The Network Stack (1)

L41 Lecture 5

Dr Robert N. M. Watson

2019-2020

This time: Introduction to Network Stacks

Rapid tour across hardware and software:

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

Networking: A key OS function (1)

- Communication between computer systems
  - **Local-Area Networks (LANs)**
  - **Wide-Area Networks (WANs)**
- A network stack provides:
  - Sockets API and extensions
  - Interoperable, feature-rich, high-performance protocol implementations (e.g., IPv4, IPv6, ICMP, UDP, TCP, SCTP, 802.1, 802.11, …)
  - Security functions (e.g., cryptographic tunneling, firewalls...)
  - Device drivers for Network Interface Cards (NICs)
  - Monitoring and management interfaces (BPF, `ioctl`)
  - Plethora of support libraries (e.g., DNS)

Networking: A key OS function (2)

- Dramatic changes over 30 years:

  1980s: Early packet-switched networks, UDP+TCP/IP, Ethernet

  1990s: Large-scale migration to IP; Ethernet VLANs

  2000s: 1-Gigabit, then 10-Gigabit Ethernet; 802.11; GSM data

  2010s: Deployment of IPv6; 40/100-Gbps Ethernet; 3G to 5G;
  ... billions → trillions of devices?

- Vanishing technologies
  - UUCP, IPX/SPX, ATM, token ring, SLIP, ...

The Berkeley Sockets API (1983)

```
close()
read()
write()
...

accept()
bind()
connect()
getsockopt()
listen()
recv()
select()
send()
setsockopt()
socket()
...
```

- **The Design and Implementation of the 4.3BSD Operating System**
  - (but APIs/code first appeared in 4.2BSD)
- Now universal TCP/IP (POSIX, Windows)
- Kernel-resident network stack serves networking applications via system calls
- Reuses file-descriptor abstraction
  - Same API for local and distributed IPC
  - Simple, synchronous, copying semantics
  - Blocking/non-blocking I/O, `select()`
- Multi-protocol (e.g., IPv4, IPv6, ISO, …)
  - TCP-focused but not TCP-specific
  - Cross-protocol abstractions and libraries
  - Protocol-specific implementations
  - “Portable” applications

BSD network-stack principles (1980s-1990s)

Multi-protocol, packet-oriented network research framework:

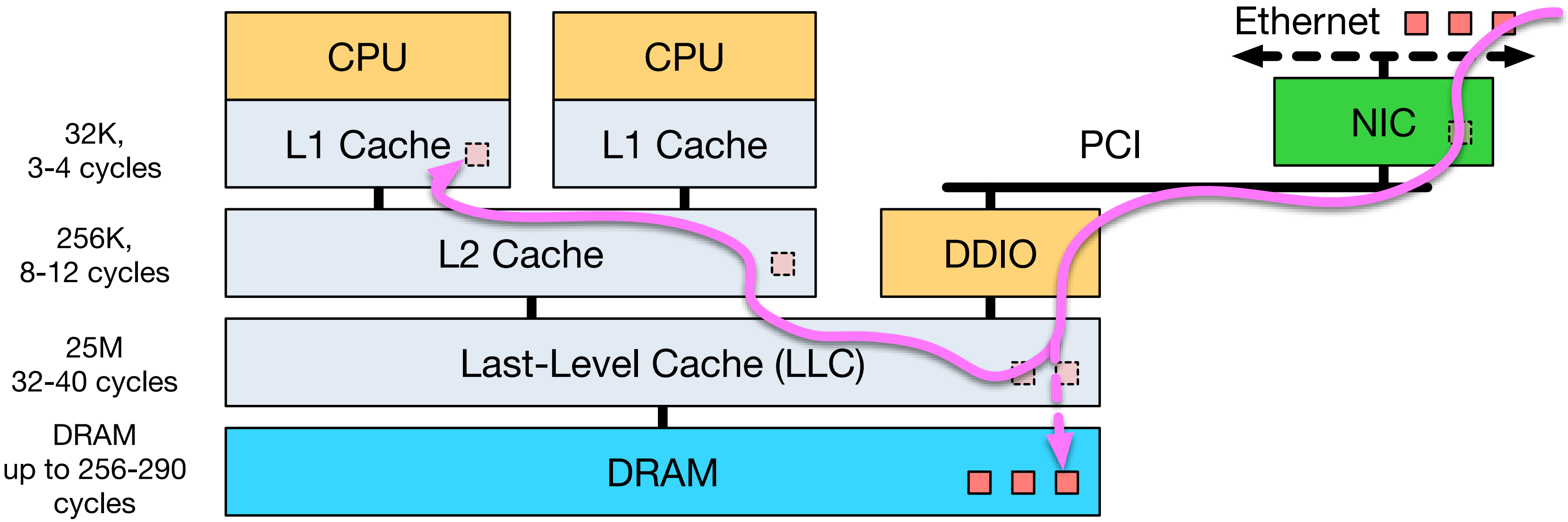
- **Object-oriented**: multiple protocols, socket types, but one API
  - **Protocol-independent**: streams vs. datagrams, sockets, socket buffers, socket addresses, network interfaces, routing table, packets
  - **Protocol-specific**: connection lists, address/routing specialization, routing, transport protocol itself – encapsulation, decapsulation, etc.
- **Packet-oriented**:
  - Packets and packet queueing as fundamental primitives
  - Best effort: If there is a failure (overload, corruption), drop the packet
  - Work hard to maintain packet source ordering
  - Differentiate ‘receive’ from ‘deliver’ and ‘send’ from ‘transmit’
  - Heavy focus on TCP functionality and performance
  - Middle-node (forwarding), not just edge-node (I/O), functionality
  - High-performance packet capture: Berkeley Packet Filter (BPF)

FreeBSD network-stack principles (1990s-2010s)

All of the 1980s features and also …

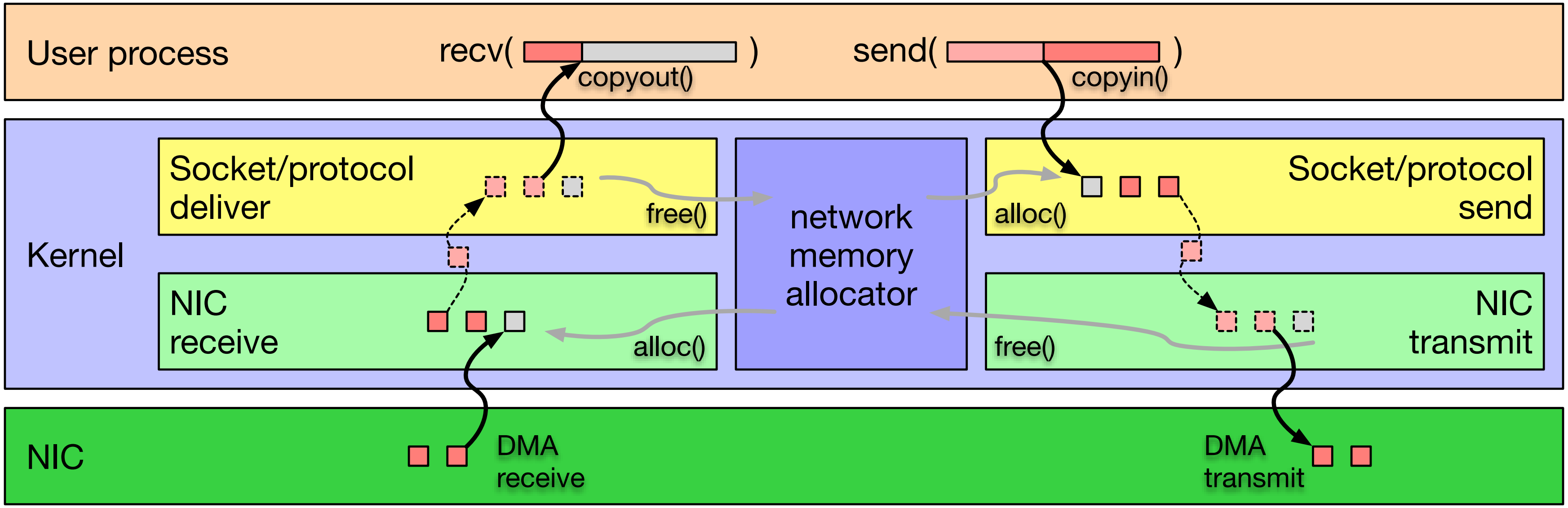
- **Hardware**:
  - Multi-processor scalability
  - NIC offload features (checksums, TSO/LRO, full TCP)
  - Multi-queue network cards with load balancing/flow direction
  - Performance to 10s or 100s of Gigabit/s
  - Wireless networking
- **Protocols**:
  - Dual IPv4/IPv6
  - Pluggable congestion control, delay-based congestion control (BBR)
  - Security/privacy: firewalls, IPSec, ...
- **Software model**:
  - Flexible memory model integrates with VM for zero-copy
  - Fine-grained locking and lockless algorithms (e.g., RCU)
  - Network-stack virtualisation
  - Userspace networking via netmap

Memory flow in hardware



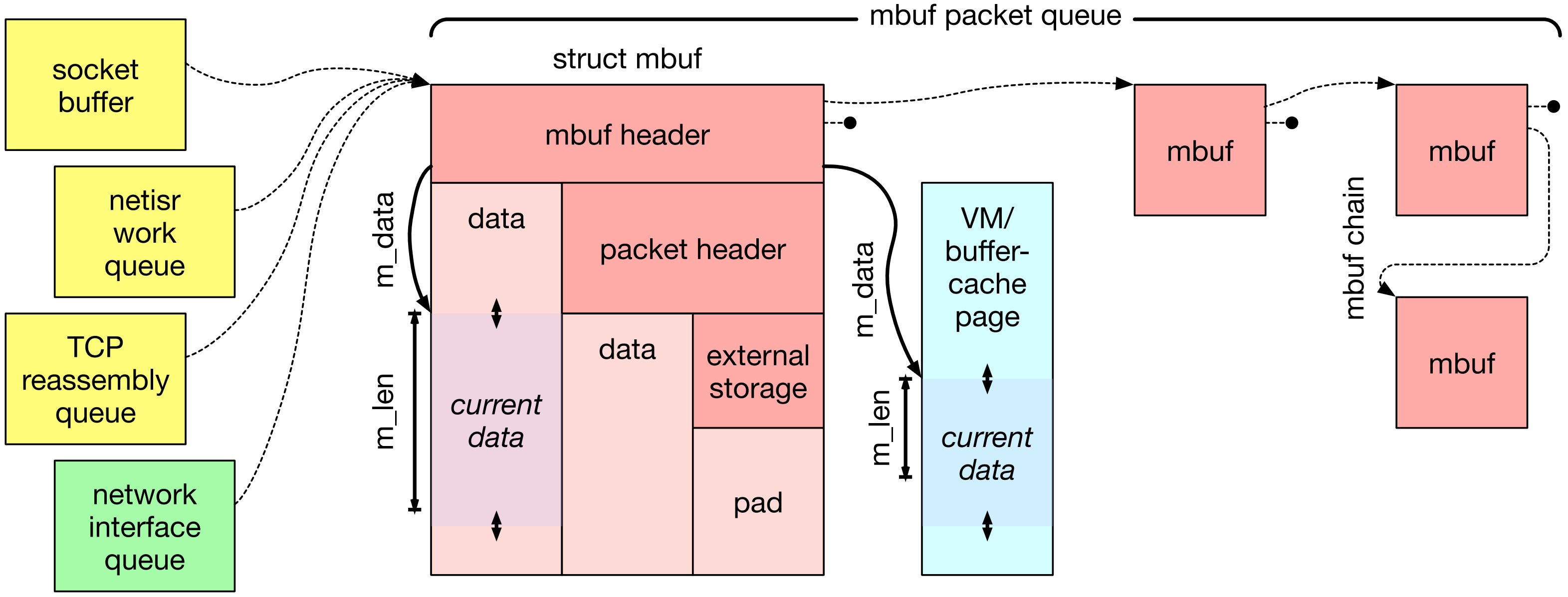
- Key idea: **follow the memory**
  - Historically, memory copying is avoided due to **instruction count**
  - Today, memory copying is avoided due to **cache footprint**
- Recent Intel CPUs push and pull DMA via the LLC (“DDIO”)
  - If we differentiate ‘send’ and ‘transmit’, ‘receive’ vs. ‘deliver’, is this a good idea?
  - … it depends on the latency between DMA and processing

Memory flow in software



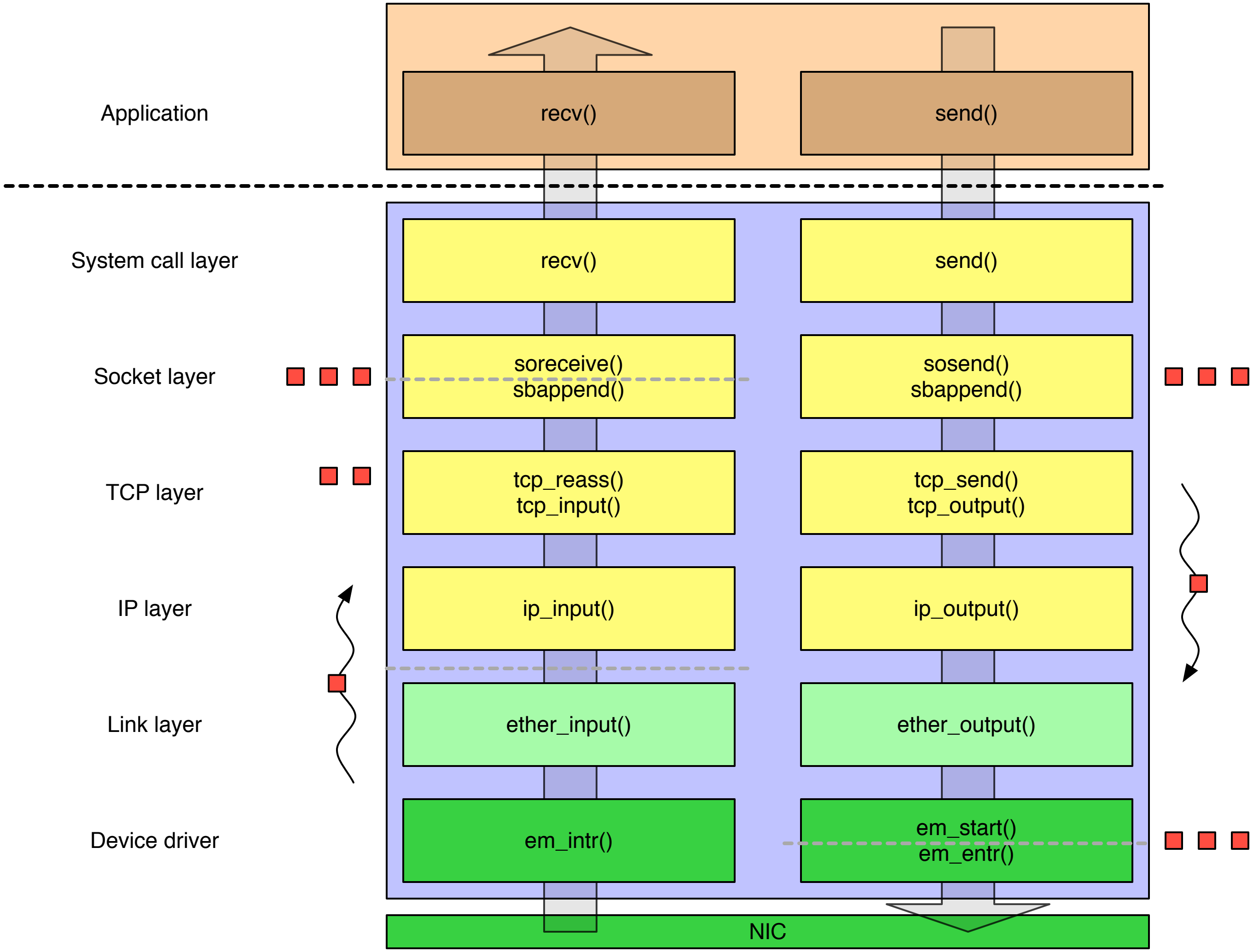
- Socket API implies **one software-driven copy** to/from user memory
  - Historically, zero-copy VM tricks for socket API ineffective
- Network buffers cycle through the slab allocator
  - Receive: allocate in NIC driver, free in socket layer
  - Transmit: allocate in socket layer, free in NIC driver
- **DMA performs second copy**; can affect cache/memory bandwidth
  - NB: what if packet-buffer working set is larger than the cache?

The mbuf abstraction



- Unit of **work allocation and distribution** throughout the stack
- `mbuf` chains represent in-flight packets, streams, etc.
  - Operations: alloc, free, prepend, append, truncate, enqueue, dequeue
  - Internal or external data buffer (e.g., VM page)
  - Reflects bi-modal packet-size distribution (e.g., TCP ACKs vs data)
- Similar structures in other OSes – e.g., `skbuff` in Linux

Send/receive paths in the network stack



Application

recv()

send()

System call layer

recv()

send()

Socket layer

soreceive()
sbappend()

sosend()
sbappend()

TCP layer

tcp_reass()
tcp_input()

tcp_send()
tcp_output()

IP layer

ip_input()

ip_output()

Link layer

ether_input()

ether_output()

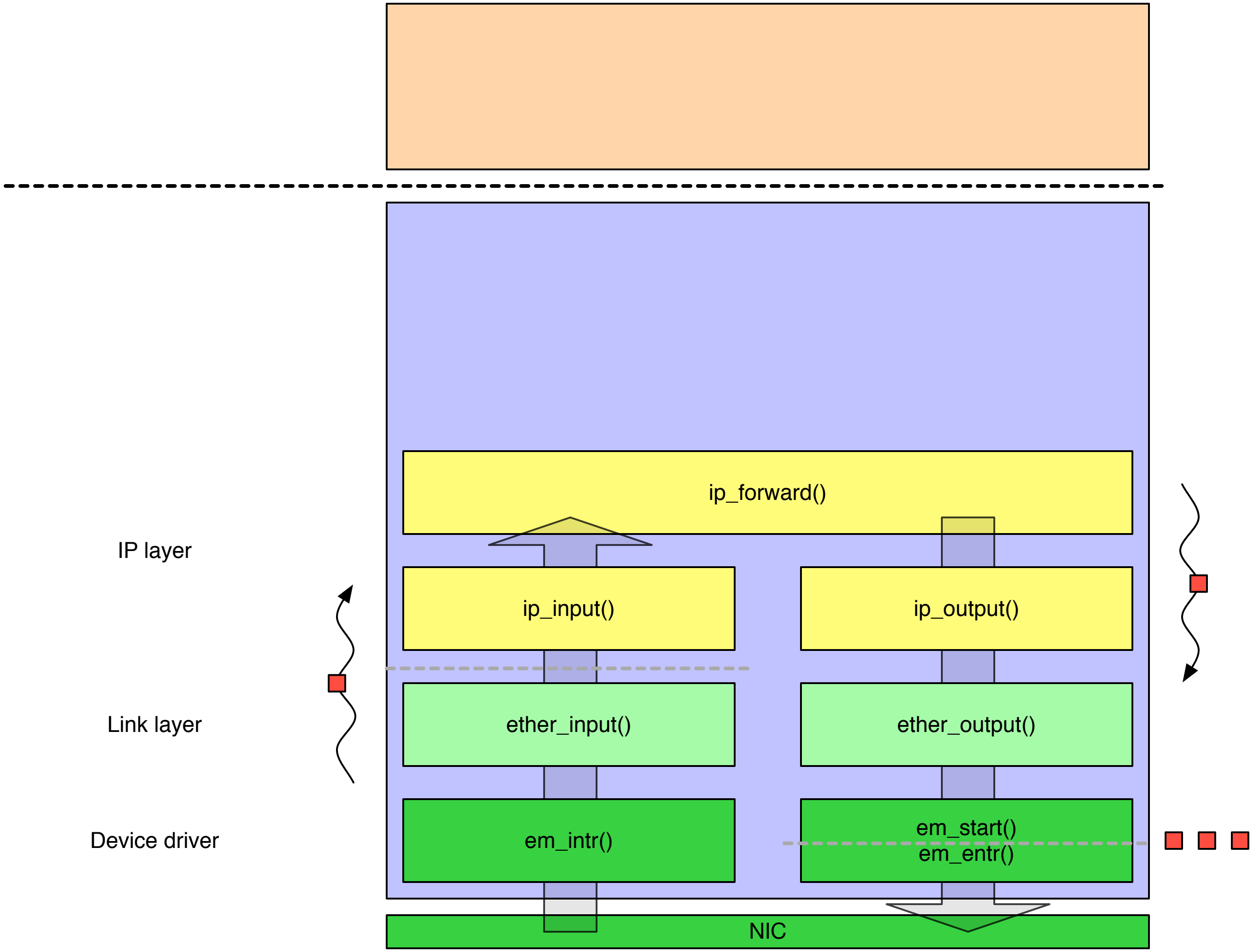
Device driver

em_intr()

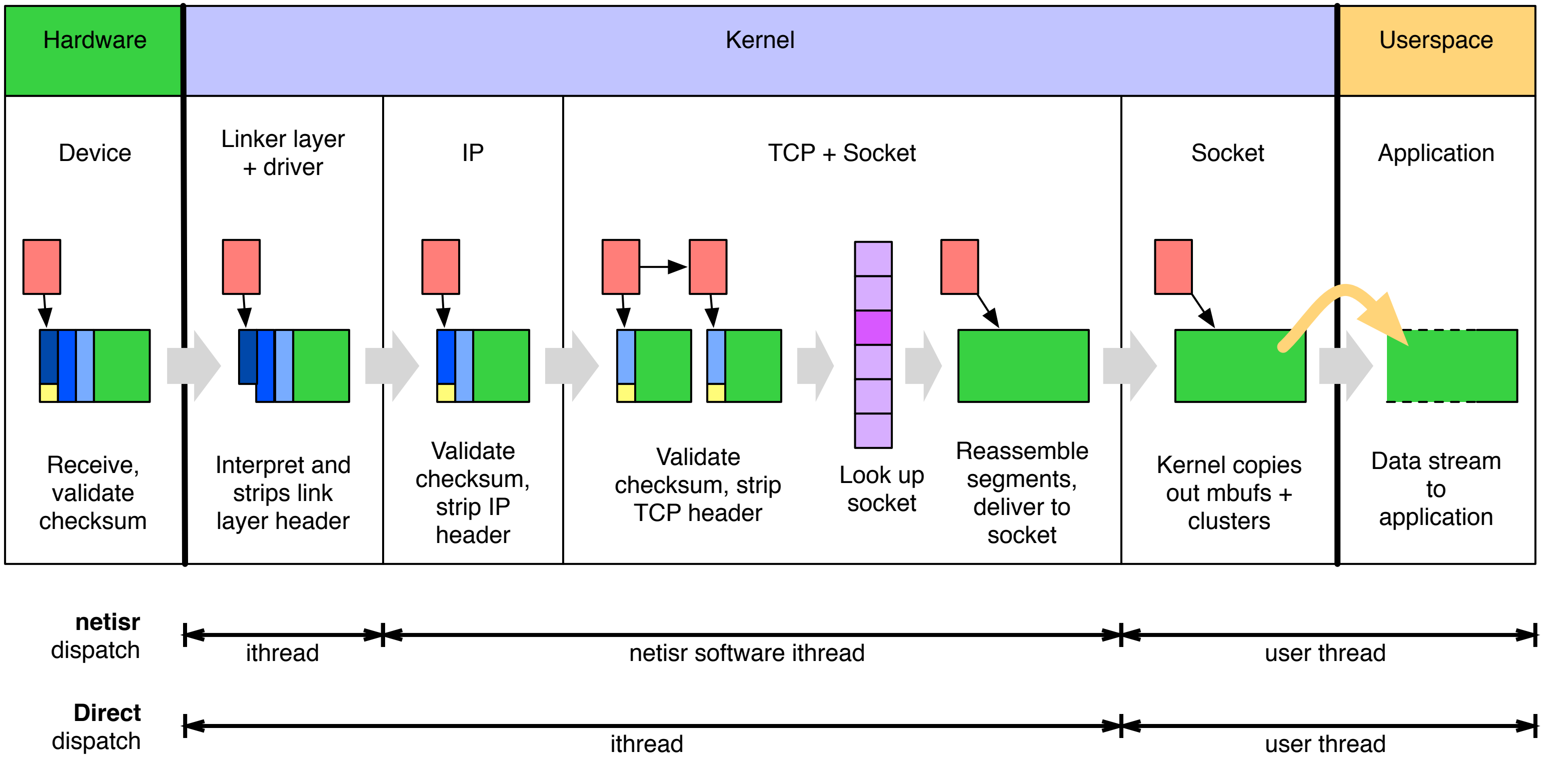
em_start()
em_entr()

NIC

Forwarding path in the network stack

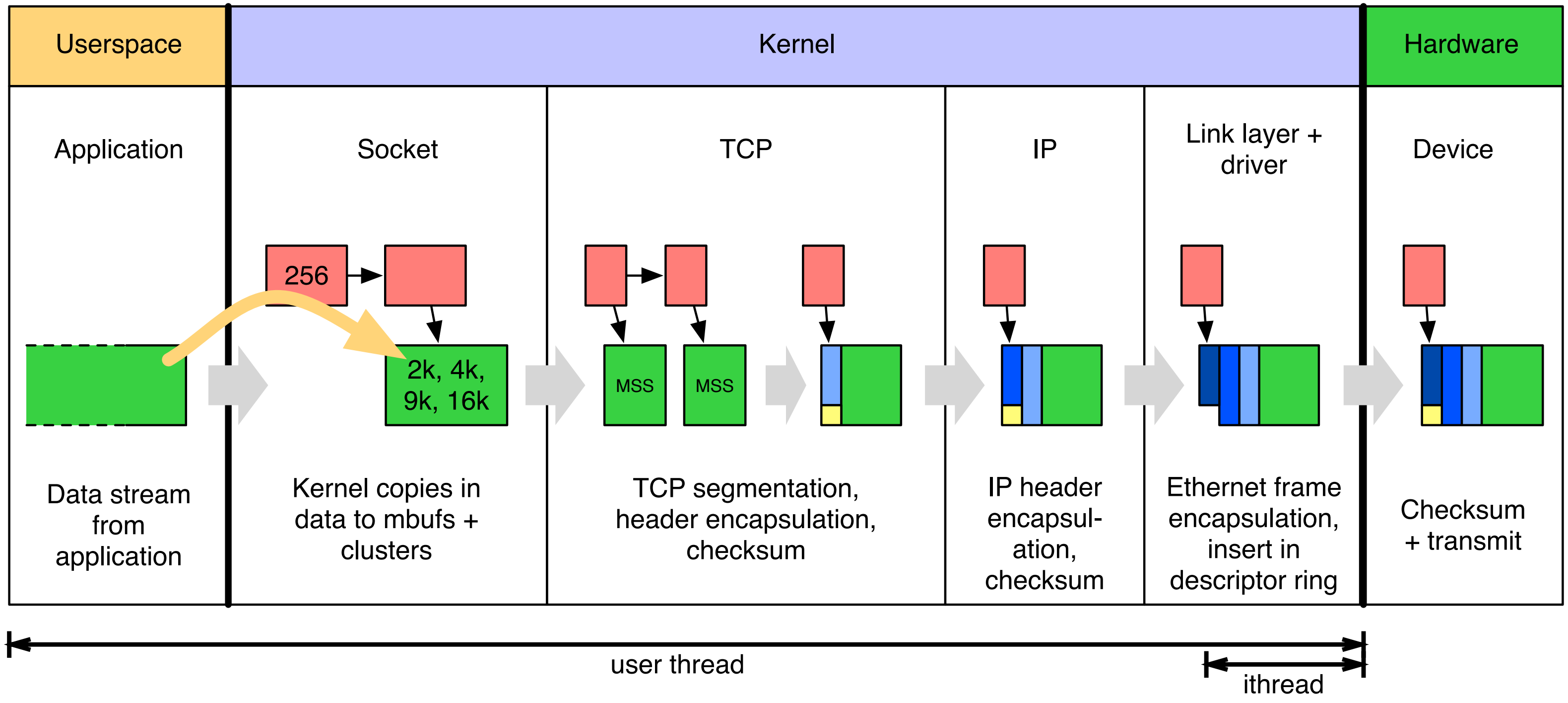


Work dispatch: input path



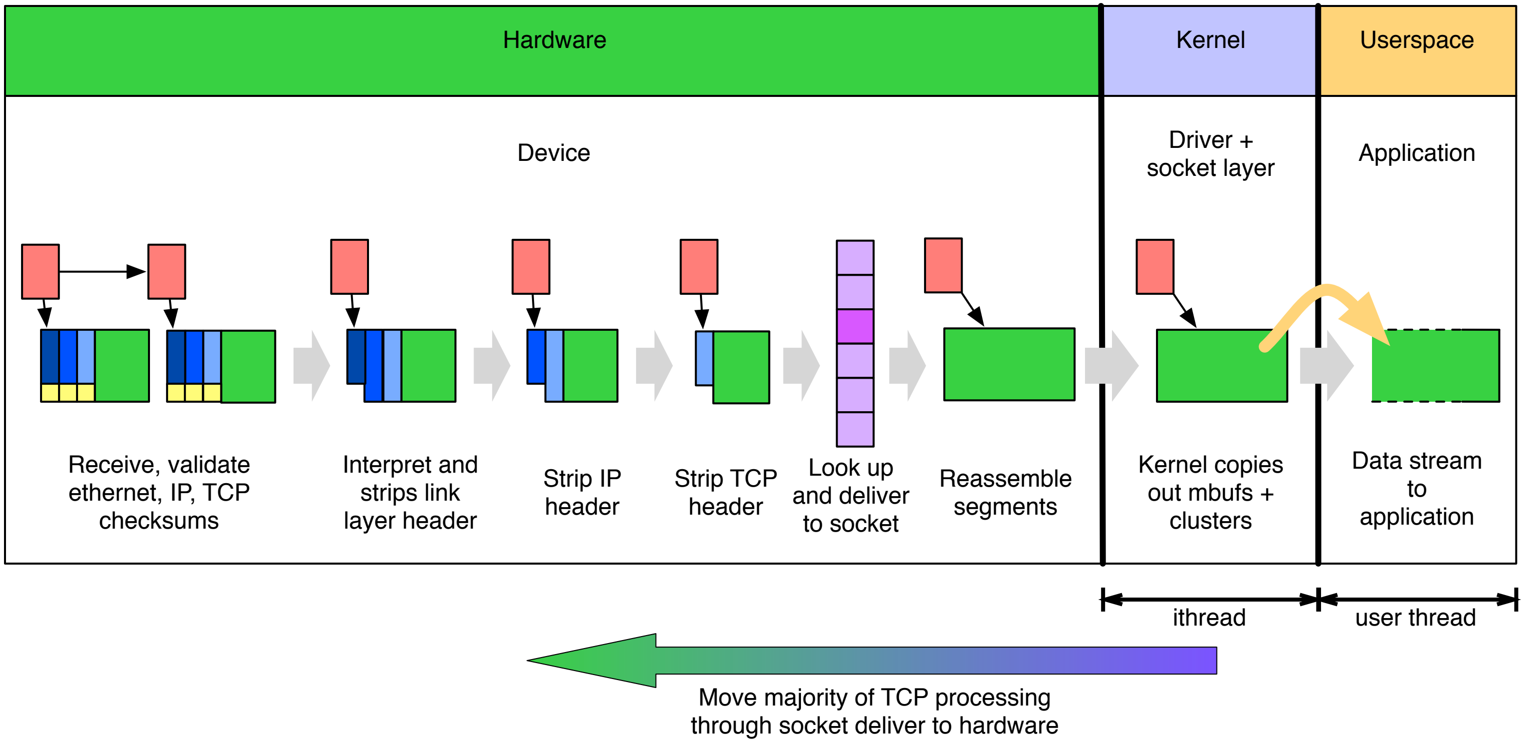
- **Deferred dispatch**: ithread → netisr thread → user thread
- **Direct dispatch**: ithread → user thread
  - Pros: reduced latency, better cache locality, drop early on overload
  - Cons: reduced parallelism and work placement opportunities

Work dispatch: output path



- Fewer deferred dispatch opportunities implemented
  - (Deferred dispatch on device-driver handoff in new `iflib` KPIs)
- Gradual shift of work from software to hardware
  - Checksum calculation, segmentation, …

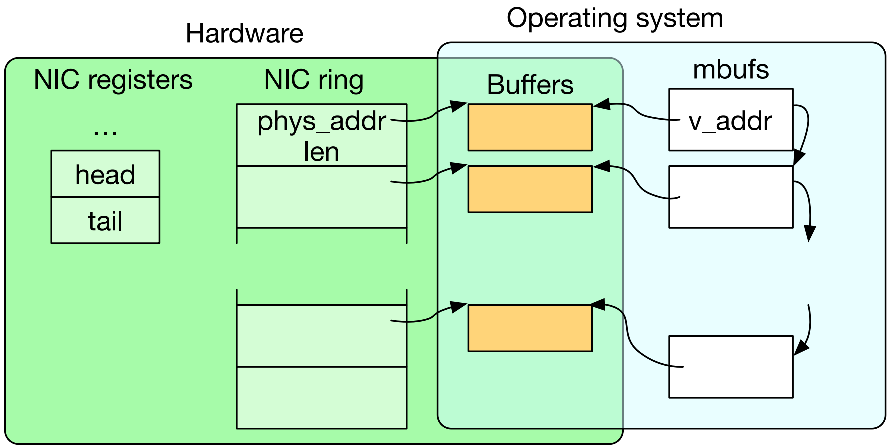
Work dispatch: TOE input path

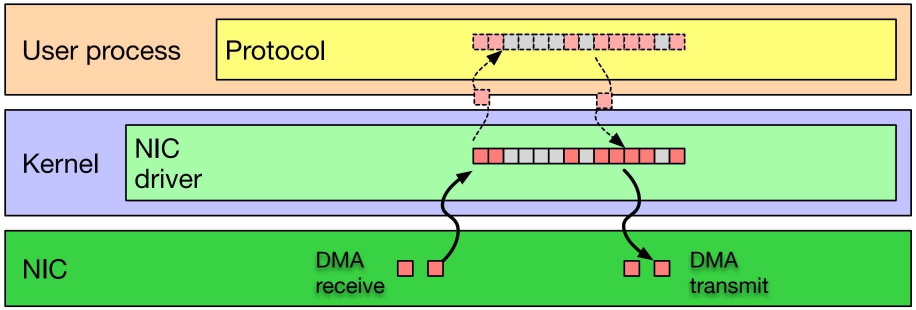


- Kernel provides socket buffers and resource allocation
- Remainder, including state, retransmissions, etc., in NIC
- But: two network stacks? Less flexible/updateable structure?
  - Better with an explicit HW/SW architecture – e.g., Microsoft Chimney

Netmap: a novel framework for fast packet I/O

Luigi Rizzo, USENIX ATC 2012 (best paper).

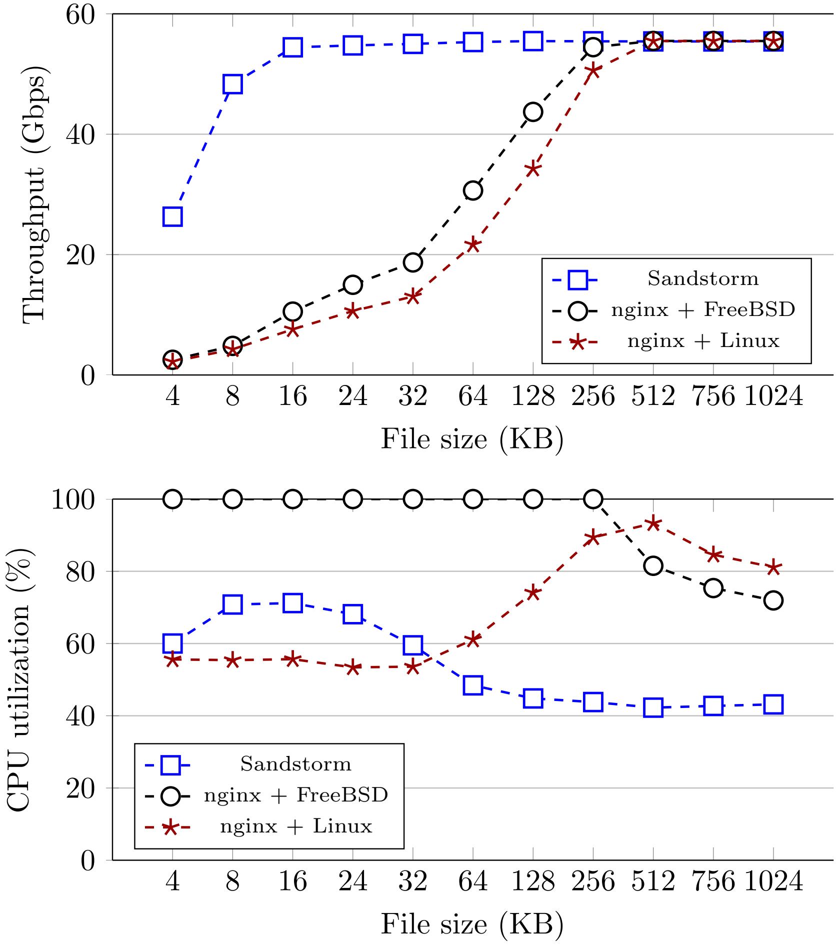




- Map NIC buffers directly into user process memory
- Not the sockets API: Zero copy to/from application
- System calls initiate DMA, block for NIC events
- Packets can be reinjected into normal stack
- Ships in FreeBSD; patch available for Linux
- Userspace network stack can be **specialised** to task (e.g., packet forwarding)

Network stack specialisation for performance

Ilias Marinos, Robert N. M. Watson, Mark Handley, SIGCOMM 2014, 2017.



- 30 years since the network-stack design developed
- Massive changes in architecture, micro-architecture, memory…
    - Optimising compilers
    - Cache-centered CPUs
    - Multiprocessing, NUMA
    - DMA, multiqueue
    - 10 Gigabit/s Ethernet
- Performance lost to 'generality' throughout stack
- Revisit fundamentals through clean-slate stack
- Orders-of-magnitude performance gains

Break

The Network Stack (2)

L41 Lecture 6

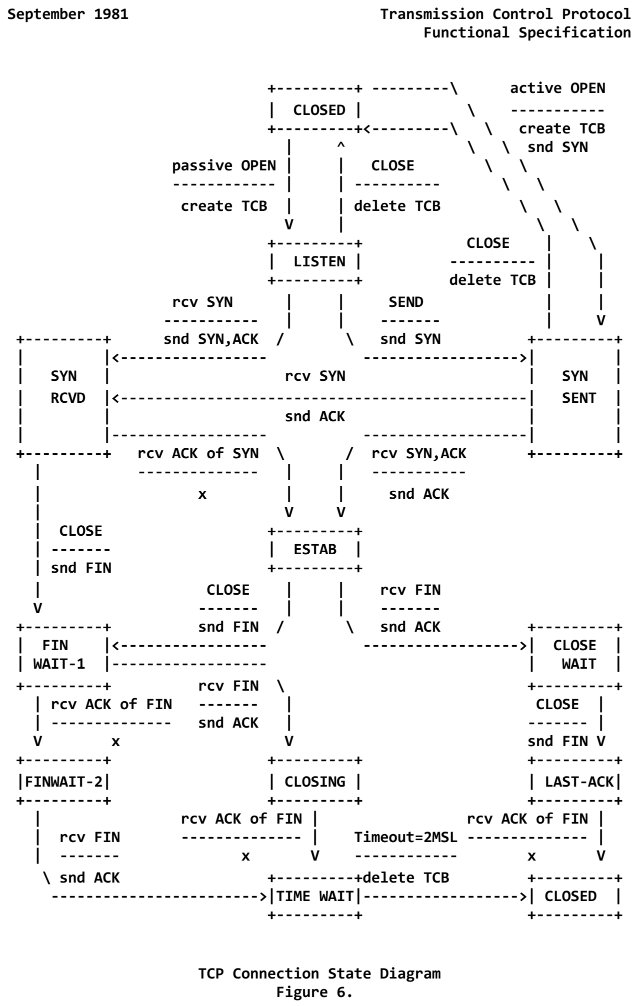
Dr Robert N. M. Watson

2019-2020

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)



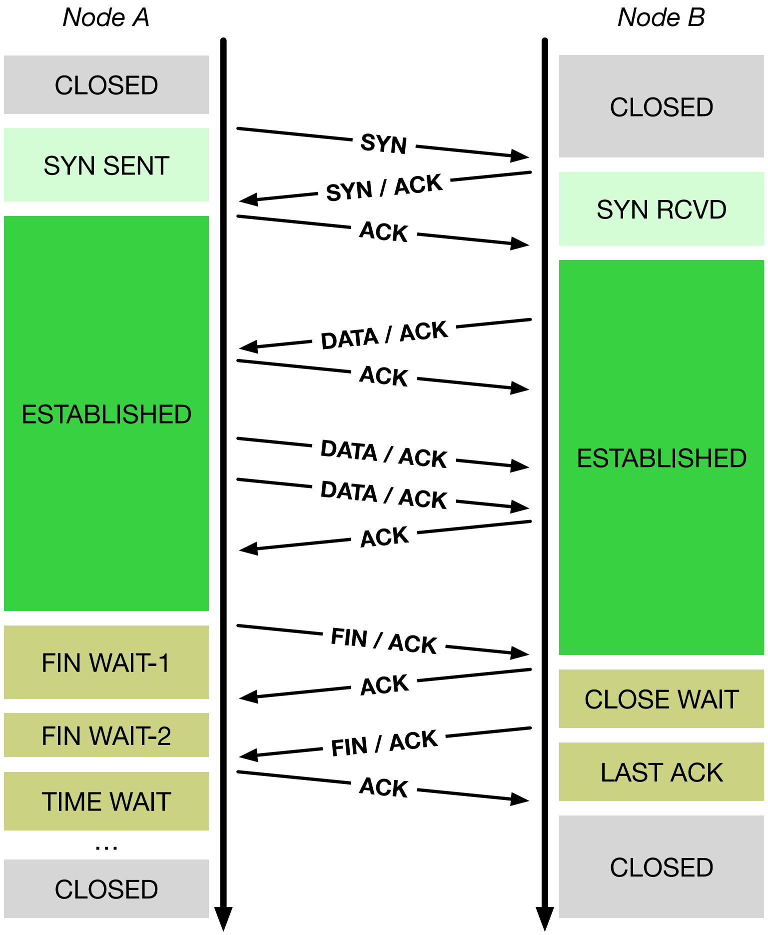
```
September 1981                                Transmission Control Protocol
                                                   Functional Specification

                              +---------+ ---------\      active OPEN
                              |  CLOSED |            \    -----------
                              +---------+<---------\   \   create TCB
                                |     ^              \   \  snd SYN
                   passive OPEN |     |   CLOSE        \   \
                   ------------ |     | ----------       \   \
                    create TCB  |     | delete TCB         \   \
                                V     |                      \   \
                              +---------+            CLOSE    |    \
                              |  LISTEN |          ---------- |     |
                              +---------+          delete TCB |     |
                   rcv SYN      |     |     SEND              |     |
                  -----------   |     |    -------            |     V
 +---------+      snd SYN,ACK  /       \   snd SYN          +---------+
 |         |<-----------------           ------------------>|         |
 |   SYN   |                    rcv SYN                     |   SYN   |
 |   RCVD  |<-----------------------------------------------|   SENT  |
 |         |                    snd ACK                     |         |
 |         |------------------           -------------------|         |
 +---------+   rcv ACK of SYN  \       /  rcv SYN,ACK       +---------+
   |           --------------   |     |   -----------
   |                  x         |     |     snd ACK
   |                            V     V
   |  CLOSE                   +---------+
   | -------                  |  ESTAB  |
   | snd FIN                  +---------+
   |                   CLOSE    |     |    rcv FIN
   V                  -------   |     |    -------
 +---------+          snd FIN  /       \   snd ACK          +---------+
 |  FIN    |<-----------------           ------------------>|  CLOSE  |
 | WAIT-1  |------------------                              |   WAIT  |
 +---------+          rcv FIN  \                            +---------+
   | rcv ACK of FIN   -------   |                            CLOSE  |
   | --------------   snd ACK   |                           ------- |
   V        x                   V                           snd FIN V
 +---------+                  +---------+                   +---------+
 |FINWAIT-2|                  | CLOSING |                   | LAST-ACK|
 +---------+                  +---------+                   +---------+
   |                rcv ACK of FIN |                 rcv ACK of FIN |
   |  rcv FIN       -------------- |    Timeout=2MSL -------------- |
   |  -------              x       V    ------------        x       V
    \ snd ACK                 +---------+delete TCB         +---------+
     ------------------------>|TIME WAIT|------------------>| CLOSED  |
                              +---------+                   +---------+

                      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.

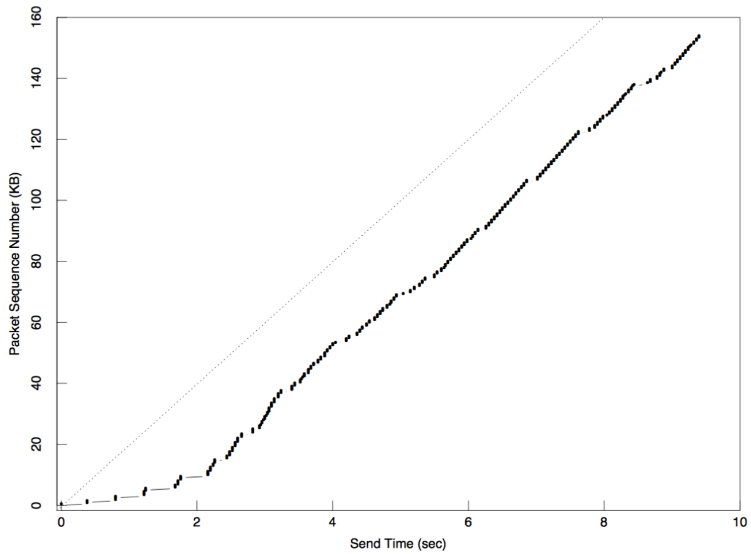
TCP principles and properties



- 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 (and, recently, via delay: BBR)

TCP congestion control and avoidance

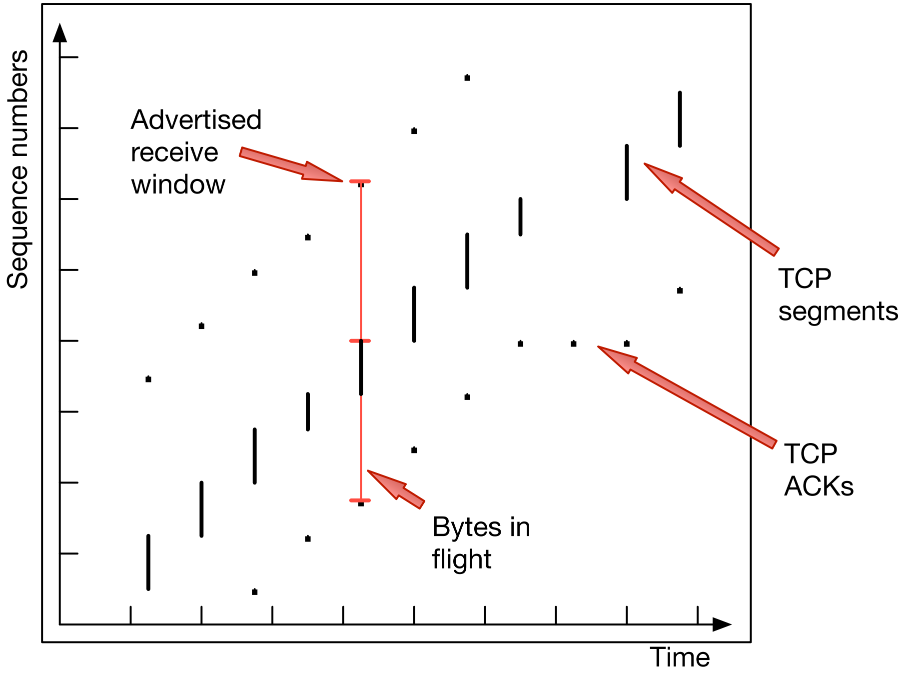
Figure 4: Startup behavior of TCP with Slow-start



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), BBR (Cardwell, ACM Queue 2016)

TCP time/sequence graphs



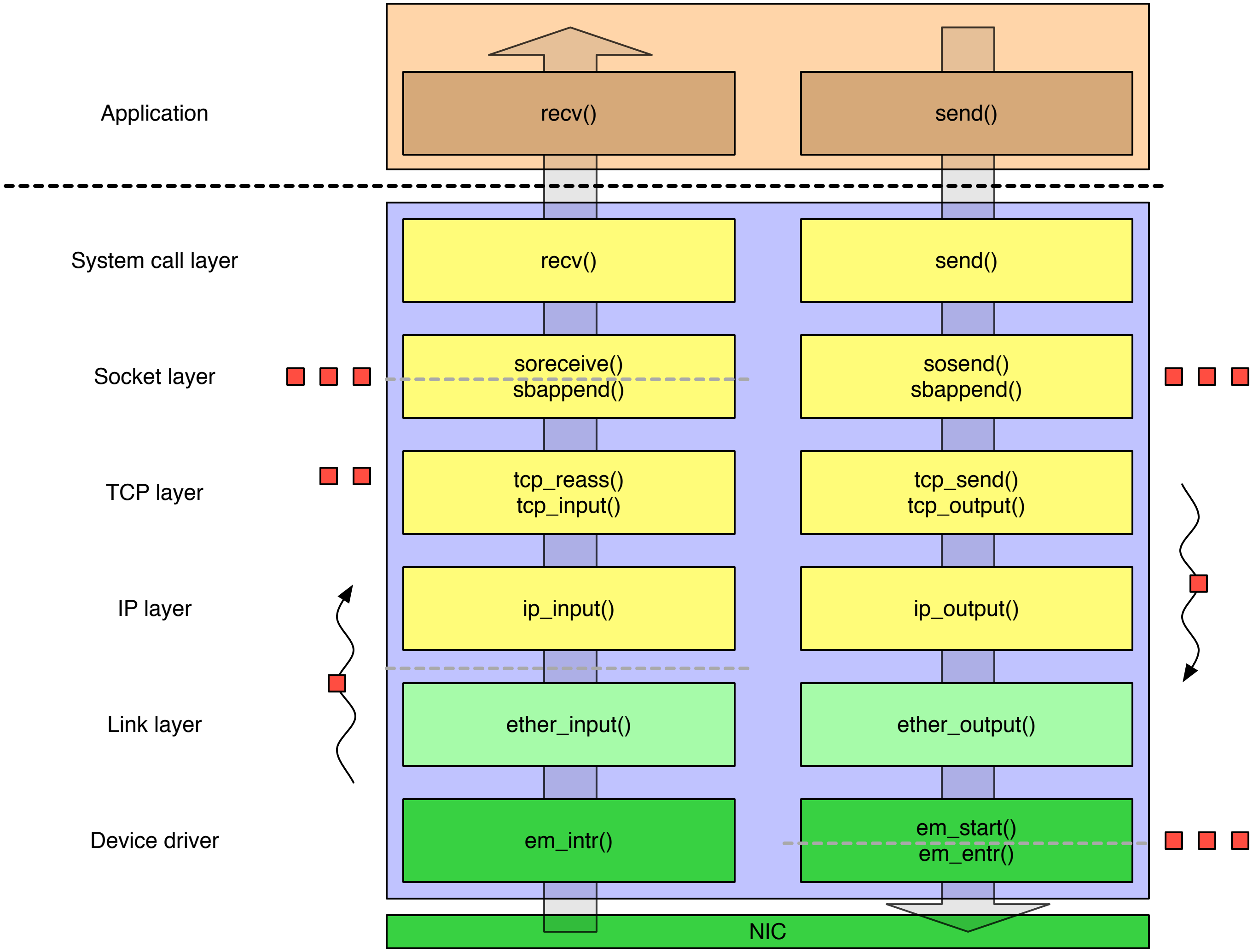
- Extracted from TCP packet traces (e.g., via `tcpdump`)
- Visualize windows, congestion response, buffering, RTT, etc:
  - X: Time
  - Y: Sequence number
- We can extract this data from the network stack directly using Dtrace
  - Allows correlation/plotting with respect to other variables / events

Evolving BSD/FreeBSD TCP implementation

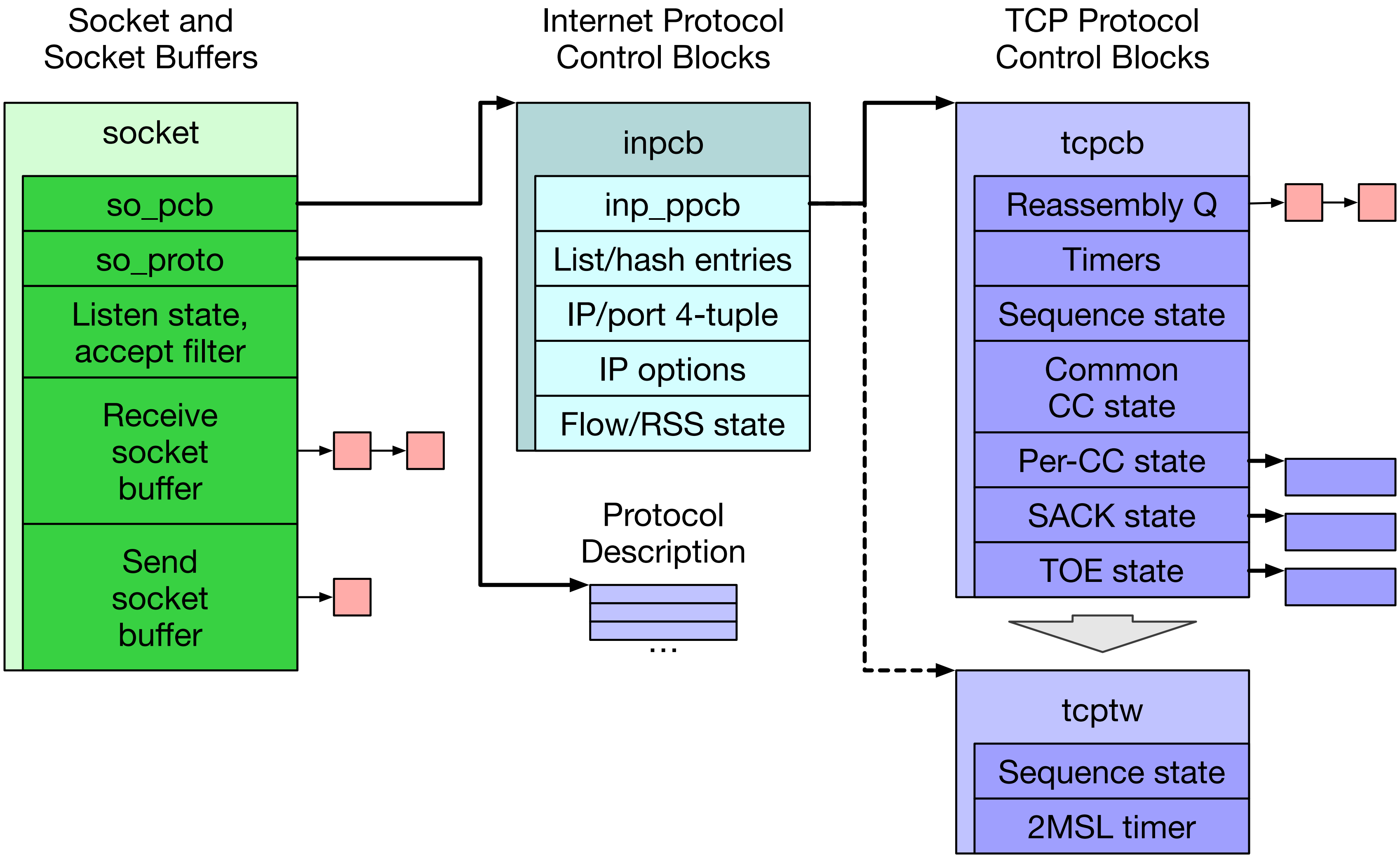
Year	Version	Feature
1983	4.2BSD	BSD sockets, TCP/IP implementation
1986	4.3BSD	VJ/Karels congestion control
1999	FreeBSD 3.1	`sendfile(2)`
2000	FreeBSD 4.2	TCP accept filters
2001	FreeBSD 4.4	TCP ISN randomisation
2002	FreeBSD 4.5	TCP SYN cache/cookies
2003	FreeBSD 5.0-5.1	IPv6, TCP TIMEWAIT state reduction
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

- … changes continue to this day … BBR, RCU, pluggable TCP, …
- 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

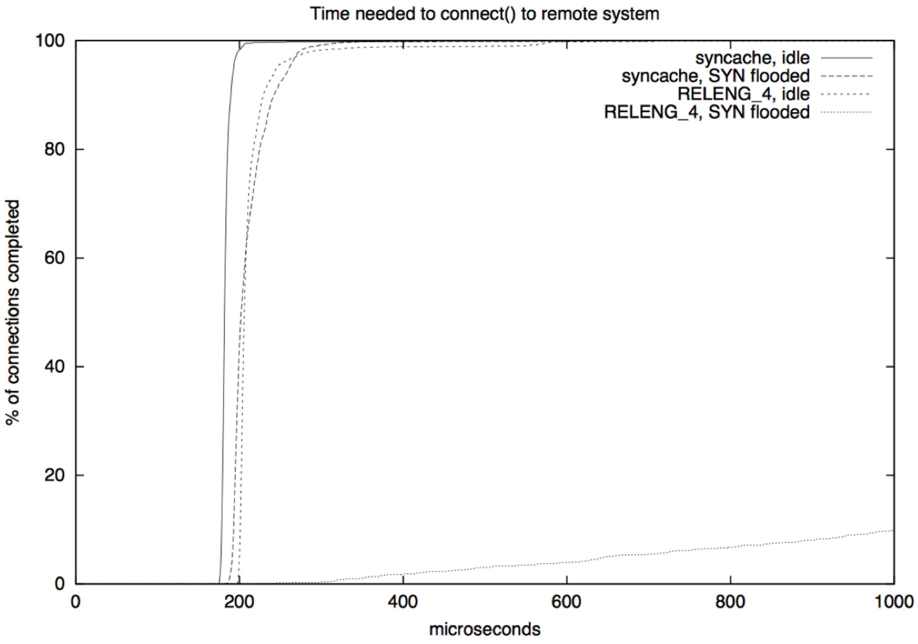
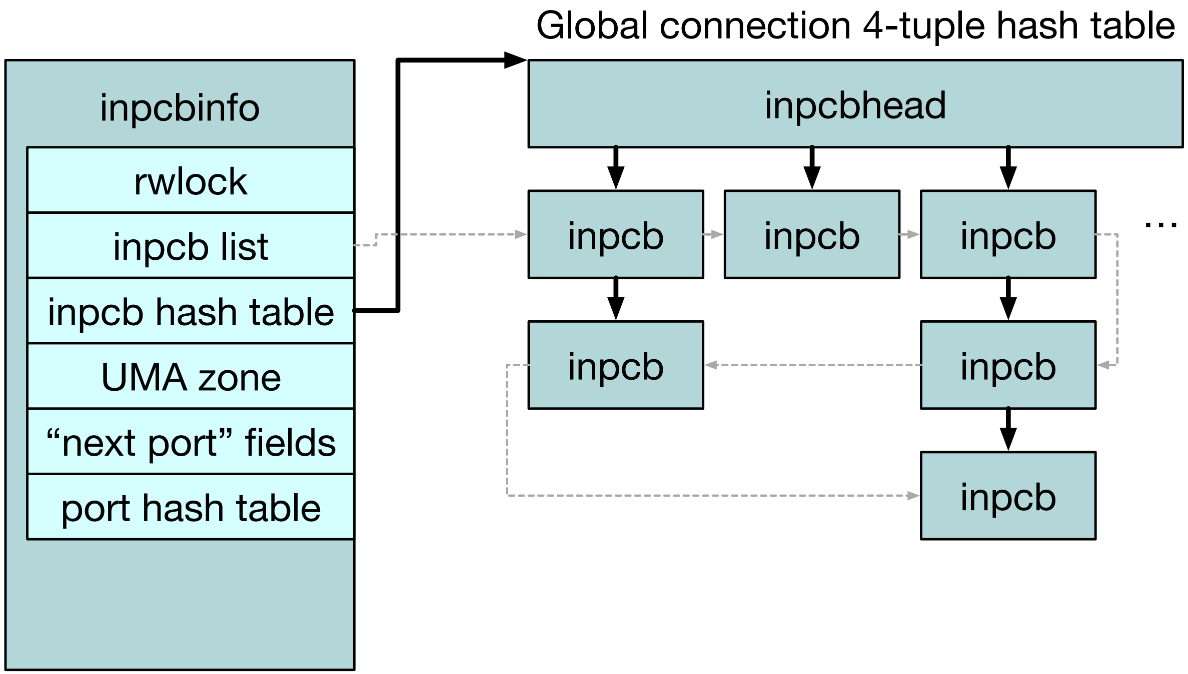


Figure 3: Time needed to connect() to remote system.

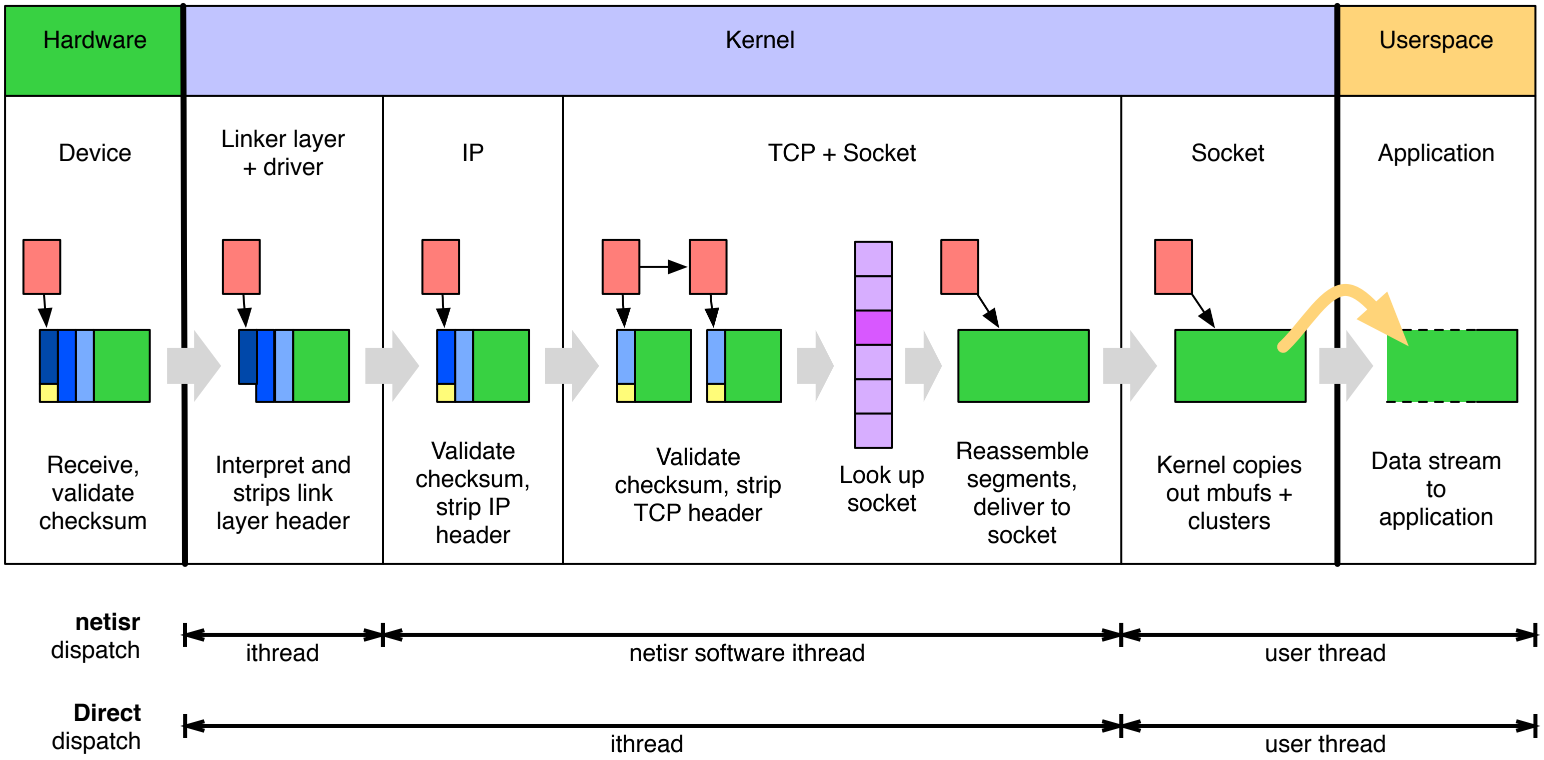
- Yahoo!, Amazon, CNN taken down by SYN floods in February 2000
- D. Borman: **TCP SYN cache** – minimise state for new connections
- D. Bernstein: **SYN cookies** – eliminate state entirely – at a cost
- J. Lemon: **TCP TIMEWAIT reduction** – minimise state during close
- J. Lemon: **TCP TIMEWAIT recycle**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

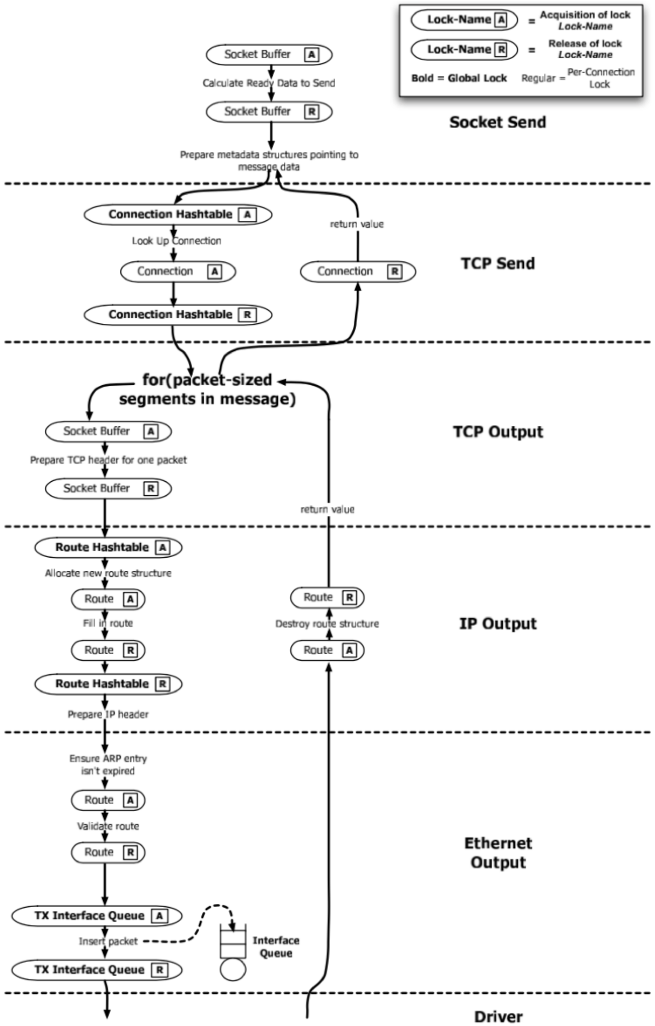
Lect. 5 - Work dispatch: input path



- **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

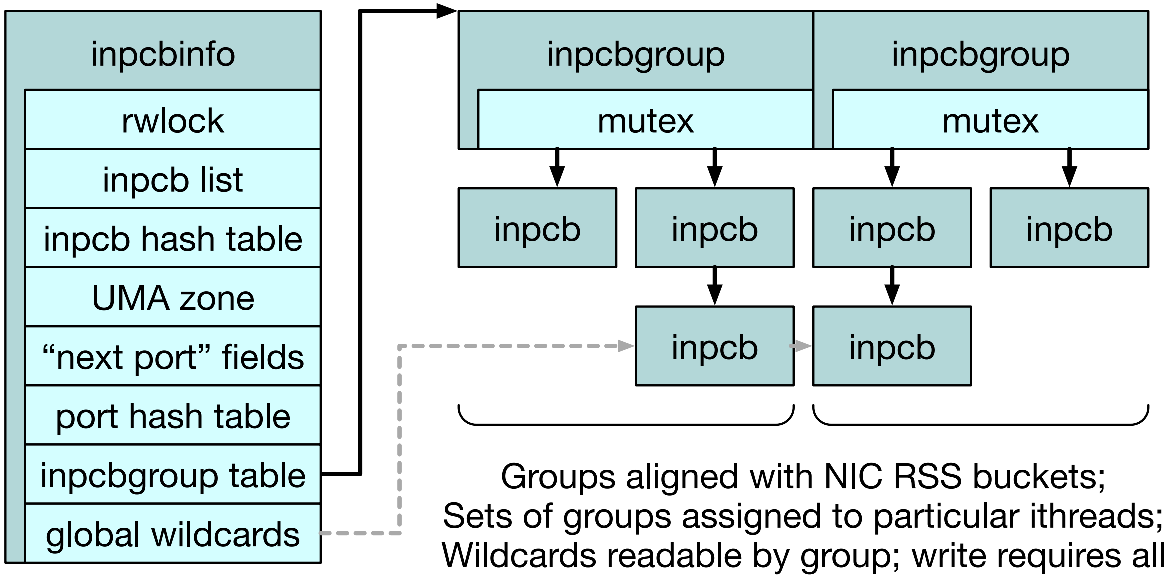
An Evaluation of Network Stack Parallelization Strategies in Modern Operating Systems

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

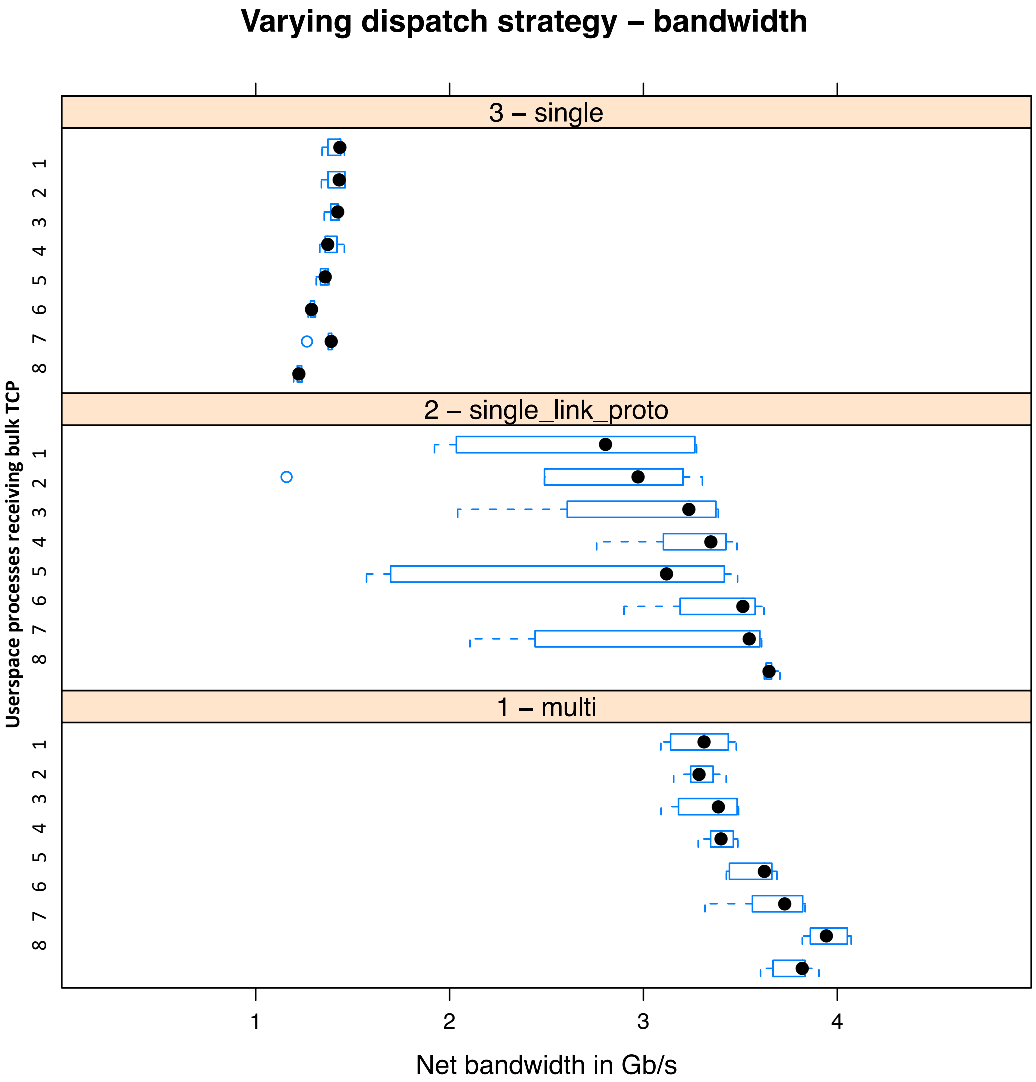
FreeBSD connection groups, RSS



Groups aligned with NIC RSS buckets;
Sets of groups assigned to particular ithreads;
Wildcards readable by group; write requires all

- **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



- 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 placement
  - 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
  - Be sure to consider probe and simulation effects
- Lab 4 – TCP state machine and latency
  - Measure the TCP state machine in practice
  - Experiment with artificially induced latency (DUMMYNET)
- Lab 5 – TCP congestion control
  - 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!