A Linux Kernel Implementation of the Homa Transport Protocol

John Ousterhout
Stanford University

PLATFORMLAB

Background

- **Homa: new transport protocol for datacenters**
  - Behnam Montazeri’s PhD dissertation, SIGCOMM 2018
  - Eliminates network congestion at downlinks
  - Low tail latency especially for small messages, even under high load
  - Great results with simulations, RAMCloud implementation
- **Can Homa replace TCP in the datacenter?**
- **This work: production-quality Linux kernel implementation**
  - Reproduce earlier results
  - Support real applications

Takeaways

1. **Results confirm those from Montazeri et al.**
   - Homa/Linux eliminates network congestion
   - 7–83x lower tail latency than TCP or DCTCP

2. **Software overheads are now the primary obstacle to networking performance**
   - Load balancing hot-spots
   - Load balancing cache contention

3. **High-performance datacenter networking requires**
   - Moving transports to the NIC (no software implementation is efficient enough)
   - Moving beyond TCP

Homa API

- **Designed for RPC-style communication in datacenters**
  - Message-oriented
  - Connectionless (but still reliable and flow-controlled)
  - RPCs are independent: no ordering guarantees
- **Connection-less API:**

```
int homa_send(int sockfd, const void *request, size_t reqlen,
        const struct sockaddr *dest_addr, socklen_t addrlen,
        uint64_t *id);

int homa_reply(int sockfd, const void *response, size_t resplen,
        const struct sockaddr *dest_addr, socklen_t addrlen,
        uint64_t id);

int homa_recv(int sockfd, void *buf, size_t len, int flags,
        struct sockaddr *src_addr, socklen_t addrlen,
        uint64_t *id);
```

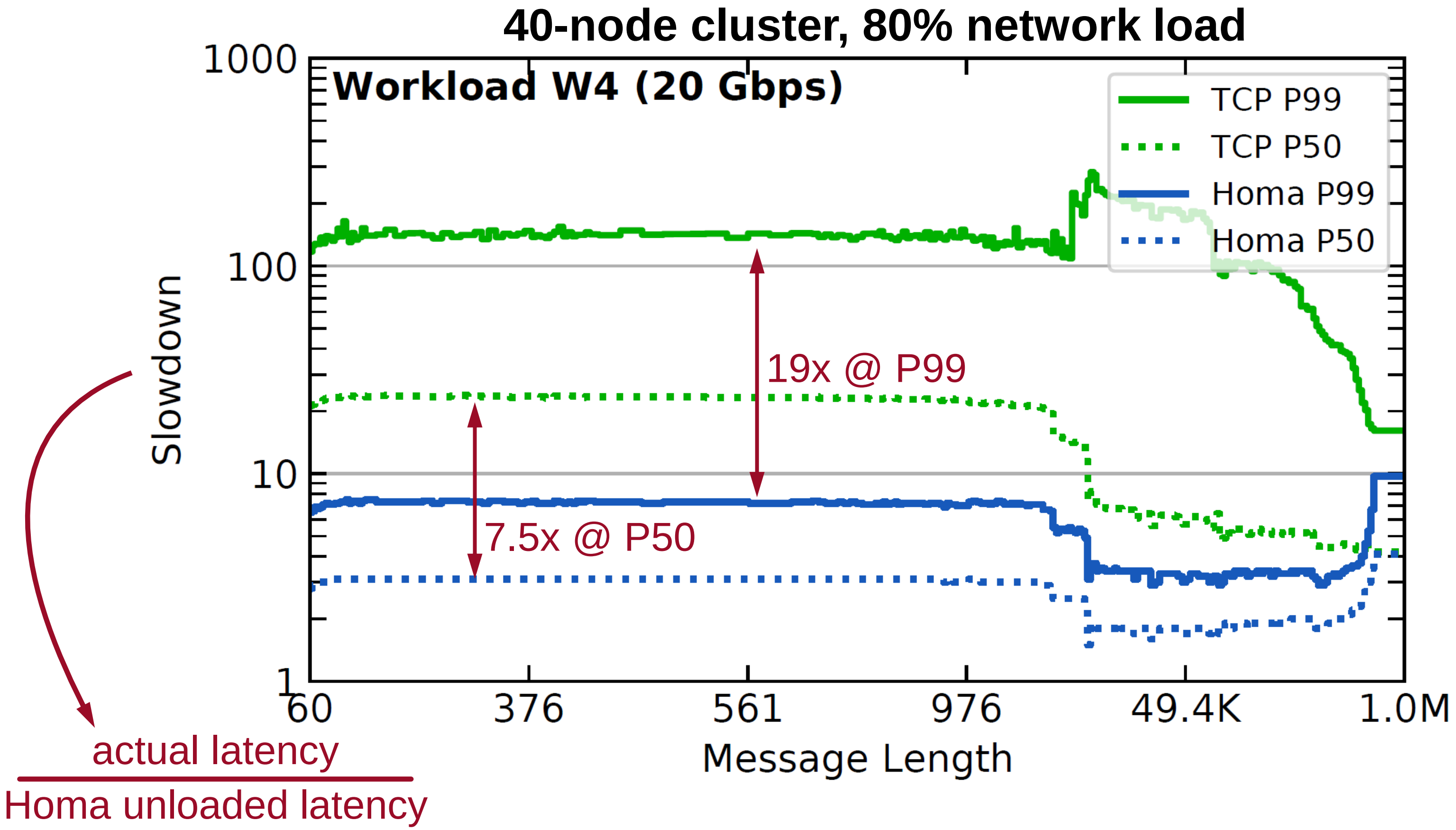
Homa Protocol

- **Goal: lowest possible latency**
  - Especially for short messages
  - Especially at the tail
  - Even under high network load
- **SRPT (Shortest Remaining Processing Time first)**
  - Best latency for short messages
  - Also benefits long messages! (run to completion)
  - Implemented using in-network priority queues
- **Receiver-driven congestion control**
  - First packets of message sent unilaterally (unscheduled)
  - Later packets sent in response to grants from receiver (scheduled)
  - Receiver determines packet priorities
- **Packets need not be received in order**

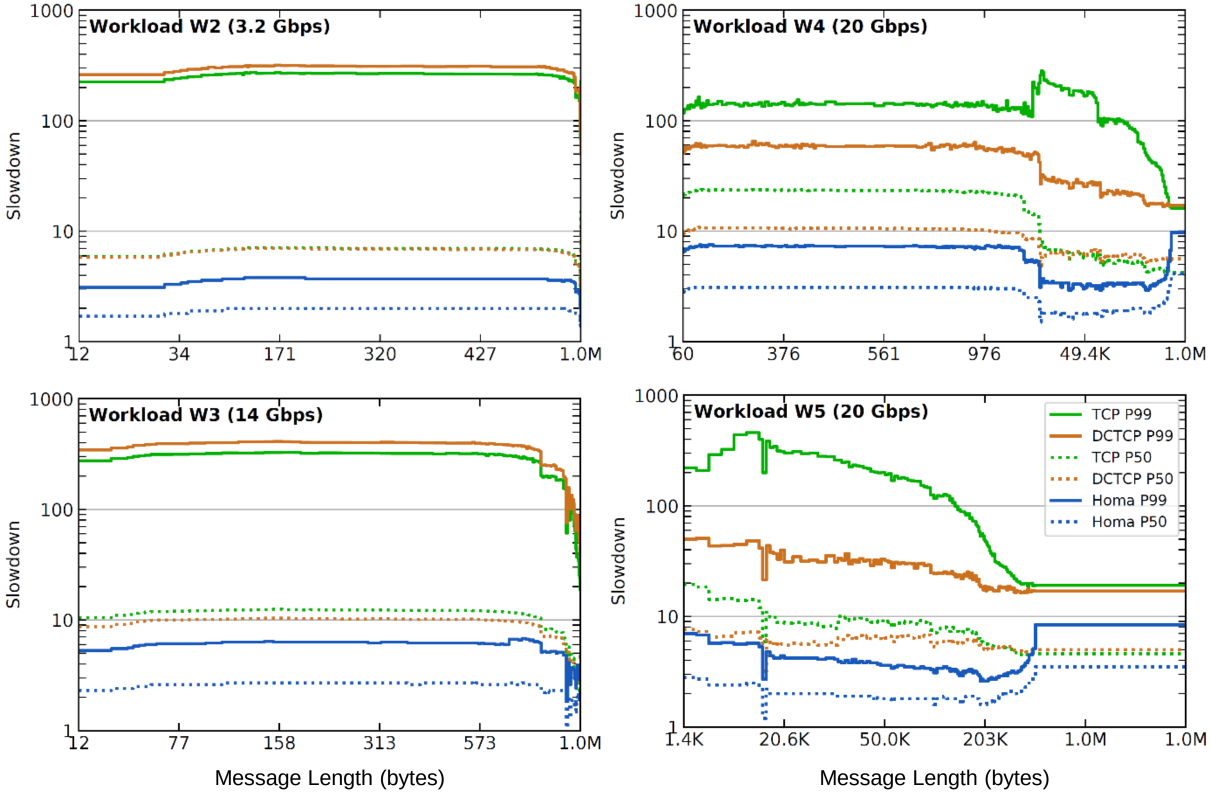
Homa/Linux

- **Dynamically loadable kernel module**
- **No kernel modifications required**
  - New system calls layered on ioctl
- **Open source: `git@github.com:PlatformLab/HomaModule.git`**
- **Currently runs on Linux 5.4.80**
- **About 10,000 lines C code (heavily commented)**
- **At or close to production quality**

Homa Latency << TCP



Homa Dominates: All Workloads, All Sizes



- **Homa's latency improvement for short messages:**

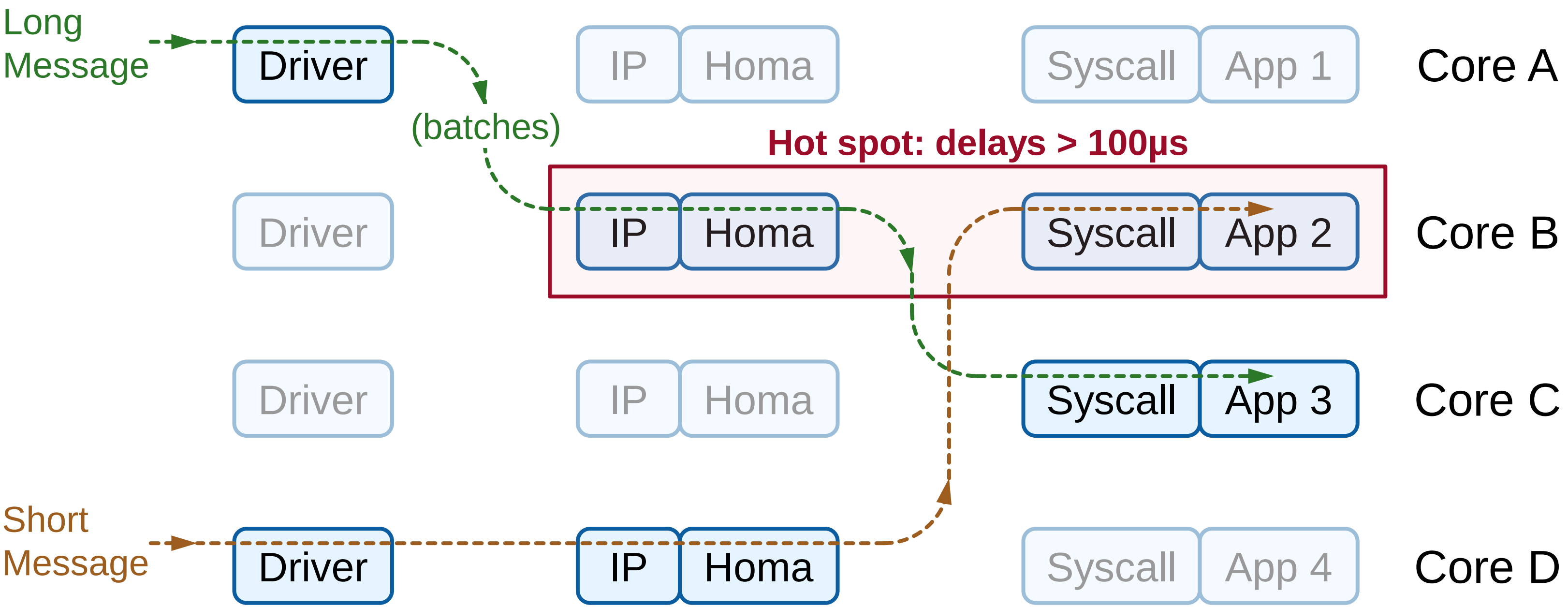
	P50	P99
vs TCP	3.5–7.5x	19–72x
vs DCTCP	2.7–3.8x	7–83x

- **P99 for Homa is better than P50 for TCP/DCTCP almost everywhere**
- **Homa eliminates congestion**

Software Overheads

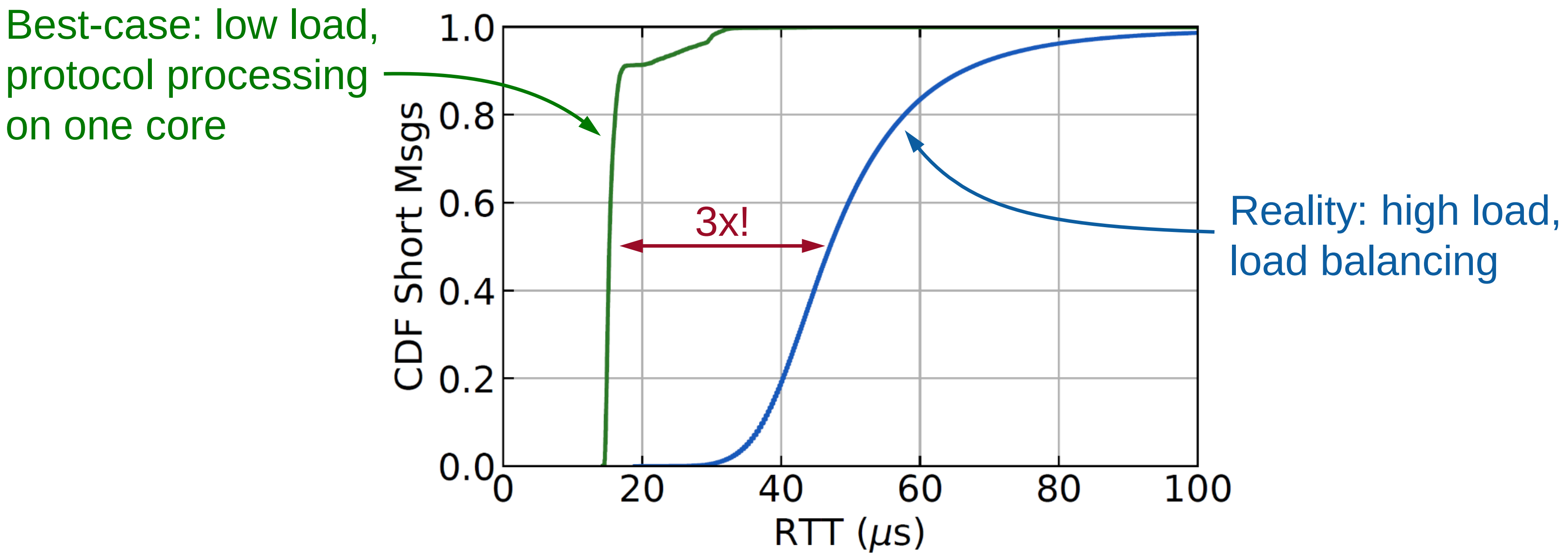
- **Homa/Linux performance still 5–10x worse than hardware potential:**
  - Small-message P99:
    Homa/Linux: 100 µs
    Homa/RAMCLoud: 14 µs (user space, kernel bypass)
- **Tail latency now caused by software overheads**
- **Load-balancing is problematic:**
  - Networks getting faster, CPUs aren't
  - Must distribute packet processing across many cores
- **Move protocols to user space? Won't help much**

Load Balancing Causes Hot Spots



Primary source of tail latency in Homa/Linux

2–3x Overhead for Load Balancing



Likely cause: cache interference

Move Transports to User Space?

- **Small-message P50 RTT:**
  - Homa/Linux: 12.6–38 µs
  - Homa/RAMCloud: 4.7 µs
  - eRPC: 3.7 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

Most user-space transports unrealistic:

- **Measured under ideal conditions**
- **No load balancing (or hand-partitioned)**
- **Unrealistic workloads: only short or long messages**
- **No congestion control**
- **Assume no shared protocol state between apps**

Many Homa overheads are inevitable

Homa/Linux vs. Snap

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

	Homa	Snap
Base latency (polling)	15.1 µs	9 µs
Cores to drive 80 Gbps bidirectional	17	7–14

- **Snap also suffers from load-balancing problems:**
  - Throughput per core drops by 3.5–7x

User-space protocols are not a long-term solution

Transports in the NIC?

- **All packet processing must move to the NIC**
  - CPUs deal only in messages
  - NIC dispatches messages directly to applications via kernel bypass
- **No existing approach is adequate:**
  - RDMA NICs: poor congestion control/load balancing, closed/proprietary
  - Many-core “Smart NICs”: just software processing in a different place
  - FPGA “Smart NICs”: too hard to program
  - P4 pipelines: no long-term state
- **Need a new NIC architecture**
  - Process packets at line rate
  - Programmable to support many protocols and functions
  - Interesting/difficult design challenge

TCP: Wrong for Datacenters In Every Way

- **Connection oriented**
  - High time/space overheads (datacenter apps have 1000's of connections)
- **Stream oriented**
  - Awkward for RPCs (transport doesn't know message boundaries)
  - Head-of-line blocking
- **Fair sharing of bandwidth**
  - Increases latency, especially for short messages
- **Sender-driven congestion control**
  - Requires buffer occupancy to detect congestion
  - Buffer occupancy → high latency
- **Requires in-order packet delivery**
  - Cripples load balancing

Conclusion

- **Homa/Linux confirms earlier results:**
  - Tail latency 10x better than TCP/DCTCP
  - Network congestion eliminated
- **Limitation going forward: software overheads**
  - Especially related to load balancing
- **Need radical changes in transport protocols:**
  - Move transport protocols to new NIC architectures
  - Replace TCP
- **Interested in users for Homa/Linux!**

Contact: ouster@cs.stanford.edu