# Homa vs TCP

Kunal Daga, Casper Guo, Dylan Mitek, Yuanzhe Liu

# Motivation

In 2018, John Outserhout published the paper: *Homa: a receiver-driven low-latency transport protocol using network priorities.* This paper introduced Homa, a Transport Protocol specialized for use in data center environments. Homa aimed to fix some of its perceived issues with TCP under data center conditions but some of its underlying assumptions spawned critiques such as Ivan Pepelnjak. In the blog post “Is It Time to Replace TCP in Data Centers?”, Pepalnjak argues that TCP isn’t necessarily worse than Homa in data centers, Ousterhout made several mischaracterizations and unfair comparisons, and TCP canbe configured to perform similarly to Homa.

Our project aims to look into both sides of the argument. We run Cloudlab experiments testing Homa against custom TCP configurations that are modified to perform better against certain workloads to see if we can match Homa

# The Protocols

## TCP

TCP is a common transport protocol primarily characterized by the following properties:

- *Connection-oriented* **–** TCP starts a connection between 2 servers, initiated with a handshake, and once that connection is up, packets can be sent down it for as long as it's open. Typically, in datacenters we establish a large number of connections all at once at startup and reuse those connections throughout the application lifetime.
- *Stream-oriented* – data is sent in a constant stream of bytes
- *Sender-driven Congestion Contro*l– When a packet gets dropped due to congestion, it's up to the sender to recognize this and resend the packets.
- *In-order Delivery* – Packets are delivered in the order that they are sent

## Homa

Homa differentiates itself from TCP by making a few distinct changes in its scheme:

- *Connectionless* – packets do not need to be sent down connections, firing packets between servers as needed.
- *Message-Based* – packets are distinctly grouped by whole messages rather than a stream. This allows for prioritization, like scheduling.
- *SRPT scheduling approximation* – It prioritizes the smaller packets first, aka the ones with shorter processing times, over the larger packets. This is opposed to TCP’s fair

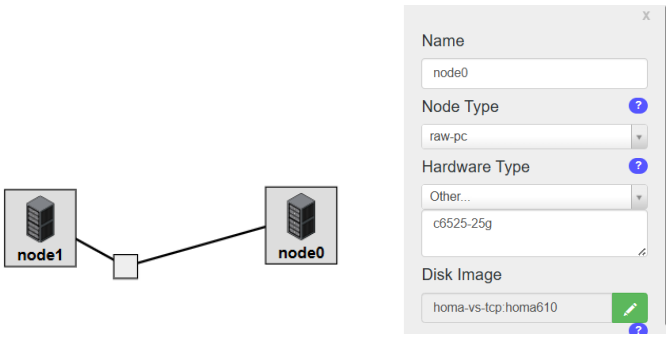
scheduling, which prioritizes all messages equally, giving small messages the same priority as larger ones.

- *Receiver Driven* – packets are only sent when receivers explicitly grant a request for data transfer via GRANT packets.
- *Out-of-Order Delivery* – A major complaint with TCP Homa was its In-Order delivery, potentially causing Head-of-Line blocking, where a single large message on a connection prevents any of the smaller connections from getting through.

# Experiment Setup

## Cloud-lab Setup

Our Cloud-lab topology consists of 2 nodes: a server and a client. While originally we intended to include 4 nodes, we struggled to get Homa's kernel functioning and decided to downgrade to 2 Nodes. Each node ran on C6525-25g with custom disk images (mostly for the correct Linux version).




On each node, we downloaded a Homa Linux Kernel implementation found in the paper: *A Linux Kernel Implementation of the Homa Transport Protocol* [github] and edited the Linux kernel's TCP protocol for our modifications.

# Workloads

## 1) Small RPC-Style Messages

### Workload Definition:

1. Fixed-Length Messages:
   We defined a set of fixed message sizes: **200B, 400B, 800B, and 1000B**. Each experiment was conducted using a single fixed size to isolate the transport protocol's behavior under tightly controlled conditions. This setting mimics microservice-style RPCs where payload formats are standardized and predictable.

2. Varied-Length Messages:
   To simulate more realistic RPC behavior, we generated messages with random sizes ranging from **10 bytes to 1 megabyte**. This reflects diverse RPC scenarios involving varying metadata, parameter sizes, and result payloads. Notably, **80% of the messages** in this configuration were kept below **1.8KB**, emphasizing short message performance, which is critical for latency-sensitive applications.

## TCP Modifications for Fair Comparison with Homa:

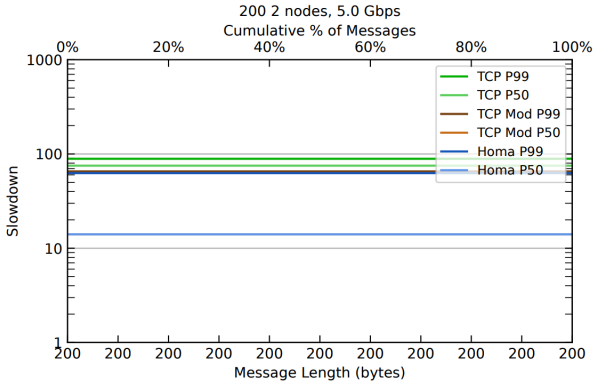
To ensure a meaningful comparison with Homa, which is inherently optimized for low-latency short message communication, we applied several optimizations to the TCP configuration:

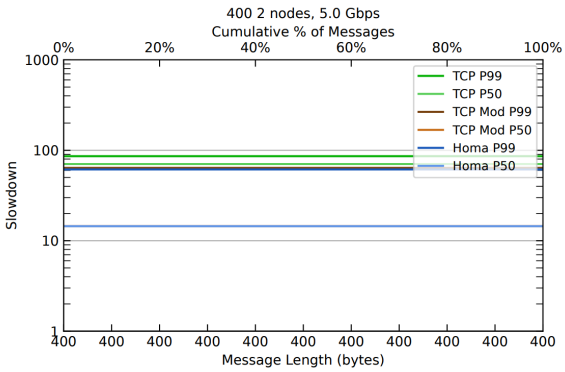
- **Low Latency Mode Enabled**: We bypassed traditional TCP delay mechanisms (such as delayed ACKs) to better support short-lived flows and small messages.
- **Socket Buffer Tuning**:
  - Increased **receive and send buffer sizes** to handle higher throughput and absorb traffic bursts effectively.
  - Allowed TCP to **dynamically scale buffer usage** during high traffic loads.
- **Retransmission Timeout Adjustment**:
  - Lowered the timeout threshold to reduce delays in detecting and recovering from packet loss.
- **DCTCP with ECN Support**:
  - Enabled **Data Center TCP (DCTCP)**, which leverages **Explicit Congestion Notification (ECN)** for more granular congestion feedback and better control over queue buildup and latency.

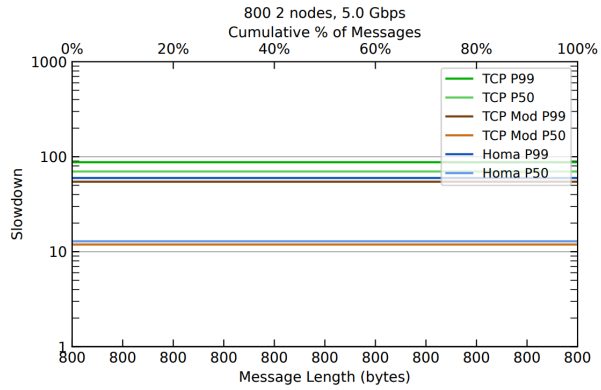
These adjustments collectively helped TCP handle short messages more effectively and provided a more balanced baseline for evaluating the advantages of Homa in mixed and short-message-heavy workloads.

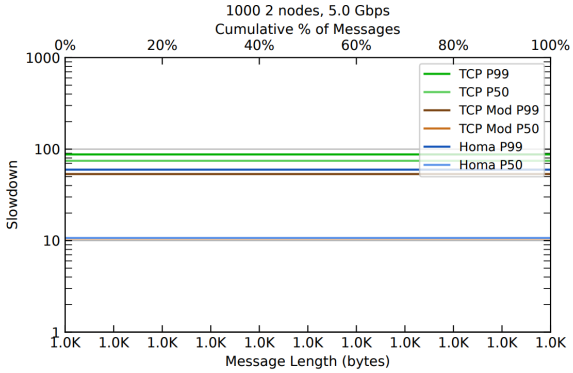
## Performance Result:

Fixed message length:

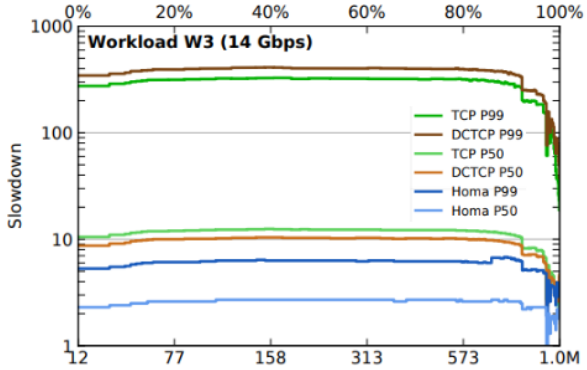


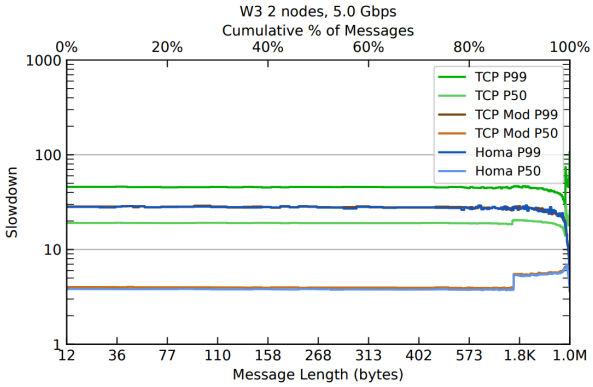






Varied Message Lengths:




## Observation:

The above figure presented at the left is the original result from the Homa paper using 40 nodes and a 14 Gbps network. On the right is our experimental replication using 2 nodes and a 5 Gbps link.

In the Homa paper(left chart):

- Homa significantly outperforms TCP, especially for short messages(which make up the majority of the workload)
- TCP experienced slowdown at both the P50 and P99(tail) latencies.
- The performance gap remains consistent across the range of message sizes until the largest messages.

In our replication(right chart):

- We observe similar trends, but with a great improvement in modified TCP performance:
    - TCP Mod P50 and P99 closely track the Homa curves for most message sizes
    - This means our modified TCP stack, with low latency mode, tuned buffer sizes, lowered retransmission timeout, ECN enabled congestion control, is improving the performance.
- While Homa still shows slightly better P99 behavior in extreme tail cases, the gap is significantly narrower than in the original paper.

## Potential implications:

This result demonstrates that TCP's performance of small message workloads can be improved with careful configuration, potentially reducing the performance gap between TCP and Homa, particularly in low-node, medium bandwidth environments(2-nodes, 5Gbps). Although this setup may not be the typical case for the modern datacenter. These findings highlight the value of protocol tuning and congestion feedback mechanisms when low latency is critical.

# 2) Bulk Data Transfers

## Background

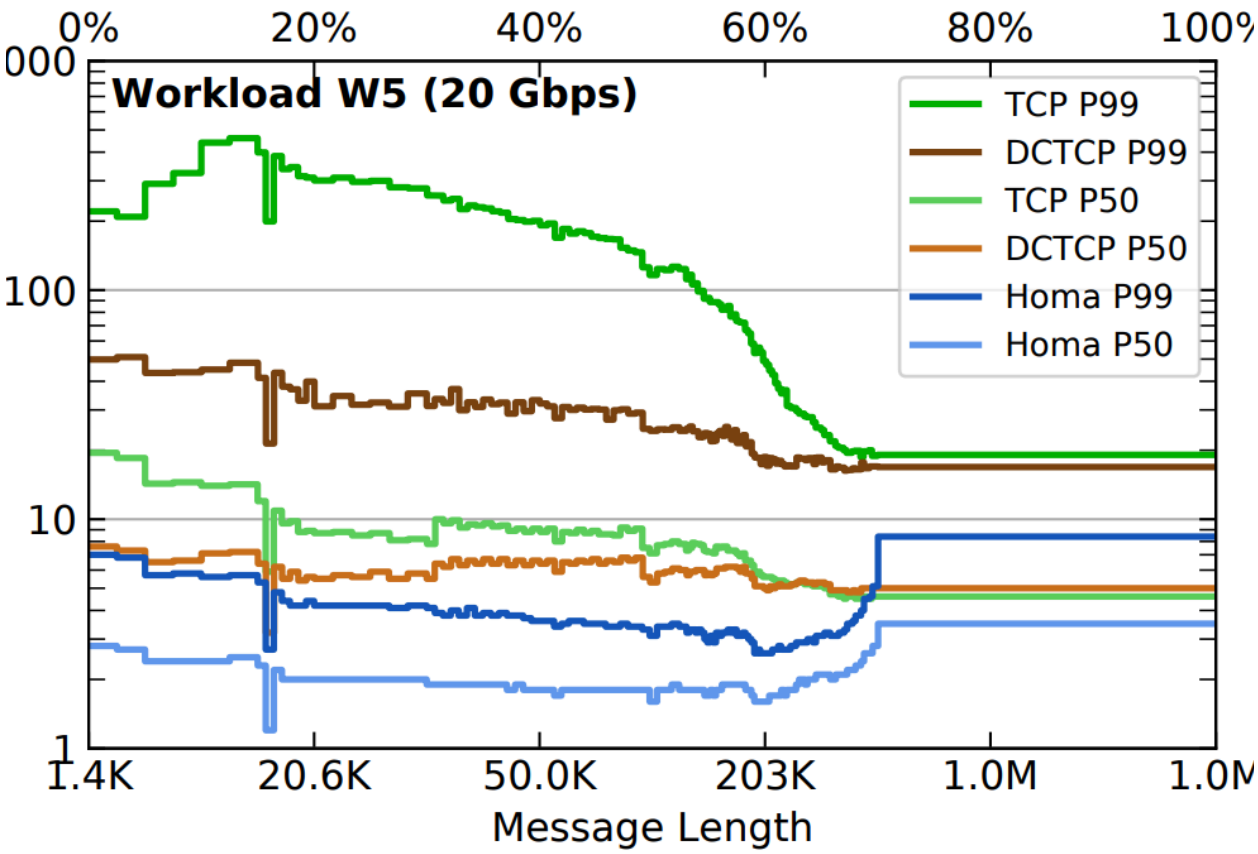
To preface this section, we acknowledge that Homa "aims to provide the lowest possible tail latency for short messages" (Ousterhout 2021) because it is designed around the observation that "many datacenter applications use request-response protocols that are dominated by very short messages (a few hundred bytes or less)" (Montazeri 2018). This assumption is met with some pushback from Pepelnjak, who says in his blog post "Is It Time to Replace TCP in Data Centers?": long-lived high-volume connections represent the majority of traffic in most data centers.

There is a fundamental divide between how the two authors characterize data center traffic, and our experiments show that their differing views are reflected in the respective strengths of Homa and TCP.

## Baseline Workload

The Homa Linux kernel by Ousterhout benchmarks Homa against TCP and DCTCP using four different workloads, named W2 through W5 by the average message length. The W5 workload, with an average message length of 385315 bytes, is the workload that most closely simulates bulk data transfer workload and serves as an useful baseline measurement.

The plot below (and in the following sections) uses slowdown as a proxy metric for overall protocol performance (y-axis), where slowdown is defined as the RTT measured during the experiment, divided by the RTT observed during low load, ideal network conditions. The experiment uses 40 nodes, all acting as both clients and servers simultaneously, with the most heavily loaded node sending and receiving at 20 Gbps. In this figure, Homa's 99th percentile performance is significantly better than both TCP and DCTCP regardless of message length. However, as the message length increases, the 50th percentile performance gap between Homa and TCP/DCTCP shrinks.

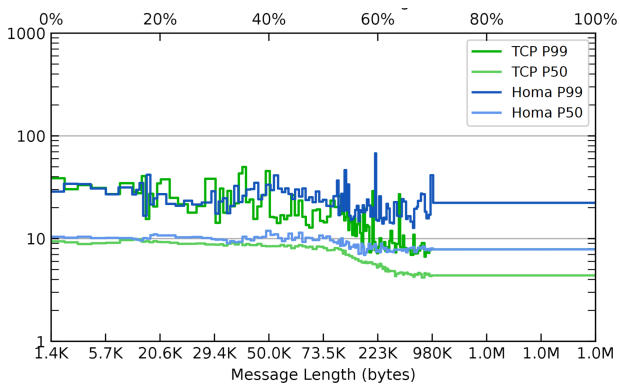


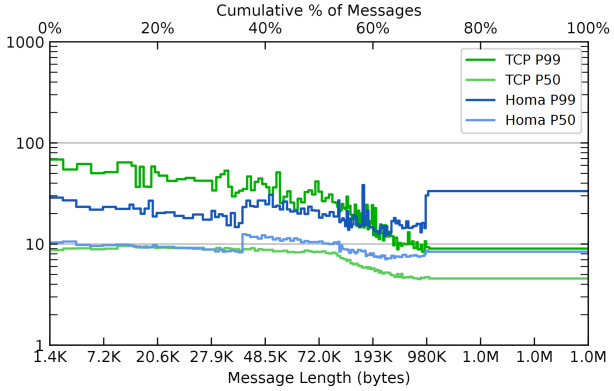

One issue with this workload is the skewness of the message length distribution. Despite claiming an average message length of 385315 bytes, the figure clearly shows that the median message length is substantially lower. This is an issue that we seek to remediate in our fixed message length experiments.

## Variable Message Length Experiment

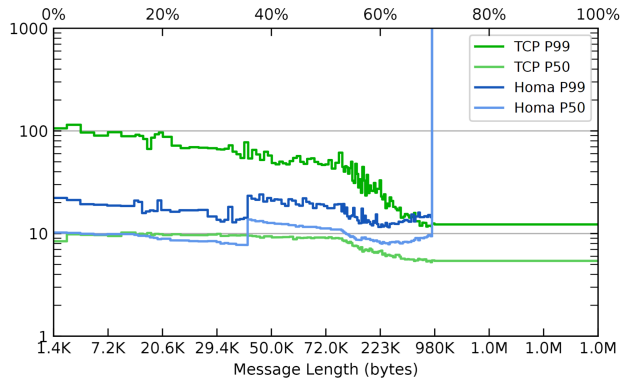
Using our 2-node cloudlab setup, we exercise Homa and TCP using a similar workload but vary the traffic throughput released into the network. We run the experiment with 2 Gbps, 5 Gbps, and 10 Gbps of traffic. We also include the performance figure generated from our 20 Gbps experiment, although we suspect that this figure may not be representative given that we found neither protocol can sufficiently exploit the 25 Gbps of available bandwidth in separate performance trials.

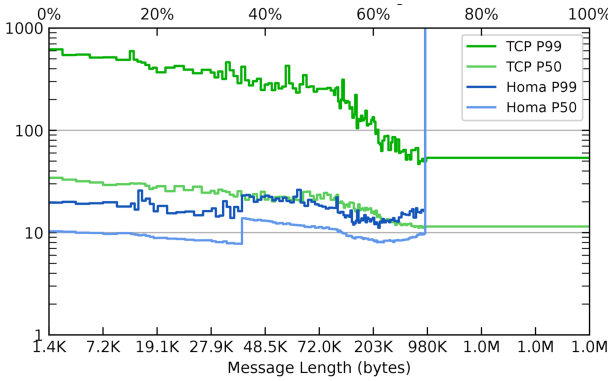
2 Gbps and 5 Gbps:




10 Gbps and 20 Gbps:



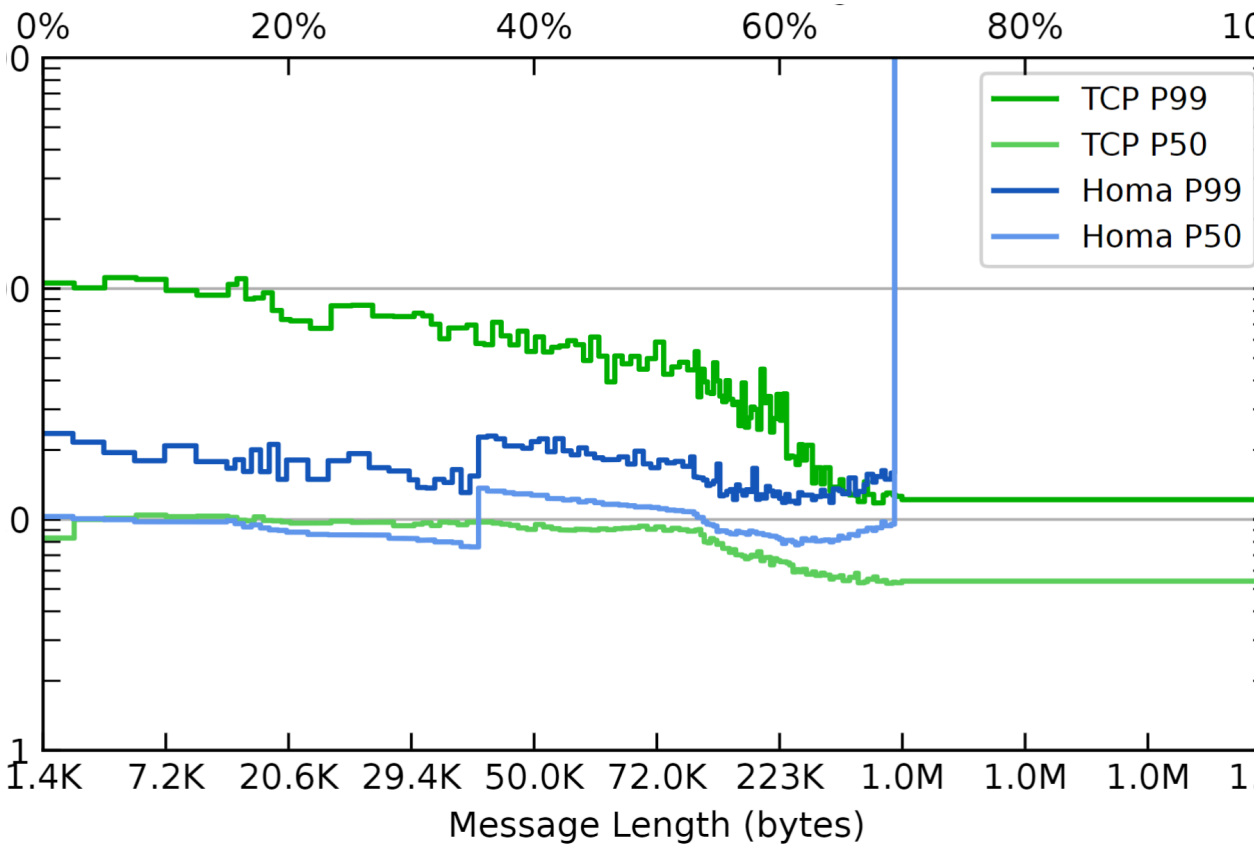

Our experiment shows that TCP frequently outperforms Homa in terms of median slowdown, especially for lower throughput and larger message sizes. In lower throughput conditions, TCP is able to match Homa closely in terms of 99th percentile slowdown. In high throughput conditions, the gap between TCP's and Homa's 99th percentile slowdown narrows as the message length increases. This is expected as Homa is more geared towards optimizing the tail latency of short messages, and our experiments confirm that it is largely successful in doing so.

## TCP Tuning

Given that TCP's median latency is competitive for the bulk data transfer workload, we focus our optimization efforts on improving its tail latency. We makes the following adjustments to the kernel networking parameters:

1. Increasing the size of send and receive buffers on both nodes to 128 MB
2. Enable TCP window scaling

By default, TCP uses the cubic AIMD algorithm for congestion control. We tried to combine the above configurations with the BBR congestion control algorithm, which was first proposed by Google in 2016 with an eye towards data center traffic. The below figure shows TCP's performance after these changes under the same workload at 10 Gbps throughput. Evidently, these optimizations do not significantly improve its tail latency.

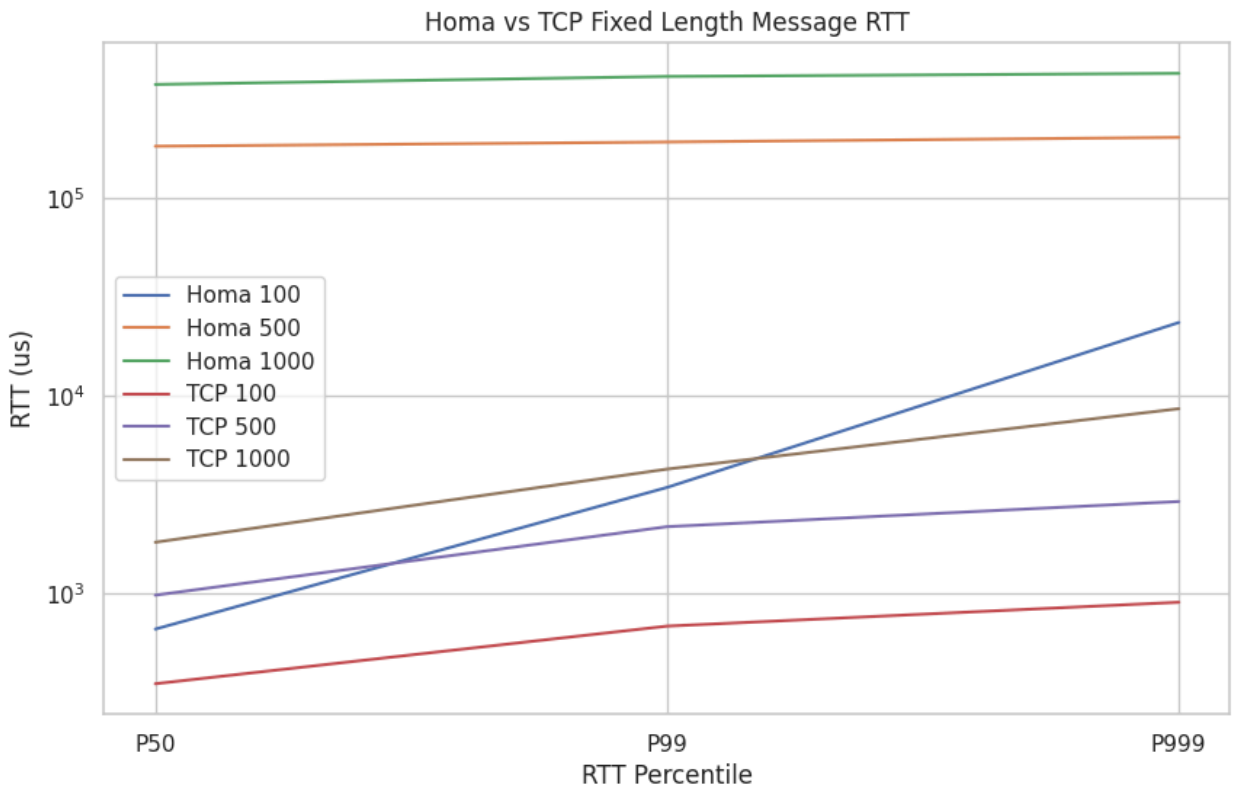



Variable message length workload at 10 Gbps, with BBR congestion control algorithm

We hypothesize that future gains can be made by disabling Nagle's algorithm via setting TCP_NODELAY = 1. This is a well-noted optimization method for data center traffic; in our research, multiple authors have independently advocated for this configuration in high bandwidth, low latency networks.

## Fixed Message Length Experiment

As discussed previously, the W5 workload in the Homa Linux kernel paper emits messages whose lengths are sampled from a distribution. We additionally subject Homa and TCP to large, fixed message length traffic and observe the resulting slowdown. We found in these conditions, TCP significantly outperforms Homa by up to an order of magnitude. We initially suspected that these results are faulty, but running the same experiment several times across different Cloudlab hardware all returned similar performance differentials.

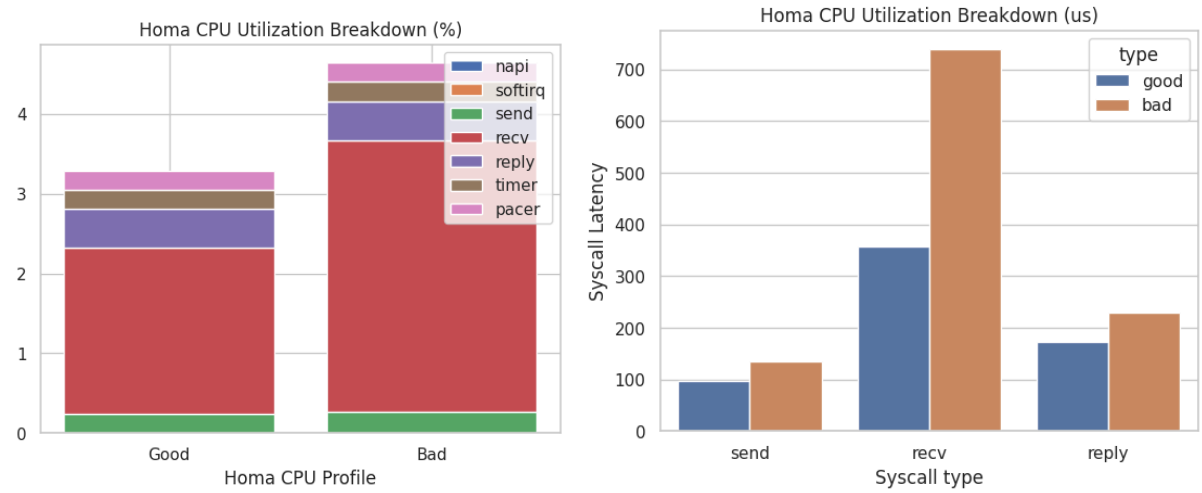



Simulated traffic throughput of 10 Gbps

## Analysis

The benchmarking suite found in the Homa Linux kernel Github repository produces detailed CPU core utilization breakdown, syscall latencies, along with a host of other network and CPU diagnostics metrics. We cross-examined these logs from the variable length 10 Gbps and fixed length 10 Gbps experiment to identify the Homa performance bottlenecks.

We observe that in the fixed length message case, Homa utilizes more CPU resources despite the network throughput being equal. We also note that the latency for the `recv` system call is significantly higher.




Given these patterns, we have some hypotheses regarding the cause for Homa's poor performance with large, fixed length messages.

- Homa uses pacer threads to regulate the length of NIC transmit queues. The pacer thread passes packets to NIC in SRPT order. The pacer thread can fall behind and/or be descheduled. If either happens, microsecond-scale delays can be introduced. Such events may be more likely under increased core utilization
- The large `recv` system call latency may be a result of RPC reaping. Reaping large RPCs in the form of freeing packet buffers can take many microseconds as well

### Caveat

Although Homa's performance is unsatisfactory for this workload, it may be due to us not fully exploiting all available optimization options. For example, Homa recommends enabling jumbo frames, which we could not get working reliably on Cloudlab. Regardless, we suspect that jumbo frames will enhance TCP and Homa performance similarly. Homa can also use TCP segmentation offload to more effectively deal with large messages, but our NIC is not on the list of devices compatible with this protocol.

## 3) High Package Drop Rate

### Workload Definition:

Our third workload was to use mixed message sizes - specifically, 80% small RPC-style messages (~1KB) and 20% large data transfer-style messages (~1MB). This was tested under two different traffic modes: bursty traffic with short spikes of transfers and random traffic that induced a steady stream of messages. Lastly, we used the inbuilt Linux `tc` command to simulate packet drop conditions varying from [0, 2, 5, 10, 20] to see how both protocols would react.

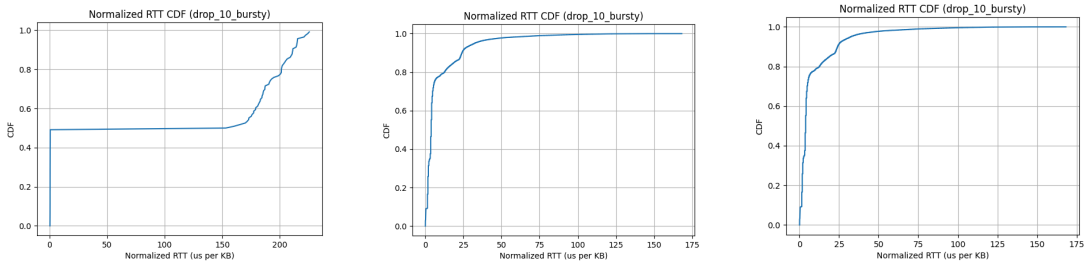
The workload was used to compare TCP, Homa, and TCP-mod, a modification to TCP with some adjusted parameters chosen to improve performance on this workload. These parameters are:

- Changed Buffer Sizes:
    - Increased the socket buffer so that TCP could better handle bursty traffic and would try to match Homa's relentless throughput-oriented design
    - Increased read and write buffers to allow TCP to dynamically scale up during higher loads, aiming to once again promote throughput to match Homa.
- Modified Retransmission Timeout:
    - Lowered the threshold for giving up on retransmission attempts from 15 to 5, to address the fairly high levels of packet loss being introduced into the network. Broken flows are identified and discarded quicker than before.

The provided files, like `cp_vs_tcp` or `cp_basic`, in the HomaModule repository did not have this particular workload, so Homa was configured using the provided `cp_node.cc` to create a new workload that would either randomly generate messages from the client or follow a bursty pattern and generate a set of messages intermittently.

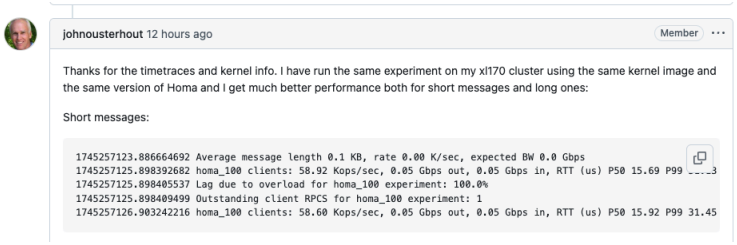
## Results:

### Effect of Burstiness on Homa and TCP:




*Left: Homa, Middle: Base TCP, Right: Modified TCP*
*Normalised RTT CDF for the [bursty, 10% packet drop] simulation*

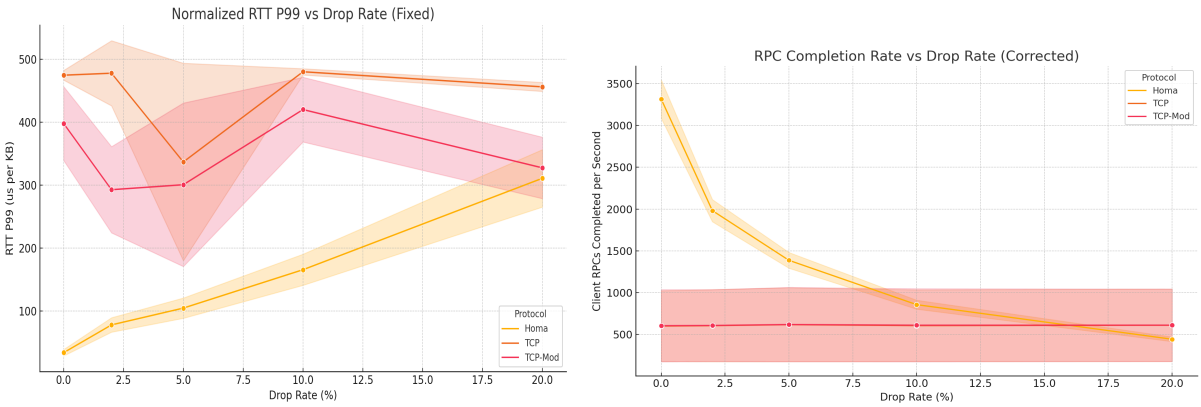
While TCP and TCP-mod show tighter RTT distributions and in general lower RTTs, Homa's CDF is much flatter, with ~P50 messages taking negligible time as compared to both P99 packets in Homa, and any packets in TCP as evidenced by the almost vertical line at the beginning. A potential hypothesis is that packets, especially larger ones which are deprioritised by Homa due to the SRTF heuristic, were experiencing prolonged queuing under overload. This led to considerable amounts of lag when sending Homa messages. Even Ousterhout's shared logs show a similar lag on his setup when trying to resolve a GitHub issue we created, but we did not get the chance to ask him about that.




The flat horizontal portion of the CDF could also be exaggerated because Homa doesn't drop messages, but rather defers them, which shifts the latency curve even though throughput stays high. So Homa appears slower here, not due to loss or failure, but more so due to its fairness, or lack of, and scheduling under both network and potential hardware constraints.

Here, our results align more with Ousterhout in that Homa is superior for latency even under some loss, while Pepelnjak's view explains why TCP remains viable in non-ideal networks.

## Comparing Throughput:




*Left: Normalized RTT for P99s vs Drop Rates, Right: RPC Completion Rate vs Drop Rates*

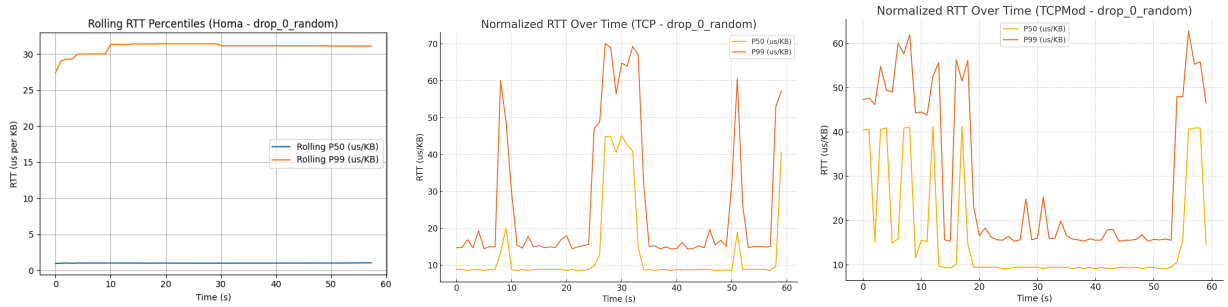
Despite showing higher latencies on burstier loads, when aggregating data across all experiment runs and across the 2 traffic conditions, Homa maintained lower average latencies for all drop rates, even up to 20% drops, while it just barely dropped below in terms of throughput by the 20% drop scenario. TCP and TCP-mod data fluctuated significantly, as seen in the wide interval range, compared to Homa's tighter spread, but in general had higher yet more stable latencies than Homa did. While Homa's performance had a downward trajectory as drop rate increased, TCP did not conclusively show such a pattern. If we had more time, we would have liked to run this experiment on higher packet drops to determine if TCP-mod would eventually overtake Homa for performance. This lines up with our theoretical understanding since TCP has extensive packet drop coverage, whereas Homa does not. In fact, Homa assumes low packet loss rates due to its localised usage in data centers where hardware and switches can guarantee high reliability. So while this may be an unfair comparison, in cases

where the aforementioned reliability is not guaranteed, Homa will suffer due to its lack of recovery mechanisms.

We also observed that TCP and TCP-mod throughput remained relatively flat across all drop rates, even at 0%. This highlights a key limitation of TCP in low-concurrency environments: with a limited number of in-flight RPCs, the protocol is unable to fully utilize available bandwidth. Because of hardware issues, we were unable to set up nodes with enough bandwidth to make use of more ports than 4 on Homa, and in aiming for fairness, we therefore only allowed 4 ports for TCP. Since Homa uses multi-port concurrency, it scaled up more effectively, maintaining high throughput, whereas TCP stayed stagnant due to its slower but more constant loss-handling mechanisms.

Our results validate some of Pepelnjak's concerns in lossy conditions. While Homa maintains throughput and average latency, TCP's consistent tail behavior under drop supports his view that it's more reliable in unpredictable networks.

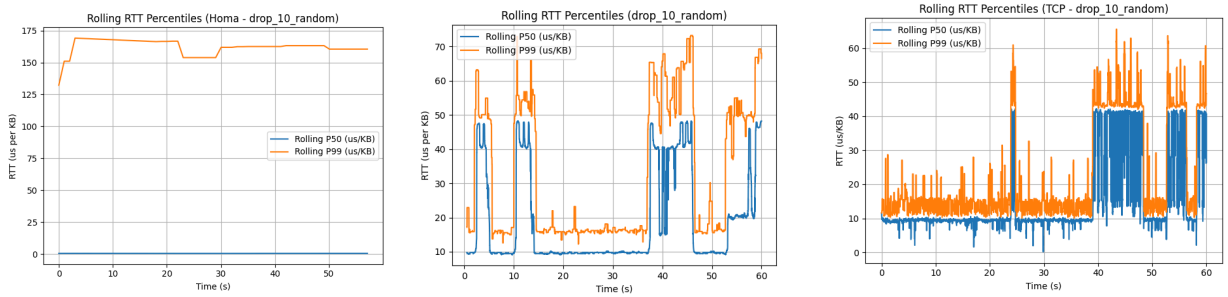
## RTT behavior under healthy conditions:




*Left: Homa, Middle: Base TCP, Right: TCP-Mod*
*Normalised RTTs for the [random, 0% packet drop] simulation*

The graphs above verify the claims that Homa thrives in low packet drop conditions. Both P50 and P99 for Homa are flat and low, whereas the P50 and P99 for both TCPs are noisy, spiky, and on average much higher than Homa (even TCP-mod). This shows that Homa's queuing system and fairness policies shine under low-congestion conditions, yielding very stable latency.

## A note on HoL Blocking




*Left: Homa, Middle: Base TCP, Right: TCP-Mod*

*Normalised RTTs for the [random, 10% packet drop] simulation*

While Pepelnjak suggests modern TCP mitigates HoL blocking, our experiment shows that under moderate loss and random workloads, HoL is still quite evident. This can be seen in the TCP graphs, where both P50 and P90 RTTs fluctuate a lot, and often move together, which suggests that shorter messages are getting stuck behind longer ones at various times, causing a sizable drop in latency. In fact, TCP-mod, which uses larger buffers and more in-flight messages, appears to have an even more exaggerated HoL problem. This could be because there are now larger queues available, so a higher likelihood of smaller flows getting stuck behind larger flows.

On the other hand, while Homa has a sharp increase in latencies for P99 as compared to P50, this can be attributed to the phenomena discussed before with regards to overload-induced lag on hardware or queueing. What we also see is that both lines are flat, indicating that messages have to wait, but they are scheduled fairly. Due to its connectionless setup and multi-port concurrency, Homa can completely avoid HoL blocking issues.

# Conclusion

Our findings confirm some of Ousterhout's claims and reiterate that Homa heavily outperforms other protocols in the scenarios for which it was designed, namely large numbers of short messages. However, it does not offer as much flexibility as TCP. When outside of its ideal traffic conditions, Homa loses its edge to or is outperformed by TCP..

We concur with Ousterhout's original evaluation that Homa is preferable to TCP in scenarios with high burstiness and large volumes of small messages. TCP, even when tuned, starts to exhibit increased queuing delays and retransmission overhead under these conditions. Pepelnjak argued that TCP can be tuned to match Homa, but our results suggest that this holds true only to a limited extent—and not under extreme bursty conditions. The discrepancy likely stems from differences in test environments: Pepelnjak focused on theoretical scalability and mature TCP tuning in practical deployments, while our results emphasize latency-critical microbursts where Homa's design shines.

In summary, Homa is an excellent low-latency protocol for highly bursty, small-message, low-loss environments, validating Ousterhout's vision. TCP remains the more robust and flexible choice in lossy or bandwidth-heavy scenarios, as Pepelnjak proposed. The divergence in their conclusions can largely be attributed to differing assumptions about the underlying network—Ousterhout assumes high control and reliability, while Pepelnjak is more pragmatic about real-world imperfections. Our results suggest that the optimal protocol choice depends heavily on workload patterns and network reliability, and in some hybrid systems, a layered or adaptive transport strategy may be most effective.