TCP connection establishment (3-way handshake):

- $X, Y$ are chosen randomly
  - → sequence number prediction
- piggybacking
2-person consensus problem: are $A$ and $B$ in agreement about the state of affairs after 3-way handshake?

$\rightarrow$ in general: impossible

$\rightarrow$ can be proven

$\rightarrow$ “acknowledging the ACK problem”

$\rightarrow$ also TCP session ending

$\rightarrow$ lunch date problem
Call Collision:

\[ \text{A} \quad \text{B} \]

- SYN = 1, Seq. No. = X
- SYN = 1, Seq. No. = Y
- SYN = 1, Seq. No. = Y
- Ack. No. = X + 1
- SYN = 1, Seq. No. = X
- Ack. No. = Y + 1

\[ \rightarrow \text{ only single TCB gets allocated} \]
\[ \rightarrow \text{ unique full association} \]
TCP connection termination:

- full duplex
- half duplex
More generally, finite state machine representation of TCP’s control mechanism:

→ state transition diagram
Features to notice:

- **Connection set-up:**
  - client’s transition to **ESTABLISHED** state without ACK
  - how is server to reach **ESTABLISHED** if client ACK is lost?
  - **ESTABLISHED** is macrostate (partial diagram)

- **Connection tear-down:**
  - three normal cases
  - special issue with **TIME WAIT** state
  - employs hack
Basic TCP data transfer:
TCP’s sliding window protocol

- sender, receiver maintain buffers $\text{MaxSendBuffer}$, $\text{MaxRcvBuffer}$
Note asynchrony between TCP module and application.

Sender side: maintain invariants

- \( \text{LastByteAcked} \leq \text{LastByteSent} \leq \text{LastByteWritten} \)
- \( \text{LastByteWritten} - \text{LastByteAcked} < \text{MaxSendBuffer} \)

\[ \rightarrow \text{buffer flushing (advance window)} \]
\[ \rightarrow \text{application blocking} \]

- \( \text{LastByteSent} - \text{LastByteAcked} \leq \text{AdvertisedWindow} \)

Thus,

\[ \text{EffectiveWindow} = \text{AdvertisedWindow} - (\text{LastByteSent} - \text{LastByteAcked}) \]

\[ \rightarrow \text{upper bound on new send volume} \]
Actually, one additional refinement:

\[ \text{CongestionWindow} \]

**EffectiveWindow** update procedure:

\[
\text{EffectiveWindow} = \text{MaxWindow} - (\text{LastByteSent} - \text{LastByteAcked})
\]

where

\[
\text{MaxWindow} = \min\{\text{AdvertisedWindow}, \text{CongestionWindow}\}
\]

How to set **CongestionWindow**.

\[ \text{domain of TCP congestion control} \]
Receiver side: maintain invariants

- \( \text{LastByteRead} < \text{NextByteExpected} \leq \text{LastByteRcvd} + 1 \)
- \( \text{LastByteRcvd} - \text{NextByteRead} < \text{MaxRcvBuffer} \)
  \(\rightarrow\) buffer flushing (advance window)
  \(\rightarrow\) application blocking

Thus,

\[
\text{AdvertisedWindow} = \text{MaxRcvBuffer} - (\text{LastByteRcvd} - \text{LastByteRead})
\]
Issues:

How to let sender know of change in receiver window size after \textit{AdvertisedWindow} becomes 0?

- trigger ACK event on receiver side when \textit{AdvertisedWindow} becomes positive
- sender periodically sends 1-byte probing packet

$\rightarrow$ design choice: smart sender/dumb receiver

$\rightarrow$ same situation for congestion control
Silly window syndrome: Assuming receiver buffer is full, what if application reads one byte at a time with long pauses?

- can cause excessive 1-byte traffic

- if $\text{AdvertisedWindow} < \text{MSS}$ then set $\text{AdvertisedWindow} \leftarrow 0$
Do not want to send too many 1 B payload packets.

Nagle’s algorithm:

- rule: connection can have only one such unacknowledged packet outstanding
- while waiting for ACK, incoming bytes are accumulated (i.e., buffered)

... compromise between real-time constraints and efficiency.

→ useful for telnet-type applications
Sequence number wrap-around problem: recall sufficient condition

\[ \text{SenderWindowSize} < \left( \text{MaxSeqNum} + 1 \right) / 2 \]

\[ \rightarrow \] 32-bit sequence space/16-bit window space

However, more importantly, time until wrap-around important due to possibility of roaming packets.

<table>
<thead>
<tr>
<th>bandwidth</th>
<th>time until wrap-around †</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (1.5 Mbps)</td>
<td>6.4 hrs</td>
</tr>
<tr>
<td>Ethernet (10 Mbps)</td>
<td>57 min</td>
</tr>
<tr>
<td>T3 (45 Mbps)</td>
<td>13 min</td>
</tr>
<tr>
<td>F/E (100 Mbps)</td>
<td>6 min</td>
</tr>
<tr>
<td>OC-3 (155 Mbps)</td>
<td>4 min</td>
</tr>
<tr>
<td>OC-12 (622 Mbps)</td>
<td>55 sec</td>
</tr>
<tr>
<td>OC-24 (1.2 Gbps)</td>
<td>28 sec</td>
</tr>
</tbody>
</table>
Even more importantly, “keeping-the-pipe-full” consideration.

<table>
<thead>
<tr>
<th>bandwidth</th>
<th>delay-bandwidth product †</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (1.5 Mbps)</td>
<td>18 kB</td>
</tr>
<tr>
<td>Ethernet (10 Mbps)</td>
<td>122 kB</td>
</tr>
<tr>
<td>T3 (45 Mbps)</td>
<td>549 kB</td>
</tr>
<tr>
<td>FDDI (100 Mbps)</td>
<td>1.2 MB</td>
</tr>
<tr>
<td>OC-3 (155 Mbps)</td>
<td>1.8 MB</td>
</tr>
<tr>
<td>OC-12 (622 Mbps)</td>
<td>7.4 MB</td>
</tr>
<tr>
<td>OC-24 (1.2 Gbps)</td>
<td>14.8 MB</td>
</tr>
</tbody>
</table>

→ 100 ms latency

Also, throughput limitation imposed by TCP receiver window size.

→ e.g., high-performance grid apps
RTT estimation

... important to not underestimate nor overestimate.

Karn/Partridge: Maintain running average with precautions

\[
\text{EstimateRTT} \leftarrow \alpha \cdot \text{EstimateRTT} + \beta \cdot \text{SampleRTT}
\]

- \text{SampleRTT} computed by sender using timer
- \(\alpha + \beta = 1; \ 0.8 \leq \alpha \leq 0.9, \ 0.1 \leq \beta \leq 0.2\)
- \text{TimeOut} \leftarrow 2 \cdot \text{EstimateRTT} \text{ or } \text{TimeOut} \leftarrow 2 \cdot \text{TimeOut} \text{ (if retransmit)}

\[\rightarrow \ \text{need to be careful when taking} \ \text{SampleRTT}\]
\[\rightarrow \ \text{infusion of complexity}\]
\[\rightarrow \ \text{still remaining problems}\]
Hypothetical RTT distribution:

\[
\begin{array}{c}
\text{# Samples} \\
\end{array}
\stackrel{\text{RTT}}{\rightarrow}
\begin{array}{c}
\text{# Samples} \\
\end{array}
\]

\[\text{need to account for variance}\]

\[\text{not nearly as nice}\]
Jacobson/Karels:

- Difference = SampleRTT − EstimatedRTT
- EstimatedRTT = EstimatedRTT + \( \delta \cdot \text{Difference} \)
- Deviation = Deviation + \( \delta(|\text{Difference}| - \text{Deviation}) \)

Here \( 0 < \delta < 1 \).

Finally,

- TimeOut = \( \mu \cdot \text{EstimatedRTT} + \phi \cdot \text{Deviation} \)

where \( \mu = 1, \phi = 4 \).

\[\rightarrow \text{ persistence timer}\]

\[\rightarrow \text{ how to keep multiple timers in UNIX}\]