Internet Technology 06. TCP: Transmission Control Protocol Paul Krzyzanowski Rutgers University Spring 2016

Last time: Reliable Data Transfer

- · Checksum: so we can determine if the data is damaged
- ARQ (Automatic Repeat reQuest) protocols
- Use acknowledgements to request retransmission
- Acknowledgement (receiver feedback)
- Retransmit if NAK or corrupt ACK
- Allow us identify duplicate segments
- No need for NAK if we use sequence numbers for ACKs
- Timeouts
 - Detect segment loss
 - time expiration = assume that a segment was lost

Last time: Reliable Data Transfer

- · Stop-and-wait protocol
- Do not transmit a segment until receipt of the previous one has been acknowledged
- Leads to extremely poor network utilization
- · Use a pipelining protocol
- Go-back-N (GBN)
 - Window size W no more than W unacknowledged segments can be sent

 - Cumulative acknowledgement
 Receipt of a sequence number n means that all segments up to and including n have been received. Timeout: retransmit <u>all</u> unacknowledged segments
- Selective Repeat (SR)
- Acknowledge individual segments
 Sender's window: N segments starting from the earliest unacknowledged segment
- Per-segment timer on sender: retransmit only that segment on timeout Receiver's window. buffer for N segments starting from the first missing segment Receiver must buffer acknowledged out-of-order segments Deliver segments to application in order

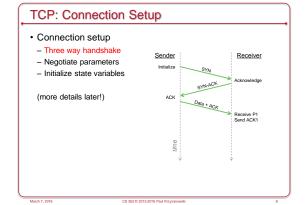
CS 352 © 2013-2016 Paul Krzyzanowski

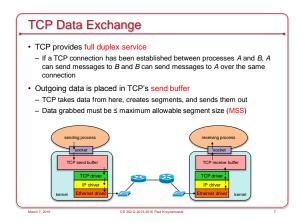
TCP: Transmission Control Protocol

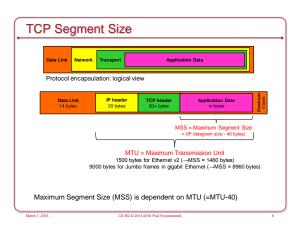
TCP

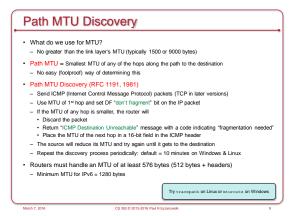
- Transport-layer protocol ... like UDP
- Connection-oriented
- Bidirectional communication channel
- Reliable data transfer
- Flow control
- · Network stacks on both end systems keep state
- "Connection" managed only in end systems
- Routers are not aware of TCP

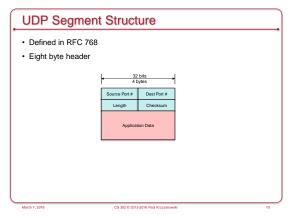
CS 352 © 2013-2016 Paul Krzyzanowski

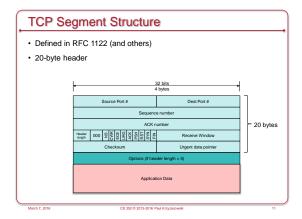


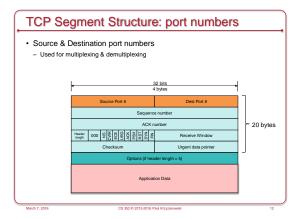


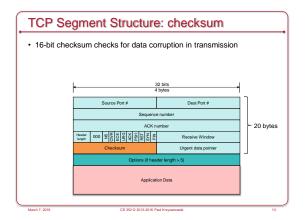


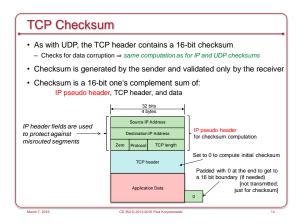


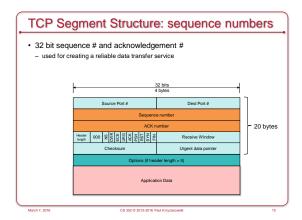


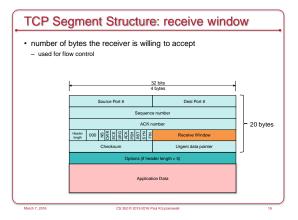


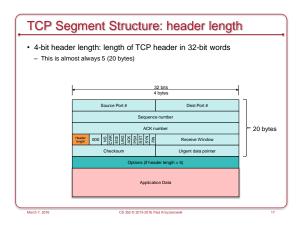


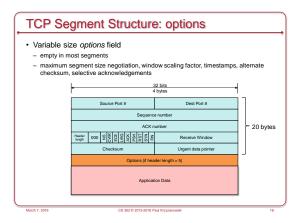


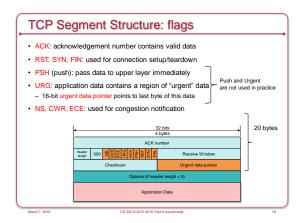


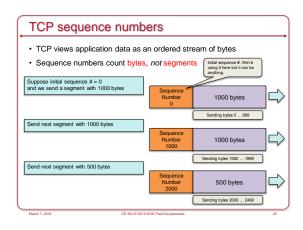


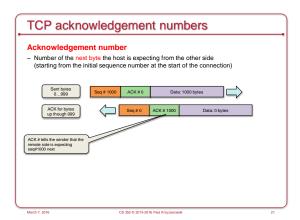


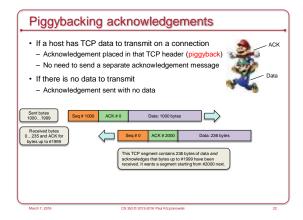


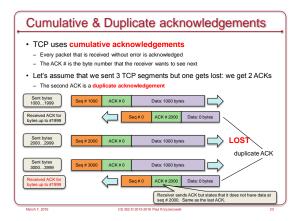












Out of order data

A segment that arrives out of order is not acknowledged
Instead, a duplicate ACK is sent asking for the missing sequence

TCP protocol does not define what happens to the received segment

We options:

Discard it

Hold on to out of order segments and wait for missing data

More complex

Du but much more efficient for the network

This is the preferred approach

Receiver action Delayed ACK. Wait up to 500 ms for the arrival of another in-order segment. Otherwise send ACK.
arrival of another in-order segment.
Send a single cumulative ACK. This acknowledges both segments.
Send duplicate ACK with sequence number of next expected byte.
Send ACK with sequence number of next unfilled byte (might be duplicate).

TCP Timeouts

Round-trip time estimation

- Round trip time:
- elapsed time from sending a segment to getting an ACK
- · RTT helps us determine a suitable timeout value
- TCP measures RTT for each non-retransmitted segment
- RTTs fluctuate
- SRTT = "Smoothed Round Trip Time" = weighted average

$$SRTT = (1 - \alpha) \cdot SRTT + \alpha \cdot RTT$$

 $\alpha = 0.125$

- Exponential weighted moving average (EWMA)
- Greater weight on recent measurements

March 7, 2016

CS 352 © 2013-2016 Paul Krzyzanowski

Round-trip time variation estimation

- Compute the average variation in round-trip time from the estimate (smoothed average)
- Another exponential weighted moving average

 $\begin{aligned} & \text{RTTVAR} = (1 - \beta) \cdot \text{RTTVAR} + \beta \cdot (\text{SRTT} - \text{RTT}) \\ & \beta = 0.25 \end{aligned} & \text{Round Trip Time} \\ & \text{Snoothed Round Trip Time}$

RTTVAR = estimate of how much RTT typically deviates from SRTT

See RFC 6298

Setting the TCP timeout interval

- Timeout ≥ SRTT
- Otherwise we'll time out too early and retransmit too often
- But don't want a value that's too high
 - Because we will introduce excessive delays for retransmission
- Use SRTT + x
- x should be large when there is a lot of variation in RTT
- x should be small when there is little variation in RTT
- This is what RTTVAR gives us!
- TCP sets retransmission timeout to:

Timeout interval = SRTT + $4 \cdot RTTVAR$

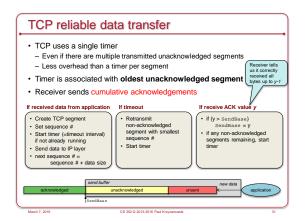
- Initial value of 1 second
- When timeout occurs, the timeout interval is doubled until the next round trip

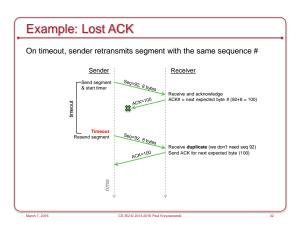
March 7, 201

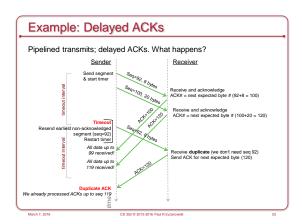
CS 352 © 2013-2016 Paul Krzyzanowski

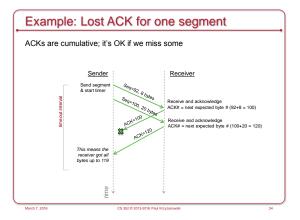
TCP Reliable Data Transfer

March 7, 2016 CS 352 © 2013-2016 Paul Krzyzanowski 3i









Timeout interval is normally set to Timeout interval = SRTT + 4 · RTTVAR But if a timeout occurs Retransmit unacknowledged segment with smallest seq # Set timer to Timeout interval = 2 · previous timeout interval If timer expires again, do the same thing: Retransmit & double the timeout This gives us exponentially longer time intervals This is a form of congestion control Any other even that requires a timer reset Set normal time interval (SRTT + 4 · RTTVAR)

CS 352 © 2013-2016 Paul Krzyzanowski

Timeouts

Property Color Past Retransmit

TCP uses pipelining
Will usually send many segments before receiving ACKs for them

If a receiver detects a missing sequence #
It means out-of-order delivery or a lost segment
TCP does not send NAKs
Instead, acknowledge every segment with the last in-order seq #
Each segment received after a missing one will generate replies with duplicate ACKs

TCP Fast Retransmit

- · Waiting for timeouts causes a delay in retransmission
- Increases end-to-end latency
- · But a sender can detect segment loss via duplicate ACKs
 - Duplicate ACK:
 - Sender receives an ACK for a segment that was already ACKed
- That means that a segment was received but not the sequentially next one
- If a sender receives three duplicate ACKs
- Sender assumes the next segment was lost (it could have been received out of order but we're guessing that's unlikely since three segments after it have been received)
- Performs a fast retransmit
- Sends missing segment before the retransmission timer expires

CS 352 © 2013-2016 Paul Krzyzanowski

GBN or SR?

- TCP looks like a Go-Back-N protocol
 - Sender only keeps track of smallest sequence # that was transmitted but not acknowledged
- · But not completely...
 - GBN will retransmit all segments in the window on timeout
 - TCP will retransmit at most one segment (lowest #)
 - TCP will retransmit no segments if it gets ACKs for highernumbered segments before a timeout
 - Most TCP receivers will hold out-of-order segments in a buffer
- · We can call it a modified Go-Back-N

March 7, 2016

CS 352 © 2013-2016 Paul Krzyzanowski

SACK: Selective Acknowledgements

- Enhancement to TCP to make it be a Selective Repeat protocol
- RFC 2018: TCP Selective Acknowledgement Options
- When receiving an out-of-order segment:
- Send duplicate ACK segment (as before)
- But append TCP option field containing range of data received
- · List of (start byte, end byte) values
- Negotiated between hosts at the start of a connection
- SACK may be used if both hosts support it

March 7, 201

CS 352 © 2013-2016 Paul Krzyzanowski

Flow Control

March 7, 201

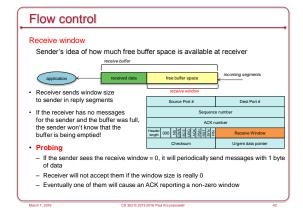
CS 352 © 2013-2016 Paul Krzyzanowsk

Flow control

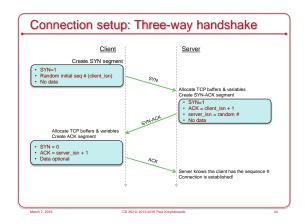
- · Incoming data goes to receive buffer
- · What if it comes in faster than the process reads it?
- · We don't want overflow!
- Flow control: match transmission rate with rate at which the app is reading data

March 7, 201

CS 352 © 2013-2016 Paul Krzyzanowski



Connection Management



SYN Flooding

- · An OS will allocate only a finite # of TCP buffers
- SYN Flooding attack
- Send lots of SYN segments but never complete the handshake
- The OS will not be able to accept connections until those time out
- SYN Cookies: Dealing with SYN flooding attacks
- Do not allocate buffers & state when a SYN segment is received
- Create initial sequence # = hash(src_addr, dest_addr, src_port, dest_port, SECRET)
- When an ACK comes back, validate the ACK # Compute the hash as before & add 1
- If valid, then allocate resources necessary for the connection & socket

MSS Announcement

- Remember the Maximum Segment Size (MSS)?
- · For direct-attached networks
 - MSS = MTU of network interface protocol headers
 - Ethernet MTU of 1500 bytes yields MSS of 1460 (1500-20-20)
- · For destinations beyond the LAN (routing needed)
- Use TCP Options field to set Maximum Segment Size
- Set MSS in SYN segment
- MSS may be obtained from PATH MTU discovery
- · Other side receives this and records it as MSS for sent messages.
- · It can respond with the MSS it wants to use for incoming messages in the SYN-ACK message
- All IP routers must support MSS ≥ 536 bytes

Special cases

- · What if the host receives a TCP segment where the port numbers or source address do not match any connection?
- Host sends back a "reset" segment (RST = 1) "I don't have a socket for this"
- For UDP messages to non-receiving ports
 - Send back an ICMP message to the sending host

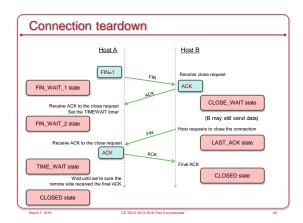
CS 352 © 2013-2016 Paul Krzyzanowski

Connection teardown

- · Either side can end a connection
- · Buffers & state variables need to be freed
- · Both sides agree to send no more messages

To close:

- 1. Send a TCP segment with the FIN bit set (FIN = Finish)
- You are saying "I will not send any more data on this connection"
- 2. Other side acknowledges this
- 3. Other side then agrees to close the connection
- Sends a TCP segment with the FIN bit set
- 4. You acknowledge receipt of this
- Then wait (TIME_WAIT state) to ensure that your ACK had time to get to the other side and that any stray segments for the connection have been received
- Wait time = 2 x maximum segment lifetime (timeout interval x 2)
 Opportunity to resend final ACK in case it is lost



TCP Congestion Control

Congestion control

· Congestion control goal

Limit rate at which a sender sends traffic based on congestion in the network

(Flow control goal was: limit traffic based on remote side's ability to process)

- · Must use end-to-end mechanisms
- The network gives us no information
- We need to infer that the network is congested
- Generally, more packet loss = more congestion

Regulating Rate: Congestion Window

- Window size = # bytes we can send without waiting for ACKs
- Receive Window (rwnd) flow control request from receiver
- # bytes that a receiver is willing to receive (reported in header)
- Congestion Window (cwnd) rate control by sender
- Window size to limit the rate at which TCP sender will transmit
- TCP will use window size = min(rwnd, cwnd)
- These are per-connection values!
- How does a window regulate transmission rate?
 - If we ignore loss and delays, we transmit cwnd bytes before waiting
 - The time we wait is the round-trip time (RTT)

Transmission rate ≈ cwnd / RTT bytes/second

March 7, 201

CS 352 © 2013-2016 Paul Krzyzanowsk

Basic mechanisms

- · Timeout or three duplicate ACKs
- Assume segment loss \rightarrow decrease cwnd = decrease sending rate
- Sender receives expected ACKs
- Assume no congestion \rightarrow increase cwnd = increase sending rate
- · ACKs pace the transmission of segments
- ACKs trigger increase in cwnd size
- If ACKs arrive slowly (slow network) → cwnd increases slowly
- TCP is self-clocking
- Bandwidth probing
- Increase rate in response to arriving ACKs
- $-\,\dots$ until loss occurs; then back off and start probing (increasing rate) again

March 7, 2016

CS 352 © 2013-2016 Paul Krzyzanowski

Basic Principle: Additive Increase (AI)

If we feel we have extra network capacity

- Increase window by 1 segment each RTT
- If we successfully send cwnd bytes, increase window by 1 MSS
- That means increase window fractionally for each ACK cwnd = cwnd + [MSS ÷ (cwnd/MSS)]
- This is Additive (linear) Increase

Tarch 7, 2016 CS 352 © 2013-2016 Paul Krzyzanowski

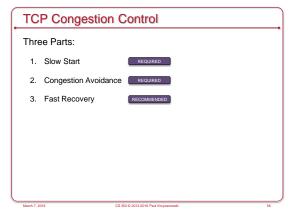
Basic Principle: Multiplicative Decrease (MD)

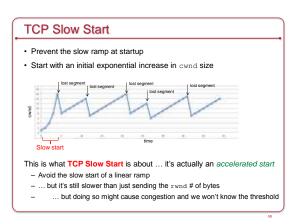
If we feel we have congestion (timeout due to lost segment)

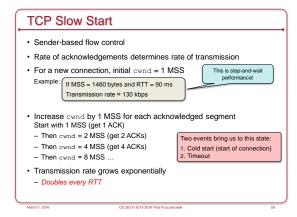
- Decrease cwnd by halving it cwnd = cwnd ÷ 2
- This is Multiplicative decrease

Additive Increase / Multiplicative Decrease (AIMD)

AIMD is a necessary condition for TCP congestion control to be stable







• "Slow Start" actually grows quickly!

• When do we stop going faster?

• On timeout (we assume this is due to congestion)

• Sender sets cwnd=1 and restarts Slow Start process

• Set slow start threshold, sethresh = cwnd/2

• When cwnd ≥ ssthresh

• switch to Congestion Avoidance mode (slow the ramp)

• This is not set at cold start; we will time out

• When three duplicate ACKs received (following a normal ACK for a segment)

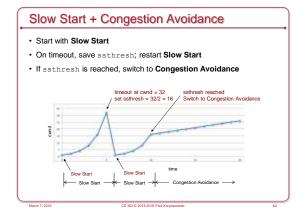
• Perform Fast Retransmit of segment

• Enter Fast Recovery State

Internet Technology 3/7/2016

Congestion Avoidance

- cwnd is 1/2 of the size when we saw congestion
- We think that's safe
- ... it worked before but doubling it gave a timeout so we're close
- · Increase rate additively: 1 MSS each RTT
- # segments in window = cwnd/MSS
- E.g., if MSS = 1460 bytes & cwnd=23360 bytes, cwnd/MSS=16
- Each ACK means we increase cwnd by MSS/(cwnd/MSS)
 - E.g., after 16 ACKs, cwnd increased by 1 MSS = increase cwnd by 1/16 MSS (~91 bytes) for each received ACK
- · Now we have a linear growth in transmission speed



Congestion Avoidance

- When do we stop increasing cwnd?
- · When we get a timeout
- Set ssthresh to 1/2 cwnd when the loss occurred
- Set cwnd set to 1 MSS and do a Slow Start
- · When we receive 3 duplicate ACKs
- We're guessing segment loss BUT the network is delivering segments

(3 · MSS) a

- Otherwise the receiver would not send ACKs
- ssthresh = cwnd/2
- cwnd = ssthresh + (3 · MSS)=
- We essentially $\frac{1}{2}$ our transmission rate
- Enter Fast Recovery state

March 7, 201

CS 352 © 2013-2016 Paul Krzyzanowski

Fast Recovery

- Fast Retransmit was used when duplicate ACKs received
 Avoid waiting for a timeout
- Duplicate ACKs means data is flowing to the receiver
 ACKs are generated only when a segment is received
- Might indicate that we don't have congestion and the loss was a rare event.
- · Don't reduce flow abruptly by going into Slow Start
- Adjust cwnd = cwnd / 2

Fast Recovery

- · Increase cwnd by 1 MSS for each duplicate ACK received
- Increase transmission rate exponentially just like slow start
- Each ACK means that the receiver received a segment ... data is flowing!
- · When ACK arrives for the missing segment (non-duplicate ACK)
- Reset cwnd to ssthresh (back to where it was)
- Enter Congestion Avoidance state
- · Resumes transmission with linear growth of the window
- · If timeout occurs
- ssthresh = cwnd /2
- cwnd = 1
- Do a Slow Start

March 7, 201

CS 352 © 2013-2016 Paul Krzyzanowski

Why the name?

- Why do we call it Fast Recovery?
- Prior to its use, TCP would set cwnd = 1 and enter Slow Start for both timeouts as well as triple duplicate ACKs
- We try to distinguish casual packet loss from packet loss due to congestion

March 7, 201

CS 352 © 2013-2016 Paul Krzyzanowski

Internet Technology 3/7/2016

