The Transmission Control Protocol (TCP)
What are transmission protocols needed for?

**Addressing:** application-to-application addressing

**Reliable delivery:** the receiver application should receive the same data stream the source puts on the net

**Segment order maintenance:** data segments should reach the application in the same order they left the sender

**Flow control:** the data sending speed should adapt itself to the receivers speed

**Congestion control:** the transmission speed cannot be faster than the speed of the slowest link traversed

**Segmentation:** data is sent in segments that provide the highest throughout
Transmission Control Protocol

- 1981

- reliable, sequenced byte stream on top of datagram

- connections with graceful close or hard reset

- retransmit if no ACK after $n \cdot RTO$ (initially, 6 s)

- $RTO = \text{average} + 4 \cdot \text{smoothed mean deviation}$

- windowed congestion control: not part of the protocol spec...

- windows:
- *flow control*: avoid overrunning slow receiver advertised by receiver
- *congestion control*: avoid network overload maintained by sender – *cwnd*

- packet header overhead: 20 bytes
Transmission Control Protocol

Maximum segment size (MSS) set during connection establishment

Reliability: acknowledgments, round trip delay estimations and data retransmission

Flow control: TCP uses a variable-window mechanism

Congestion control: “slow start” and related schemes

June 6, 1998
### TCP Header

<table>
<thead>
<tr>
<th>Field</th>
<th>0</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit source port number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16-bit destination port number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32-bit sequence number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32-bit acknowledgment number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-bit header length</td>
<td>reserved (6 bits)</td>
<td>U</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>C</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>G</td>
<td>K</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>I</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>16-bit window size</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16-bit TCP Checksum</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16-bit urgent pointer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>options (if any)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data (if any)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The most common option is the maximum segment size
option.
Connection Establishment and Termination

three-way handshake

Each side can just close its transmission side resulting in a
half close.
Connection Establishment

- *tao* sends a SYN segment with an initial sequence number (ISN) and the maximum segment size (MSS) it is willing to receive.

- *lupus* replies with a SYN segment acknowledging ISN and announcing its MSS.

- MSS can be at the most as large as the interface segment size minus 40 bytes.
TCP

Connection Establishment

**Server**
- `socket()`
- `bind()`
- `listen()`
- `accept` (blocks until connection from client)

**Client**
- `socket()`
- `connect()`
- `write()`
- `read()`

Data Request
- `read()` (process request)

Data Reply
- `write()`
- `read()`

Connection Establishment
- `read()` -> `write()`
- `write()` -> `read()`
Connection Termination

- Each side terminates its half of the connection by sending a FIN segment.
- After acknowledging the FIN, the other can still send data: *(half close)*.
- Connection can be aborted with an RST segment.
Interactive Data Transfer

- Data received from the application is usually sent in segments of MSS
- Interactive applications (rlogin, telnet) type one character, generate one packet
- The sender can force the delivery of small packets using the PSH (push) flag
- With delayed acknowledgments, the receiver delays sending the acknowledgments until it has some data to send ("piggy-back") or a 200 ms timer expires.
Interactive Data Transfer

0.0
PSH 0:1(1) ack1 (char)
ack1
0.01
PSH 0:1(1) ack 2 (echo char)
ack 2
0.015
0.016
0.0
PSH 0:1(1) ack 2 (echo char)
delayed
ack
0.015
0.03
0.0
PSH 1:2(1) ack1 (echo char)
Ack 2
0.0
PSH 0:1(1) ack1 (char)
Interactive Data Transfer

- Sending a lot of small segments can add congestion to a wide area network

- *Nagle Algorithm*: a sender can at most have one outstanding small segment (that has not yet been acknowledged).

- All data arriving at TCP from the application are queued until the currently outstanding segment is acknowledged.
Flow Control in TCP

- sliding window mechanism to adjust the sender transmission speed to receiver speed

- sliding window permits sending of multiple segments before waiting for an acknowledgment

- ACK segments indicate the last correctly received byte and the number of bytes the receiver is still willing to accept (window)
Flow Control in TCP

TCP

SYN 141:141(0) <mss 1024> WIN 4096
SYN 181:181(0) <mss 1024>
ACK 142 WIN 3072
ACK 1, WIN 4096
1:1025 (1024) ACK 1, WIN 4096
1025:2049 (1024) ACK 1, WIN 4096
2049:3073 (1024) ACK 1, WIN 4096
ACK 3073, WIN 2049
3073:4097 (1024) ACK 1, WIN 4096

June 6, 1998
Acknowledgments and Retransmission

- TCP receiver acknowledges the last correctly received byte

- After sending a segment the sender starts a timer.

- If the timer expires before receiving an acknowledgment for the sent segment the segment is considered lost and must be retransmitted.

- The timeout value is calculated dynamically according to the measured round trip times ($RTT$).

\[
\begin{align*}
\text{Err} & \quad = \quad RTT - A \\
A & \quad = \quad A + g \text{Err}
\end{align*}
\]

\[A = \text{smoothed RTT}\]
\[\text{gain } g = 1/8\]
\[ D = D + h(|\text{Err}| - D) \quad D = \text{smoothed mean deviation} \]
\[ \text{RTO} = A + 4D \]
Round-Trip Time Measurement

- TCP implementations use a 500-ms clock for time measurements and timeout determination.
- Only one measurement is done at a time.
- At the start of a measurement a counter is set to 0 and is then incremented every time the 500 ms TCP timer is invoked and the number of the sent segment is remembered.
- Only after acknowledging the sent segment can a new measurement start.
- After a retransmission the timeout value is not updated until an acknowledgment for a segment arrives that was
not retransmitted (Karn’s algorithm).
Round-Trip Time Measurement

As the 500-ms timer is used for determining the RTT the values used for updating the timeout value might differ up to ±500 ms from the actual value.
Congestion Control in TCP

- A connection’s rate is determined as transmission window/round trip time.

- When the sum of the connection rates over a link is higher than the link rate, segments can be dropped.

- TCP uses packet drops and timeouts as congestion indication.
Slow Start and Congestion Avoidance

- To avoid congestion in advance, the sender must adapt its transmission window to the available link bandwidth.

- connection establishment: window = 1 MSS: congestion window \((cwnd)\)

- The congestion window is increased by 1 MSS for each acknowledged segment.

- At any time the sender has a transmission window of

  \[
  \text{transmission window} = \min(\text{advertised window}, \text{congestion window})
  \]
Slow Start and Congestion Avoidance

- slow start ➤ congestion window increases exponentially
- congestion (packet drops)
- timeout ➤ \( cwnd = 1 \) MSS.
- reduce slow start threshold \( ssthresh \) by half
- congestion avoidance = after reaching \( ssthresh \): add \( 1/cwnd \) for each ACK ➤ add one segment for RTT
Fast Retransmission and Fast Recovery

- Using only timeouts as loss indication leads to long idle periods.

- *fast retransmission*: receiver acknowledges out-of-order segment with an ACK of the last correctly received segment

- 3 duplicate ACKs $\Rightarrow$ retransmission of the last missing segment

- older TCP versions: slow start as for time-out

- *fast recovery*: $cwnd \leftarrow ssthresh + 3$
Congestion Control

slow start

$65535$

$ssthresh$

$cwnd$

congestion avoidance

$t$

slow start

$+$

$+$

June 6, 1998
Congestion Example

- source and receiver buffers: 8192 bytes
- router buffer: 2128 bytes
- link bandwidth: 2128 bytes/s
- MSS = 1024
- round trip delay: 1 s
Congestion Example: Slow Start

<table>
<thead>
<tr>
<th>time</th>
<th>tao</th>
<th>lupus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>segment 3, win=1024</td>
<td>0:1025 (1024) ACK 1</td>
</tr>
<tr>
<td>2</td>
<td>win=2048</td>
<td>ACK 1025, WIN 8192</td>
</tr>
<tr>
<td>2.0001</td>
<td>segment 5</td>
<td>1025:2049 (1024) ACK 1</td>
</tr>
<tr>
<td>2.0002</td>
<td>segment 6</td>
<td>2049:3073 (1024) ACK 1</td>
</tr>
<tr>
<td>3.001</td>
<td>win=3072</td>
<td>ACK 2049, WIN 8192</td>
</tr>
<tr>
<td>4.0003</td>
<td>win=4096</td>
<td>ACK 3073, WIN 8192</td>
</tr>
<tr>
<td>4.0003</td>
<td>segment 9</td>
<td>3073:4097 (1024) ACK 1</td>
</tr>
<tr>
<td>4.5004</td>
<td>segment 10</td>
<td>4097:5121 (1024) ACK 1</td>
</tr>
<tr>
<td>5.5005</td>
<td>segment 11</td>
<td>5121:6145 (1024) ACK 1</td>
</tr>
<tr>
<td>6.5006</td>
<td>segment 12</td>
<td>6145:7169 (1024) ACK 1</td>
</tr>
<tr>
<td>5.0004</td>
<td>win=5121</td>
<td>ACK 4097, WIN 8192</td>
</tr>
<tr>
<td>5.5006</td>
<td>win=6144</td>
<td>ACK 5121, WIN 8192</td>
</tr>
<tr>
<td>6.5004</td>
<td>win=7168</td>
<td>ACK 6145, WIN 8192</td>
</tr>
<tr>
<td>7.5006</td>
<td>win=8192</td>
<td>ACK 7169, WIN 8192</td>
</tr>
<tr>
<td>8.0007</td>
<td>segment 15</td>
<td>7169:8193 (1024) ACK 1</td>
</tr>
<tr>
<td>8.5008</td>
<td>segment 16</td>
<td>9217:10241 (1024) ACK 1</td>
</tr>
<tr>
<td>8.5009</td>
<td>segment 18</td>
<td>10241:11265 (1024) ACK 1</td>
</tr>
</tbody>
</table>
Congestion Example: Fast Retransmission

<table>
<thead>
<tr>
<th>time</th>
<th>tao</th>
<th>lupus</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.501</td>
<td>segment 19</td>
<td>11265:12289 (1024) ACK 1</td>
</tr>
<tr>
<td>8.501</td>
<td>segment 20</td>
<td>12289:13313 (1024) ACK 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACK 12289, <strong>WIN 8192</strong></td>
</tr>
<tr>
<td>9.0001</td>
<td></td>
<td>segment 21</td>
</tr>
<tr>
<td>9.0001</td>
<td>segment 22</td>
<td>13313:14337 (1024) ACK 1</td>
</tr>
<tr>
<td>10.002</td>
<td>segment 23</td>
<td>14337:15361 (1024) ACK 1</td>
</tr>
<tr>
<td>10.503</td>
<td>segment 24</td>
<td>15361:16385 (1024) ACK 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACK 12289, <strong>WIN 7168</strong></td>
</tr>
<tr>
<td>10.001</td>
<td></td>
<td>segment 25</td>
</tr>
<tr>
<td>11.002</td>
<td></td>
<td>ACK 12289, <strong>WIN 6144</strong></td>
</tr>
<tr>
<td>11.501</td>
<td></td>
<td>ACK 12289, <strong>WIN 5120</strong></td>
</tr>
<tr>
<td>11.501</td>
<td>segment 28</td>
<td>12289:13313 (1024) ACK 1</td>
</tr>
</tbody>
</table>
Congestion Example: Fast Retransmission

<table>
<thead>
<tr>
<th>Time</th>
<th>tao</th>
<th>ACK</th>
<th>WIN</th>
<th>Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:00</td>
<td>WIN 1024</td>
<td>16385:17409 (1024)</td>
<td>8192</td>
<td>29</td>
</tr>
<tr>
<td>12:00</td>
<td>segment 30</td>
<td>ACK 17409, WIN 8192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13:00</td>
<td>WIN 2048</td>
<td>17409:18433 (1024)</td>
<td>ACK 1</td>
<td>31</td>
</tr>
<tr>
<td>14:00</td>
<td>segment 32</td>
<td>ACK 18433, WIN 8192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14:03</td>
<td>segment 33</td>
<td>ACK 19457, WIN 8192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15:04</td>
<td>WIN 3072</td>
<td>19457:20481 (1024)</td>
<td>ACK 1</td>
<td>34</td>
</tr>
<tr>
<td>15:04</td>
<td>WIN 4096</td>
<td>ACK 20481, WIN 8192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17:01</td>
<td>WIN 4352</td>
<td>ACK 20481, WIN 8192</td>
<td></td>
<td>37</td>
</tr>
</tbody>
</table>

June 6, 1998
Congestion Example: Fast Recovery

<table>
<thead>
<tr>
<th>time</th>
<th>tao</th>
<th>lupus</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.501</td>
<td>WIN 4096</td>
<td>ACK 16385, WIN 8192</td>
</tr>
<tr>
<td>13.002</td>
<td>segment 31</td>
<td>17409:18433 (1024) ACK 1</td>
</tr>
<tr>
<td>13.503</td>
<td>segment 32</td>
<td>18433:19457 (1024) ACK 1</td>
</tr>
<tr>
<td>14.503</td>
<td>segment 32</td>
<td>19457:20481 (1024) ACK 1</td>
</tr>
<tr>
<td>13.505</td>
<td>WIN 4352</td>
<td>ACK 17409, WIN 8192</td>
</tr>
<tr>
<td>14.004</td>
<td>WIN 4593</td>
<td>ACK 18433, WIN 8192</td>
</tr>
<tr>
<td>14.504</td>
<td>WIN 4821</td>
<td>ACK 19457, WIN 8192</td>
</tr>
<tr>
<td>15.504</td>
<td>WIN 5038</td>
<td>ACK 20481, WIN 8192</td>
</tr>
</tbody>
</table>
Silly Window Syndrome and Probe Packets

- ACKs can get lost ➤ window update lost
- advertised window size ↓ 0 (freeze) ➤ prevent at sender and receiver
- sender: window probes of size 1 at increasing intervals
- receiver: advertise 0 or ≥ MSS
- sender: send full-sized segment or wait until everything ACKed
TCP throughput

Keep pipe occupied \( \Rightarrow \text{window} \uparrow \) bandwidth-delay product

Bandwidth-delay product: capacity of “pipe” \( \Rightarrow \) continental T3 = 45 Mb/s \( \times \) 5000 km \( \times \) 5 \( \mu s \) / km \( \approx \) 140 kB

- “long, fat pipes”

- currently, 64 kB window \( \Rightarrow \) window scale option

- RTO update only once per RTT \( \Rightarrow \) time stamp option

- non-congestion losses (wireless) \( \Rightarrow \) selective acknowledgements
TCP throughput

\[ R_{TCP} = \frac{1.22 \cdot MTU}{RTT \sqrt{\text{loss}}} \]

for losses < 16%

derivation:

\[ R = \frac{W \cdot MTU}{RTT} \]

For TCP, packet drop halve window, takes \( W/2 \) times to reach \( W \) again:

\[ R_{TCP} = \frac{0.75W \cdot MTU}{RTT} \]
T/TCP

- Common TCP application: request to a server, reply to the client, done.

- 10 TCP segments

- Connection establishment + termination: $\geq 2$ RTT + server processing time
T/TCP

- distinguish between consecutive transactions
- connection count (CC) option added to header
- host maintains a per-host cache of the last seen timeout value, MSS, window size and the CC value.
- Client can combine the SYN, FIN, data request and the current CC value in one segment.
- If received CC value is larger than the cached CC, the server can combine the SYN, FIN, ACK of the sender’s SYN and the reply in one segment.
- The client acks the server’s SYN and FIN in one segment.
● \[\text{transaction time} = \text{RTT} + \text{the processing time}\]