The Transmission Control Protocol

Dorgham Sisalem
GMD Fokus, Berlin
sisalem@fokus.gmd.de

TU Berlin, WS 1995/96

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 can not be faster than the speed of the slowest link traversed on the connections path

Segmentation: data is sent in segments that provide the highest throughout.

Transmission Control Protocol

- TCP is connection oriented and full duplex.
- The maximum segment size *(MSS)*is set during connection establishment.
- Reliability is achieved using acknowledgments, round trip delay estimations and data retransmission.
- TCP uses a variable window mechanism for flow control.
- Congestion control and avoidance is reached using slow start and congestion avoidance schemes.

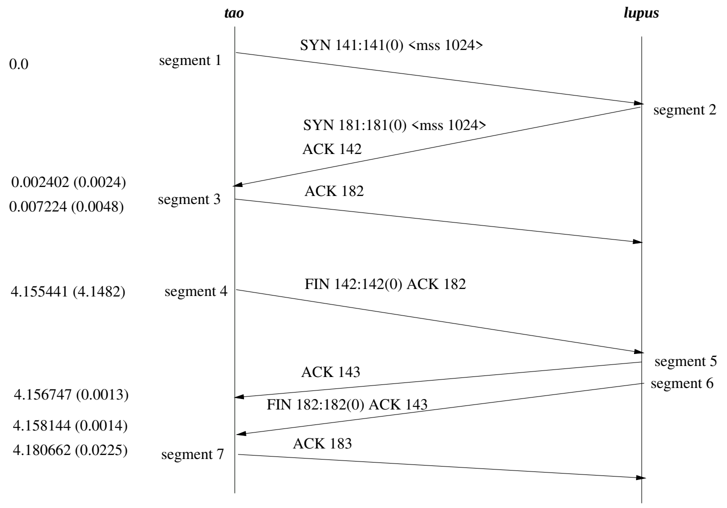
TCP Header

<table>
<tr><td colspan="8">0</td><td>16 … 31</td><td></td></tr>
<tr><td colspan="8">16-bit source port number</td><td>16-bit destination port number</td><td rowspan="5">20 bytes</td></tr>
<tr><td colspan="9">32-bit sequence number</td></tr>
<tr><td colspan="9">32-bit acknowledgment number (next byte expected)</td></tr>
<tr><td>4-bit header length</td><td>reserved (6 bits)</td><td>U R G</td><td>A C K</td><td>P S H</td><td>R S T</td><td>S Y N</td><td>F I N</td><td>16-bit window size</td></tr>
<tr><td colspan="8">16-bit TCP Checksum</td><td>16-bit urgent pointer</td></tr>
<tr><td colspan="9">options (if any)</td><td></td></tr>
<tr><td colspan="9">data (if any)</td><td></td></tr>
</table>

- The most common option is the maximum segment size option.

Connection Establishment and Termination

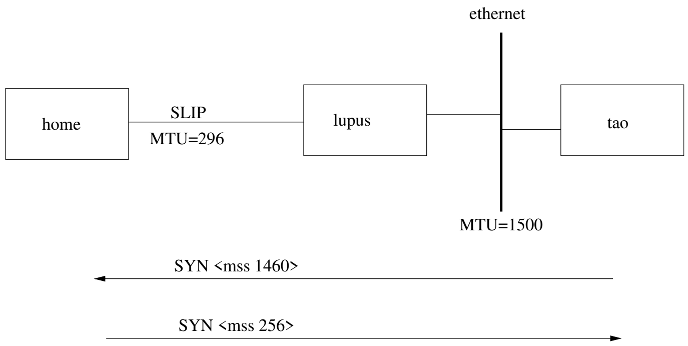
- Connection establishment is done with a three way handshake



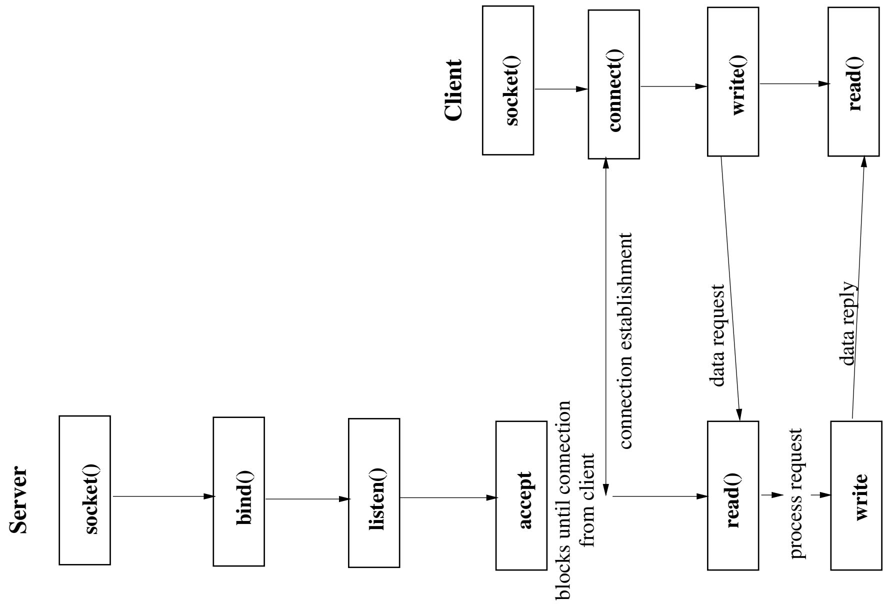
- Each side can just close its transmission side resulting in a half close.

Connection Establishment

- *tao* sends a SYN segment with an initial sequence number (ISN) and 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.



Connection Establishment

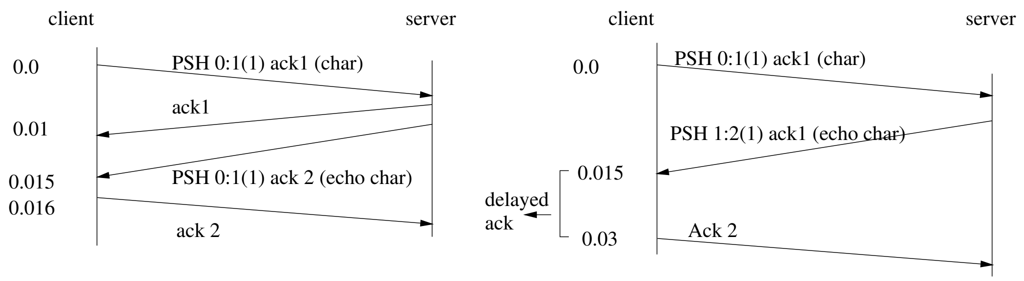


Connection Termination

- A sender terminates its part of the connection by sending a FIN segment.
- After acknowledging the FIN the receiver can still send data on its part of the connection *(half close)*.
- A connection can be aborted with a RST segment.

Interactive Data Transfer

- Data received from the application is usually sent in segments of MSS.
- In the case of interactive applications -rlogin, telnet- 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 or a 200 ms timer expires.



GMD fokus

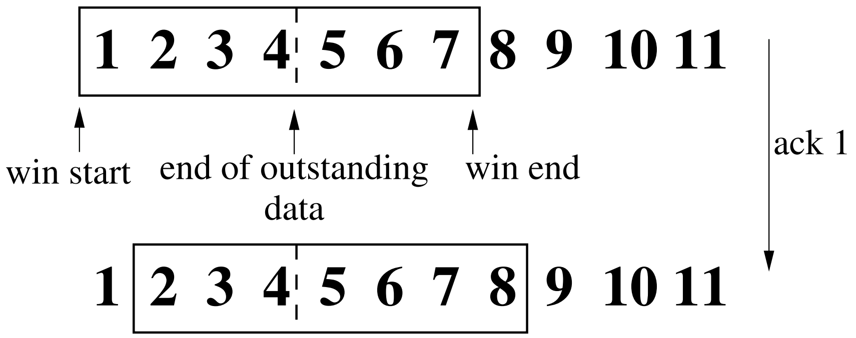
Interactive Data Transfer

- 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.

GMD fokus

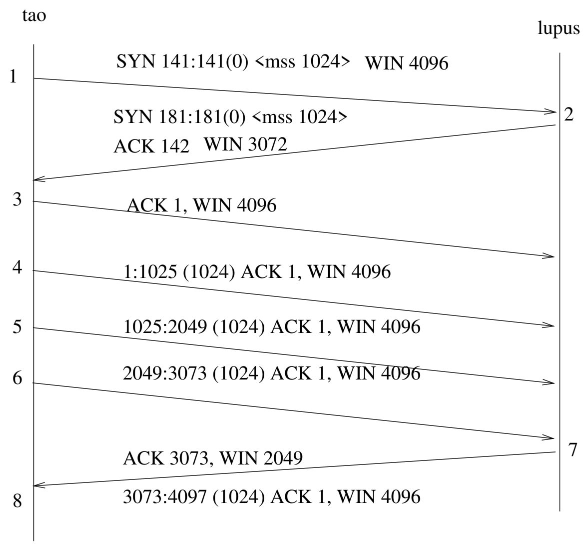
Flow Control in TCP

- TCP uses a sliding window mechanism to adjust the senders transmission speed to that of the receiver.
- The sliding window permits the 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.



Flow Control in TCP

- Ack segments indicate the last correctly received byte and the number of bytes the receiver is still willing to accept.



Acknowledgments and Retransmission

- A TCP receiver always 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)*.

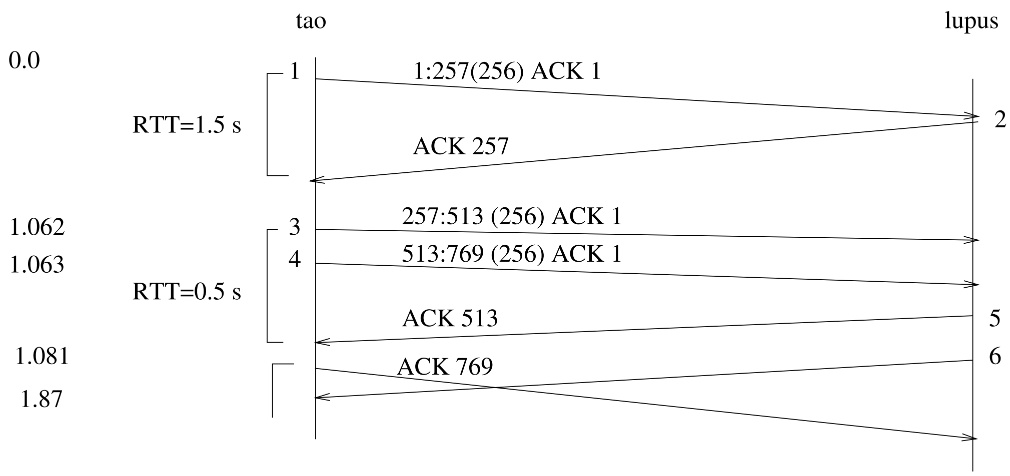
$$
\begin{array}{rcll}
\text{Err} & = & \text{RTT} - A & A = \text{smoothed RTT} \\
A & = & A + g\text{Err} & \text{gain } g = 1/8 \\
D & = & D + h\left(|\text{Err}| - D\right) & D = \text{smoothed mean deviation} \\
\text{RTO} & = & A + 4D &
\end{array}
$$

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 *(Karns's algorithm)*.

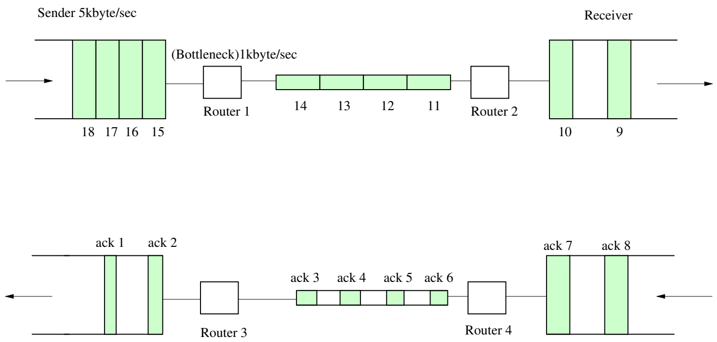
Round-Trip Time Measurement

- 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's 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.
- On connection establishment TCP uses a window of the size of 1 MSS *Congestion Window*.
- The congestion window is increased by 1 MSS for each acknowledged segment.
- At any time the sender has has a transmission window of

$$\text{transmission window} = \min(\text{advertised window, congestion window})$$

Slow Start and Congestion Avoidance

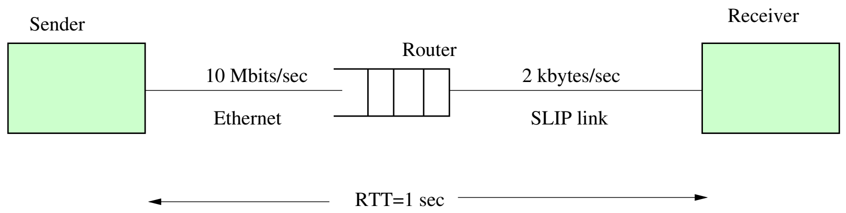
- With the slow start scheme the congestion window is exponentially increased.
- This can quickly congest the network and cause packet drops.
- After a timeout the congestion window is set again to 1 MSS.
- Slow start is reused but only until the congestion window reaches half of its value before the timeout.
- Afterwards the congestion window is increased only by 1/congestion window for each acknowledged segment *(congestion avoidance)*.

Fast Retransmission and Fast Recovery

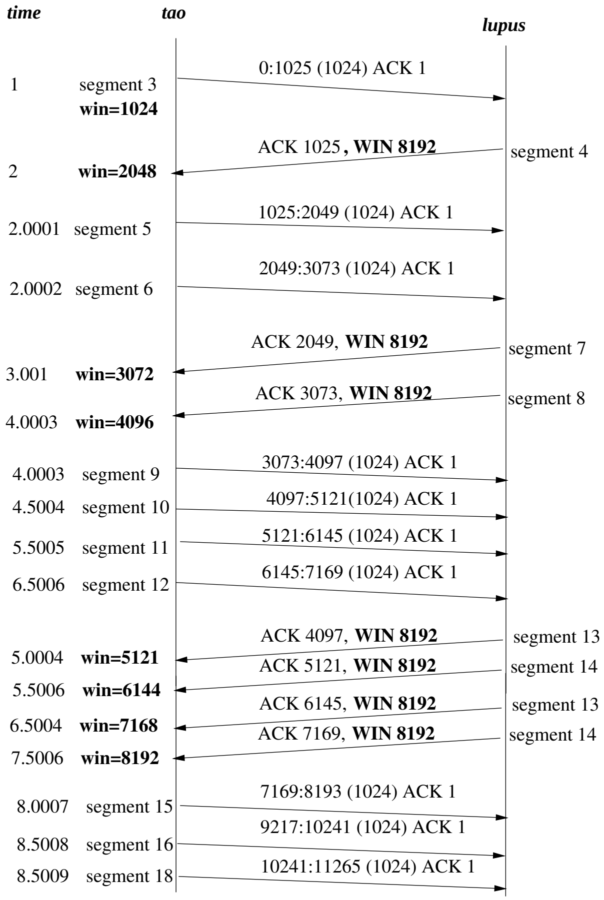
- Using only timeouts as loss indication leads to long idle periods.
- With the *fast retransmission* scheme the receiver acknowledges each out of order segment with an ack of the last correctly received segment.
- Receiving 3 duplicate acks triggers at the source the retransmission of the last acked segment.
- In the older TCP versions the same actions taken after a timeout are used in this case as well.
- in TCP versions using *fast recovery* the congestion window is only reduced by half after each loss.

Congestion Example

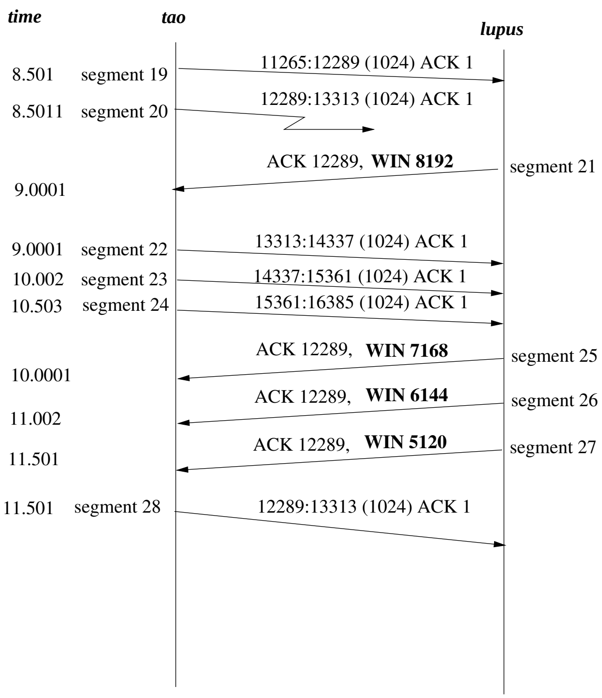
- Both source and receiver have buffers up to 8192 bytes.
- The router has a buffer of 2128 bytes.
- The link has a bandwidth of 2128 bytes/sec.
- MSS=1024.
- The configuration has a round trip delay of 1 sec.



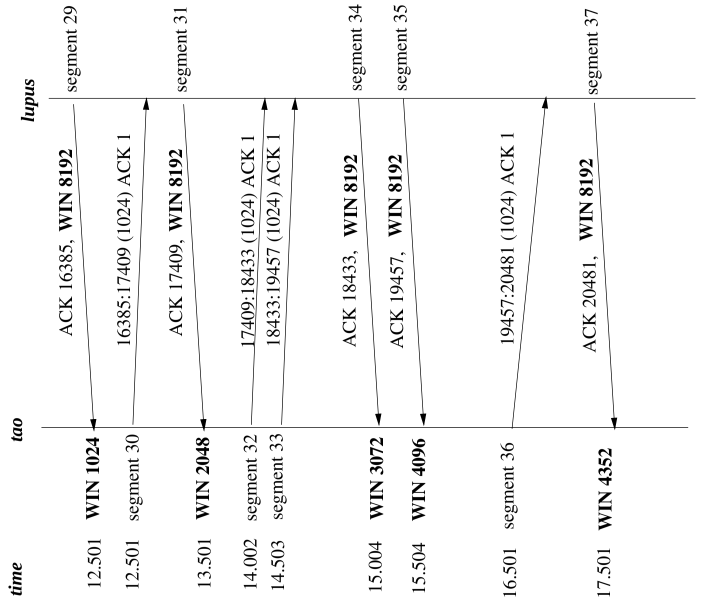
Congestion Example: Slow Start



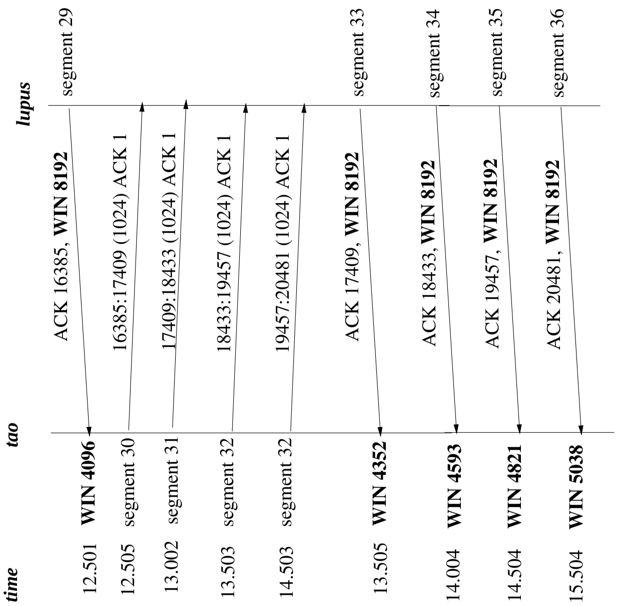
Congestion Example: Fast Retransmission



Congestion Example: Fast Retransmission



Congestion Example: Fast Recovery



Silly Window Syndrome and Probe Packets

- It is possible for the advertised window size to go to 0.
- After freeing some buffer, the receiver sends an update message with the size of the available buffer.
- After receiving an ack with WIN=0 the sender starts a persist timer.
- On the expiration of the persist timer a small packet of 1 byte payload is sent to see if a window update message got lost -such a packet is called *probe packet*.
- To avoid sending small packets the receiver must not advertise small segments, i.e., segments smaller than MSS *(silly window syndrome)*.

GMD fokus

TCP Future and Performance

- The capacity of a link is measured as

$$\text{capacity} = \text{bandwidth*RTT}$$

- The throughput of TCP is limited to

$$\text{throughput} = \frac{\text{max window size}}{\text{RTT}}$$

- Using a window scale option improves performance on long fat pipes.
- Updating the RTO value every RTT leads to aliasing effects.
- More accurate timeout calculations can be reached using a time stamp option

GMD fokus

T/TCP

- Lots of TCP transactions consist simply of a request to a server and a reply to the client.
- This simple transaction require the sending of 10 segments.
- Due to the connection establishment and termination a simple transaction requires at least two RTT times plus the processing time required at the server.

T/TCP

- To distinguish between consecutive transactions a connection count option is added to the header.
- To avoid unnecessary overhead a host might maintain a per-host cache of the last seen timeout value, MSS, window size and the CC value.
- A client can combine the SYN, FIN, data request and the current CC value in one segment.
- If the 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.
- This minimal transactions reduces the time needed for the transaction to a minimum of RTT plus the processing time at the server.