The Network Stack (2)

L41 Lecture 6

Dr Robert N. M. Watson

27 January 2017

Reminder: Last time

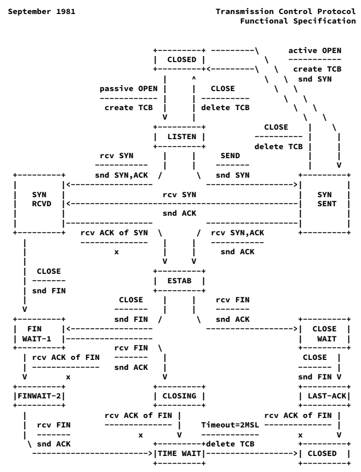
Rapid tour across hardware and software:

- Networking and the sockets API
- Network-stack design principles: 1980s and today
- Memory flow across hardware and software
- Network-stack construction and work flows
- Recent network-stack research

This time: The Network Stack (2)

- The Transmission Control Protocol (TCP)
  - The TCP state machine
  - TCP congestion control
  - TCP implementations and performance
  - The evolving TCP stack
  - Labs 4 + 5 on TCP
- Wrapping up the L41 lecture series

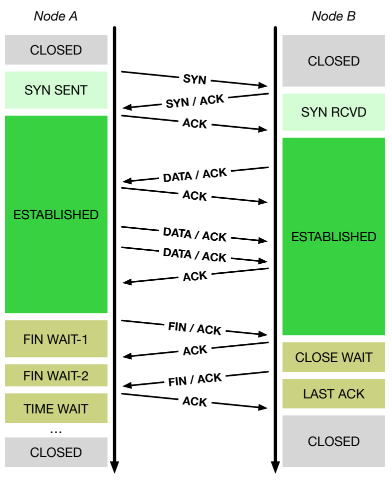
The Transmission Control Protocl (TCP)



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.

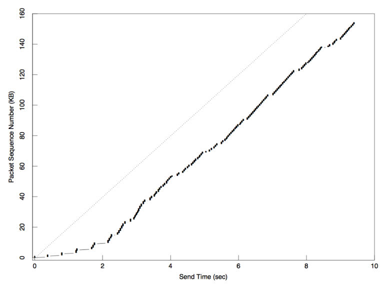
TCP principles and properties



- Network may delay, (reorder), drop, corrupt packets
- TCP: Reliable, ordered, stream transport protocol over IP
  - Three-way handshake: SYN / SYN-ACK / ACK (mostly!)
  - 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

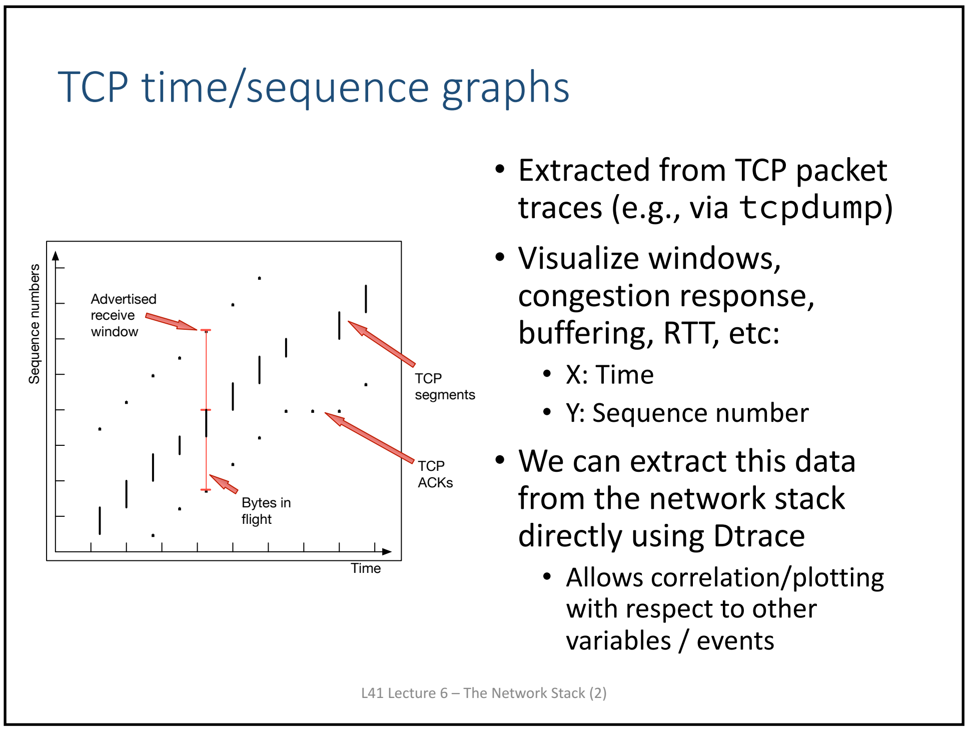
TCP congestion control and avoidance

Figure 4: Startup behavior of TCP with Slow-start

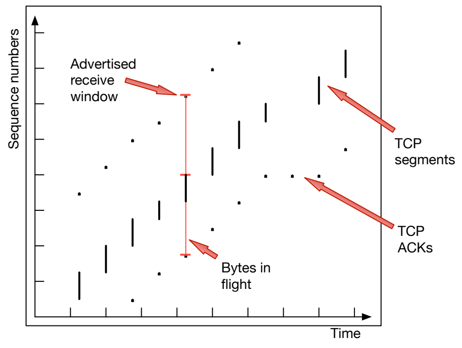


Same conditions as the previous figure (same time of day, same Suns, same network path, same buffer and window sizes), except the machines were running the 4.3^{+}TCP with slow-start. No bandwidth is wasted on retransmits but two seconds is spent on the slow-start so the effective bandwidth of this part 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.)

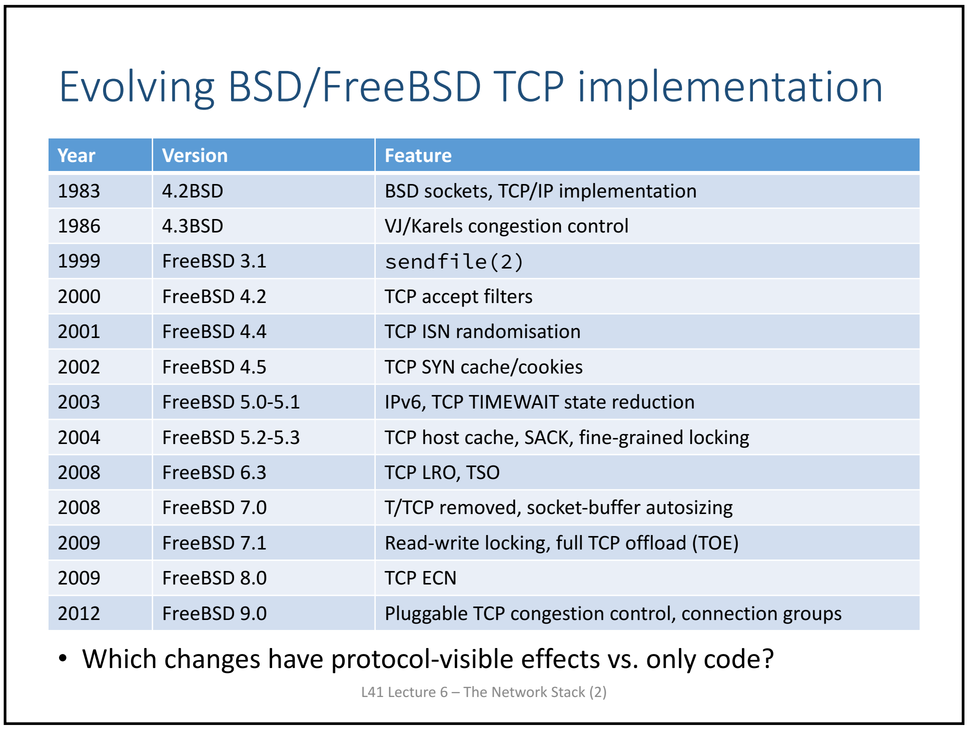
- 1986 Internet CC collapse
  - 32Kbps → **40bps**
- Van Jacobson, SIGCOMM 1988
  - Don't send more data than the network can handle!
  - **Conservation of packets** via ACK clocking
  - Exponential retransmit timer, slow start, aggressive receiver ACK, and dynamic window sizing on congestion
- ECN (RFC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



TCP time/sequence graphs



- 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

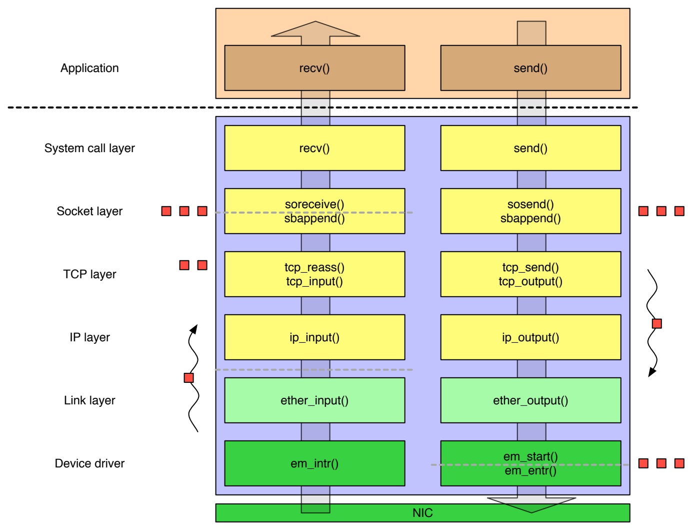


Evolving BSD/FreeBSD TCP implementation

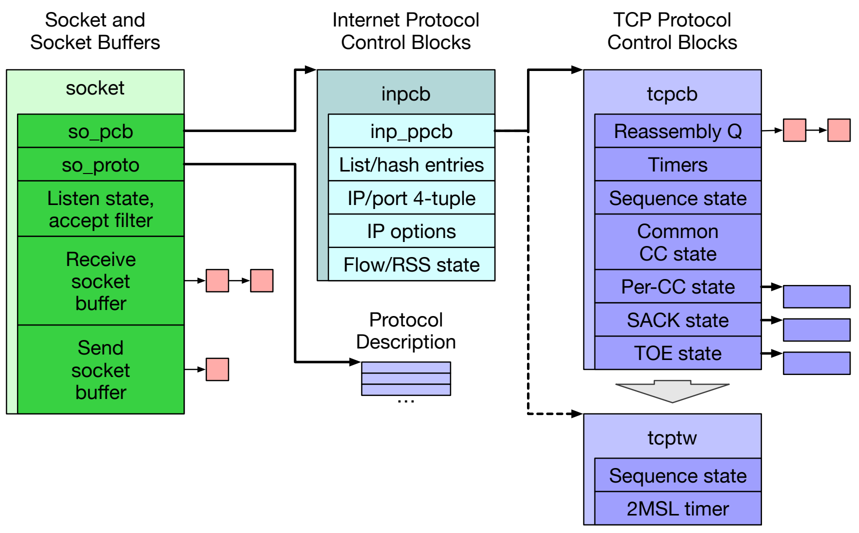
Year	Version	Feature
1983	4.2BSD	BSD sockets, TCP/IP implementation
1986	4.3BSD	VJ/Karels congestion control
1999	FreeBSD 3.1	`sendfile(2)`
2000	FreeBSD 4.2	TCP accept filters
2001	FreeBSD 4.4	TCP ISN randomisation
2002	FreeBSD 4.5	TCP SYN cache/cookies
2003	FreeBSD 5.0-5.1	IPv6, TCP TIMEWAIT state reduction
2004	FreeBSD 5.2-5.3	TCP host cache, SACK, fine-grained locking
2008	FreeBSD 6.3	TCP LRO, TSO
2008	FreeBSD 7.0	T/TCP removed, socket-buffer autosizing
2009	FreeBSD 7.1	Read-write locking, full TCP offload (TOE)
2009	FreeBSD 8.0	TCP ECN
2012	FreeBSD 9.0	Pluggable TCP congestion control, connection groups

- Which changes have protocol-visible effects vs. only code?

Lect. 5 - Send/receive paths in the network stack



Data structures – sockets, control blocks



Denial of Service (DoS) – state minimisation

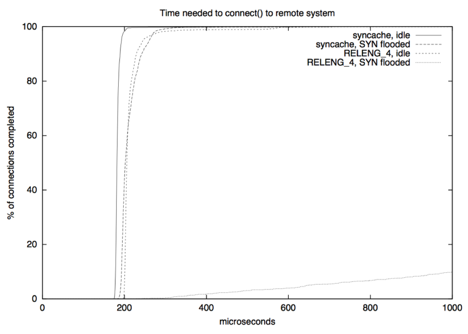
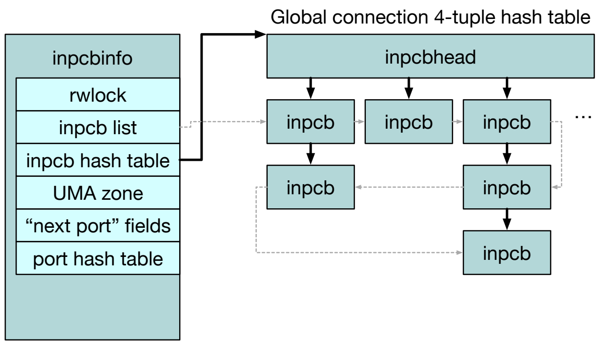


Figure 3: Time needed to connect() to remote system.

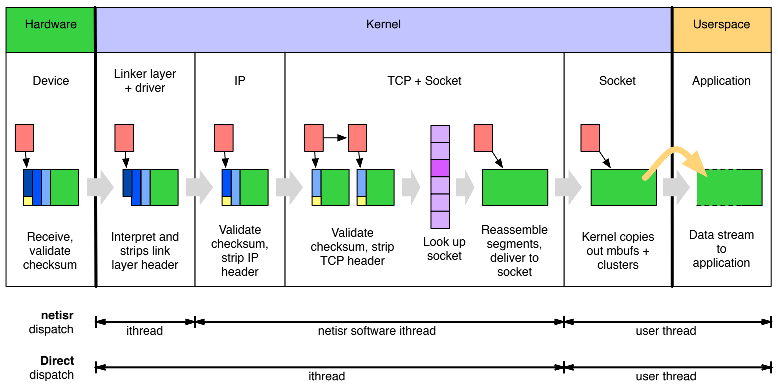
- Yahoo!, Amazon, CNN taken down by SYN floods in February 2000
- D. Borman: **TCP SYN cache** – minimise state for new connections
- D. Bernstein: **SYN cookies** – eliminate state entirely – at a cost
- J. Lemon: **TCP TIMEWAIT reduction** – minimise state during close
- J. Lemon: **TCP TIMEWAIT recycle** – release state early under load

TCP connection lookup tables



- Global list of connections for monitoring (e.g., netstat)
- Connections are installed in a global hash table for lookup
- Separate (similar) hash table for port-number allocations
- Tables protected by global read-write lock as reads dominate
  - New packets are more frequent than new connections

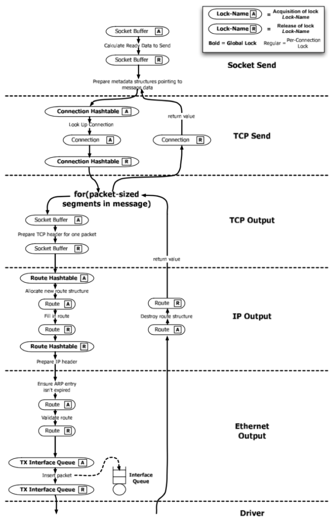
Lect. 5 - Work dispatch: input path


Hardware
Kernel
Userspace
Device
Linker layer + driver
IP
TCP + Socket
Socket
Application
Receive, validate checksum
Interpret and strips link layer header
Validate checksum, strip IP header
Validate checksum, strip TCP header
Look up socket
Reassemble segments, deliver to socket
Kernel copies out mbufs + clusters
Data stream to application
netisr dispatch
ithread
netisr software ithread
user thread
Direct dispatch
ithread
user thread

- **Deferred dispatch**: ithread → netisr thread → user thread
- **Direct dispatch**: ithread → user thread
  - Pros: reduced latency, better cache locality, drop early on overload
  - Cons: reduced parallelism and work placement opportunities

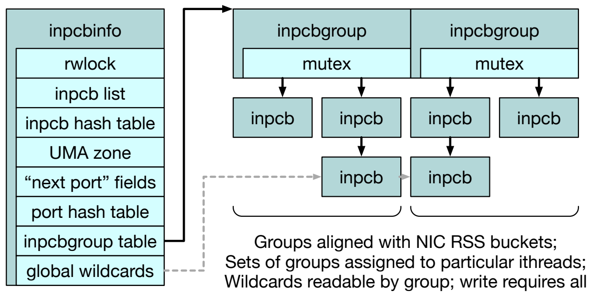
An Evaluation of Network Stack Parallelization Strategies in Modern Operating Systems

Paul Willmann, Scott Rixner, and Alan L. Cox, USENIX ATC, 2006



- Network bandwidth growth > CPU frequency growth
- Locking overhead (space, contention) substantial
  - Getting 'speedup' is hard!
- Evaluate different strategies for TCP processing parallelisation
  - Message-based parallelism
  - Connection-based parallelism (threads)
  - Connection-based parallelism (locks)
- Coalescing locks over connections:
  - reduces overhead
  - increases parallelism

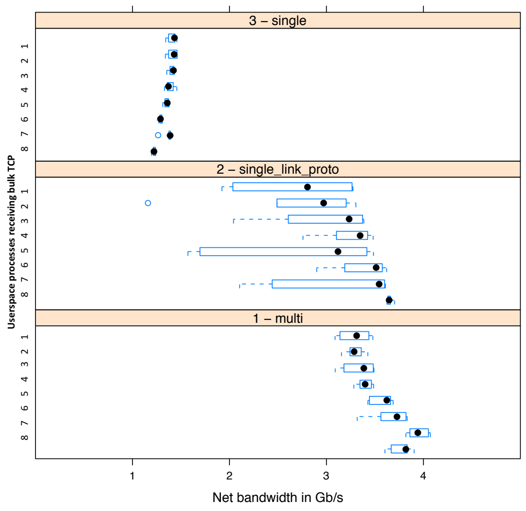
FreeBSD connection groups, RSS



- **Connection groups** blend MsgP and ConnP-L models
  - PCBs assigned to group based on 4-tuple hash
  - Lookup requires group lock, not global lock
  - Global lock retained for 4–tuple reservation (e.g., setup, teardown)
- Problem: have to look at TCP headers (cache lines) to place work!
- Microsoft: NIC **Receive-Side Scaling (RSS)**
  - Multi-queue NICs deliver packets to queues using hash of 4-tuple
  - Align connection groups with RSS buckets / interrupt routing

Performance: dispatch model and locking

Varying dispatch strategy – bandwidth



- 2010 8-core x86 multicore server
- TCP LRO disabled (maximise PPS)
- Configurations:
  - 1 queue (no dispatch), 1 thread on 1 core
  - 1 queue (SW dispatch), 8 threads on 8 cores
  - 8 queues (HW dispatch), 8 threads on 8 cores

Architectural → micro-architectural + I/O optimisation

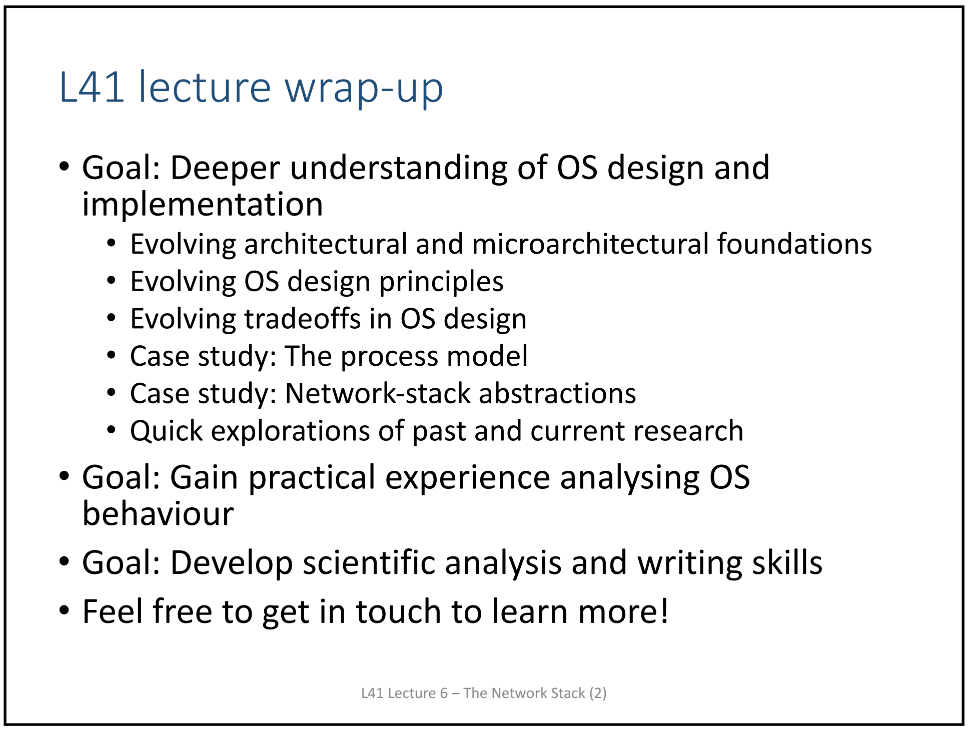
- Hardware, software, protocol co-design causes change to optimisation approach over time:
  - Counting instructions → counting cache misses
  - Reducing lock contention → cache-line contention
  - Adding locking → identifying new parallelism

  - Work ordering, classification, and distribution
  - Vertically integrated distribution and affinity

  - NIC offload of further protocol layers, crypto
  - DMA/cache interactions
- Convergence of networking and storage technologies?

Labs 4 + 5: TCP

- From abstract to concrete understanding of TCP
  - Use tools such as `tcpdump` and `DUMMYNET`
  - Explore effects of latency on TCP performance
- Lab 4 – TCP state machine and latency
  - Measure the TCP state machine in practice
  - Start looking at TCP latency vs. bandwidth (`DUMMYNET`)
  - At what transfer sizes are different latencies masked?
- Lab 5 – TCP congestion control
  - Draw time-sequence-number diagrams
  - Explore OS buffering strategies
  - Explore slow-start vs. steady state as latency changes
  - Explore OS and microarchitectural performance interactions



L41 lecture wrap-up

- Goal: Deeper understanding of OS design and implementation
  - Evolving architectural and microarchitectural foundations
  - Evolving OS design principles
  - Evolving tradeoffs in OS design
  - Case study: The process model
  - Case study: Network-stack abstractions
  - Quick explorations of past and current research
- Goal: Gain practical experience analysing OS behaviour
- Goal: Develop scientific analysis and writing skills
- Feel free to get in touch to learn more!