Data over TCP: Flow Control

Nick DeMarinis

Based partly on lecture notes by Rodrigo Fonseca, David Mazières, Phil Levis, John Jannotti
Administrivia

- Make sure you have signed up for IP grading
- TCP milestone I: Thurs, Apr 14
- TCP gear-up early next week, look for details
Topics for today

- Flow control: Sliding window
- Computing RTO
- Connection termination
Sliding window: in abstract terms

- Window of size $w$
- Can send at most $w$ packets before waiting for an ACK
Sliding window: in abstract terms

- Window of size \( w \)
- Can send at most \( w \) packets before waiting for an ACK
- Goal
  - Network "pipe" always filled with data
  - ACKs come back at rate data is delivered => "self-clocking"

\[ \text{DEPENDS ON} \]
- NETWORK CONDITIONS
- RECEIVED/APP USAGE

\[ \text{ACK R} \]
On ACK for A

IF LDA ≠ LBS = OUT OF WINDOW, DROP, SINCE OLD INVALID

OTHERWISE

LBA += (SIZE OF SEGMENT)

IF ACK FULLY COVERS A UNACKED SEGMENT, CAN DEQUEUE.
LAST BYTE ACKED

\[ \begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array} \]

NEXT TO SEND

- Receiver advertises how much data it can accept (advertized window, AW)
- Send segments to fill up to AW
  - For each one, record timestamp for retransmit

\[ \text{IN FLIGHT} = 1234 \quad \text{LBA} = 0 \quad \text{NBS} = 5 \]

Each segment should be ideally
\[ \text{MSS BYTES} = (\text{1000 BYTES}) \]

If you get ACK for SEG A

- Receiver has data up to SEG A ("cumulative ACKs")
- LBA = A
- CAN NOW SEND MORE DATA
10 BYTE SEGMENTS
\[ LBA = 10, w = 4 \]
\[ \{11, 12, 13, 14\} \]

If you get ACK 12
\[ w = \{12, 13, 14, 15\} \]

If 10 BYTE SEGMENTS
\[ LBA = 10, w = 40 \, \text{BYTES} \]

If ACK for 21
- R has DATA up to 20

\[ \text{SEGMENTS IN FLIGHT: } \{16, 20, 30, 40\} \]

So

If ACK for 15:
- Might need to retransmit
- PART OF SEGMENT

For each segment in flight:
- TIMESTAMP of last sent time
- Retransmit if this expires
Available window
(advertized window)

Bytes ready for app
App calling recv()

Last byte read by app

Early arrivals
Out of order packets

Available/advertized window

- What you send back to sender to indicate how much it can send
- Can be 0
S

FOR

LAST

BYTE

YOU

GET IN ORDER

NEXT BYTE

EXPECTED

RETRANSMIT

0

1

2

3

4

5

6

NEXT BYTE

EXPECTED
Next byte expected =  
Next seq num you expect 

to get

Valid
range of sequence numbers

(next, next seq + avail window)

If you get seq w/ seq 5

If S ≥ NBE and S < NBE + avail window

- Add to buffer at
  position S
- NBE + = segment size

- Check early arrival
queue, move up to next contiguous block.
Sender example
Receiver example
Flow Control: Sender

Invariants
- $\text{LastByteSent} - \text{LastByteAcked} \leq \text{AdvertisedWindow}$
- $\text{EffectiveWindow} = \text{AdvertisedWindow} - (\text{BytesInFlight})$
- $\text{LastByteWritten} - \text{LastByteAcked} \leq \text{MaxSendBuffer}$
Flow Control: Sender

Invariants
- LastByteSent – LastByteAcked ≤ AdvertisedWindow
- EffectiveWindow = AdvertisedWindow – (BytesInFlight)
- LastByteWritten – LastByteAcked ≤ MaxSendBuffer

Useful Sliding Window
Terminology:
RFC 793, Sec 3.3
Flow control: receiver

- Can accept data if space in window
- Available window = BufferSize – (NextByteExpected - 1) - LastByteRead

- On receiving segment for byte S
  - if s is outside window, ignore packet
  - if s == NextByteExpected:
    - Deliver to application (Update LastByteReceived)
    - If next segment was early arrival, deliver it too
  - If s > NextByteExpected, but within window
    - Queue as early arrival

- Send ACK for highest contiguous byte received, available window
Flow control: receiver

- Can accept data if space in window
- Available window = BufferSize - ((NextByteExpected-1) - LastByteRead)

- On receiving segment for byte S
  - if s is outside window, ignore packet
  - if s == NextByteExpected:
    - Deliver to application (Update LastByteReceived)
    - If next segment was early arrival, deliver it too
  - If s > NextByteExpected, but within window
    - Queue as early arrival

- Send ACK for highest contiguous byte received, available window

Useful Sliding Window Terminology: RFC 793, Sec 3.3
Flow Control

- Advertised window can fall to 0
  - How?
  - Sender eventually stops sending, blocks application

- Resolution: zero window probing: sender sends 1-byte segments until window comes back > 0
Sequence numbers
(Circumference = 0 to 2^32 slots)

Initial sequence number

Data received, acknowledged and delivered to application

Data received, acknowledged, but not yet delivered to application

Data received, but not acknowledged

Unfilled buffer

Window shifts

Receiver window (Allocation buffer)
Up to 2^16-1 slots

MEM FOR R BUF
Some Visualizations

- Normal conditions: https://www.youtube.com/watch?v=zY3Sxvj8kZA
- With packet loss: https://www.youtube.com/watch?v=IkJ27yiITOvU
How do ACKs work?

- ACK contains next expected sequence number
- If one segment is missed but new ones received, send duplicate ACK
- Retransmit when:
  - Receive timeout (RTO) expires
  - Receive 3 Duplicate ACKs
- How to set RTO?
When to time out?

Should expect an ACK within one Round Trip Time (RTT)

- Problem: RTT can be highly variable

- Strategy: expected RTT based on ACKs received
  - Use exponentially weighted moving average (EWMA)
  - RFC793 version ("smoothed RTT"):

RFC793, Sec 3.7
When to time out?

Should expect an ACK within one Round Trip Time (RTT)

• Problem: RTT can be highly variable

• Strategy: expected RTT based on ACKs received
  – Use exponentially weighted moving average (EWMA)
  – RFC793 version ("smoothed RTT"): 
    \[
    SRTT = (\alpha \cdot SRTT) + (1 - \alpha) \cdot RTT_{\text{Measured}}
    \]
    \[
    RTO = \min(RTO_{\text{Min}}, \max(\beta \cdot SRTT, RTO_{\text{Max}}))
    \]

    \(\alpha = \text{"Smoothing factor"}: 0.8-0.9\)
    \(\beta = \text{"Delay variance factor"}: 1.3—2.0\)

RFC793, Sec 3.7
This is only the beginning…

- Problem 1: what if segment is a retransmission?
This is only the beginning...

• Problem 1: what if segment is a retransmission?
  – Solution: don’t update RTT if segment was retransmitted
This is only the beginning…

• Problem 1: what if segment is a retransmission?
  – Solution: don’t update RTT if segment was retransmitted

• Problem 2: RTT can have high variance
  – Initial implementation doesn’t account for this
This is only the beginning...

- Problem 1: what if segment is a retransmission?
  - Solution: don’t update RTT if segment was retransmitted

- Problem 2: RTT can have high variance
  - Initial implementation doesn’t account for this
  - Congestion control: modeling network load
When to Transmit?

Nagle’s algorithm

- Goal: reduce the overhead of small packets
  - if (there is data to send) and (window >= MSS)
    - Send a MSS segment
  else
    - if there is unAckered data in flight
      - buffer the new data until ACK arrives
    else
      - send all the new data now
- Receiver should avoid advertising a window <= MSS after advertising a window of 0
Delayed Acknowledgments

- Goal: Piggy-back ACKs on data
  - Delay ACK for 200ms in case application sends data
  - If more data received, immediately ACK second segment
  - Note: never delay duplicate ACKs (if missing a segment)
Delayed Acknowledgments

• Goal: Piggy-back ACKs on data
  – Delay ACK for 200ms in case application sends data
  – If more data received, immediately ACK second segment
  – Note: never delay duplicate ACKs (if missing a segment)

• Warning: can interact badly with Nagle for some applications
  – Nagle waits for ACK until send => Temporary deadlock
  – App can disable Nagle with TCP_NODELAY
  – App should also avoid many small writes
Summary: flow control

• Flow control provides correctness: reliable, in order delivery
• Need more for performance
  – What if the network is the bottleneck?
• Sending too fast will cause queue overflows, heavy packet loss
• Need more for performance: congestion control
Connection Termination

• When you have no more data to send, send a FIN
  – Sent by close() or shutdown()
• Both sides close connection separately!
Connection Termination

- When you have no more data to send, send a FIN
  - Sent by `close()` or `shutdown()`
- Both sides close connection separately!
- **TIME_WAIT**: initiating side should wait for 2*MSL before deleting TCB
  - MSL = Longest time a segment might be delayed (configurable, ~1min)
TCP State Diagram
AIMD Implementation

- In practice, send MSS-sized segments
  - Let window size in bytes be $w$ (a multiple of MSS)

- Increase:
  - After $w$ bytes ACKed, could set $w = w + \text{MSS}$
  - Smoother to increment on each ACK
    - $w = w + \text{MSS}/(\#\text{ acks}/w) = w + \text{MSS}/(w/\text{MSS})$
    - $= w + \text{MSS}^2/w$

- Decrease:
  - After a packet loss, $w = w/2$
  - But don’t want $w < \text{MSS}$
  - So react differently to multiple consecutive losses
  - Back off exponentially (pause with no packets in flight)