack parameters

• A small demo
Info on (tuning) TCP stack parameters

