TCP Congestion Avoidance

Joshua Gancher

November 10, 2016

A little history

- Late 1950's: SAGE radar station

A little history

- Late 1950's: SAGE radar station
- 1961: Leonard Kleinrock – queueing theory $\implies$ packet switching

A little history

- Late 1950's: SAGE radar station
- 1961: Leonard Kleinrock – queueing theory $\Longrightarrow$ packet switching
- 1964: Dartmouth Time Sharing System

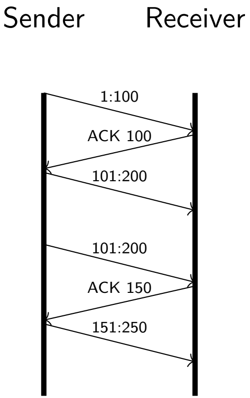
A little history

- Late 1950's: SAGE radar station
- 1961: Leonard Kleinrock – queueing theory $\Longrightarrow$ packet switching
- 1964: Dartmouth Time Sharing System
- 1969: Beginning of ARPANET – UCLA, SRI, UCSB, Utah
  - Initially over NCP

"We typed the L and we asked on the phone, "Do you see the L?"
"Yes, we see the L," came the response.
"We typed the O, and we asked, "Do you see the O."
"Yes, we see the O."
"Then we typed the G, and the system crashed...
Yet a revolution had begun..."

Kleinrock, at UCLA

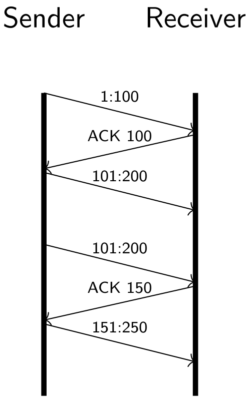
TCP in One Slide



$(wsize = 100; \text{rate} = wsize/RTT)$

- 1974: RFC 675
  - (coined the term Internet)

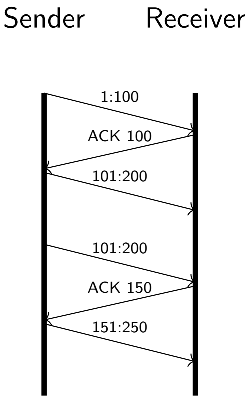
TCP in One Slide



$(wsize = 100; \text{rate} = wsize/RTT)$

- 1974: RFC 675
  - (coined the term Internet)
- All data carries a sequence number

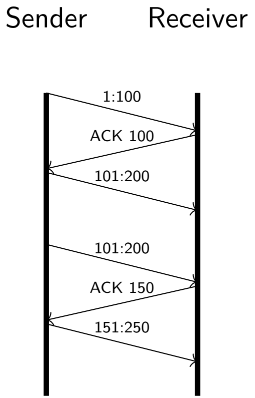
TCP in One Slide



$$(wsize = 100; \text{rate} = wsize/RTT)$$

- 1974: RFC 675
  - (coined the term Internet)
- All data carries a sequence number
- Receiver sends back cumulative acknowledgement (ACKs)

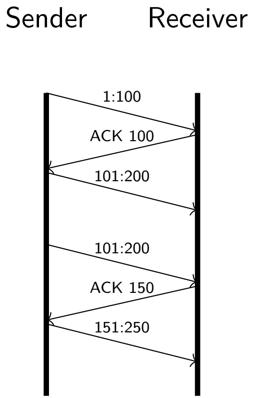
TCP in One Slide



$(wsize = 100; \text{rate} = wsize/RTT)$

- 1974: RFC 675
  - (coined the term Internet)
- All data carries a sequence number
- Receiver sends back cumulative acknowledgement (ACKs)
  - If no ACK, retransmit from last ACK

TCP in One Slide



$(wsize = 100; \text{rate} = wsize/RTT)$

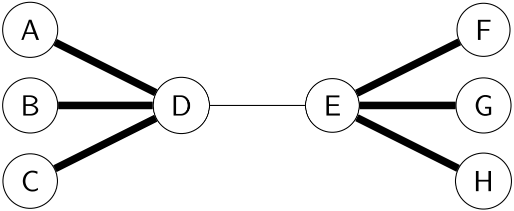
- 1974: RFC 675
  - (coined the term Internet)
- All data carries a sequence number
- Receiver sends back cumulative acknowledgement (ACKs)
  - If no ACK, retransmit from last ACK
- Receiver advertises window size in header

TCP Sending Behavior

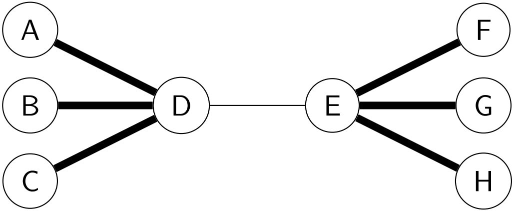
Repeat:

1. Send packet
2. Wait for ack
3. If no ack within timeout, retransmit until acknowledged

Bottleneck Buffers

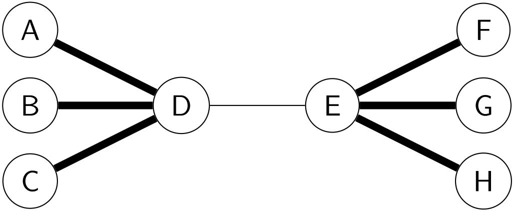


Bottleneck Buffers



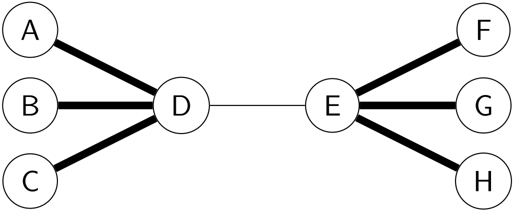
- F,G,H advertise large window

Bottleneck Buffers



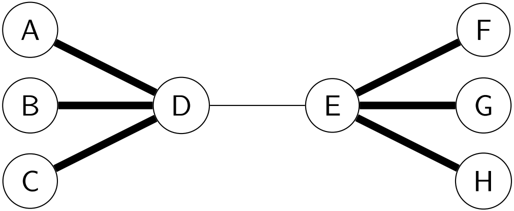
- F,G,H advertise large window
- A,B,C send large window

Bottleneck Buffers



- F,G,H advertise large window
- A,B,C send large window
- E's buffer reaches capacity

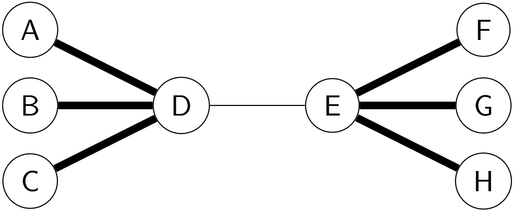
Bottleneck Buffers



- F,G,H advertise large window
- A,B,C send large window
- E's buffer reaches capacity

$\Longrightarrow$ A,B,C all must retransmit lost packets, after timeout

Bottleneck Buffers



- F,G,H advertise large window
- A,B,C send large window
- E's buffer reaches capacity

$\Longrightarrow$ A,B,C all must retransmit lost packets, after timeout

What if timeout range is smaller than transmit time?

Congestion Collapse

1986: NSFNET dropped from 32 Kb/s to 40 b/s

> *[Hosts] will begin to introduce more and more copies of the same datagrams into the net. The network is now in serious trouble... Hosts are sending each packet several times, and eventually some copy of each packet arrives at its destination. This is congestion collapse. – RFC 896*

Optimistic Case / Worst Case

- Low demand on network
- No major bottleneck
- Little packet loss

Optimistic Case / Worst Case

- Low demand on network
- No major bottleneck
- Little packet loss

⇓

- Low load stable state
- Low round trip time

Optimistic Case / Worst Case

- Low demand on network
- No major bottleneck
- Little packet loss

- High demand on network
- Bottleneck
- High packet loss

$\Downarrow$

- Low load stable state
- Low round trip time

Optimistic Case / Worst Case

Optimistic Case	Worst Case
▶ Low demand on network ▶ No major bottleneck ▶ Little packet loss	▶ High demand on network ▶ Bottleneck ▶ High packet loss
⇓	⇓
▶ Low load stable state ▶ Low round trip time	▶ High load stable state ▶ High round trip time

Congestion Avoidance and Control

Van Jacobson*

University of California
Lawrence Berkeley Laboratory
Berkeley, CA 94720
van@helios.ee.lbl.gov

- From Berkeley; now at UCLA
- Major contributions to TCP/IP
- Member of the Internet Hall of Fame

Conservation of Packets

Conservation

Under stable conditions, new packets enter the stream only when old packets leave.

Conservation of Packets

Conservation

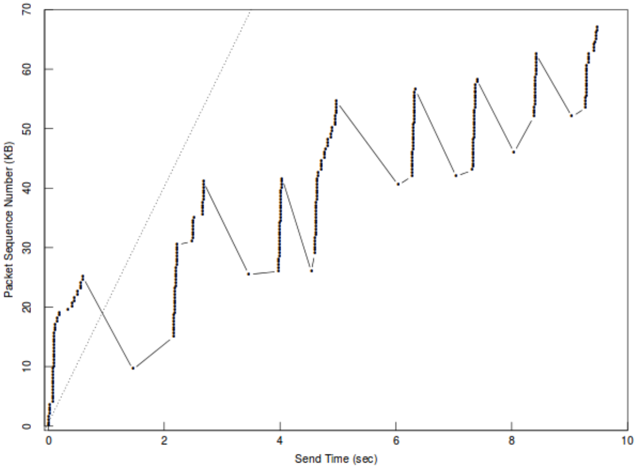
Under stable conditions, new packets enter the stream only when old packets leave.

Can be violated by:

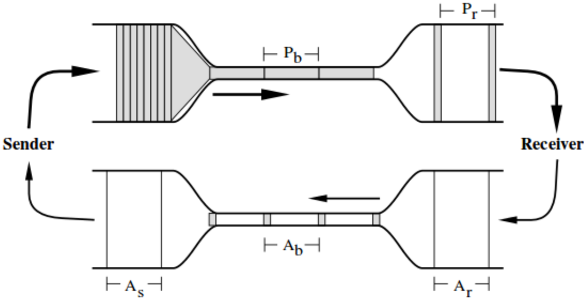
- The connection doesn't stabilize
- A new packet enters before an old packet is received
- In-transit packet loss

Stability

Problem 1: Stability



Self-Clocking TCP



Slow Start

Congestion Windows

```
Initialize: cwnd := 1
On ack: cwnd++
On packet loss: set cwnd := 1
On send: send min(cwnd, receiver's window size)
```

Slow Start

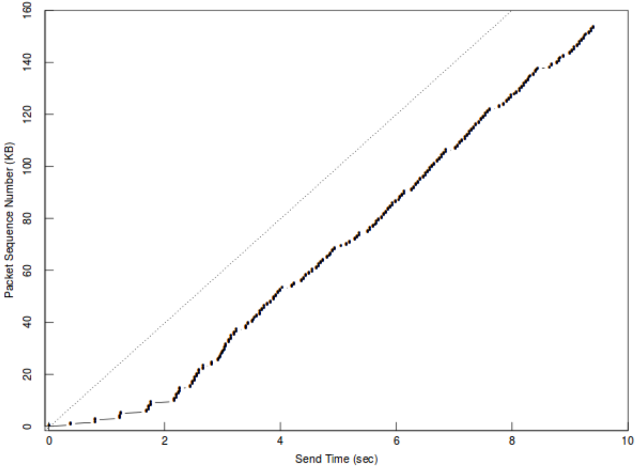
Congestion Windows

```
Initialize: cwnd := 1
On ack: cwnd++
On packet loss: set cwnd := 1
On send: send min(cwnd, receiver's window size)
```

⇓

- Exponential acceleration to receiver's window size ($R \log W$ time to reach window size of W)
- Reset back to 1 on failure (will be amended)

Execution with Slow Start



Conservation of Packets

Conservation

Under stable conditions, new packets enter the stream only when old packets leave.

Can be violated by:

- ~~The connection doesn't stabilize~~
- A new packet enters before an old packet is received
- In-transit packet loss

Problem 2: Packet duplication

Problem 2: Packet duplication

- Need a good estimator of round-trip time (RTT)

Problem 2: Packet duplication

- Need a good estimator of round-trip time (RTT)

Jacobson's insight: account for the variation of RTT

- Each ACK: $RTT := \alpha \cdot RTT + (1 - \alpha) \cdot M$
  - Where $\alpha \approx 0.9$

Problem 2: Packet duplication

- Need a good estimator of round-trip time (RTT)

Jacobson's insight: account for the variation of RTT

- Each ACK: $RTT := \alpha \cdot RTT + (1 - \alpha) \cdot M$
  - Where $\alpha \approx 0.9$
- Set timeout to $\beta \cdot RTT$
  - Where $\beta \approx 2$

Round-Trip Time Estimation

Figure 5: Performance of an RFC793 retransmit timer

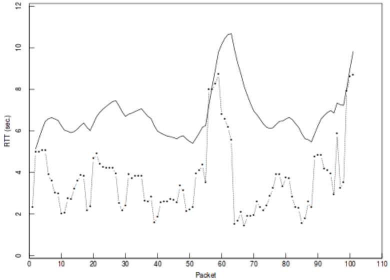
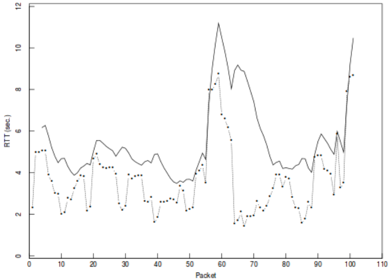


Figure 6: Performance of a Mean+Variance retransmit timer



Conservation of Packets

Conservation

Under stable conditions, new packets enter the stream only when old packets leave.

Can be violated by:

- ~~The connection doesn't stabilize~~
- ~~A new packet enters before an old packet is received~~
- In-transit packet loss

Congestion Avoidance

Problem 3: packet loss in-transit

Congestion Avoidance

Problem 3: packet loss in-transit

- 99% of packet loss due to buffer overflow

Congestion Avoidance

Problem 3: packet loss in-transit

- 99% of packet loss due to buffer overflow
- Need local state to keep track of network allowance

Congestion Avoidance

Problem 3: packet loss in-transit

- 99% of packet loss due to buffer overflow
- Need local state to keep track of network allowance
  - Congestion window: slow start
  - Congestion threshold: exploratory window opening

Congestion Avoidance

Problem 3: packet loss in-transit

- 99% of packet loss due to buffer overflow
- Need local state to keep track of network allowance
  - Congestion window: slow start
  - Congestion threshold: exploratory window opening

Queueing theory:

- Low load: average buffer length $\approx$ constant
- Congestive load: average buffer length $\approx$ exponential growth

Congestion Avoidance

Problem 3: packet loss in-transit

- 99% of packet loss due to buffer overflow
- Need local state to keep track of network allowance
  - Congestion window: slow start
  - Congestion threshold: exploratory window opening

Queueing theory:

- Low load: average buffer length ≈ constant
- Congestive load: average buffer length ≈ exponential growth

Jacobson's insight:

- Use timeouts to determine congestion
- No congestion: log growth
- Congestion: exponential decay

New Congestion Window Algorithm

```
On timeout: ssthresh := cur window size / 2
            cwnd := 1
On ack: if cwnd < ssthresh, cwnd++ // slow start
        else, cwnd += 1/cwnd // exploratory growth
```

Congestion Avoidance

Figure 8: Multiple, simultaneous TCPs with no congestion avoidance

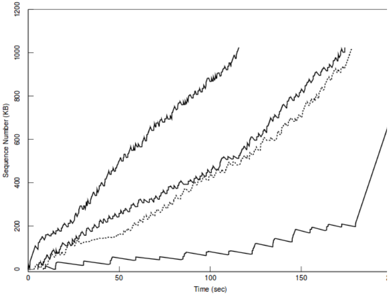
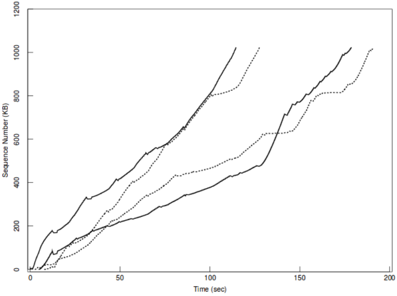


Figure 9: Multiple, simultaneous TCPs with congestion avoidance



Biggest lesson learned:

Biggest lesson learned:

- Analytic methods $\Longrightarrow$ tiny codebase which does a whole lot

TCP Congestion Control with a Misbehaving Receiver

Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson
Department of Computer Science and Engineering
University of Washington, Seattle

TCP Congestion Control with a Misbehaving Receiver

- Van Jacobson paper assumes coordination

TCP Congestion Control with a Misbehaving Receiver

- Van Jacobson paper assumes coordination
- Attacks: Malicious receivers can encourage unfriendliness

TCP Congestion Control with a Misbehaving Receiver

- Van Jacobson paper assumes coordination
- Attacks: Malicious receivers can encourage unfriendliness
- Modifications to disable such attacks

Attack 1: ACK Division

ACK Granularity

During slow start, acks assumed to be in units of segments

Attack 1: ACK Division

ACK Granularity

During slow start, acks assumed to be in units of segments

Attack:

- Send many acks for each segment received

Attack 1: ACK Division

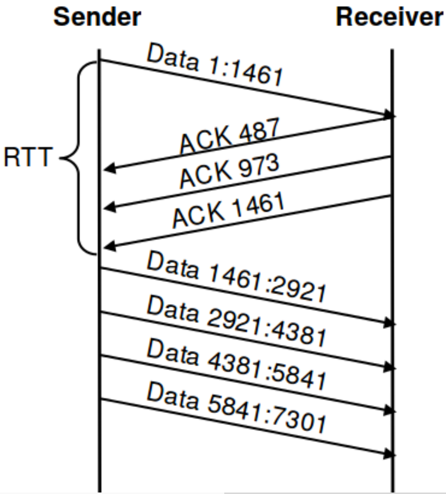
ACK Granularity

During slow start, acks assumed to be in units of segments

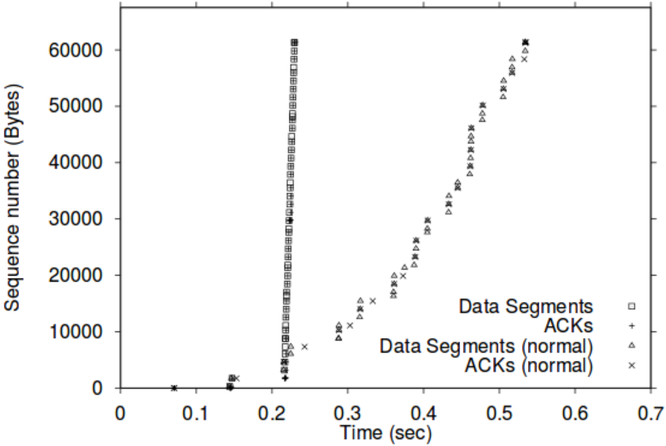
Attack:

- Send many acks for each segment received
- Causes congestion window to increase many times

Attack 1: ACK Division



Attack 1: ACK Division



Attack 1: ACK Division

Solution:

- Require unambiguous ACK granularity

Attack 1: ACK Division

Solution:

- Require unambiguous ACK granularity
- Either byte-level or segment-level

Attack 2: Duplicate ACKs

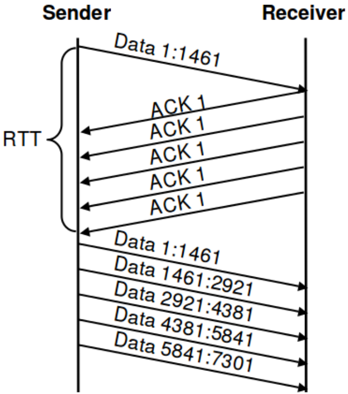
Duplicate ACKs

Duplicate ACKs interpreted as duplicate packets leaving the network; each ACK increases cwnd

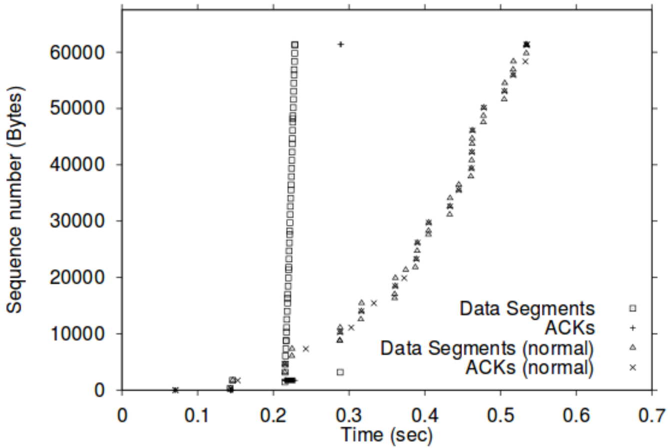
Attack:

- Flood connection with duplicate ACKs

Attack 2: Duplicate ACKs



Attack 2: Duplicate ACKs



Attack 2: Duplicate ACKs

Solution:

- Attach nonces to retransmitted data

Attack 3: Optimistic ACKing

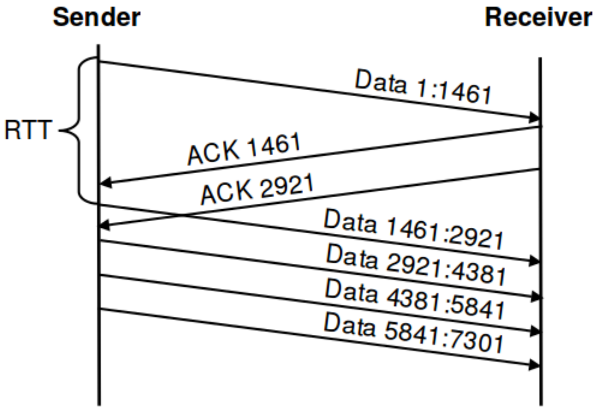
Optimistic ACKs

ACKs can be sent before data is received, obtaining artificially low RTT

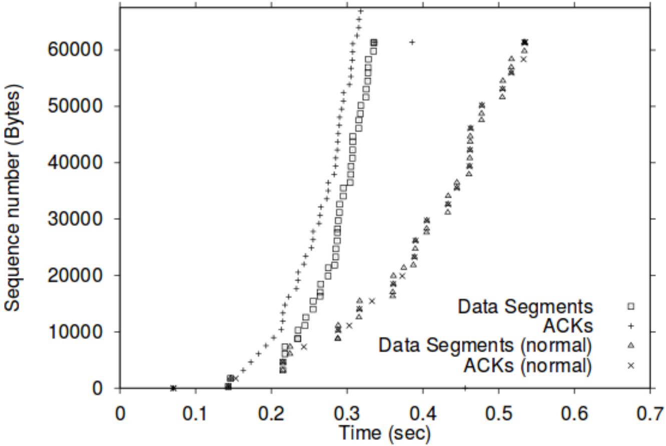
Attack:

- Send ACKs before data is received
- Time so that ACK received just after data is sent

Attack 3: Optimistic ACKing



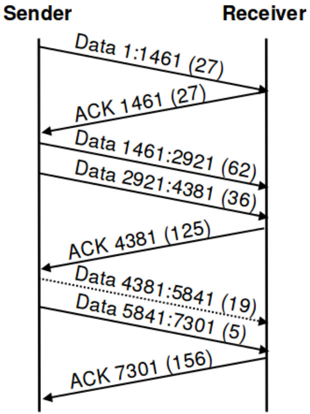
Attack 3: Optimistic ACKing



Attack 3: Optimistic ACKing

Solution:

- Use cumulative nonces to enforce causality



	ACK Division	DupACK Spoofing	Optimistic Acks
Solaris 2.6	Y	Y	Y
Linux 2.0	Y	Y (**N**)	Y
Linux 2.2	**N**	Y	Y
Windows NT4/95	Y	**N**	Y
FreeBSD 3.0	Y	Y	Y
DIGITAL Unix 4.0	Y	Y	Y
IRIX 6.x	Y	Y	Y
HP-UX 10.20	Y	Y	Y
AIX 4.2	Y	Y	Y

Lesson learned:

- Must assume malicious behavior in wide area networks!!
- More important now than ever