The Network Stack (2)

Lecture 6, Part 2: TCP Implementation

Prof. Robert N. M. Watson

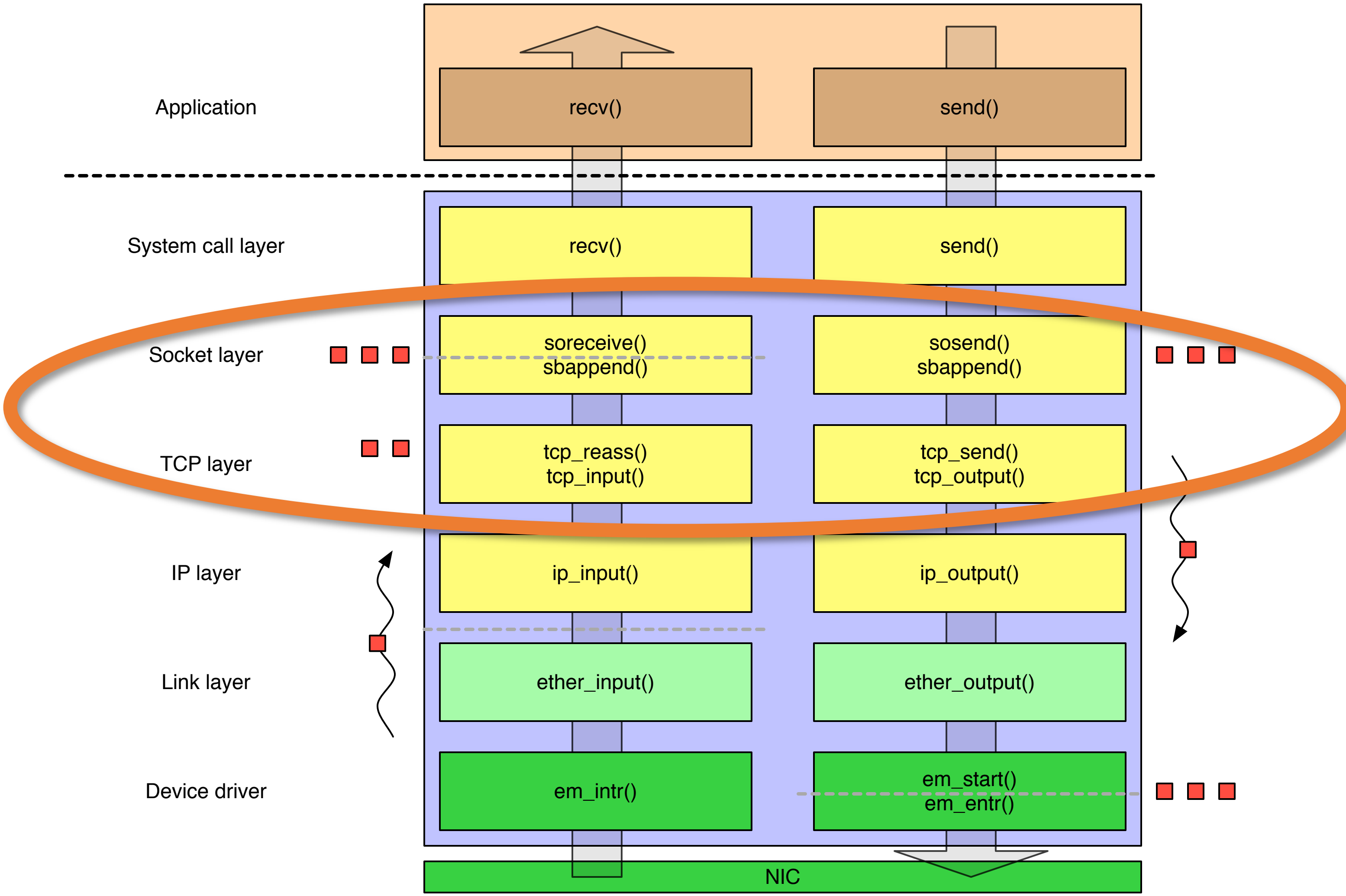
2022-2023

Evolving BSD/FreeBSD TCP implementation

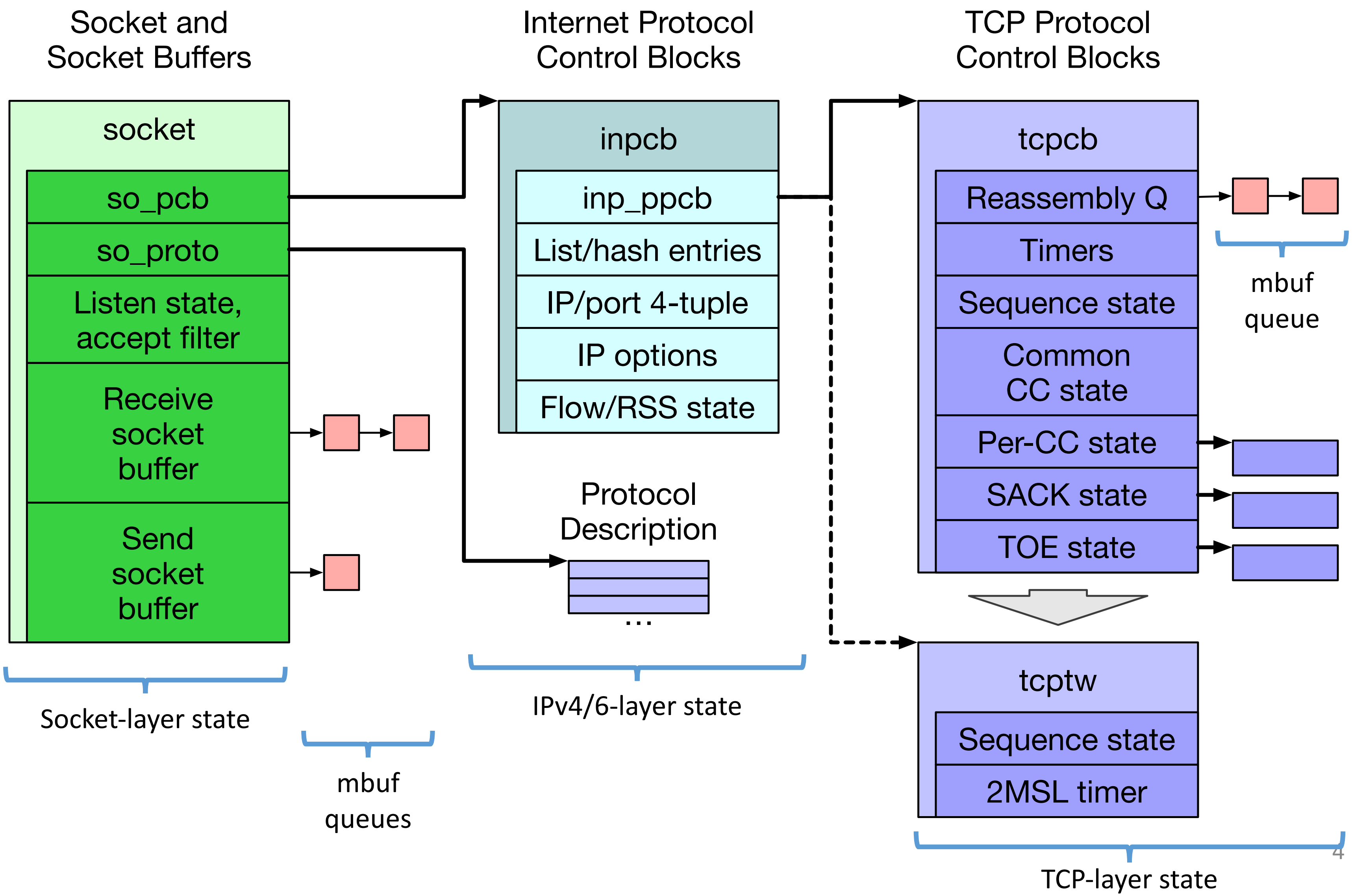
Year	Version	Feature	
1983	4.2BSD	BSD sockets, TCP/IP implementation	Initial TCP R&D
1986	4.3BSD	VJ/Karels congestion control	
1999	FreeBSD 3.1	`sendfile(2)`	The commercial Internet and web hosting
2000	FreeBSD 4.2	TCP accept filters	
2001	FreeBSD 4.4	TCP ISN randomisation	
2002	FreeBSD 4.5	TCP SYN cache/cookies	
2003	FreeBSD 5.0-5.1	IPv6, TCP TIMEWAIT state reduction	
2004	FreeBSD 5.2-5.3	TCP host cache, SACK, fine-grained locking	Multicore, 10gbps, …
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	More CC, more SMP
2012	FreeBSD 9.0	Pluggable TCP congestion control, connection groups	

- … changes continue to this day … BBR, RCU, pluggable TCP, KTLS, …
- Which changes have protocol-visible effects vs. only code?

Reminder: Send/receive paths in the network stack



Data structures – sockets, control blocks



Denial of Service (DoS) – state minimisation

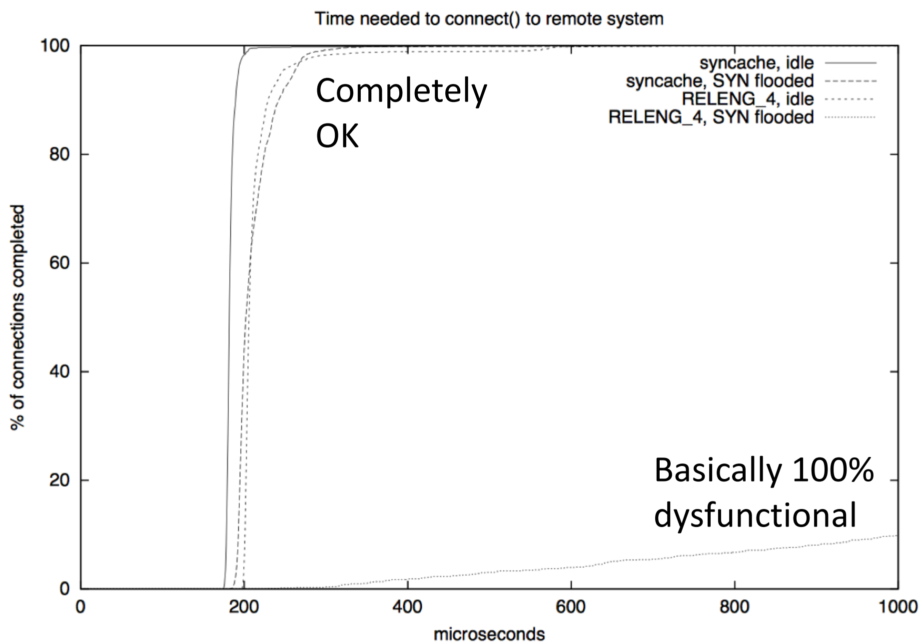
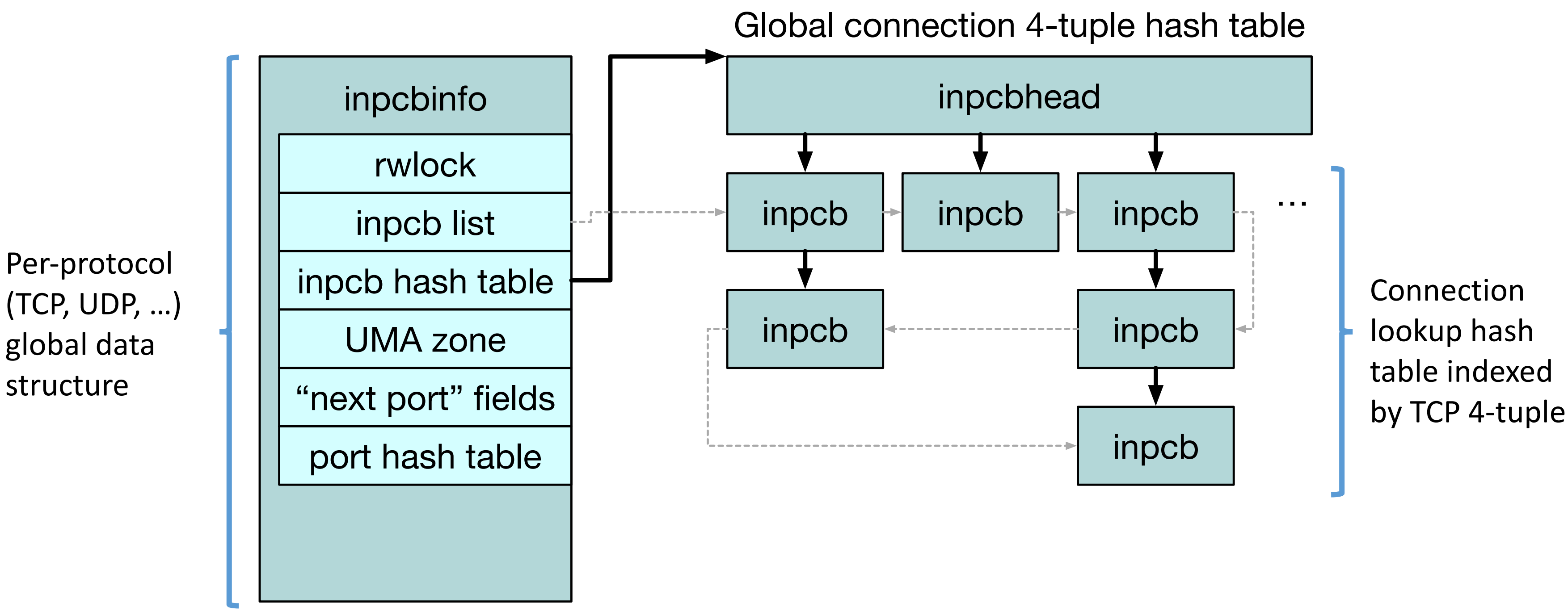


Figure 3: Time needed to connect() to remote system.

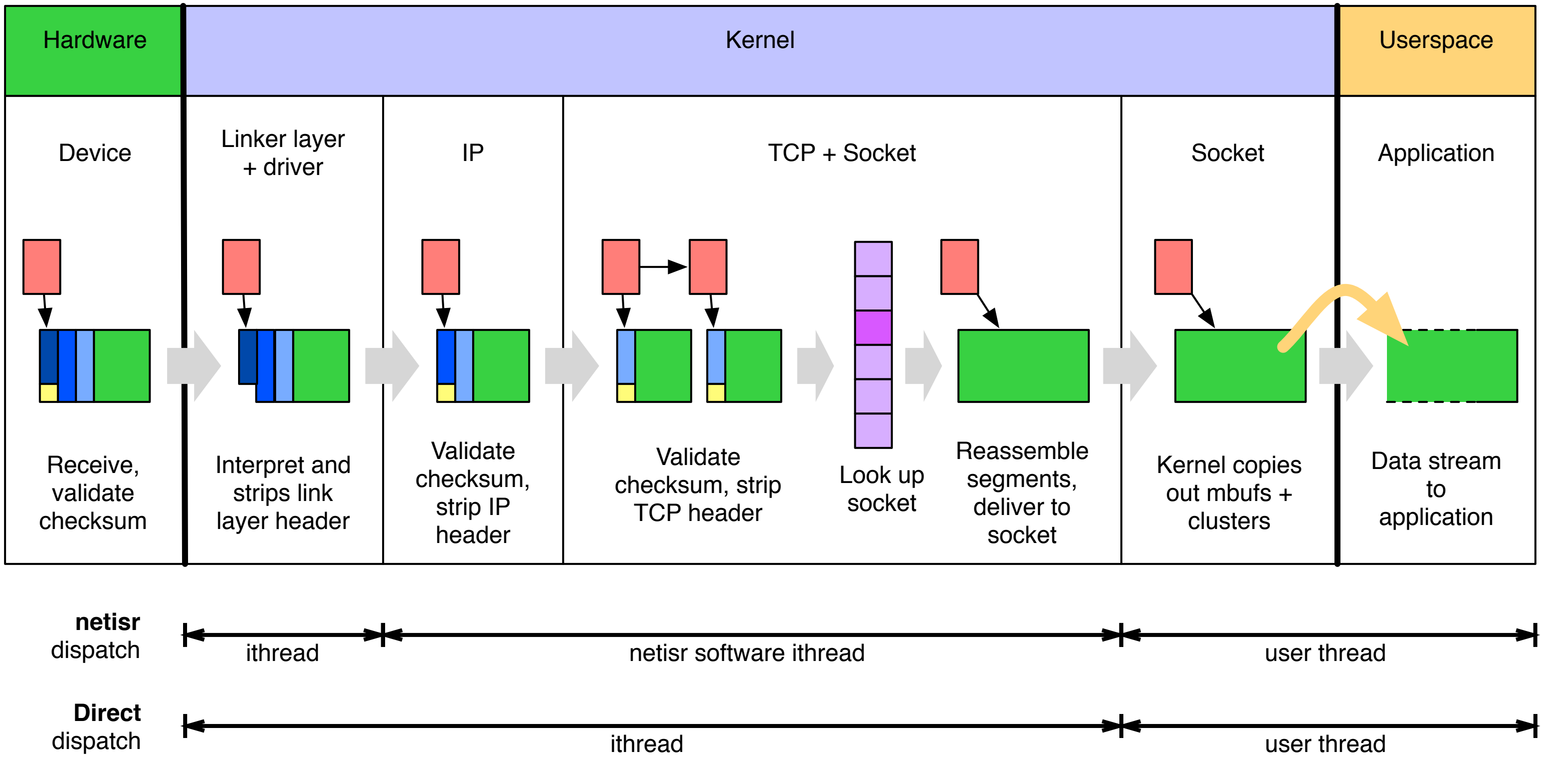
- Yahoo!, Amazon, CNN taken down by SYN floods in February 2000
- Attackers exploit automatic state allocation to overload servers
  - TCP state itself
  - Underlying routing state
  - Cost of walking data structures
- Attackers spoof SYN packets with random source addresses
  - IPv4 address use is sparse, so no RST
- 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 long close sequences (e.g., 2MSL)
- J. Lemon: **TCP TIMEWAIT recycle** – release state early under load

TCP connection lookup tables (original BSD)



- Global list of connections for monitoring (e.g., netstat)
- Connections are installed in a global hash table for lookup
  - NB: separate (similar) hash table for 2-tuple port-number allocations
- Tables protected by global read-write lock as reads dominate
  - New packets are more frequent than new connections

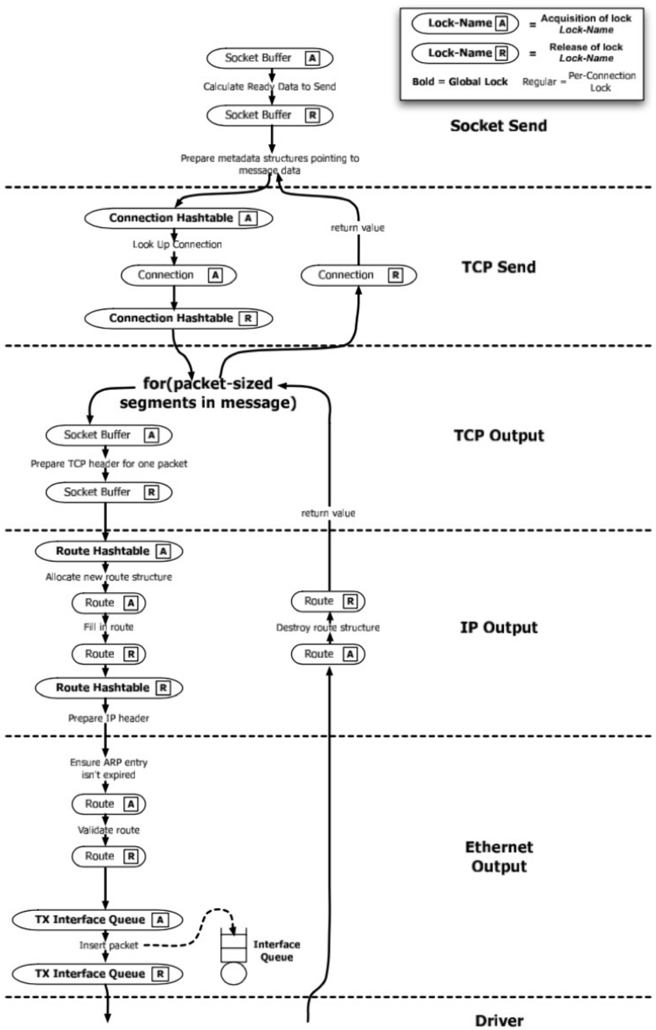
Reminder - Work dispatch: input path



- **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

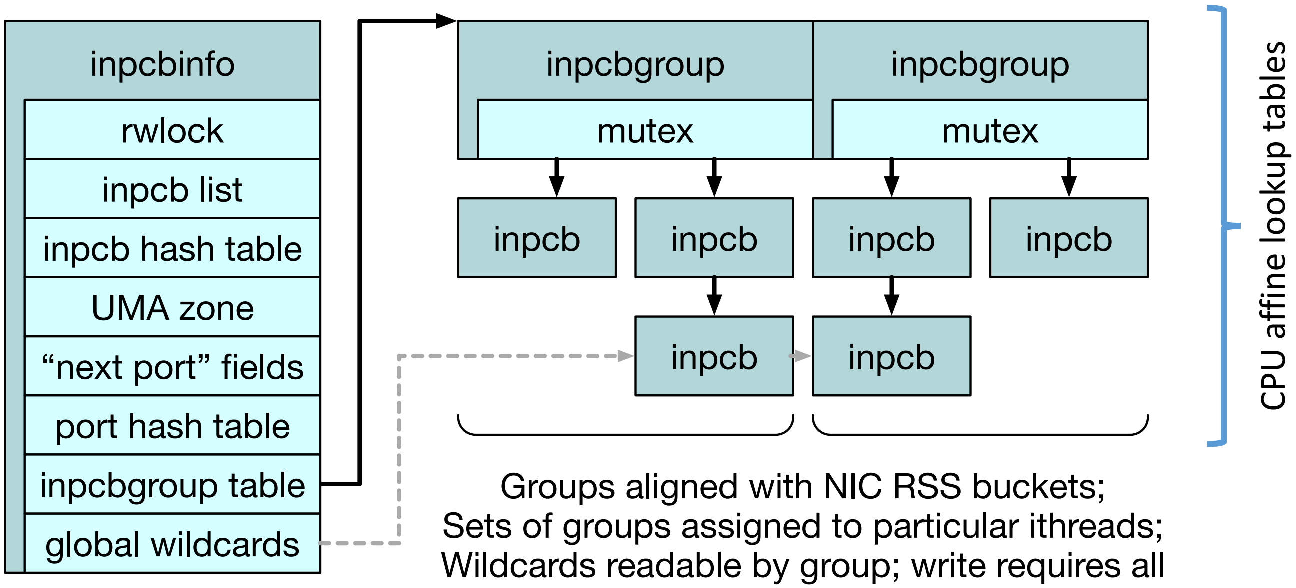
An Evaluation of Network Stack Parallelization Strategies in Modern Operating Systems

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

Connection groups, RSS (FreeBSD)



- From FreeBSD 9.x: **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 Steering (RSS)**
  - Multi-queue NICs deliver packets to queues using hash of 4-tuple
  - Align connection groups with RSS buckets / interrupt routing
- From FreeBSD 12.x: **Read-Copy-Update (RCU)** rather than RW locks protect lists