It's Time to Replace TCP in the Datacenter

John Ousterhout
Stanford University

Datacenter Networking

- **Extraordinary hardware advances:**
  - Link speeds: 10 Gbps → 25 Gbps → 40 Gbps → 100 Gbps → ??
  - RTTs ~ 5 µsec
  - Cost-effective switching chips
- **Raw network potential not accessible to applications**
  - Especially latency, throughput for small messages
  - Cause: network stack overheads
- **Solution: redesign the network stack**
  - Replace TCP protocol
  - Lighter weight RPC framework
  - Eliminate software stack implementations: move transport protocols to NICs

Goals for Datacenter Networks

High Performance

Unique to Datacenters

Data Throughput	Low Tail Latency	Message Throughput
Full link speed for large messages	< 10 µsec for short messages?	10-100M short messages per second?

Application-level performance is what matters

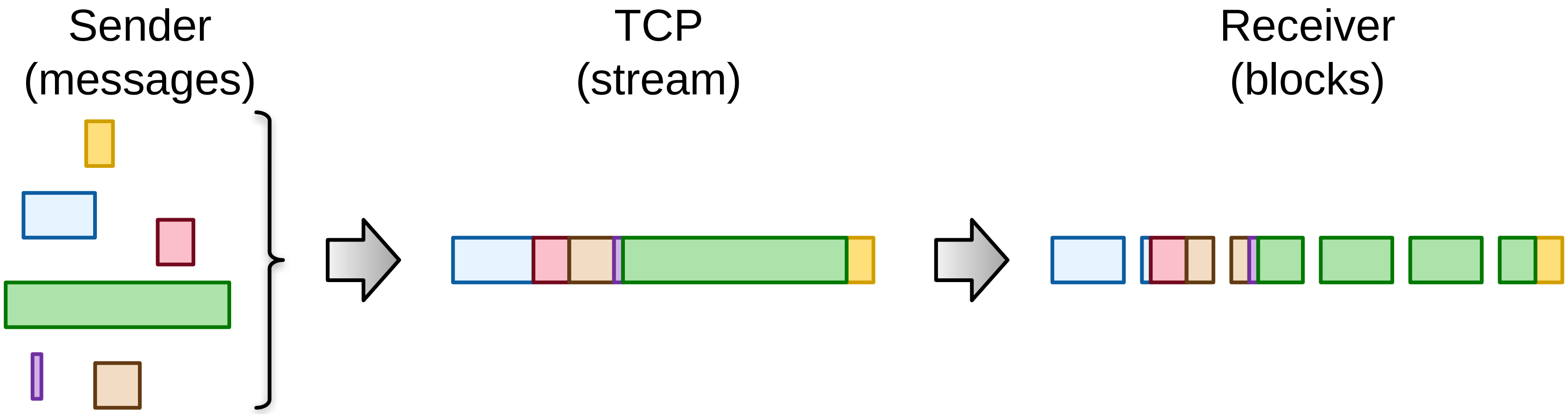
Implied Requirements

- **Load balancing across cores:**
  - One core cannot sustain link speeds > 10 Gbps
  - Hot spots limit throughput, drive up tail latency
- **Congestion control in the network:**
  - Core fabric (avoidable with good load balancing)
    - Poor load balancing reduces throughput
  - At the edge (unavoidable due to fan-in)
    - Buffer buildup increases latency

Part 1: Replacing TCP

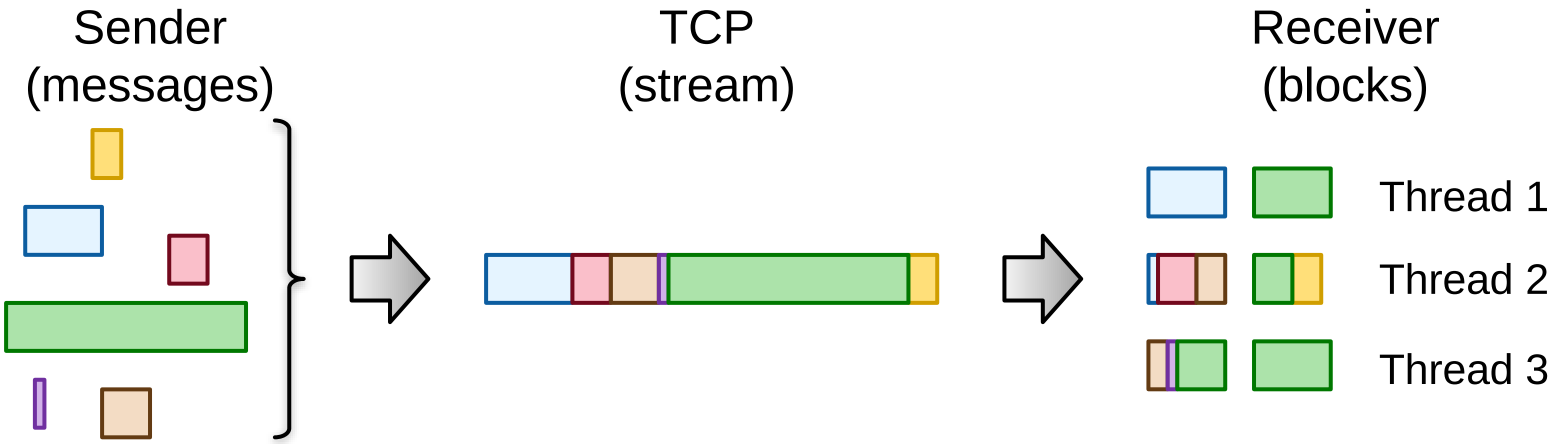
- **TCP has been tremendously successful**
- **But, every aspect of its design is wrong for the datacenter:**
  1. Stream-oriented → Message-based
  2. Connection-oriented → Connectionless
  3. Fair scheduling (bandwidth sharing) → Run to completion (SRPT)
  4. Sender-driven congestion control → Receiver-driven congestion control
  5. Assumes in-order packet delivery → No ordering requirements
- **Must find a way to introduce a TCP replacement:**
  - Homa is a good candidate
  - Insert underneath RPC frameworks such as gRPC and Thrift?

1. TCP Data Model: Byte Stream



- **Applications care about messages, but TCP drops boundary info**
- **Extra complexity/overhead for message reassembly**

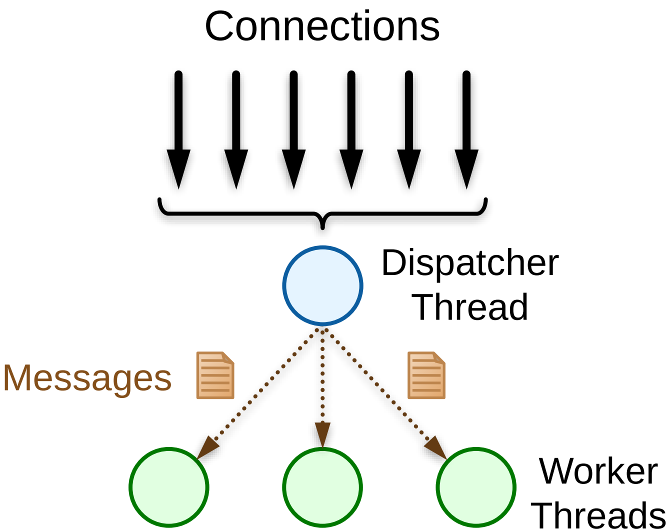
1. TCP Byte Streams, cont'd



- **Disastrous for load balancing**
  - Can't share one stream among multiple threads
  - Can't offload dispatching to NIC

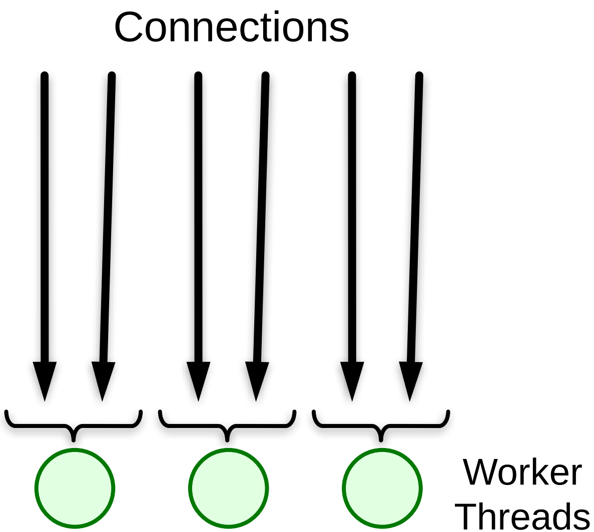
Load Balancing Choices

Choice #1: dispatcher thread



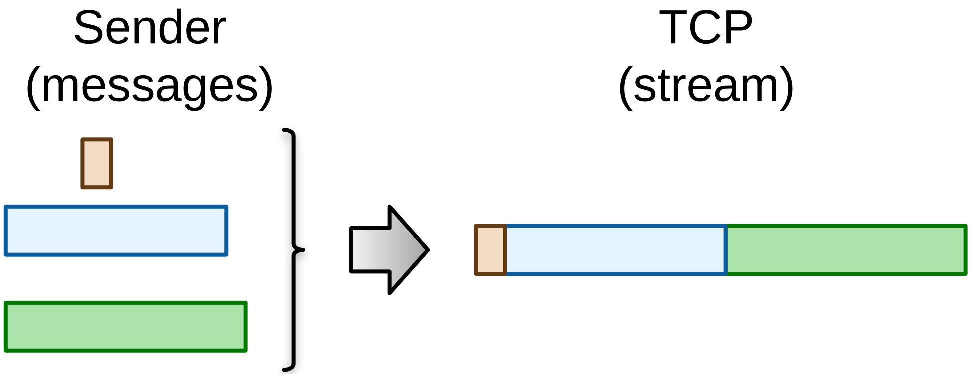
- **Extra latency for worker handoff**
- **Dispatcher is throughput bottleneck (~1M msgs/sec)**

Choice #2: partition connections



- **Static load balancing: prone to hot spots**

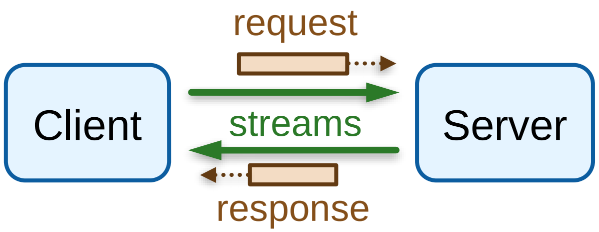
1. TCP Byte Streams, cont'd



- **Head-of-line blocking:**
  - Short messages can get stuck behind long ones
  - High tail latency

Stream-Level Reliability Inadequate

- **Clients want round-trip guarantees:**
  - Deliver request
  - Ensure it is processed
  - Deliver response
  - Or, notify of error
- **Stream guarantees are weaker:**
  - Best-effort delivery of request or response
  - No notification if server machine crashes
- **Clients must implement additional timeout mechanisms**
  - Even though TCP already implements timers

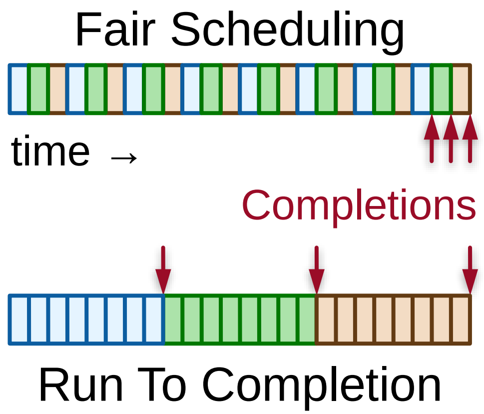


2. TCP is Connection-Oriented

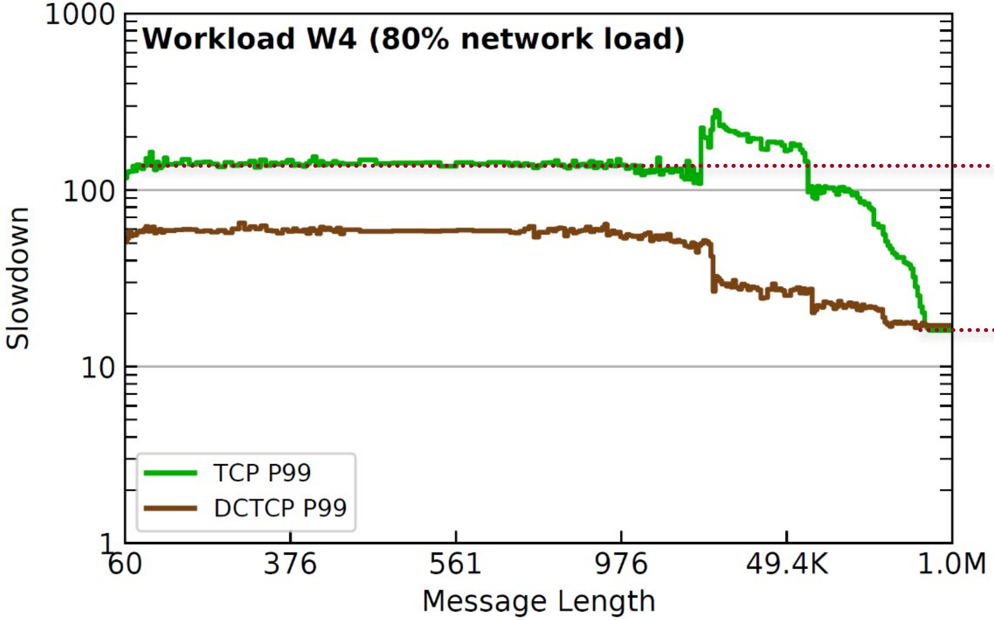
- **Requires long-lived state for each stream**
  - ~2000 bytes per connection in Linux, not including packet buffers
  - Individual datacenter apps can have thousands of connections
  - Mitigate with connection pooling/proxies (e.g. Facebook)? Adds overhead
  - Challenging for NIC offloading (e.g. Infiniband): thrashing in connection caches
- **Before sending any data, must pay round-trip for connection setup**
  - Problematic in serverless environments: can't amortize setup cost
- **Motivation for connections:**
  - Enable reliable delivery, flow control, congestion control
  - But, all these can be achieved without connections

3. TCP Uses Fair Scheduling

- **When loaded, share bandwidth equally among active connections**
- **Well-known to perform poorly: everyone finishes slowly**
- **Run-to-completion approaches (e.g. SRPT) are better**
  - But requires message sizes



TCP Isn't Actually Fair!



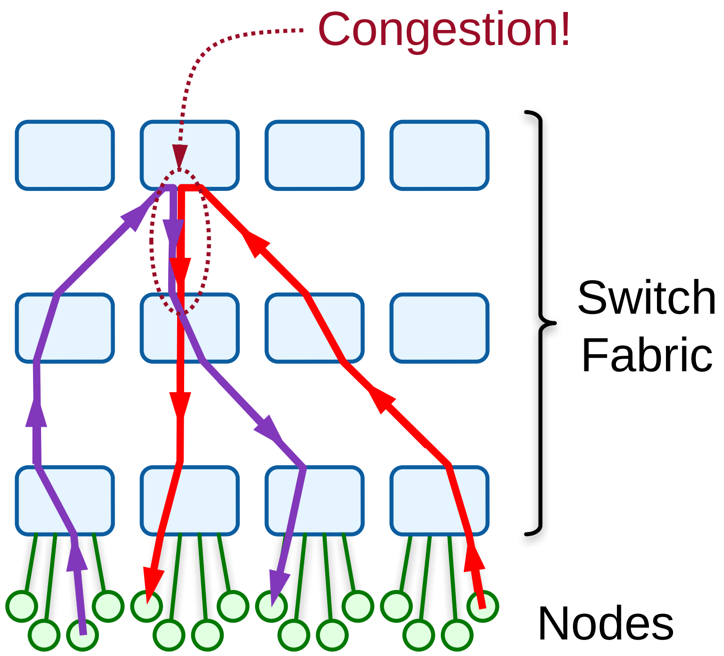
~9x bias against short messages

4. TCP: Sender-Driven Congestion Control

- **Senders responsible for scaling back transmission rates when needed**
  - But, they have no direct knowledge of congestion
- **Congestion signals based on buffer occupancy:**
  - Packets dropped if queues overflow
  - Congestion notifications based on queue length
- **Problems:**
  - Significant buffer occupancy when system is loaded
  - Queuing causes delays, especially for short messages

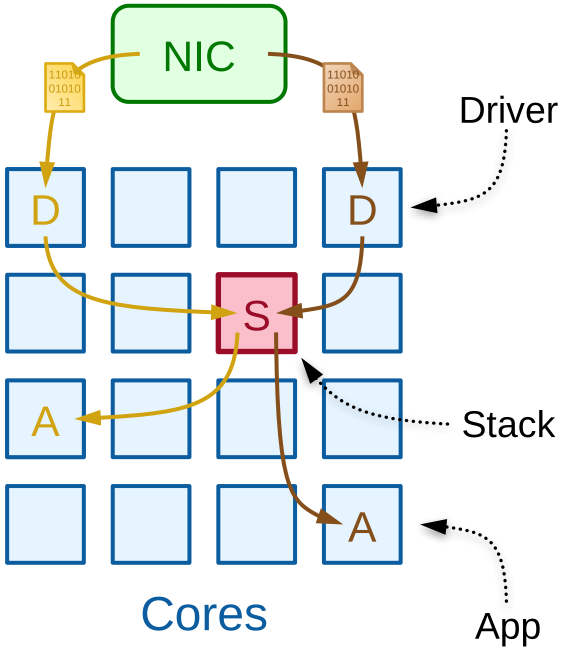
5. TCP Expects In-Order Delivery

- **Packets must arrive in same order as transmitted**
  - Out-of-order arrivals assumed to indicate packet drops
- **Severe damage to load-balancing:**
  - Hot spots in both hardware and software
  - High tail latency
- **Network: must use flow-consistent routing**
  - Overloaded links inevitable even at low loads
  - Dominant cause of core congestion in datacenter networks?



Software Hot Spots

- **In-order delivery requires packets for flow to traverse the same cores:**
  - Driver (NAPI/GRO)
  - Stack (SoftIRQ)
  - Application
- **Result: uneven core loading, hot spots**
- **Dominant source of software-induced tail latency**



TCP is Beyond Repair

- **Too many problems**
- **Problems are fundamental, interrelated**
  - Lack of message boundaries makes it hard to implement SRPT
- **There is no part worth keeping**
- **Need a replacement protocol that is different from TCP in every aspect**
- **Homa!**
  - Clean-slate design for datacenters
  - Solves TCP's problems
  - Design elements are synergistic
  - (For datacenters only, not WANs)

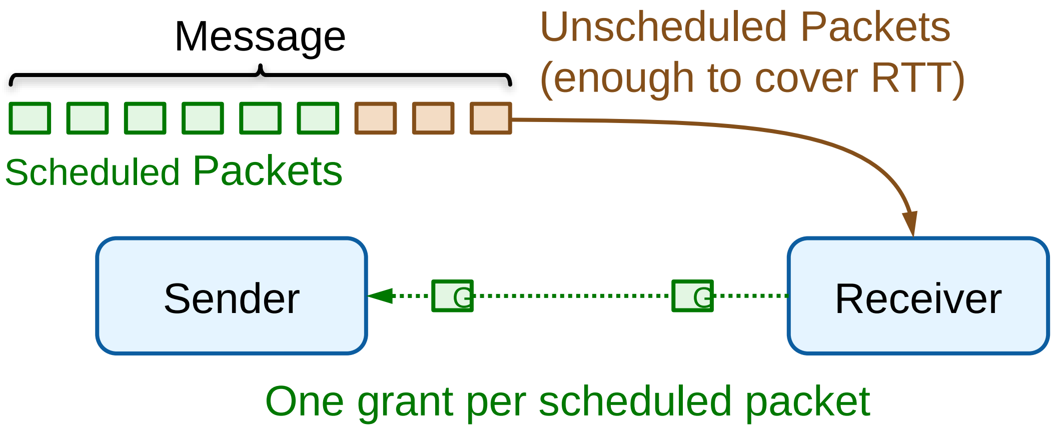
1. Homa is Message-Based

- **Dispatchable units are explicit in the protocol**
- **Enables efficient load balancing**
  - Multiple threads can safely read from a single socket
  - Future NICs can dispatch messages directly to threads
- **Enables run-to-completion (e.g. SRPT)**

2. Homa is Connectionless

- **Fundamental unit is a remote procedure call (RPC)**
  - Request message
  - Response message
  - RPCs are independent
- **No long-lived connection state**
  - (But there is long-lived per-peer state: ~200 bytes)
- **No connection setup overhead**
  - Use one socket to communicate with many peers
- **Homa ensures end-to-end RPC reliability**
  - No need for application-level timers

3. Homa: Receiver-Driven Congestion Control

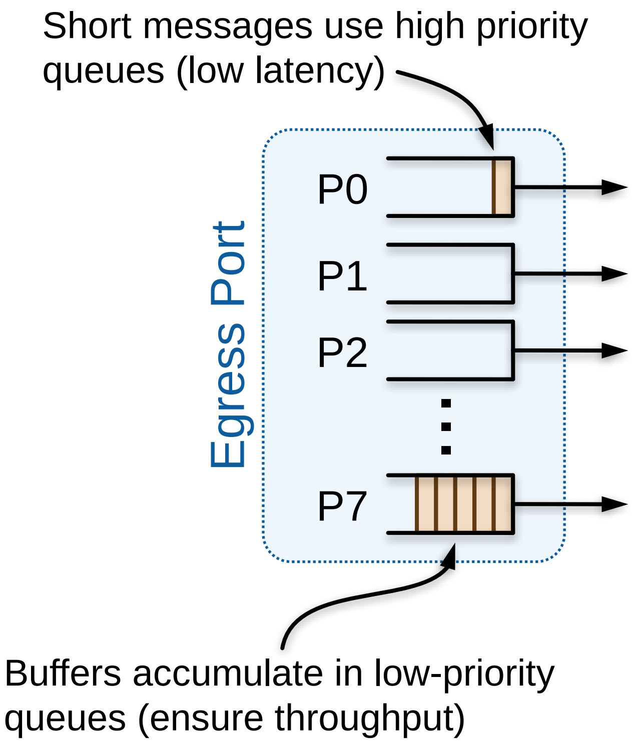


- **Receiver can delay grants to:**
  - Reduce congestion in TOR
  - Prioritize shorter messages
- **Message sizes allow receivers to predict the future:**
  - Faster, more accurate response to congestion

Homa Uses Priority Queues

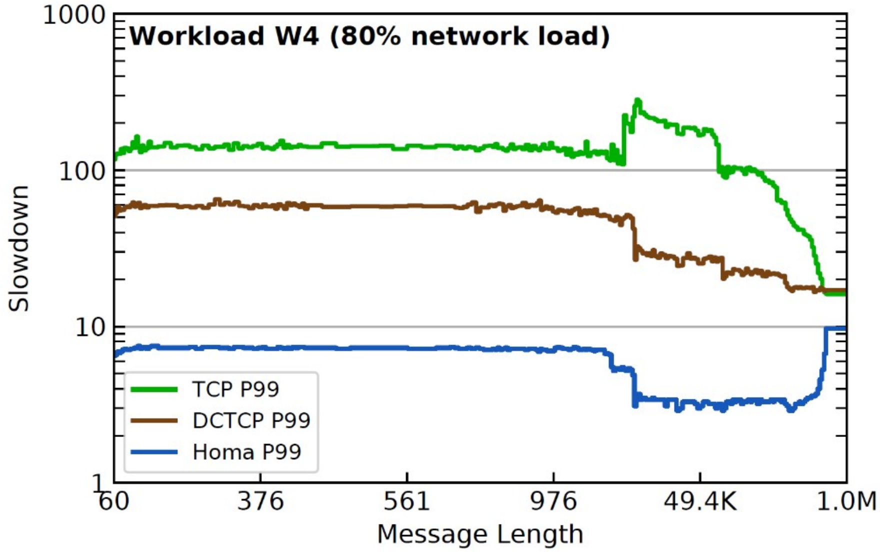
- **Modern switches: 8–16 priority queues per egress port**
- **Homa receivers select priorities for SRPT:**
  - Favor shorter messages
- **Achieve both high throughput and low latency**
  - Need buffering to maintain throughput (e.g. if sender doesn't respond to grant)
  - But buffers can result in delays
  - Solution: overcommitment:
    - Grant to multiple messages
    - Different priority for each message

Overcommitment



4. Homa: SRPT

- **Combination of grants, priorities**
- **Run-to-completion improves performance for every message length!**
- **Starvation risk for longest messages?**
  - Use 5-10% of bandwidth for oldest message



5. Homa: No Order Requirement

- **Can use packet spraying in datacenter networks**
  - Hypothesis: will eliminate core congestion
    (unless core fabric systemically overloaded)
- **Better load balancing across CPU cores**

Can Homa Replace TCP?

- **Will be difficult: TCP deeply entrenched**
- **My personal mission: either**
  - Figure out a way for Homa to take over from TCP in the datacenter, or
  - Learn why this is not possible
- **First step: widely available production quality implementation**

Homa Kernel Module for Linux

- **Open source: https://github.com/PlatformLab/HomaModule**
- **Dynamically loadable**
- **No kernel modifications required**
  - New system calls layered on ioctl
- **Currently runs on Linux 5.17 and 5.18**
- **About 12,000 lines (including heavy comments)**
- **Near production quality**

Homa Dominates TCP/DCTCP

- **All workloads, all message sizes**
- **Latency improvement for short messages:**

	P50	P99
vs TCP	3.5–7.5x	19–72x
vs DCTCP	2.7–3.8x	7–83x

- **P99 for Homa almost always better than P50 for TCP/DCTCP**
- **See USENIX ATC 2021 paper for details**

Challenge: API Incompatibility

- **Homa requires software modifications:**
  - Message-based API different from TCP sockets
- **Impractical to convert 1000s of apps that layer directly on sockets**
- **Instead, focus on apps designed for datacenters**
  - They layer on an RPC framework, not sockets
  - Only a few popular frameworks: gRPC and Thrift?
- **Solution: integrate Homa with major frameworks**
  - Then apps can convert with ~one-line changes
- **Work in progress: gRPC support (GitHub: PlatformLab/grpc_homa)**
  - C++ integration is working (without encryption)
  - Java integration is underway

gRPC is Slow

- **Best-case round-trip latency for short RPCs:**

	Network	Client	Server	Total
gRPC/TCP	30 µs	30 µs	30 µs	90 µs
gRPC/Homa	20 µs	16 µs	19 µs	55 µs

- **gRPC is faster with Homa than TCP**
- **Can’t achieve network hardware potential with gRPC**

Will eventually need a lighter weight RPC framework

Part 2: Eliminating Software

- **Replacing TCP makes a big difference, but can do even better**
  - 5–10x additional improvement available
- **Software implementations of transport layer no longer make sense**
  - Network speeds increasing faster than CPU speeds
- **High software overheads in OS**
  - 9.5 µs for tiny Homa RPCs
- **Moving to user space doesn't help enough**

Must move transport layer to NIC hardware

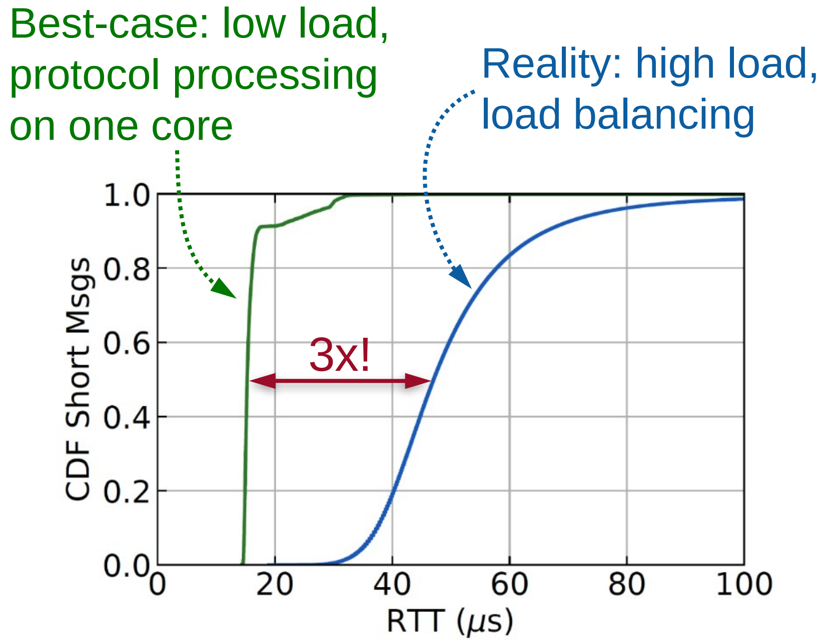
Tail Latency

- **Small-message best case (RTT): 15 µs**
- **Small-message P99:**

  Homa/Linux: 100 µs
  Homa/RAMCLoud: 14 µs (user space, kernel bypass)
- **Primary source of tail latency: software overheads**
- **Primary culprit: load balancing**
  - Multi-core approaches sacrifice efficiency
  - Can’t eliminate hot spots

3x Overhead for Load Balancing

Homa/Linux



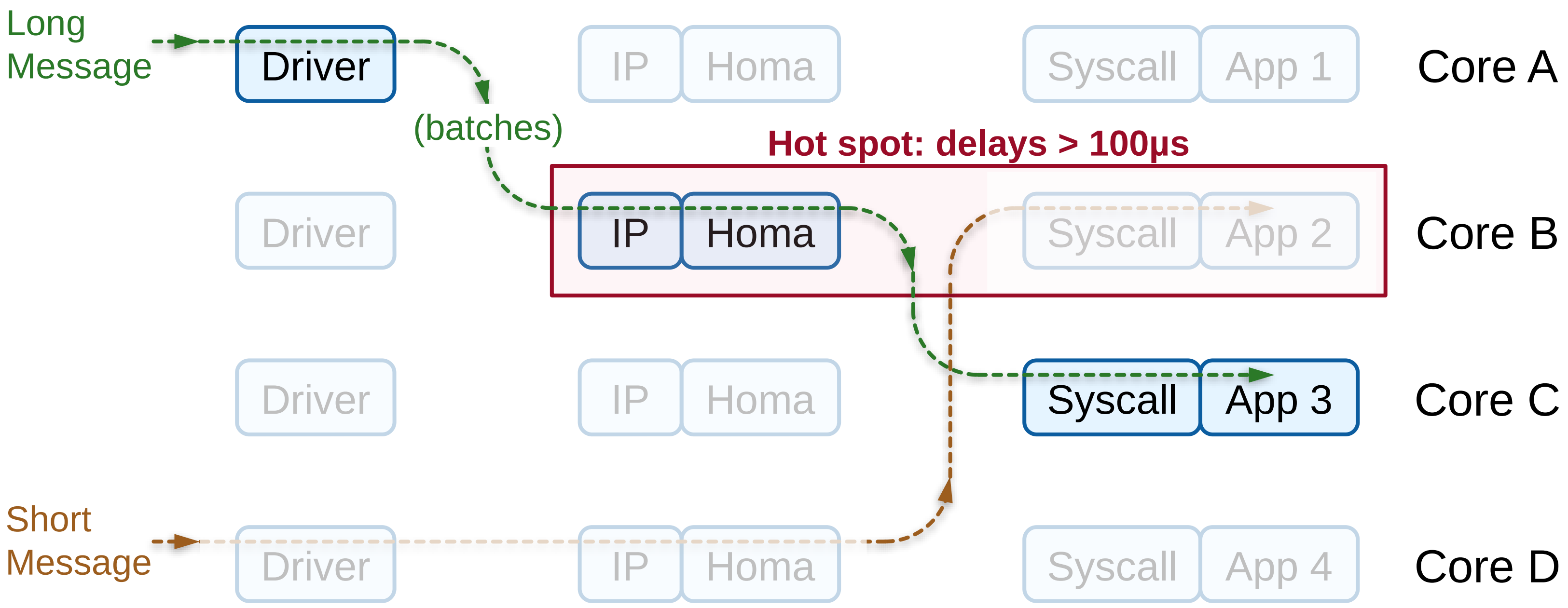
Google Snap/Pony

	Thruput	Cores
No load balancing	70 Gbps	1
Load balancing	80 Gbps	4.5–7

4-6x efficiency loss

Hypothesis: cache interference

Load Balancing Causes Hot Spots



Primary source of tail latency in Homa/Linux

Move Transports to User Space?

- **Small-message P50 RTT:**
  - Homa/Linux: 15 µs
  - Homa/RAMCloud: 5 µs
  - eRPC: 4 µs
- **Small-message P99 RTT:**
  - Homa/Linux: 100 µs
  - Homa/RAMCloud: 14 µs
- **Small-message throughput (M RPCs/sec/core)**
  - Homa/Linux: 0.1
  - Homa/RAMCloud: 1.0
  - Shenango: 1.0
  - eRPC: 2.5

No load balancing (or hand-partitioned)

Measured under ideal conditions

Only short messages

Research prototypes oversimplified

No shared state in protocol

No congestion control

Homa/Linux vs. Snap

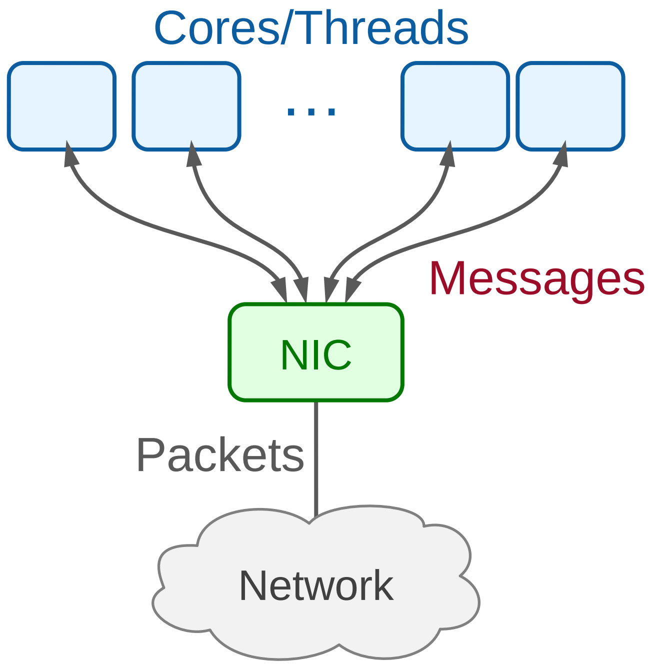
- **Snap: Google's user-space protocol implementation**
- **Snap < 2x better than Homa/Linux:**

	Homa	Snap
Best-case latency (polling)	15 µs	9 µs
Cores to drive 80 Gbps bidirectional	17	9–14

User-space protocols are not a long-term solution

Better Solution: New NIC

- **Move transport to NIC hardware:**
  - Kernel bypass
  - Message-based interface
- **Other NIC features:**
  - Dispatching/load balancing (pick idle app thread)
  - Virtualization/mgmt (e.g. rate limiting)
  - Encryption/authentication



Need a New NIC Architecture

- **Requirements:**
  - Process packets at line rate
  - Programmable to support multiple protocols and functions
  - Important for protocol implementations to be open source
- **Existing “smart NICs” are inadequate:**
  - Many-core designs: still software, but with slower CPUs
  - FPGA approach: design environment too awkward?
  - P4 pipelines: no long-term state
- **A difficult/interesting challenge in special-purpose architecture**

Why Not Infiniband?

Strengths:

- **Kernel bypass**
- **NIC implementation of transport**
- **Very fast NICs (e.g. Mellanox)**

Wrong abstractions:

- **One-sided RDMA operations have limited applicability**
  - Microscopically efficient, macroscopically inefficient
- **Reliable queue pairs use connections and streams**
  - Limited cache space for connections hurts performance
  - Same problems with streams as TCP
- **Unreliable datagrams are … unreliable**

Poor congestion control (PFC)

Controversy over Homa

- **Recent papers claim:**
  - Problems with Homa (e.g. unsustainable buffer usage)
  - Better alternatives
- **Examples:**
  - Aeolus (SIGCOMM 2020)
  - PowerTCP (NSDI 2022)
  - dcPIM (SIGCOMM 2022)
- **All of these have major flaws**
  - Unrealistic configurations
  - Hobbled/incorrect Homa implementations
- **See the Homa Wiki for details:**

  https://homa-transport.atlassian.net/wiki/spaces/HOMA/overview

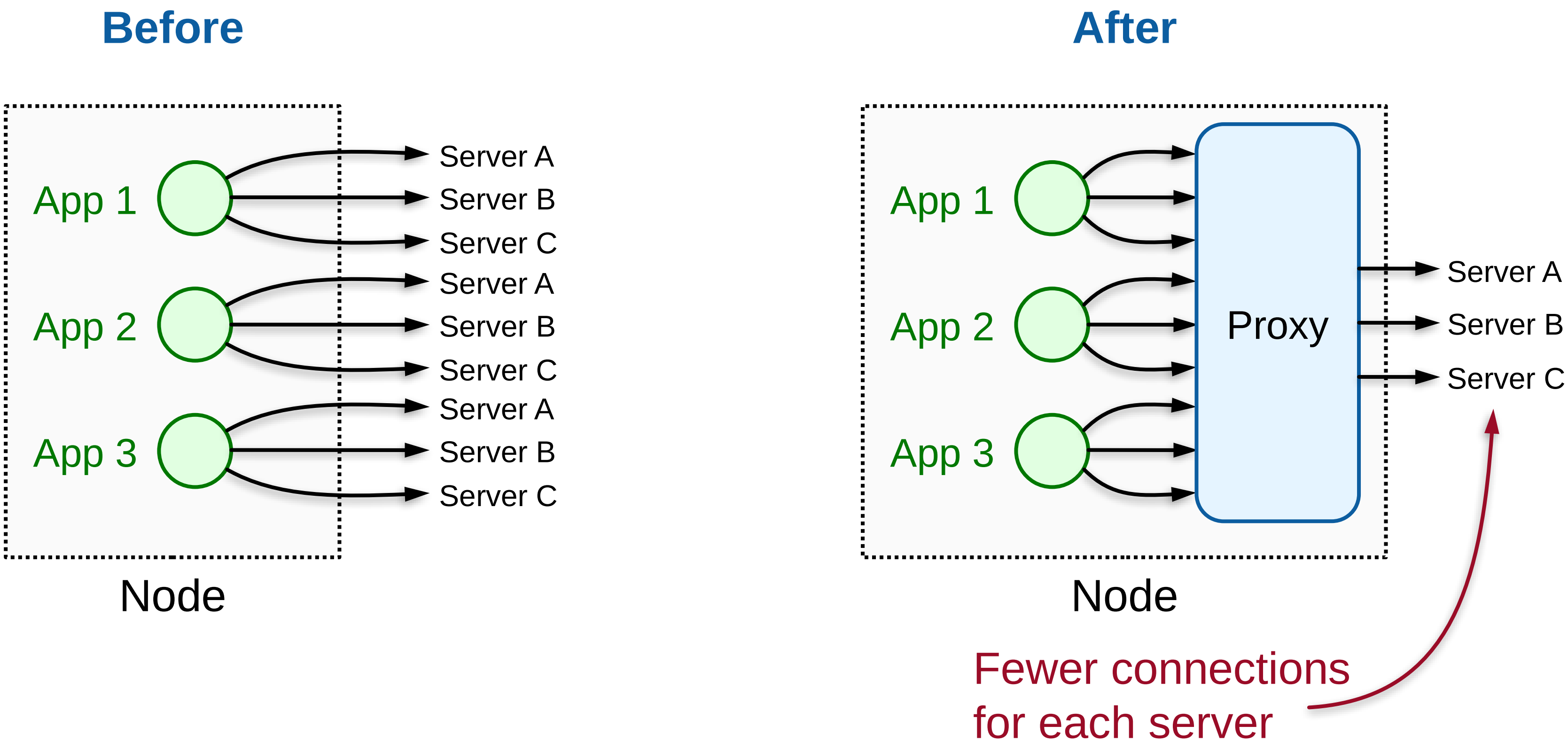
If We Build It, Will They Come?

- **Is it important for applications to harness the full power of network hardware?**
- **Today: no-chicken-no-egg cycle**
  - Apps must make do with existing networking performance
  - No incentive to make networking faster
- **Will faster networking enable new applications?**
- **If you know of such apps, let me know!**

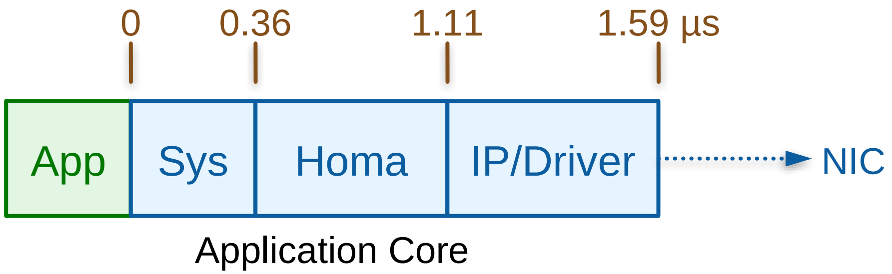
Conclusion

- **Datacenter network architects have created fabulous hardware**
- **Do we want to make capabilities available to apps?**
- **If so, need a new networking stack for datacenter software:**
  - New transport protocol (Homa?)
  - New lightweight RPC framework
  - NIC implementation of transport layer

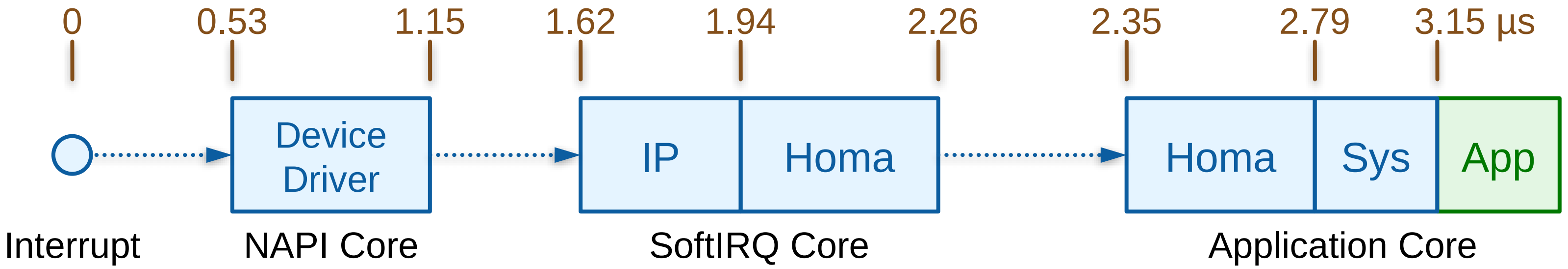
Facebook Connection Pooling



Best-Case Latency (100B Messages)



	Total	Homa
Receive	3.15 µs	0.76 µs
Send	1.59 µs	0.75 µs



Total round-trip software overhead: 9.5 µs

Notes

- **Mention single class of service for TCP?**

- **Experiences with Linux kernel, gRPC**
  - gRPC far worse
  - Linux kernel: main problem is lack of documentation
  - Joe Damato's "Sending and Receiving Data" pages were invaluable
  - Where have I spent my time?
    - Hot spots
    - Documentation
    - Bandwidth optimizations