

The Network Stack (2)

Lecture 6, Part 1: TCP

Dr Robert N. M. Watson

2020-2021



The Network Stack (2)

- The Transmission Control Protocol (TCP)
 - The TCP state machine
 - TCP congestion control
 - TCP implementations and performance
 - The evolving TCP stack
 - Lab 3 on TCP
- Wrapping up the Advanced Operating Systems lecture series

Lecture 6, Part 1

Lecture 6, Part 2

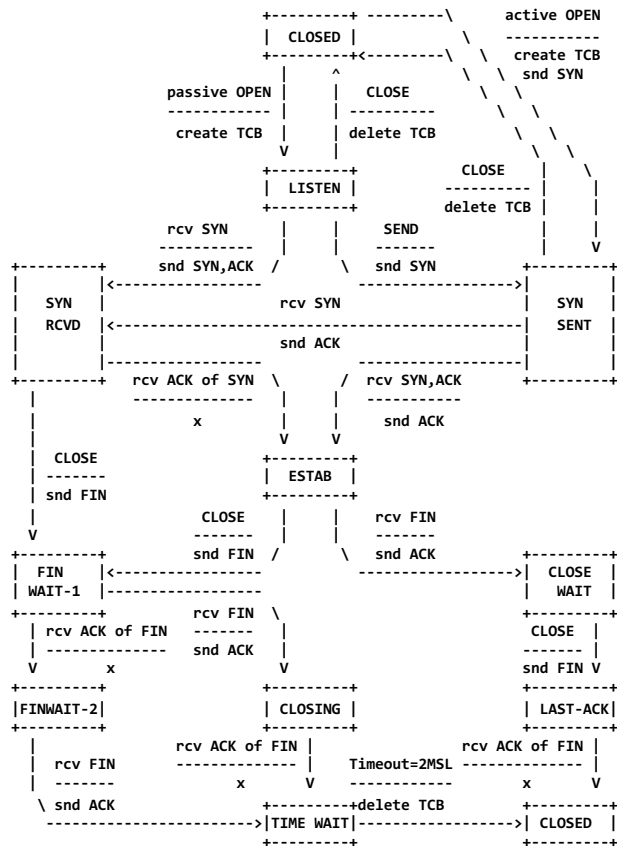
Lecture 6, Part 3



The Transmission Control Protocol (TCP)

September 1981

Transmission Control Protocol
Functional Specification



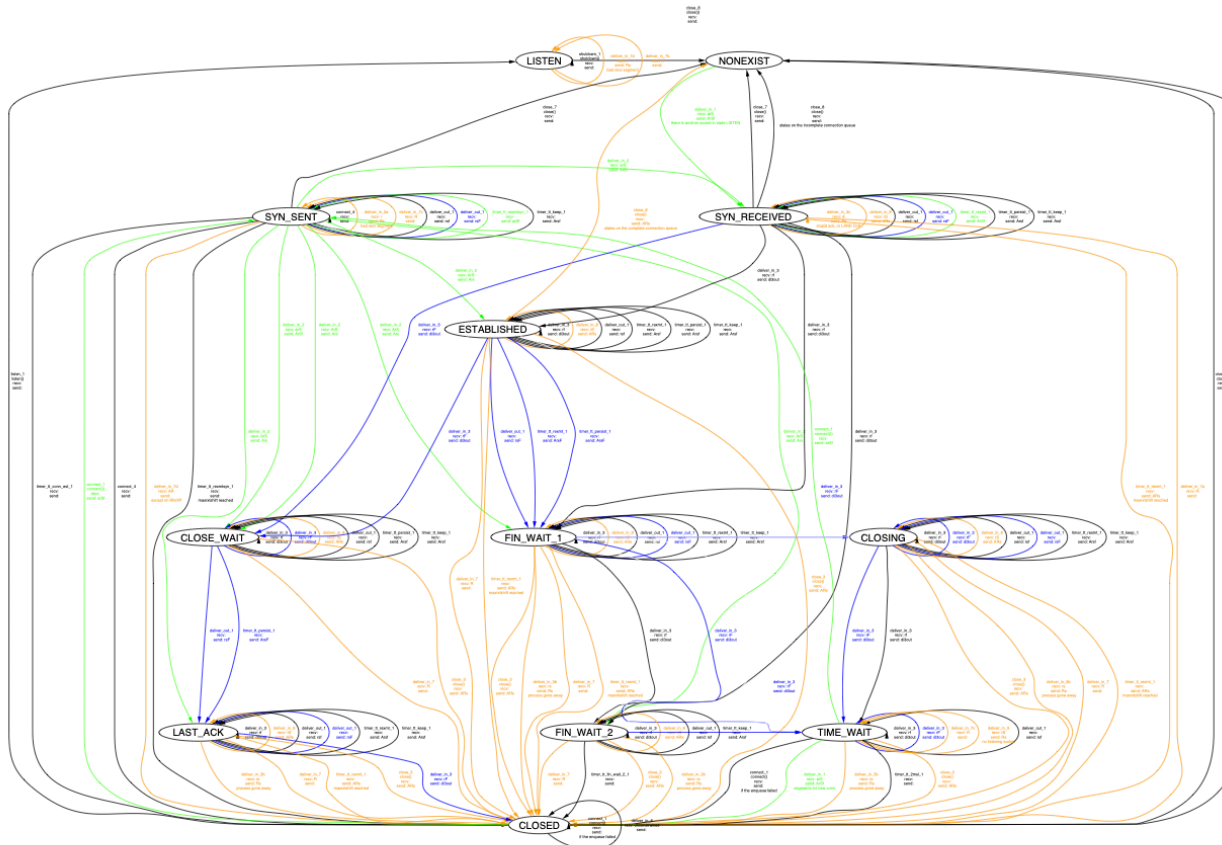
TCP Connection State Diagram
Figure 6.

- V. Cerf, K. Dalal, and C. Sunshine, ***Transmission Control Protocol (version 1)***, INWG General Note #72, December 1974.
- In practice: J. Postel, Ed., ***Transmission Control Protocol: Protocol Specification***, RFC 793, September, 1981.



Compare to Bishop, et al (2005)

TCP: an approximation to the real state diagram



What Is This?

This graph shows an approximation to the Host Transition System of the TCP specification.

TCP: UDP and Sockets: rigorous and experimentally-validated behavioural specification. Volume 1: Overview. Volume 2: The Specification. Steven Bishop, Matthew Fairbairn, Michael Norrish, Peter Sewell, Michael Smith, and Keith Wansbrough. 2005.

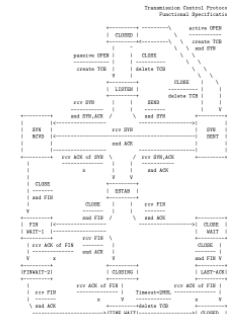
The states are the classic 'TCP states', though note that these are only a tiny part of the protocol endpoint state, in the specification or in implementations. The transitions are an over-approximation to the set of all the transitions in the model which (1) affect the TCP state of a socket, and/or (2) involve processing segments from the host's input queue or adding them to its output queue, except that transitions involving ICMP are omitted, as are transitions arising from the pathological BSD behaviour in which arbitrary sockets can be moved to LISTEN states. Transitions are labelled by their Host LTS rule name (e.g. socket.L, deliver.m.2, etc.), any socket call involved (e.g. close()), and constraints on the flags of any TCP segment received and sent, with e.g. R indicating that RST is set and r indicating RST is clear. Transitions involving segments (either inbound or outbound) with RST set are coloured orange; others that have SYN set are coloured green; others that have FIN set are coloured blue; others are coloured black. The FIN indication includes the case of FINs that are constructed by reassembly rather than appearing in a literal segment.

The graph is based on data extracted manually from the HOL specification. The data does not capture all the invariants of the model, so some disabled transitions may not be reachable in the model (or in practice). Similarly, the constraints on flags shown may be overly weak.

Transition Rules

- listen.0: Successfully creates user of a synchronous socket.
- listen.1: Successfully closes user of the listener to enable it to be CLOSED.
- listen.2: SYN_RECV = SYN_RECEIVED state.
- listen.3: Successfully closes user of the listener to a listening TCP socket.
- listen.4: Begins connection establishment for creating a SYN and trying to respond to each connection.
- listen.5: Full socket has pending error.
- listen.6:1: Partially receives SYN and SYN_ACK.
- listen.6.2: Full connection exists, ready and ready to accept and other processes a RST segment or aborts it. Once the incoming segment of the connection is successfully constructed.
- listen.6.3: Completion of active open (in SYN_RECV receive SYN_ACK and send ACK in pending state) in SYN_RECV receive SYN and send SYN_ACK.
- listen.6.4: Receive host being disrupted and RST is open to SYN_RECV state.
- listen.6.5: Receive SYN, FIN, and ACK in a connected state.
- listen.6.6: Receive data after successful open.
- listen.6.7: Receive input ACK or LAMP_ACK in SYN_RECEIVED state.
- listen.6.8: Receive and drop (possibly) a segment that requires CLOSED state.
- listen.7: Receive RST and set non-CLOSED, LISTEN, SYN_RECV.
- listen.7a: Receive RST and set SYN_RECEIVED state.
- listen.7b: Receive RST and set SYN_RECV state.
- listen.7c: Receive RST and set SYN_RECV (transmission unit) = TIME_WAIT state.
- listen.7d: Receive RST and set SYN_RECV (transmission unit) = FIN_WAIT_1 state.
- listen.7e: Receive RST and set CLOSED, LISTEN, SYN_RECV.
- listen.7f: Receive SYN in TIME_WAIT state if there is no matching LISTEN socket in response number to be received.
- listen.7g: Cannot open TCP socket.
- listen.7h: Successfully set socket to LISTEN state.
- listen.7i: The flow of control to the LISTEN state from any non-connection.1.
- listen.7j: Successfully set socket to LISTEN state.
- listen.7k: The flow of control to the LISTEN state from any non-connection.1.
- listen.7l: Successfully set socket to LISTEN state.
- listen.7m: Successfully set socket to LISTEN state.
- listen.7n: Successfully set socket to LISTEN state.
- listen.7o: Successfully set socket to LISTEN state.
- listen.7p: Successfully set socket to LISTEN state.
- listen.7q: Successfully set socket to LISTEN state.
- listen.7r: Successfully set socket to LISTEN state.
- listen.7s: Successfully set socket to LISTEN state.
- listen.7t: Successfully set socket to LISTEN state.
- listen.7u: Successfully set socket to LISTEN state.
- listen.7v: Successfully set socket to LISTEN state.
- listen.7w: Successfully set socket to LISTEN state.
- listen.7x: Successfully set socket to LISTEN state.
- listen.7y: Successfully set socket to LISTEN state.
- listen.7z: Successfully set socket to LISTEN state.

The RFC793 Original

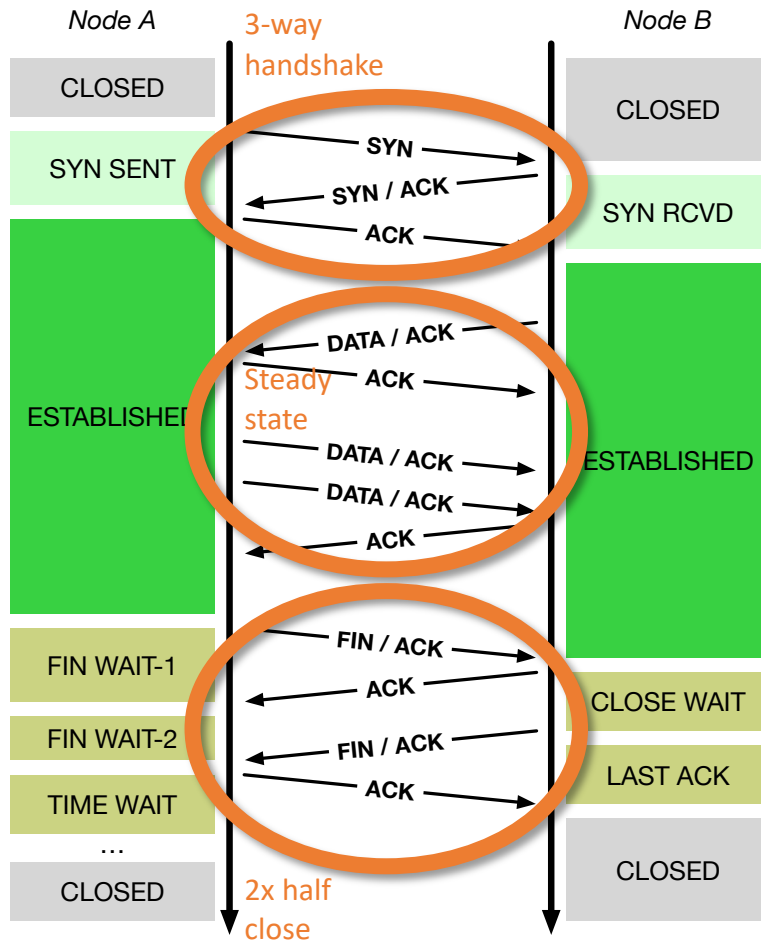


<http://www.cl.cam.ac.uk/users/ps20/Netsem>
March 18, 2005

Steve Bishop, Matthew Fairbairn, Michael Norrish, Peter Sewell, Michael Smith, and Keith Wansbrough. **Rigorous Specification and Conformance Testing Techniques for Network Protocols, as Applied to TCP, UDP, and Sockets.** Proceedings of SIGCOMM 2005, ACM, 2005.



TCP principles and properties

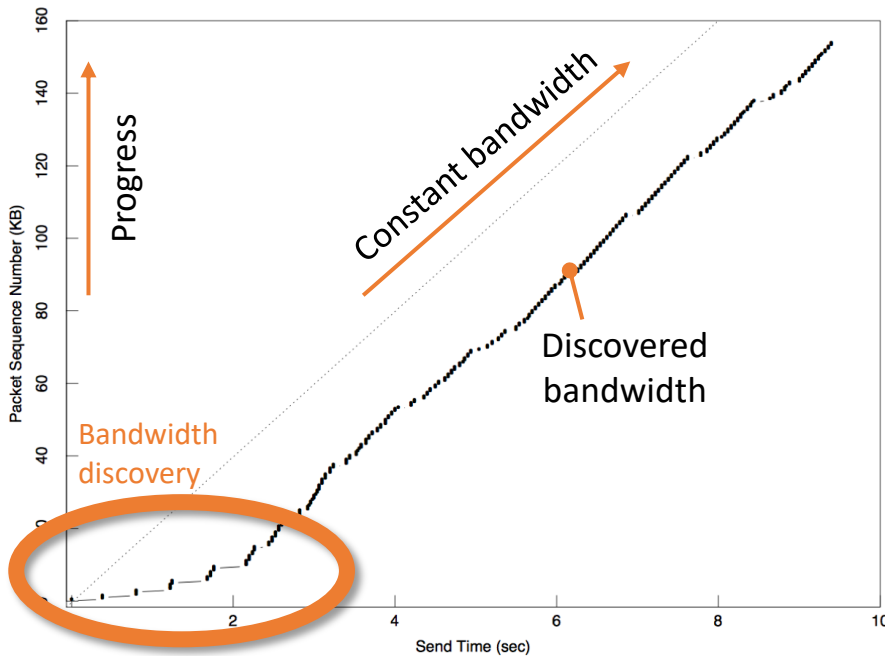


- Assumptions: Network may delay, (reorder), drop, corrupt IP packets
- TCP implements reliable, ordered, stream transport protocol over IP
- Three-way handshake: SYN / SYN-ACK / ACK (mostly!)
- Steady state
 - Sequence numbers ACK'd
 - Round-Trip Time (RTT) measured to time out loss
 - Data retransmitted on loss
 - Flow control via advertised window size in ACKs
 - Congestion control ("fairness") detects congestion via loss (and, recently, via delay: BBR)
- NB: "Half close" allows communications in one direction to end while the other continues



TCP congestion control and avoidance

Figure 4: Startup behavior of TCP with Slow-start

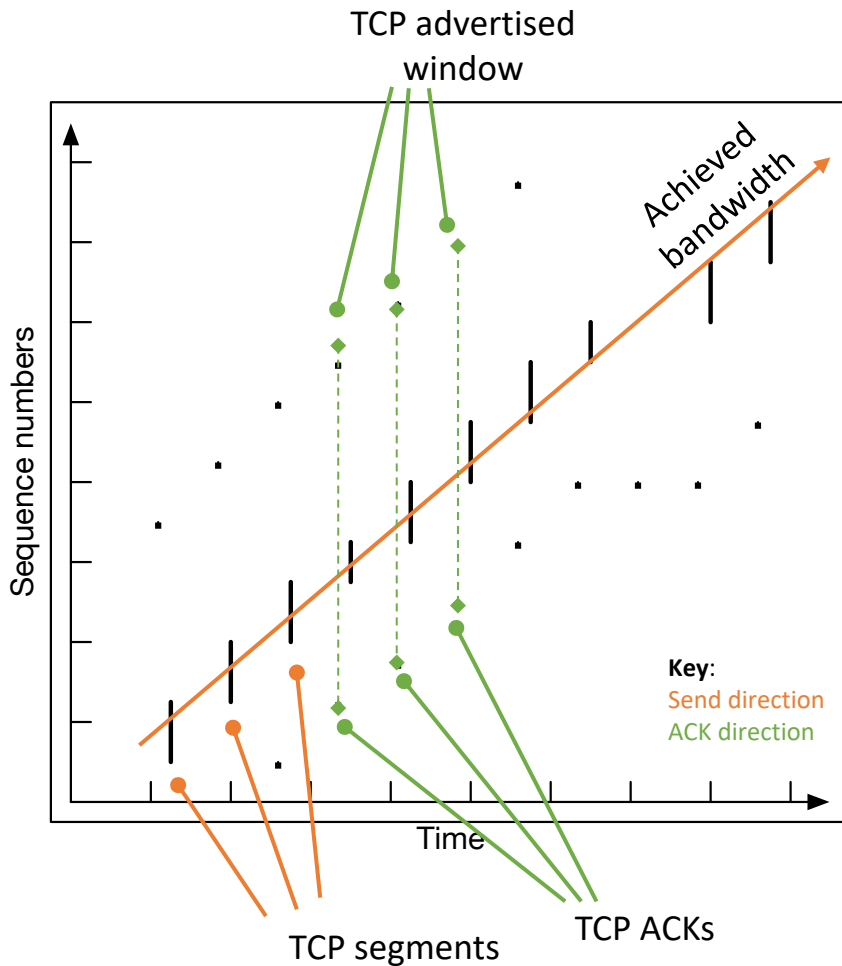


Same conditions as the previous figure (same time of day, same Suns, same network path, same buffer and window sizes), except the machines were running the 4.3⁺ TCP with slow-start. No bandwidth is wasted on retransmits but two seconds is spent on the slow-start so the effective bandwidth of this part of the trace is 16 KBps — two times better than figure 3. (This is slightly misleading: Unlike the previous figure, the slope of the trace is 20 KBps and the effect of the 2 second offset decreases as the trace lengthens. E.g., if this trace had run a minute, the effective bandwidth would have been 19 KBps. The effective bandwidth without slow-start stays at 7 KBps no matter how long the trace.)

- 1986 Internet CC collapse
 - 32Kbps → **40bps**
- Van Jacobson, SIGCOMM 1988
 - Don't send more data than the network can handle!
 - **Conservation of packets** via ACK clocking
 - Exponential retransmit timer, slow start, aggressive receiver ACK, dynamic window sizing on congestion, and (later) ABC
- ECN (RFC 3168), ABC (RFC 3465), Compound (Tan, et al, INFOCOM 2006), Cubic (Rhee and Xu, ACM OSR 2008), BBR (Cardwell, ACM Queue 2016)



TCP time/sequence graphs (Van Jacobson)



- Extracted from TCP packet traces (e.g., via tcpdump)
- Visualize windows, congestion response, buffering, RTT, etc:
 - X: Time
 - Y: Sequence number
- We can extract this data from the network stack directly using DTrace
 - Allows correlation/plotting with respect to other variables / events
 - E.g., TCP and socket-buffer state
- TCP time/sequence diagrams have since been extended to represent additional information
 - E.g., SACK (selective acknowledgement) blocks

