**Sliding Window in TCP**

**Goals of sliding window**
- Guarantee end-to-end reliable data delivery, and in order
- Enforce flow control between sender and receiver

![Diagram of Sliding Window in TCP](image.png)

**Example:**
- LastByteRcvd = 5, LastByteRead = 1, NextByteExpected = 6,
  MaxRcvBuffer = 10
  \[
  5 - 1 \leq 10 \\
  AdvertisedWindow = 10 - (6 - 1) - 1 = 10 - 4 = 6
  \]

**Flow Control in TCP**

**Receiver advertises window no larger than its buffer size**
- LastByteRcvd - LastByteRead ≤ MaxRcvBuffer
- AdvertisedWindow = MaxRcvBuffer - (NextByteExpected - 1) - LastByteRead

**Example:**
- LastByteRcvd = 5, LastByteRead = 1, NextByteExpected = 6,
  MaxRcvBuffer = 10
  \[
  5 - 1 \leq 10 \\
  AdvertisedWindow = 10 - (6 - 1) - 1 = 10 - 4 = 6
  \]

**Sender adheres to the AdvertisedWindow by**
- LastByteSent - LastByteAcked ≤ AdvertisedWindow
- EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)

**Example:**
- sent 7, ACK 5, EffectiveWindow = 6 - (7 - 5) = 4
Why Ignore LastByteReceived

Example
MaxReceiverBuffer = 10
Receiver buffer: 1, 2, 3, 5, 6, 7, 8,
AdvertisedWindow = 7 (though buffer space is only 3)
Since only packet 3 is ACKed, sender can only have 4 - 10 UnAcked.
Sender will send 9, 10 and wait for any ACK.
Packet 4 timeouts and gets retransmitted.
An Ack arrives for packet 8, with an updated Advertised Window

Flow Control in TCP

Buffer at the Sender
LastByteWritten – LastByteAcked ≤ MaxSendSize

Fast sender slow receiver scenario

Sender’s buffer
Receiver’s buffer

Learning about Advertised Window

Sender sends data
Receiver replies with ACK and Advertised Window

Issue with fast sender slow receiver
Sender is blocked, receiver will not have data to reply to

Sender’s buffer
Receiver’s buffer

When receiver advertises window of 0, sender periodically sends one byte of data

Smart sender/Dumb receiver rule
Sequence Number Wraparound

AdvertisedWindow: 16 bits long, up to $2^{16}$
Sequence number field: 32 bits long up to $2^{32}$
$SN >> 2$ AdvertisedWindow, so there would be no problem in ordering

How fast are sequence numbers exhausted

<table>
<thead>
<tr>
<th>Protocol</th>
<th>AdvertisedWindow</th>
<th>Sequence number field</th>
<th>Time (hrs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (1.5MBps)</td>
<td>$2^{16} \times 8$</td>
<td>$2^{32}$</td>
<td>6.4hrs</td>
</tr>
<tr>
<td>Ethernet (10 Mbps)</td>
<td>57 mins</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3 (45Mbps)</td>
<td>13 mins</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fast Ethernet (100 Mbps)</td>
<td>6 mins</td>
<td>TTL = 120 sec</td>
<td></td>
</tr>
<tr>
<td>OC-3 (155Mbps)</td>
<td>4 mins</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OC-12 (622 Mbps)</td>
<td>.55 sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OC-48 (2.5 Gbps)</td>
<td>.14 sec</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Failure due to Seq # Wraparound

Sequence number wrap-around on the current connection
A seq number may be "wrapped" (cycled) within the time that a segment is delayed in queues

Earlier incarnation of the connection
A delayed segment from the terminated connection could fall within the current window for the new incarnation and be accepted as valid

Solution
Add a timestamp filed on every packet
Reject a packet with a higher sequence number if earlier ones have a later timestamp

Keeping the Pipe Full

Assume the receiver has adequate buffer space
End-to-end Delay x Bandwidth product

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Delay x Bandwidth (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (1.5MBps)</td>
<td>1.5 x 10^6 x 0.1 / 8 = 19 KB</td>
</tr>
<tr>
<td>Ethernet (10 Mbps)</td>
<td>122KB</td>
</tr>
<tr>
<td>T3 (45Mbps)</td>
<td>549 KB</td>
</tr>
<tr>
<td>Fast Ethernet (100 Mbps)</td>
<td>1.2 MB</td>
</tr>
<tr>
<td>OC-3 (155Mbps)</td>
<td>1.8 MB</td>
</tr>
<tr>
<td>OC-12 (622 Mbps)</td>
<td>7.4 MB</td>
</tr>
<tr>
<td>OC-48 (2.5 Gbps)</td>
<td>29.6 MB</td>
</tr>
</tbody>
</table>

AdvertisedWindow field: 16 bits – $2^{16} = 64$KB
Handled by TCP extension specification. A scale factor is indicated in the options header of a SYN packet
Triggering TCP Transmissions

Mechanism for triggering transmission of a segment
Maximum Segment Size (MSS) parameter
MSS = MTU of underlying network – Headers = 1500 – 20 = 1480

Push Operation
Sending process using TCP request to empty the buffer

Timer expiration
A timer with specific task expires and all buffered packets are sent

So far we ignored flow control
AdvertisedWindow may be smaller than MSS, or closed
Aggressive strategy: Send whatever you have in an aggressive manner

Example

Silly Window Syndrome

Introduces tiny segments that stay in the system indefinitely

To avoid small segments, receiver advertises open window after MSS buffer size is available
To coalesce tiny segments, receiver may delay ACKs so buffer space empties up
Still, receiver cannot know exactly how much to delay

Nagle’s Algorithm

Use received ACKs as a timer to send more data

When the application has data to send
  if both data written in buffer and the window ≥ MSS
    send full segment
  else
    if there are unACKed data in flight
      buffer new data till ACK received
    else
      send all data now
TCP Adaptive Retransmission

**Problem**
Estimate RTT between two hosts to set up timeout for retransmissions

**Original Algorithm**
Use a running average of RTT

\[
\text{SampleRTT} = \text{RTT of last segment ACKed} \\
\text{EstimatedRTT} = \alpha \times \text{EstimatedRTT} + (1 - \alpha) \times \text{SampleRTT} \\
\text{Timeout} = 2 \times \text{EstimatedRTT}
\]

\(\alpha\) usually in the range 0.8 – 0.9

Karn/Patridge Algorithm

**Problem with SampleRTT measurement**

**Solution**
Ignore all SampleRTT for segments sent more than once

**Additional improvement**
Double the timeout every time you do a retransmission

Exponential backoff mechanism for TCP

Jacobson/Karels Algorithm

**Previous approaches did not take into account the variance of RTT**

If variance was well known, no need to be too conservative
TCP Extensions

Use the Options field and the Hdrlen field to define extensions

Extension #1: Timestamps
- Put a 32-bit timestamp in a segment
- Receiver simply reflects this timestamp in the acknowledgment
- Compute round-trip time based on the time ACK was received and the timestamp in ACK
- Eliminates discrepancies in RTT estimates

Extension #2: Expand the 32-bit sequence number field
- Use the 32-bit timestamp to extend the sequence number space
- Timestamps always increase

| 32-bit timestamp | 32-bit sequence # |

TCP Extensions

Extension #3: Extend the window size
- Increase the unit from one byte to several bytes (e.g. 16 bytes)
- Max Window size increases from $2^{16}$ to $2^{20}$
- Window size has to be a multiple of unit (e.g. 16)

Extension #4: Use of Cumulative and Selective ACKs (SACKs)
- Acknowledge normally for segments received in sequence in the standard header
- Indicate the out-of-sequence packets received in optional headers
- If SACK not used, transmitter can use Go-back-N, which may unnecessarily retransmit some segments