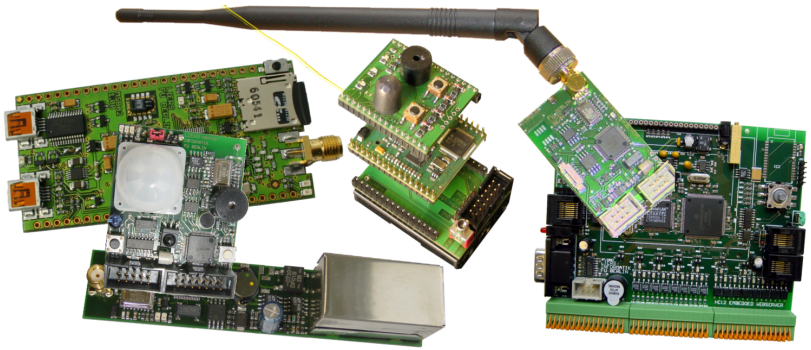


19531 - Telematics

11th Tutorial - Transmission Control Protocol

Bastian Blywis

Department of Mathematics and Computer Science
Institute of Computer Science
20. January, 2011

Outline

Transmission Control Protocol (TCP), fundamental features:

- Transport layer protocol, layer 4
- Reliable end-to-end connection
- Bidirectional connection
- In-order delivery of data (to the application layer)
- Stream-oriented, does not preserve message boundaries (application layer protocol has to reassemble messages)
- “Byte pipe” between hosts for higher layer protocols
- Multiplexing based on ports

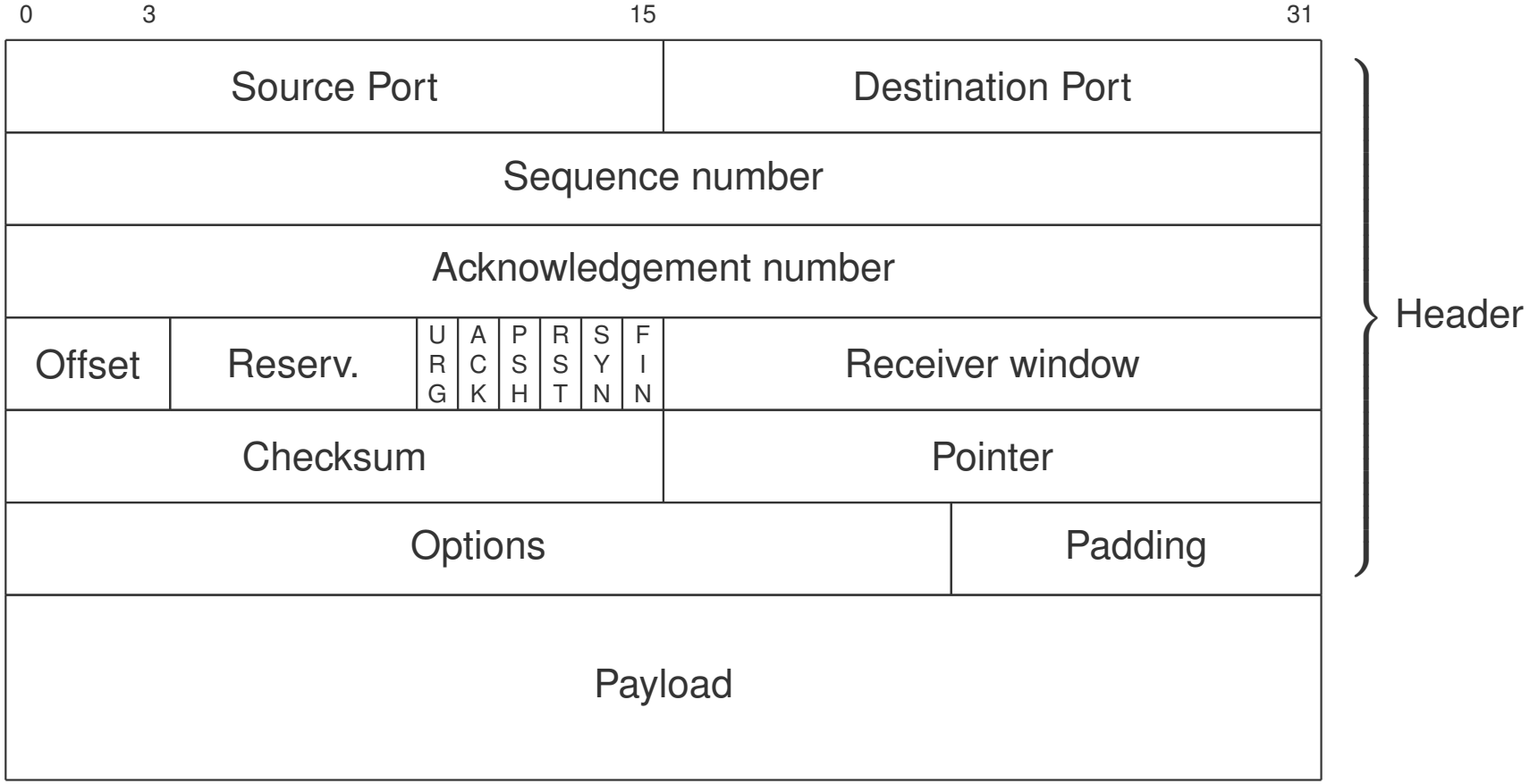


Figure: TCP Segment

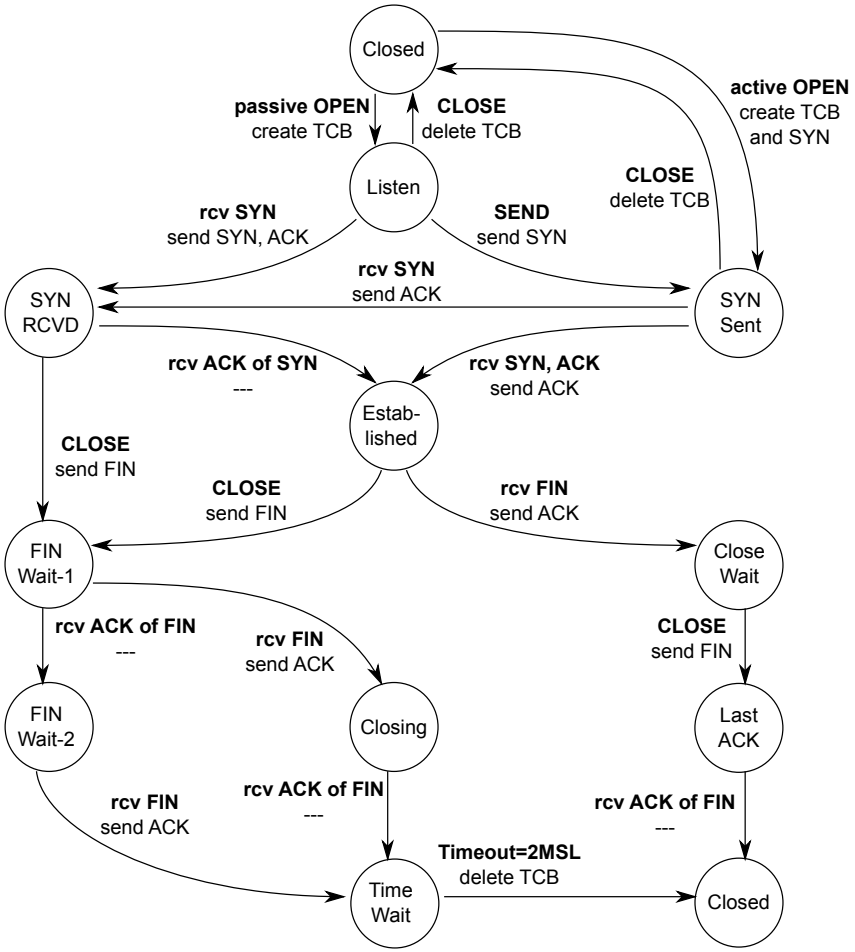


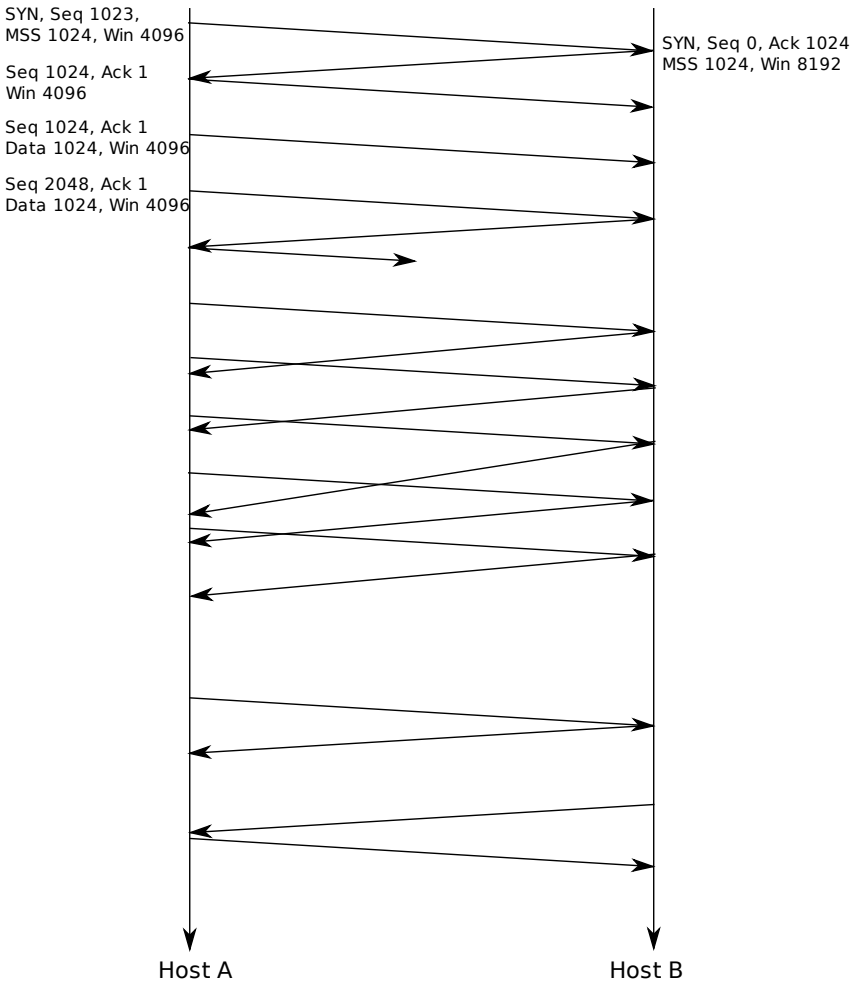
Figure: TCP State Machine

TCP Connection

Consider the following message sequence chart of a TCP connection. The horizontal arrows represent the transmission of segments between two hosts while the vertical arrows represent the time. The labels show an excerpt of the header fields (values and flags) of the TCP segments.

1. Discuss the exchange of the the first three segments and the values of the header fields.
2. Host A transmits 7 segments with a payload size of 1024 Byte to host B and closes the connection afterwards. The first two segments carrying payload are already annotated in the message sequence chart. Label the remaining segments with the values of the headers header fields considering the following information:
   - One of the segments is lost in the network (indicated by an arrow that does not reach the right side).
   - Assume that host A supports fast retransmit and no timeouts due to the lost segment occur in A's TCP implementation.

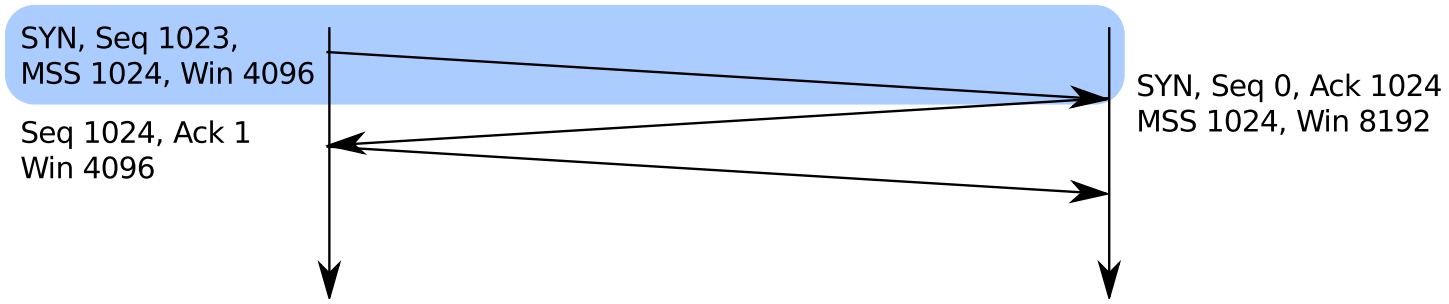
TCP Connection



3-Way-Handshake (connection establishment) initialized by host A

A → B: Active connection establishment by sending a segment with set synchronization flag (SYN)

- (Random) Initialization of sequence number counter (Seq) with value 1023
- Segments sent by host B may have a maximum segment size (MSS)[1] of 1024 Byte
- Receiver window (buffer) of 4096 Bytes is allocated by host A

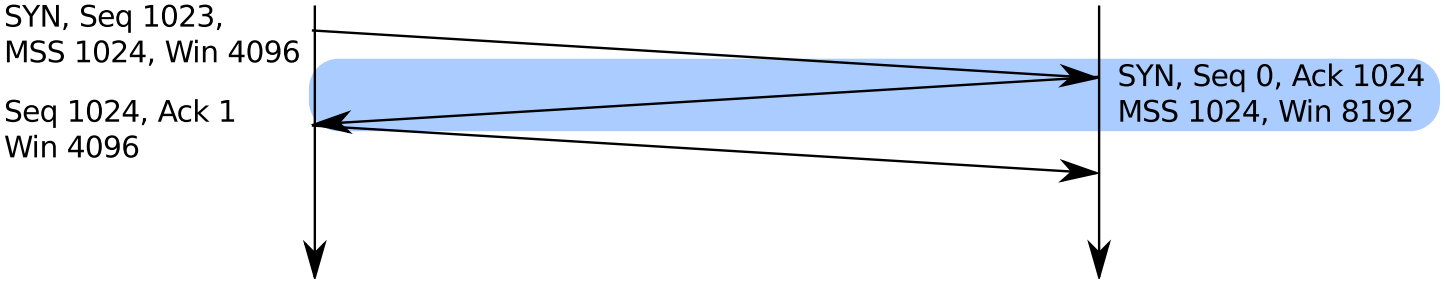


[1] Default MSS is 536 Bytes if the TCP MSS Option is not used

TCP Connection

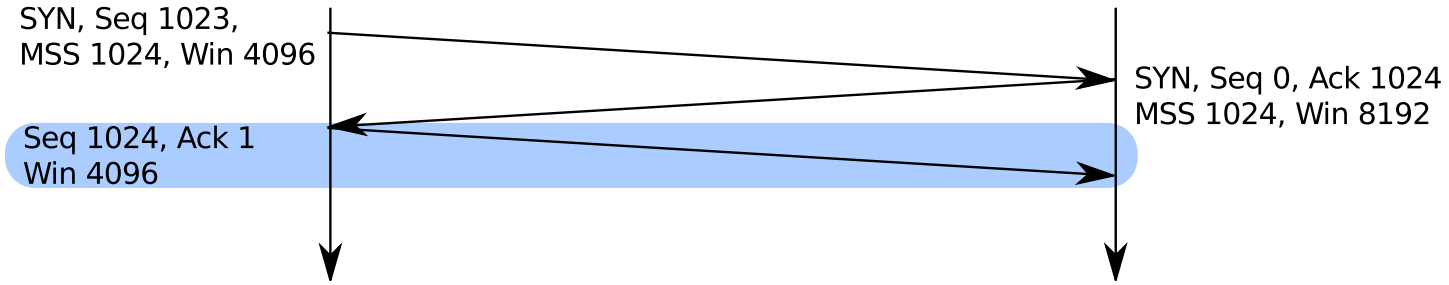
B → A: Passive connection establishment by sending a synchronization segment (SYN-ACK)

- Acknowledgement of Seq=1023 from A with ACK=1024
- Segment with set SYN and ACK flags
- (Random) Initialization of sequence number counter with value 0
- Segment sent by host A may have a MSS of 1024 Byte
- Receiver window of 8192 Bytes is allocated by host B

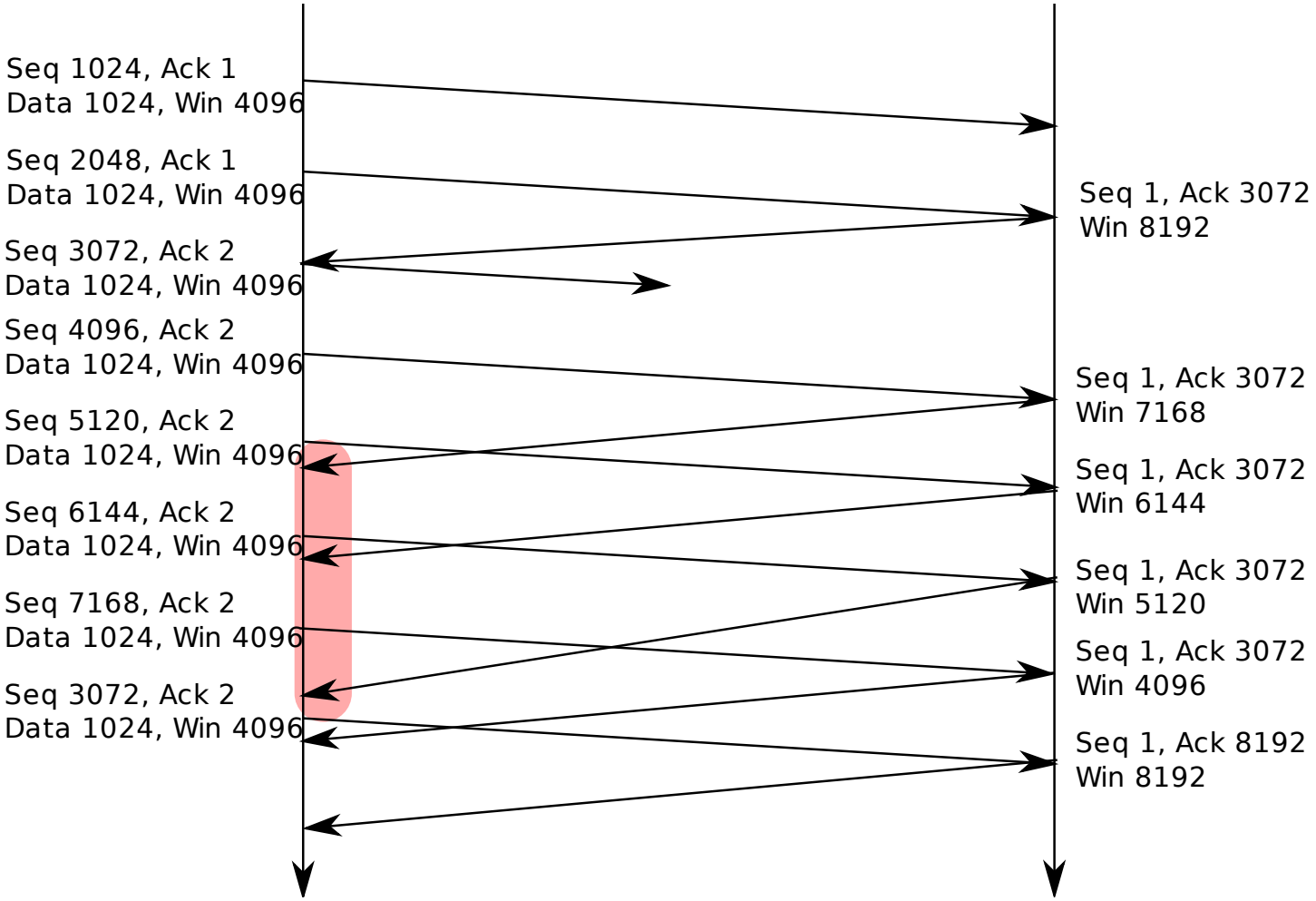


A → B: Finishing the three-way handshake

- When the segment is received by host A, it knows that the connection has been established (but host B does not know it yet)
- Acknowledgement of the SYN-ACK, Seq=0 sent by B by segment with ACK=1

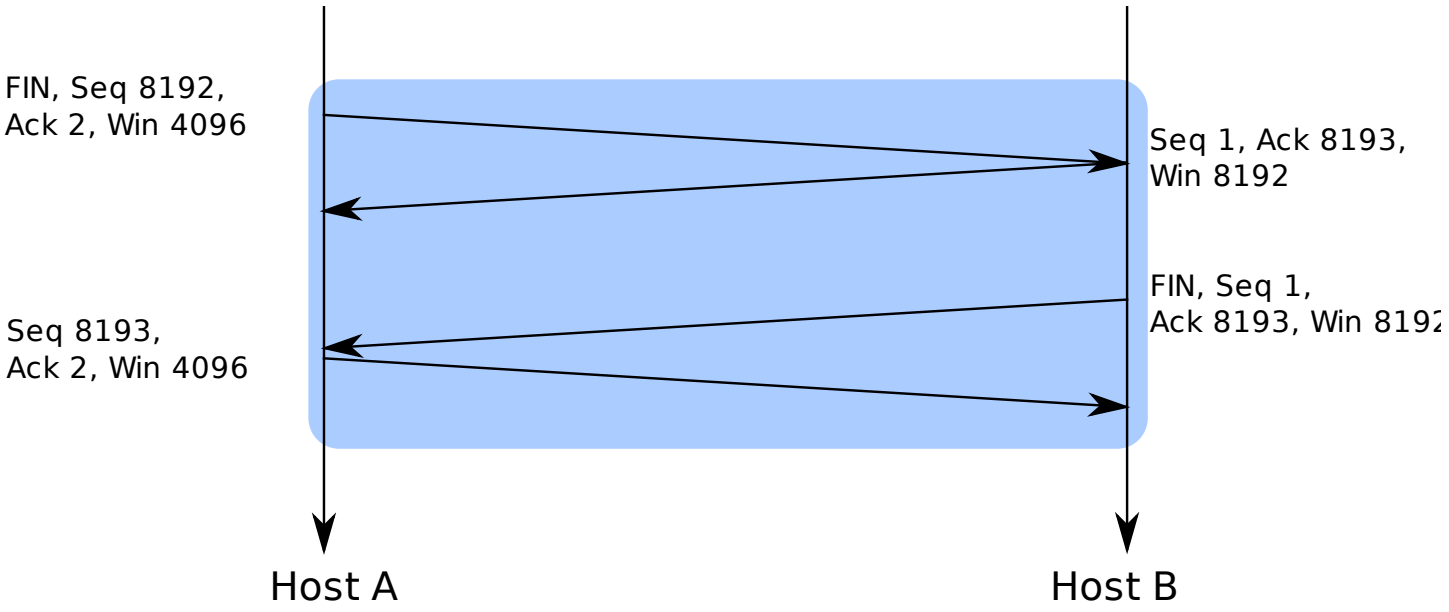


Data Transfer including a retransmission due to *fast retransmit* feature (3 duplicate ACKs)



Connection Termination

- Similar to three-way handshake
- Connection can optionally be closed in one direction but still be open in the other



Notes:

- RFC 793 defines the TCP protocol
- Fast retransmit is part of congestion control defined in RFC 2581
- Sequence numbers
  - Acknowledgements without data do not use up a sequence numbers
  - Only data bytes increment the sequence number
  - Exception: See connection establishment and termination
- The 3rd segment with data (Seq=3072) is lost. Therefore host B sends the acknowledgement (ACK = 3072) multiple times.
- After the third duplicate acknowledgement (see problem statement) host A assumes packet loss (fast retransmit) and retransmits the segment
- The second and third segments of the connection termination from host A could have been merged

Basic TCP Features

The TCP protocol as defined in RFC 793 specifies several features besides to provision of a reliable end-to-end connection. Explain the following features / properties:

1. Push function
2. Urgent data transport
3. TCP Options
4. Connection reset

Basic TCP Features

1. Push Flag
   - Problem: TCP can buffer data to create larger segments for transmission to reduce overhead (increases delay)
   - Problem: TCP does not immediately forward received data to the application layer to reduce context switches
   - Solution: Immediate data transfer signaled by push flag
   - Meaning on the sender side: “Send now, even if segment is small”
   - Meaning on the receiver side: “Push data to application”

Basic TCP Features

2. Urgent Flag
   - Problem: All bytes in TCP stream have the same priority (it is just one stream)
   - Problem: Sometimes the application has needs to signal some urgent information
   - Solution: Multiplex urgent data in normal TCP stream
   - Sender side: Creates segment with urgent data followed by regular data
   - Receiver side: Urgent data is forwarded to the application immediately, rest of data is processed normally

Basic TCP Features

Urgent data transport feature rarely used

- RFC 793, page 17: “The urgent pointer points to the sequence number of the octet **following the urgent data**.”
- RFC 961, page 6: “Page 17 <of RFC 793> is wrong. The urgent pointer points to the **last octet of urgent data** (not to the first octet of non-urgent data).”
- RFC 1122, page 84: “Urgent Pointer: RFC-793 Section 3.1. The second sentence is in error: the urgent pointer points to the sequence number of the **LAST octet** (not LAST+1) in a sequence **of urgent data**. The description on page 56 (last sentence) is correct.”

⇒ most TCP implementations use the definition of RFC 793
⇒ most applications make no use of this feature because of the incompatibility of TCP implementations

TCP Options

- Problem: TCP feature set is limited
- Solution: Extend TCP by options
- Similar approach to IPv4 options and IPv6 extensions
- Connection establishment phase: Signal which options are supported or want to be used
- Some options, e.g., MSS option are simply appended to the header; the receiver will either evaluate the option or ignore it

⇒ Options are optional, not all are implemented and commonly used

Basic TCP Features

Kind	Length	Meaning	Reference
0	-	End of Option List	RFC 793
1	-	No-Operation	RFC 793
2	4	Maximum Segment Size	RFC 793
3	3	WSOPT - Window Scale	RFC 1323
4	2	SACK Permitted	RFC 2018
5	N	SACK	RFC 2018
6	6	Echo (obsoleted by option 8)	RFC 1072
7	6	Echo Reply (obsoleted by option 8)	RFC 1072
8	10	TSOPT - Time Stamp Option	RFC 1323
9	2	Partial Order Connection Permitted	RFC 1693
10	3	Partial Order Service Profile	RFC 1693
11		CC	RFC 1644
12		CC.NEW	RFC 1644
13		CC.ECHO	RFC 1644
14	3	TCP Alternate Checksum Request	RFC 1146
15	N	TCP Alternate Checksum Data	RFC 1146
16		Skeeter	Knowles
17		Bubba	Knowles
18	3	Trailer Checksum Option	Subbu, Monroe
19	18	MD5 Signature Option	RFC 2385
20		SCPS Capabilities	Scott
21		Selective Negative Acknowledgements	Scott
22		Record Boundaries	Scott
23		Corruption experienced	Scott
24		SNAP	Sukonnik
26		TCP Compression Filter	Bellovin
27	8	Quick-Start Response	RFC 4782
28	4	User Timeout Option	RFC 5482
253	N	RFC3692-style Experiment 1	RFC 4727
254	N	RFC3692-style Experiment 2	RFC 4727

Table: List of TCP Options as assigned by IANA

Basic TCP Features

- TCP uses the Internet Checksum for error detection (see IPv4)
- Alternate checksum algorithms can be requested by a TCP option

Number	Description	Reference
0	TCP Checksum	RFC 1146
1	8-bit Fletchers's algorithm	RFC 1146
2	16-bit Fletchers's algorithm	RFC 1146
3	Redundant Checksum Avoidance	Kay

Table: TCP Alternate Checksum Numbers as assigned by IANA

Reset Flag

- Problem: TCP connection has to be reliable but errors can happen
  - Bug in TCP implementation
  - Segment with set SYN flag send to closed TCP port
- Solution: Send segment with set reset flag
- TCP connection is reset only if
  1. Sequence number is valid
  2. TCP is not in LISTEN state

The History of TCP

The Transmission Control Protocol has gotten many modifications/improvements over time. Discuss when the following got added, list the relevant RFCs (if any), and briefly explain the extension/algorithm/feature:

1. Fast Retransmit
2. Fast Recovery
3. Congestion Control
4. Flow Control
5. Karn's Algorithm
6. Nagle' Algorithm
7. Selective Acknowledgements
8. TCP for networks with high bandwidth-delay product

List TCP variants that implement these features.

The History of TCP

TCP Variant	Year	RFC	
	1973		Kahn und Cerf begin research for a reliable packet-oriented transfer protocol
(Original) TCP	1981	793	includes flow control
	1983		Widespread adoption, BSD Version 4.2
Nagles's Algorithm	1984	896	Prevent small segments
Karn's Algorithm	1987	2988	Improved RTT estimation
TCP/Tahoe	1988	2001, 2581	Slow start and congestion avoidance (by Jacobson), BSD 4.3
	1988	1072	Selective Acknowledgements
TCP Reno	1990	2581	Fast retransmit and fast recovery (by Jacobson), BSD 4.3
	1992	1323	Window scaling, optimized RTT estimation
TCP Vegas	1994		RTT estimation with timestamps, proactive approach
TCP NewReno	1999	2582	Partial acknowledgments

Table: TCP Timeline (excerpt)

Nagles's Algorithm

- Application can generate limited data chunks, e.g., keystrokes of Telnet session
- Problem: Small segments are generated, large protocol overhead, increases number of packets in network
- Solution: Buffer data if waiting for acknowledgment of other segment; sent buffered data when ACK is received
- Increases delay
- Badly interacts with TCP *Delayed Acknowledgements*

Karn's Algorithm

- Problem: RTO of TCP depend on accurate RTT estimation
- Problem: Retransmitted segment have higher delay and distort RTT estimation
- Problem: ACK for retransmitted segment might acknowledge any copy of the segment (ambiguity)
- Solution: Ignore retransmitted segments for RTT estimation

Compatibility

There are many TCP implementations/variants available and used in today's operating systems. Discuss if and why these variants are (not) fully compatible.

TCP Variants, congestion control algorithms

- Tahoe
- Reno
- New Reno
- Vegas
- Westwood, Westwood+
- BIC
- CUBIC
- Scalable TCP
- Hybla TCP
- Compound TCP
- TCP-Peach
- Bandwidth Aware TCP
- TCP-LP
- HSTCP-LP
- High Speed TCP
- TCP-Illinois
- YeAH-TCP
- Fast TCP
- H-TCP
- ...

Compatibility

- All TCP implementations have to be compatible with the protocol described in RFC 793
  - Connection states, protocol procedure
  - Sequence & acknowledgement numbers
  - Receiver window
  - Header flags
  - Checksum
- Most TCP variants try to improve TCP for particular scenarios
  - Wireless networks
  - Networks with high bandwidth-delay-product
  - Use only the bandwidth left in the network and do not try to get a fair amount (TCP-LP)
- Most often only the congestion control algorithm is adapted
- Congestion control is a sender-side mechanism
- Both hosts of a TCP connection can run totally different and independent congestion control algorithms (or even none at all)

Which TCP variants and features are supported by modern operating systems? Give examples.
How can you configure your TCP implementation?

Congestion Control Algorithms implemented in the current Linux kernel

- Binary Increase Congestion (BIC) control
- CUBIC TCP
- TCP Westwood+
- H-TCP
- High Speed TCP
- TCP-Hybla congestion control algorithms
- TCP Vegas
- Scalable TCP
- TCP Low Priority
- TCP Veno
- YeAH TCP
- TCP Illinois

```
$ sysctl net.ipv4.tcp_available_congestion_control
```

Configurable TCP parameters from the Linux TCP Manual Page (excerpt)

- tcp_abort_on_overflow
- tcp_adv_win_scale
- tcp_app_win
- tcp_bic
- tcp_bic_low_window
- tcp_bic_fast_convergence
- tcp_dsack
- tcp_ecn
- tcp_fack
- tcp_fin_timeout
- tcp_frto
- tcp_keepalive_intvl
- tcp_keepalive_probes
- tcp_keepalive_time
- tcp_low_latency
- tcp_max_orphans
- tcp_max_syn_backlog
- tcp_max_tw_buckets
- tcp_mem
- tcp_orphan_retries
- tcp_reordering
- tcp_retrans_collapse
- tcp_retries1
- tcp_retries2
- tcp_rfc1337
- tcp_rmem
- tcp_sack
- tcp_stdurg
- tcp_synack_retries
- tcp_syncookies
- tcp_syn_retries
- tcp_timestamps
- tcp_tw_recycle
- tcp_tw_reuse
- tcp_window_scaling
- tcp_vegas_cong_avoid
- tcp_westwood
- tcp_wmem

```
$ sysctl -a | grep "net\.ipv4\.tcp_"
$ sysct -w net.ipv4.tcp_sack=0
```

Sketch a diagram depicting the behavior of a TCP implementation that detects congestion in the network (no ACKs are received for some segments). Assume three cases:

1. The TCP implementation does not support congestion avoidance.
2. The TCP implementation supports slow start and congestion avoidance.
3. The TCP implementation supports fast retransmit and fast recovery (implies slow start and congestion avoidance support).

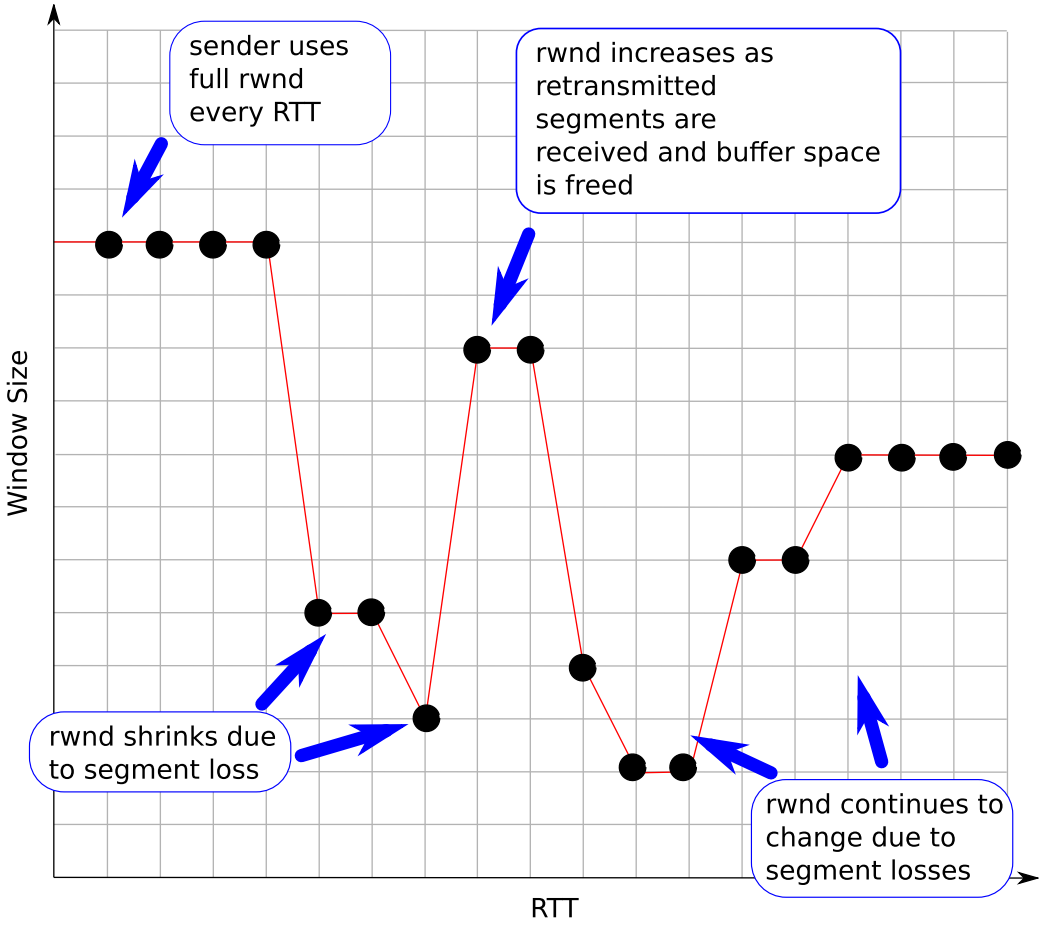


Figure: Without Congestion Aviodance

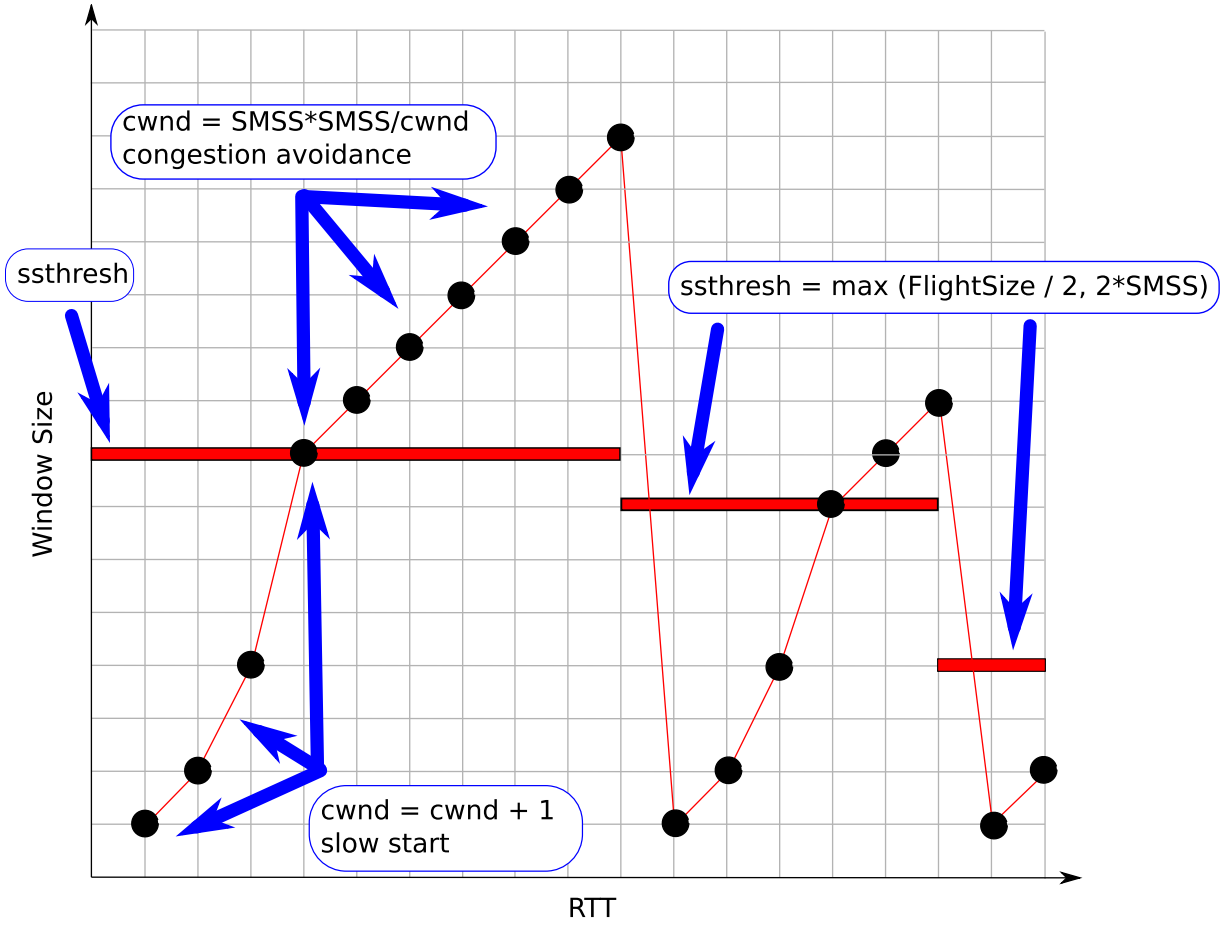


Figure: Slow Start and Congestion Aviodance

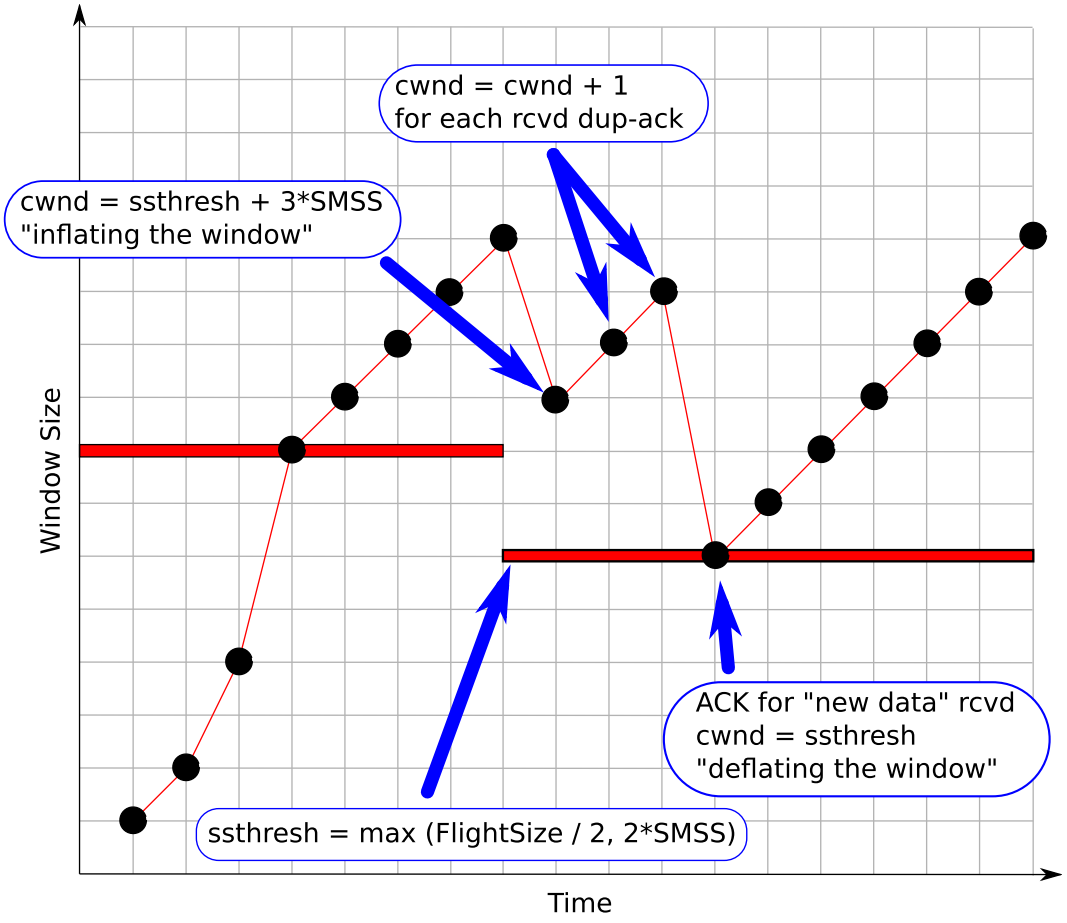


Figure: Fast Retransmit and Fast Recovery

Congestion Control Continued

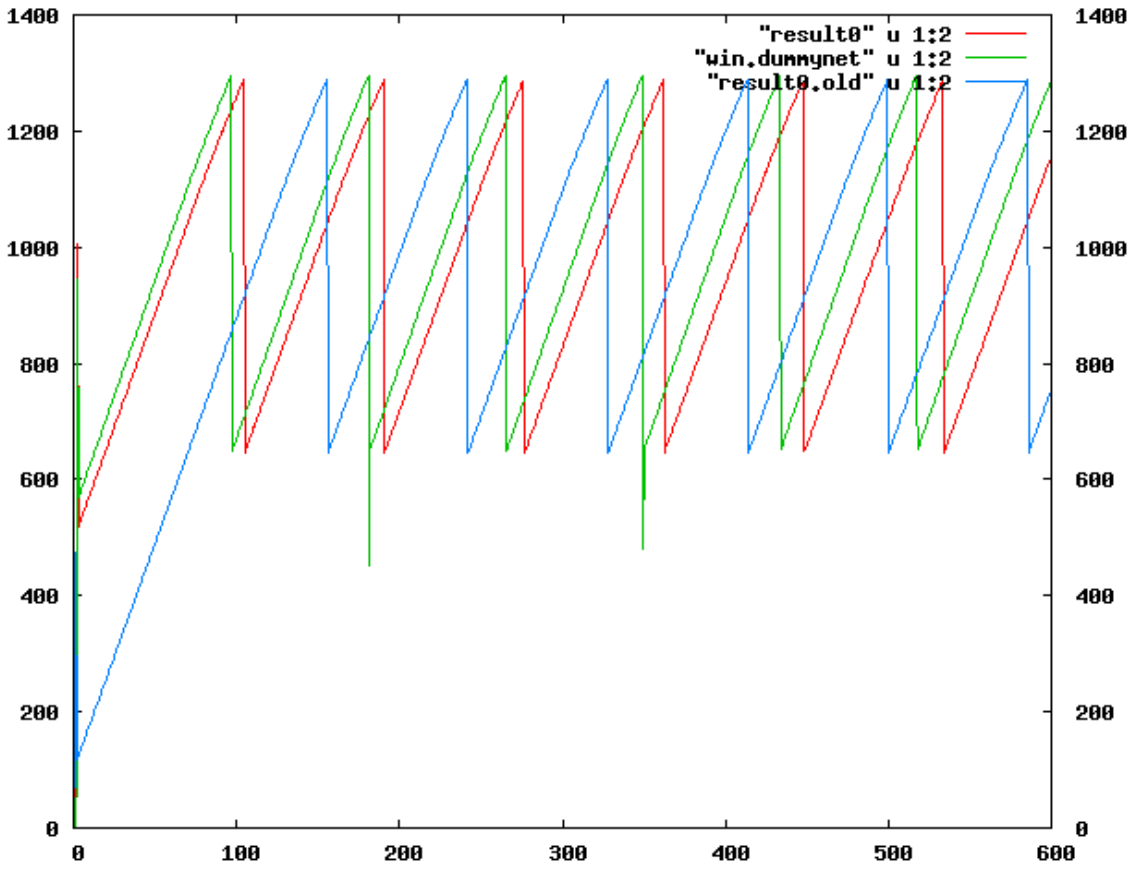


Figure: Reno [Source]

Congestion Control Continued

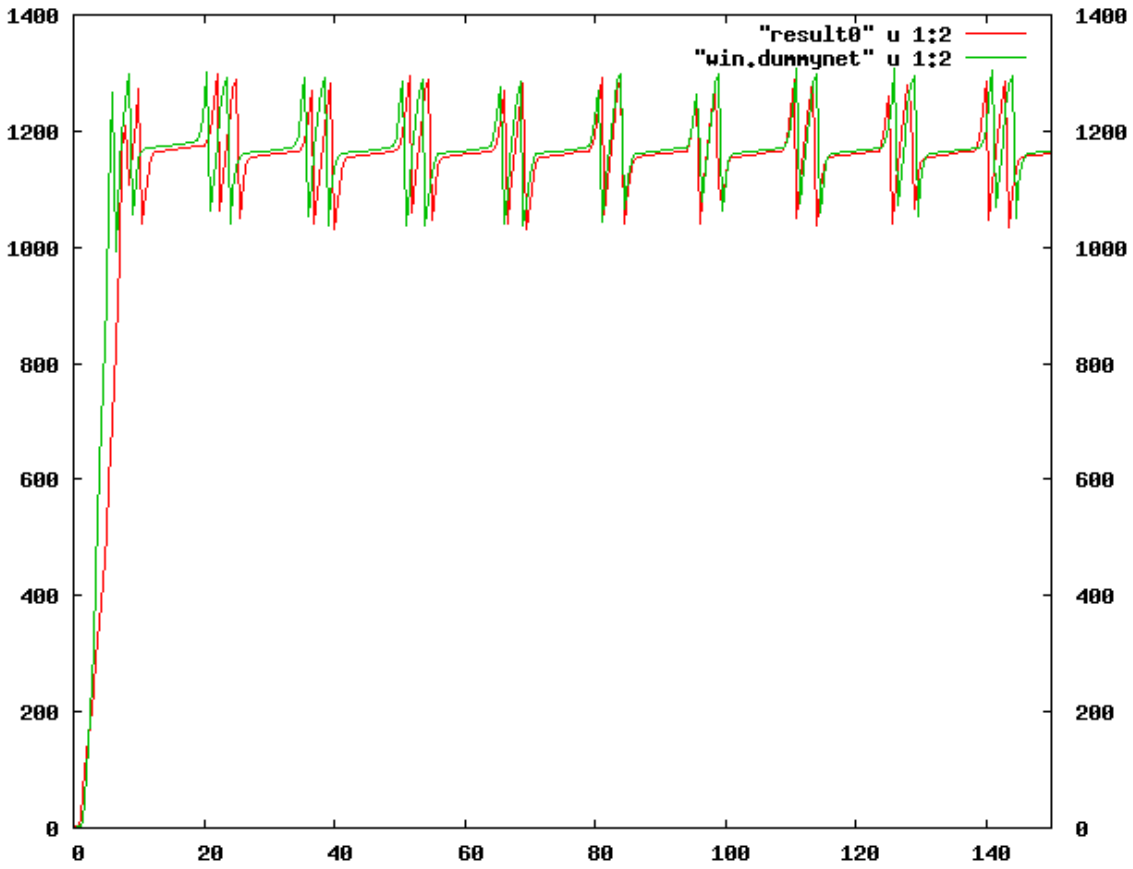


Figure: BIC [Source]

Congestion Control Continued

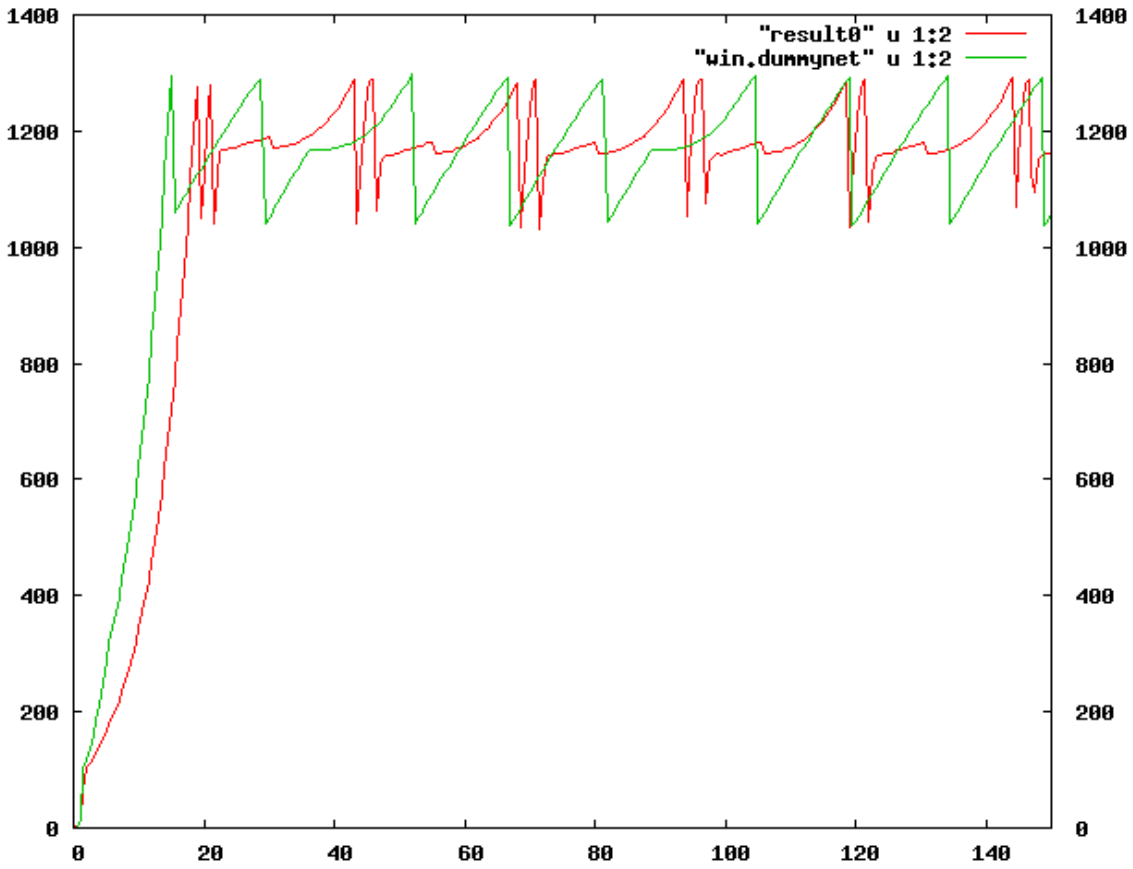


Figure: CUBIC [Source]

Congestion Control Continued

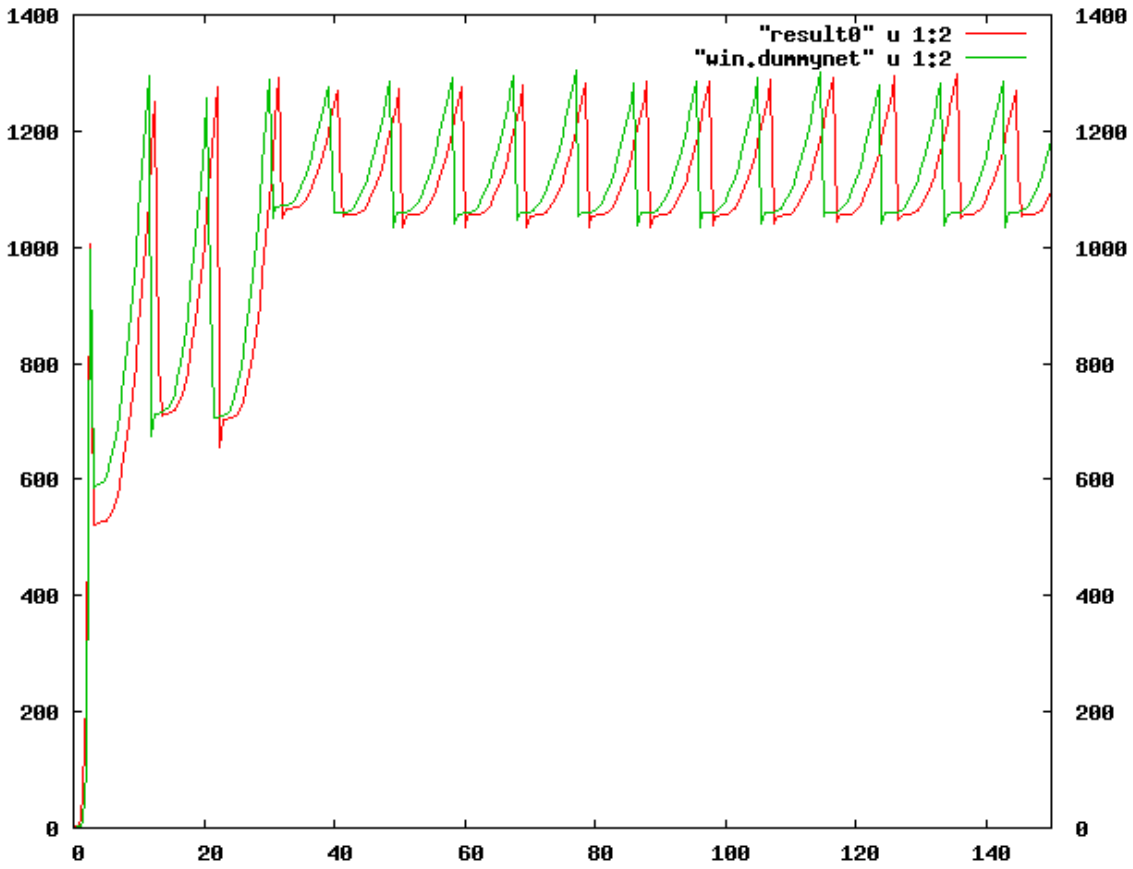


Figure: H-TCP [Source]

TCP Remarks

- Slow Start Threshold after segment loss

$$ssthresh = max(\text{FlightSize}/2, 2 * SMSS) \quad (1)$$

FlightSize := amount of outstanding (unacknowledged) data in the network.

RFC 2581:

"An easy mistake to make is to simply use cwnd, rather than FlightSize, which in some implementations may incidentally increase well beyond rwnd."

Note: In most descriptions found in books the ssthresh is halved after segment loss. Many TCP implementations also behave this way.

TCP Remarks

Confusion about the meaning of *"half-open"*

- Definition of half-open connections found in RFC 793:

  > *"An established connection is said to be "half-open" if one of the TCPs has closed or aborted the connection at its end without the knowledge of the other, or if the two ends of the connection have become desynchronized owing to a crash that resulted in loss of memory."*

- Embryonic state:
  - After host A sends a SYN to host B $\Rightarrow$ embryonic connection for A
  - After B send SYN-ACK to A $\Rightarrow$ embryonic connection for B

Note: Both terms are often confused or used synonymously.

TCP Remarks

Sender Maximum Segment Size (SMSS)

- Largest segment that the sender can transmit
- Minimum of network MTU, path MTU discovery or RMSS
- Does not include TCP/IP headers and options

Receiver Maximum Segment Size (RMSS)

- Largest segment size the receiver can/will accept
- Signaled by MSS option
- Defaults to 536 bytes
- Does not include TCP/IP headers and options

TCP Remarks

The initial value of cwnd as defined in RFC 2581

$$cwnd_{initial} \leq 2 * SMSS \tag{2}$$

Note: Might have a different value for other congestion control algorithms

TCP Remarks

Congestion window size incrementation

- Slow start: increment cwnd by at most SMSS bytes for each received ACK that acknowledges new data $\Rightarrow$ exponential increase
- Congestion avoidance: cwnd is incremented by 1 full-sized segment per round-trip time (RTT) $\Rightarrow$ linear increase
- Slow start phase ends when cwnd exceeds or reaches ssthresh
- Slow start phase end when congestion is detected

TCP Slow Start and Congestion Avoidance

Consider a TCP implementation . . . Sketch the size of the congestion window and the slow start threshold . . .

- Initial slow start threshold of 8 kB
- Maximum segment size of 1 kB
- Receiver window of 16 kB (fixed)
- Fast retransmit and fast recovery are not supported
- Timeouts after the 8th, the 11th, and the 17th round trip time

TCP Slow Start and Congestion Avoidance

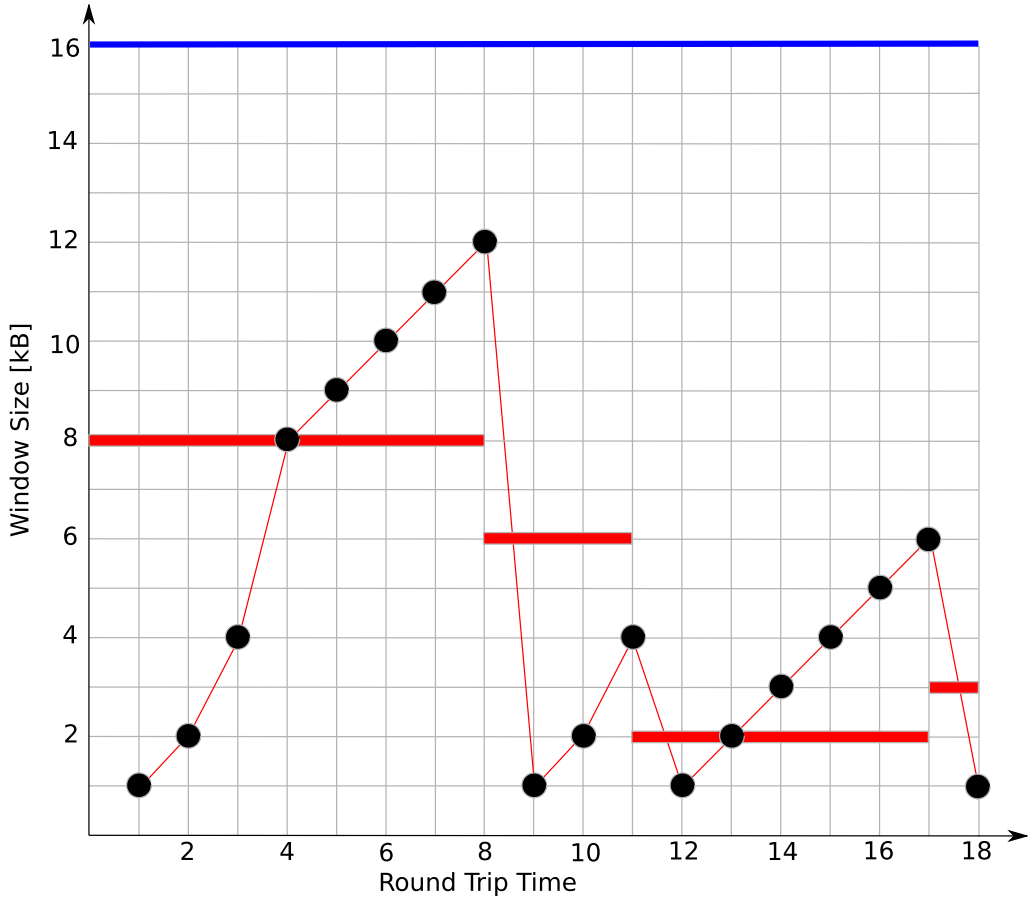


Figure: Slow Start and Congestion Avoidance

Question: What would happen, when cwnd reaches the receiver window value?

TCP Limitations

TCP is a reliable transport protocol but reliability can decrease throughput and increase delay.

1. The maximum payload size of a TCP segment is limited to 65495 Byte. Explain this value.
2. Consider a communication channel with a data rate of 1 GBit/s and a delay of 10 ms. What is the maximum throughput a TCP connection can achieve? How efficient is the TCP connection?

TCP Limitations

- TCP segments are usually sent as payload in IP datagrams
- Total-Length-Field of the IP header represents the size of the IP header and IP payload
- Field is 16 bit, thus the maximum value is 65,535
- Headers of TCP and IP usually have a size of 20 Bytes each (options are used rarely)
- Maximum TCP payload size of $65,535 - 2 * 20 = 65,495$ Byte is a good choice to prevent IP fragmentation
- Payload is usually much smaller, due to limitation of lower layers protocols (see MTU)

TCP Limitations

Maximum throughput:

1. The window size is restricted to 65,535 Byte
2. An acknowledgement is required at least every 65,535 Byte
3. Transmission of 65.535 Byte incl. acknowledgements takes at least 20 ms
4. $50 * 65,535$ Byte can be transmitted per second
5. The maximum throughput is limited to $50 * 65,535 * 8 = 26,214,000$ bps

Efficiency: $\frac{26,214,000}{1*10^9} = 0.0262,14 \approx 2.6\%$
For this reason, the congestion window has to be scaled to larger sizes to achieve a higher efficiency within modern networks.

Discuss attacks on the TCP protocol and how these can be detected or prevented.
Hint: Watch the following conference talk. TCP DoS Vulnerabilities

TCP Security

Attacks:

- TCP Session Hijacking: Redirect TCP stream to another host
- TCP Reset: Reset the TCP connection between two hosts
- SYN Flood: Do not finish three-way handshake (binds resources)
- RTT Timer manipulation: Effect a false RTT estimation by (cumulatively) acknowledging not yet received segments
- Shrew attacks: Low-rate DoS attack with timed bursts of packets

Prevention:

- Sequence number randomization to prevent hijacking and reset attacks (see 12th assignment)
- SYN Cookies: Delay allocation of resources and store state in sequence number
- Encryption on higher layers to prevent hijacking
- Watch ARP cache to prevent hijacking

Note: There are many more attacks. TCP is not focused on security.

```
# netstat -atn
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):3930      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):3316      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):4147      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):3854      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):1500      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):3931      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):2325      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):1652      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):1499      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):1710      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):1125      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):1913      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):2445      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):3929      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):1119      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):4602      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):3518      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):1529      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):1551      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):1502      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):3122      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):1311      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):3529      SYN_RECV
tcp        0      0 (MY-SERVER-IP):80       (ATTACKER-IP):3856      SYN_RECV
                                    ...
```

Figure: Host experiencing a TCP SYN-Flood Attack

The Last Slide™

Thank you for your attention.

Questions?