How does TCP decide to transmit a segment?

- TCP supports a byte stream abstraction
- Application programs write bytes into streams
- It is up to TCP to decide that it has enough bytes to send a segment
Triggering Transmission

TCP has three mechanisms to trigger the transmission of a segment:

1) TCP maintains a variable **MSS (maximum segment size)** and sends a segment as soon as it has collected MSS bytes from the sending process.

- MSS is usually set to the size of the largest segment TCP can send without causing local IP to fragment.
- MSS: MTU of directly connected network – (TCP header + and IP header)
Triggering Transmission

2) Sending process has explicitly asked TCP to send it
   - TCP supports push operation

3) When a timer fires
   - Resulting segment contains as many bytes as are currently buffered for transmission
Silly Window Syndrome

Of course, we can’t ignore flow control.

- Suppose the TCP sender is stopped for awhile (AdvertisedWindow = 0)
- Now suppose TCP sender receives an ACK that opens the window up to half of MSS
- Should the sender transmit?
Silly Window Syndrome

- The strategy of aggressively taking advantage of any available window leads to *silly window syndrome*
- Once smaller segment size is introduced into TCP segment system, it will stay around indefinitely
Nagle’s Algorithm

- If there is data to send but the window is open less than MSS, then we may want to wait some amount of time before sending the available data
  - If we wait too long, then we hurt interactive applications
  - If we don’t wait long enough, then we risk sending a bunch of tiny packets and falling into the silly window syndrome
- The solution is to introduce a timer and to transmit when the timer expires
Nagle’s Algorithm

- We could use a clock-based timer, for example one that fires every 100 ms
- Nagle introduced an elegant self-clocking solution

Key Idea
- As long as TCP has any data in flight, the sender will eventually receive an ACK
- This ACK can be treated like a timer firing, triggering the transmission of more data
Nagle’s Algorithm

When the application produces data to send
  if both the available data and the window ≥ MSS
    send a full segment
  else /* window < MSS */
    if there is unACKed data in flight
      buffer the new data until an ACK arrives
    else
      send all the new data now
TCP Retransmission (Timeouts)

- How should TCP set its timeout value?
  - Too short will unnecessarily retransmit segments
  - Too long will introduce unnecessary delay
Adaptive Retransmission

Q: how to set TCP timeout value?
- longer than RTT
  - but RTT varies
- too short: premature timeout
  - unnecessary retransmissions
- too long: slow reaction to segment loss

Q: how to estimate RTT?
- SampleRTT: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- SampleRTT will vary, want estimated RTT “smoother”
  - average several recent measurements, not just current SampleRTT
Adaptive Retransmission

- Original Algorithm
  - Measure SampleRTT for each segment/ACK pair
  - Compute weighted average of RTT
    - $\text{EstRTT} = \alpha \times \text{EstRTT} + (1 - \alpha) \times \text{SampleRTT}$
    - $\alpha$ between 0.8 and 0.9 (typically 0.875)
  - Set timeout based on EstRTT
    - $\text{TimeOut} = 2 \times \text{EstRTT}$
Adaptive Retransmission Problem

Problem

- ACK does not really acknowledge a transmission
  - It actually acknowledges the receipt of data
- When a segment is retransmitted and then an ACK arrives at the sender
  - It is impossible to decide if this ACK should be associated with the first or the second transmission for calculating RTTs
Adaptive Retransmission Problem

To which transmission does the ACK belong?
Karn/Partridge Algorithm

- Whenever TCP retransmits a segment, it does not take sample of RTT
- Double timeout after each retransmission
  - Exponential backoff
Karn/Partridge Algorithm

- Karn-Partridge algorithm was an improvement over the original approach, but it does not eliminate congestion.

- We need to understand how timeout is related to congestion.
  - If you timeout too soon, you may unnecessarily retransmit a segment which adds load to the network.
Karn/Partridge Algorithm

- Main problem with the original computation is that it does not take variance of Sample RTTs into consideration.
  - If the variance among Sample RTTs is small
    - Then the Estimated RTT can be better trusted
  - Otherwise
    - The Estimated RTT may be very wrong
Jacobson/Karels proposed a new scheme for TCP retransmission

- Takes variance (deviation term) into account
Jacobson/Karels Algorithm

\[ \text{Difference} = \text{SampleRTT} - \text{EstimatedRTT} \]
\[ \text{EstimatedRTT} = \text{EstimatedRTT} + (\delta \times \text{Difference}) \]
\[ \text{Deviation} = \text{Deviation} + \delta(|\text{Difference}| - \text{Deviation}) \]
\[ \text{TimeOut} = \text{EstimatedRTT} + 4 \times \text{Deviation} \]

- When the variance is small
  - TimeOut is close to EstimatedRTT
- When the variance is large
  - Deviation term dominates the TimeOut calculation
- \(\delta\) is between 0 and 1
TCP: retransmission scenarios

lost ACK scenario

premature timeout
TCP retransmission scenarios (more)

Cumulative ACK scenario

Host A

Seq=92, 8 bytes data

Seq=100, 20 bytes data

loss

X

Host B

ACK=100

ACK=120

time

timeout

Cumulative ACK scenario
TCP ACK generation

- How quickly should the receiver send an ACK?
- If ACKs are cumulative, it makes sense that the receiver would not send an ACK for every single packet received
# TCP ACK generation [RFC 1122, RFC 2581]

<table>
<thead>
<tr>
<th>Event at Receiver</th>
<th>TCP Receiver action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed</td>
<td>Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK</td>
</tr>
<tr>
<td>Arrival of in-order segment with expected seq #. One other segment has ACK pending</td>
<td>Immediately send single cumulative ACK, ACKing both in-order segments</td>
</tr>
<tr>
<td>Arrival of out-of-order segment higher-than-expect seq. #. Gap detected</td>
<td>Immediately send duplicate ACK, indicating seq. # of next expected byte</td>
</tr>
<tr>
<td>Arrival of segment that partially or completely fills gap</td>
<td>Immediately send ACK, provided that segment starts at lower end of gap</td>
</tr>
</tbody>
</table>
Fast Retransmit

- Time-out period often relatively long:
  - long delay before resending lost packet
- Detect lost segments via duplicate ACKs.
  - Sender often sends many segments back-to-back
  - If segment is lost, there will likely be many duplicate ACKs.
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - fast retransmit: resend segment before timer expires
TCP Extensions

- Timestamp inserted into the header
  - Help measure RTT
  - Detect which ACKs are for what transmissions

- Timestamp used to help prevent sequence number wraparound

- AdvertisedWindow can measure larger units than 1 byte
  - Left-shift scaling factor

- Selective (SACKs) for out of order segments