

The Network Stack (2)

Lecture 6, Part 1: TCP

Prof. Robert N. M. Watson

2023-2024

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

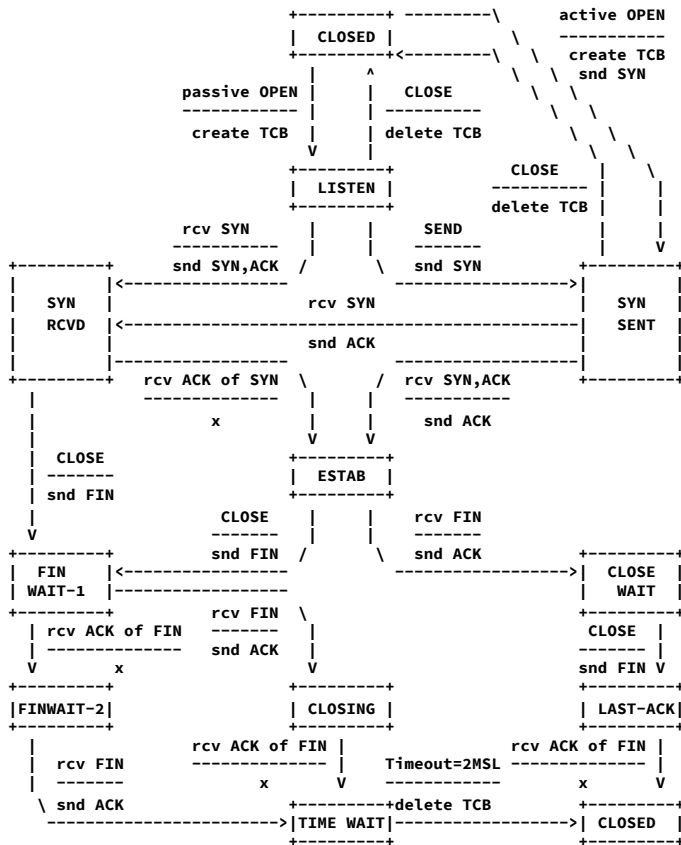


Lecture 6, Part 3

The Transmission Control Protocol (TCP)

September 1981

Transmission Control Protocol
Functional Specification

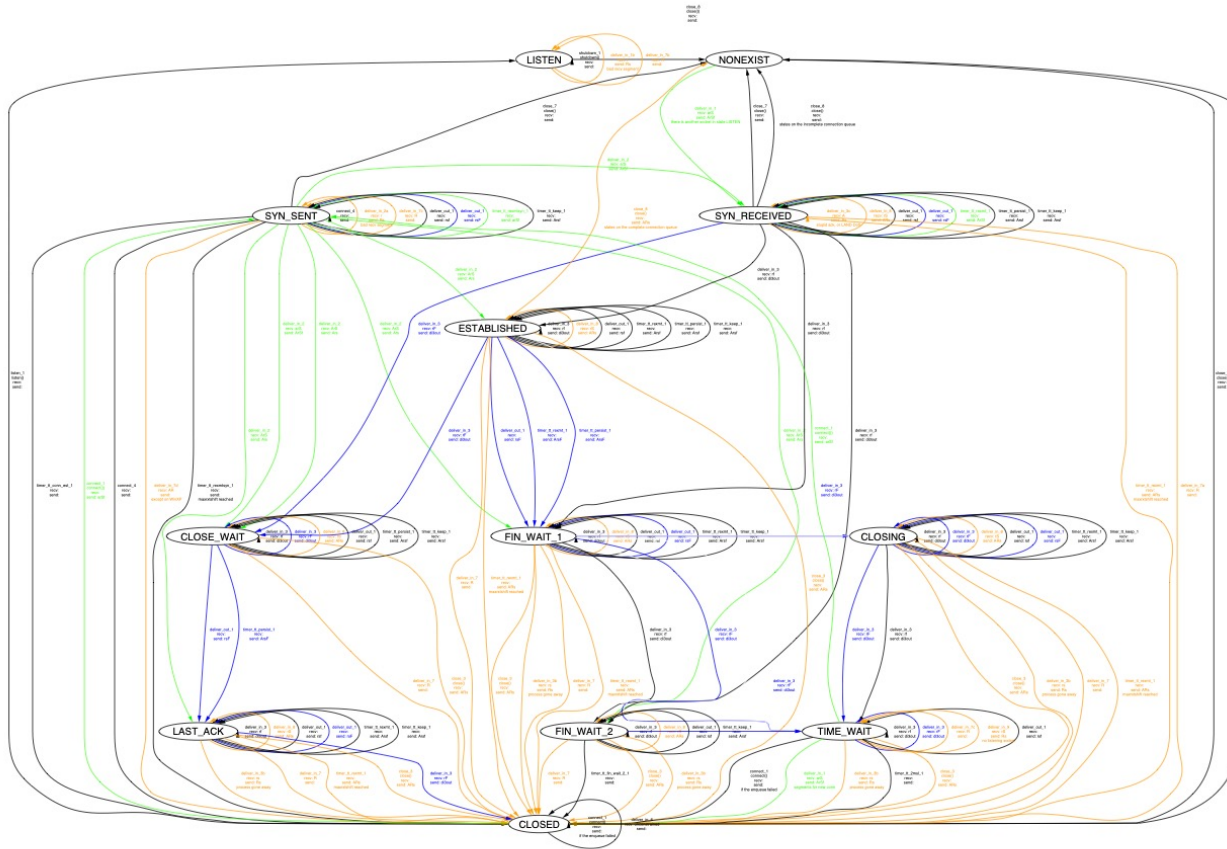


TCP Connection State Diagram
Figure 6.

- 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



<http://www.cl.cam.ac.uk/users/pes20/Netsem>
March 18, 2005

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.

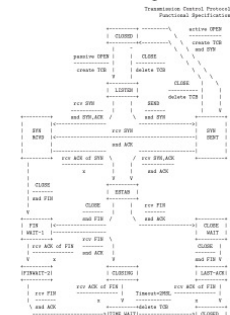
The states are the classic 'TCP states', though note that these are only a tiny part of the protocol endpoint state, in the specification or in implementations. The transitions are an over-approximation to the set of all the transitions in the model which (1) affect the TCP state of a socket, and/or (2) involve processing segments from the host's input queue or adding them to its output queue, except that transitions involving ICMPs 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

- socket.0 Successfully creates a new socket
- socket.1 Successfully closes the host's behaviour to enable the CLOSED state
- socket.2 SYN_RECV → SYN_RECEIVED state
- socket.3 Successfully closes the host's behaviour for a listening TCP socket
- socket.4 Relys connection establishment for creating a SYN and trying to respond to a host connection
- socket.5 Full socket has pending error
- socket.6 Partially receives SYN and SYN_RECV
- socket.7 For incoming SYN, sets and r flag and segment and other properties of EST segment of socket. Once the incoming segment of the current connection is received, constructs a R
- socket.8 Construction of active open (in SYN_RECV) receives SYN_RECV and sends ACK in pending state (in SYN_RECV) receives SYN_RECV and sends SYN_RECV
- socket.9a Receives host is being destroyed and RST is given to SYN_RECV
- socket.9b Receives SYN_RECV and ACK in a connected state
- socket.9c Receives data after connection has gone
- socket.9d Receives and drops (possibly) a host segment that requires CLOSED state
- socket.10a Receives RST and sets non-CLOSED, LISTEN, SYN_RECV
- socket.10b Receives RST and sets SYN_RECEIVED state
- socket.10c Receives RST and sends to SYN_RECV (transmission unit) in TIME_WAIT state
- socket.10d Receives RST and sets SYN_RECV (transmission unit) state
- socket.10e Receives FIN, r, and non-CLOSED, LISTEN, SYN_RECV
- socket.10f Receives SYN in TIME_WAIT state if there is no matching LISTEN state in response to the host's request
- socket.10g Cancels open TCP socket
- socket.10h Successfully sets socket to LISTEN state from any non-CLOSED LISTEN state or CLOSED
- socket.10i The host calls setsockopt to set LISTEN state from any non-CLOSED LISTEN state or CLOSED
- socket.10j Specially states a new SYN descriptor for a host socket
- socket.10k PMS, time expires
- socket.10l connection establishment time expires
- socket.10m setsockopt to set SYN_RECV state expires
- socket.10n send timeout expires
- socket.10o receive timeout expires
- socket.10p SYN timeout time expires

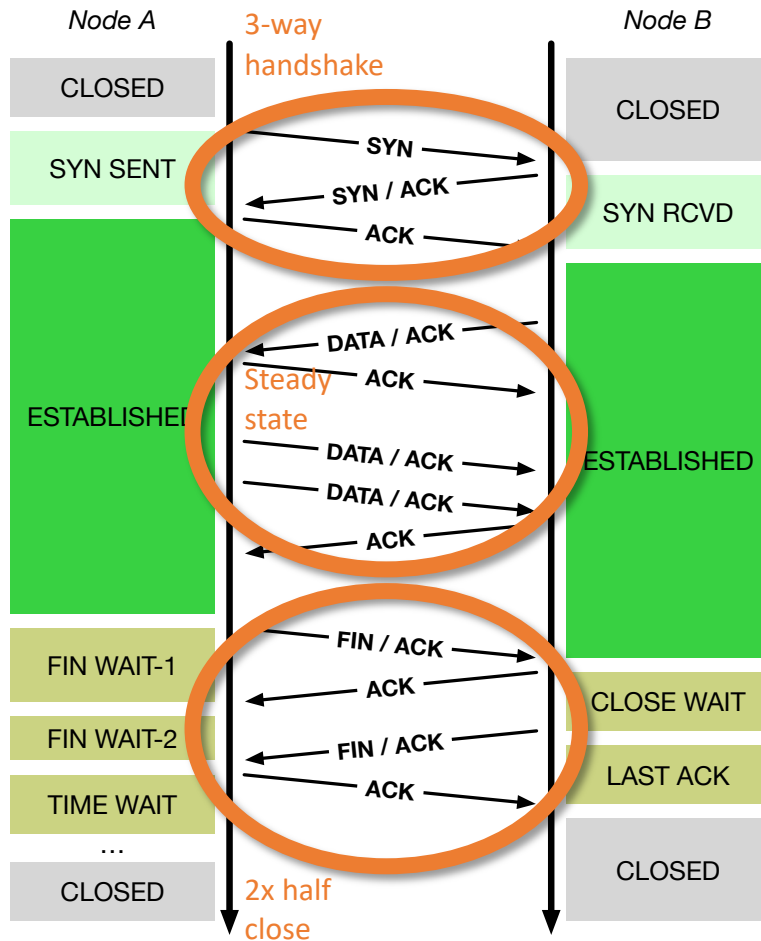
The RFC793 Original



TCP Observation Diagram
Page 6
September 1981

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.

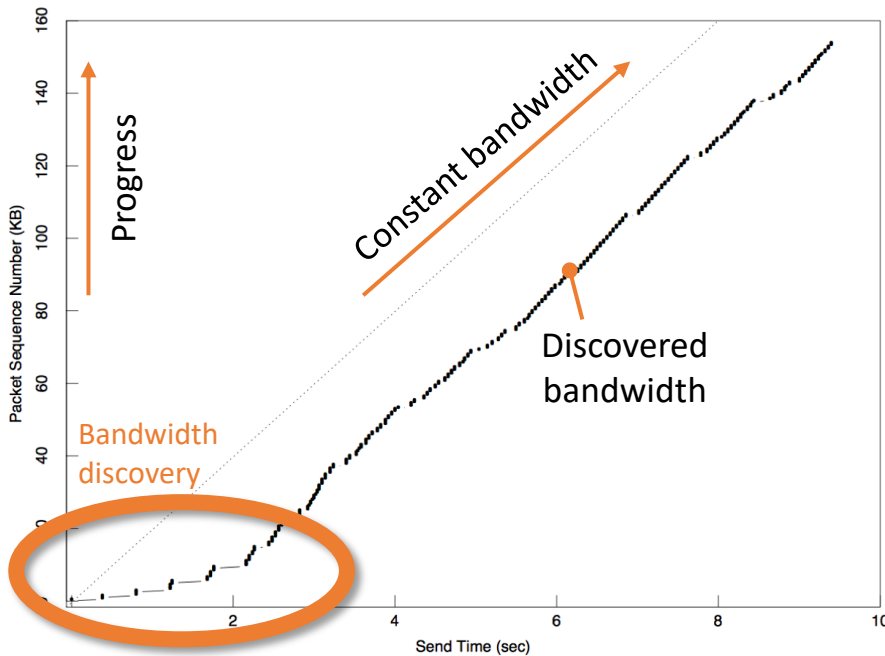
TCP principles and properties



- 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

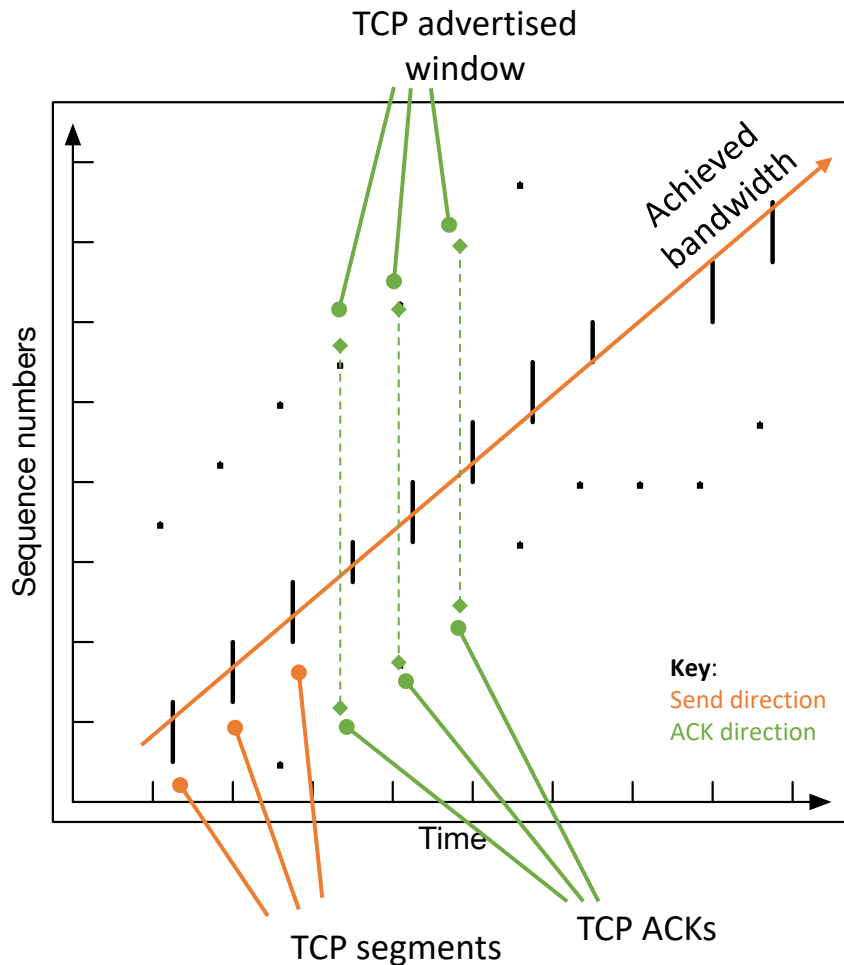
Figure 4: Startup behavior of TCP with Slow-start



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