SNIA DEVELOPER CONFERENCE

SDC 24

BY Developers FOR Developers

September 16-18, 2024
Santa Clara, CA

Complementing TCP with Homa

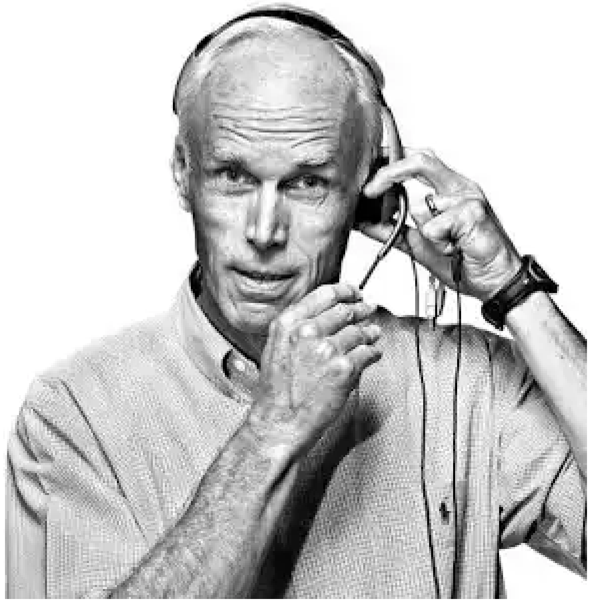
Stanford's Reliable, Rapid Request-Response Protocol

Endric Schubert, Ph.D. & Ulrich Langenbach

Presentation Outline

1. The Homa Protocol from John Ousterhout @Stanford University
2. Can Homa coexist peacefully with TCP/IP?
3. RRRRP - Homa, FPGA Accelerated
4. Call for Collaboration

Acknowledgements



John Ousterhout, Stanford University

Team MLE in Berlin and Neu-Ulm, Germany

Björn Petersen, Institute for Micro Electronics, Ulm University, Germany

Part 1

The Homa Protocol

Homa - Started by John Ousterhout et al.

It's Time to Replace TCP in the Datacenter

It's Time to Replace TCP in the Datacenter

John Ousterhout
Stanford University

January 18, 2023

This position paper has been updated since its original publication in October of 2022 in order to correct errors and add clarification. Updates are in italics; none of the original text has been modified. The paper has triggered discussion and dissent; for pointers to comments on the paper, see the Homa Wiki: `https://homa-transport.atlassian.net/wiki/spaces/HOMA/overview#replaceTcp`.

arXiv:2210.00714v2 [cs.NI] 19 Jan 2023

Abstract

In spite of its long and successful history, TCP is a poor transport protocol for modern datacenters. Every significant element of TCP, from its stream orientation to its expectation of in-order packet delivery, is wrong for the datacenter. It is time to recognize that TCP's problems are too fundamental and interrelated to be fixed; the only way to harness the full performance potential of modern networks is to introduce a new transport protocol into the datacenter. Homa demonstrates that it is possible to create a transport protocol that avoids all of TCP's problems. Although Homa is not API-compatible with TCP, it should be possible to bring it into widespread usage by integrating it with RPC frameworks.

1 Introduction

The TCP transport protocol [9] has proven to be phenomenally successful and adaptable. At the time of TCP's design in the late 1970's, there were only about 100 hosts attached to the existing ARPANET, and network links had speeds of tens of kilobits/second. Over the decades since then, the Internet has grown to billions of hosts and link speeds of 100 Gbit/second or more are commonplace, yet TCP continues to serve as the workhorse transport protocol for almost all applications. It is an extraordinary engineering achievement to have designed a mechanism that could survive such radical changes in underlying technology.

However, datacenter computing creates unprecedented challenges for TCP. The datacenter environment, with millions of cores in close proximity and individual applications harnessing thousands of machines that interact on microsecond timescales, could not have been envisioned by the designers of TCP, and TCP does not perform well in this environment. TCP is still the protocol of choice for most datacenter applications, but it introduces overheads on many levels, which limit application-level performance. For example, it is well-known that TCP suffers from high tail latency for short messages under mixed workloads [2]. TCP is a major contributor to the "datacenter tax" [3, 12], a collection of low-level overheads that consume a significant fraction of all processor cycles in datacenters.

This position paper argues that TCP's challenges in the datacenter are insurmountable. Section 3 discusses each of the major design decisions in TCP and demonstrates that every one of them is wrong for the datacenter, with significant negative consequences. Some of these problems have been discussed in the past, but it is instructive to see them all together in one place. TCP's problems impact systems at multiple levels, including the network, kernel software, and applications. One example is load balancing, which is essential in datacenters in order to process high loads concurrently. Load balancing did not exist at the time TCP was designed, and TCP interferes with load balancing both in the network and in software.

Section 4 argues that TCP cannot be fixed in an evolutionary fashion; there are too many problems and too many interlocking design decisions. Instead, we must find a way to introduce a radically different transport protocol into the datacenter. Section 5 discusses what a good transport protocol for datacenters should look like, using Homa [19, 21] as an example. Homa was designed in a clean-slate fashion to meet the needs of datacenter computing, and virtually every one of its major design decisions was made differently than for TCP. As a result, some problems, such as congestion in the network core fabric, are eliminated entirely. Other problems, such as congestion control and load balancing, become much easier to address. Homa demonstrates that it is possible to solve all of TCP's problems.

Complete replacement of TCP is unlikely anytime soon, due to its deeply entrenched status, but TCP can be displaced for many applications by integrating Homa into a small number of existing RPC frameworks such as gRPC [6]. With this approach, Homa's incompatible API will be visible only to framework developers and applications should be able to switch to Homa relatively easily.

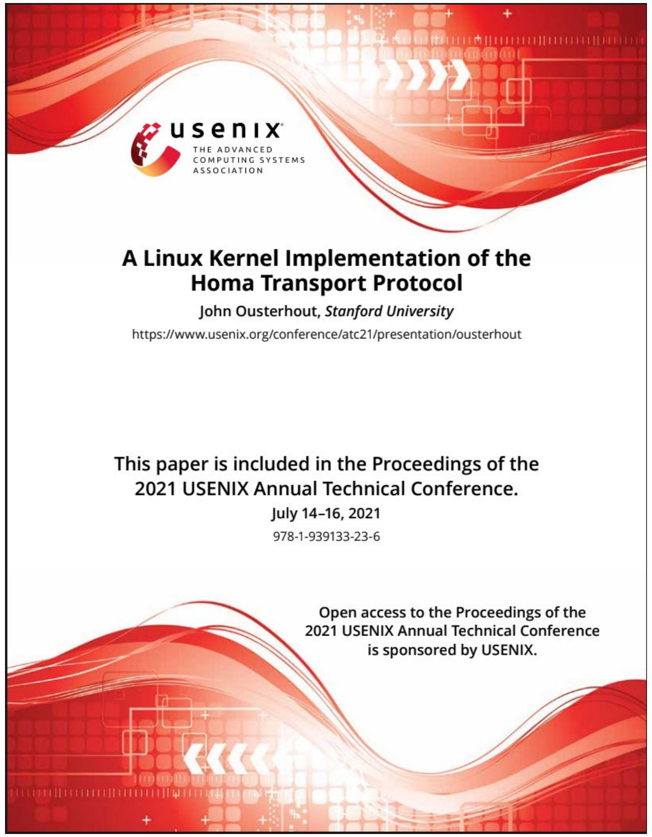
2 Requirements

Before discussing the problems with TCP, let us first review the challenges that must be addressed by any transport protocol for datacenters.

Reliable delivery. The protocol must deliver data reliably from one host to another, in spite of transient failures in the

1

A Linux Kernel Implementation of the Homa Transport Protocol



Homa Reduces Tail Latencies in Loaded Networks

- Experimental results
- 25 GigE Network
- Compares Linux kernel space implementation of
  - TCP/IPv4
  - Homa/IPv4
- X-axis is distribution of message mengths in workload
- Y-axis is Slowdown RTT_loaded / RTT_unloaded

Courtesy of John Ousterhout, Stanford University

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)**

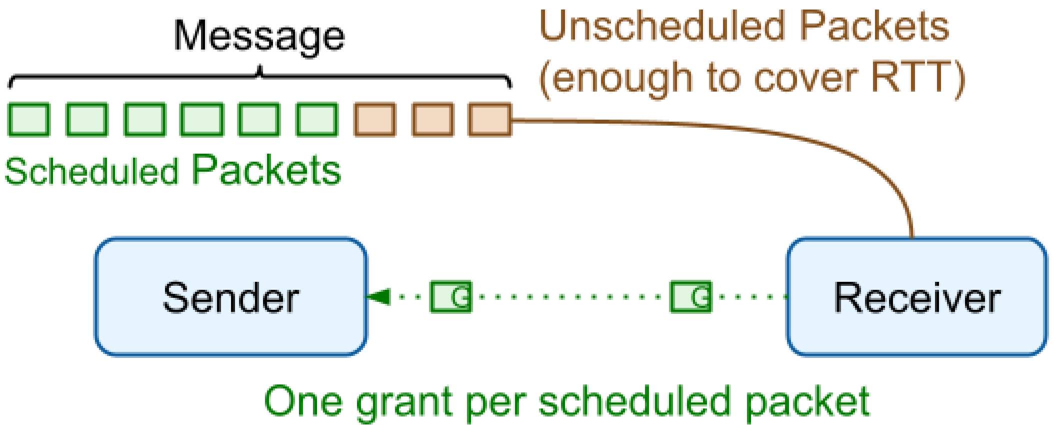
Courtesy of John Ousterhout, Stanford University

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

Courtesy of John Ousterhout, Stanford University

3. Homa: Receiver-Driven Congestion Control



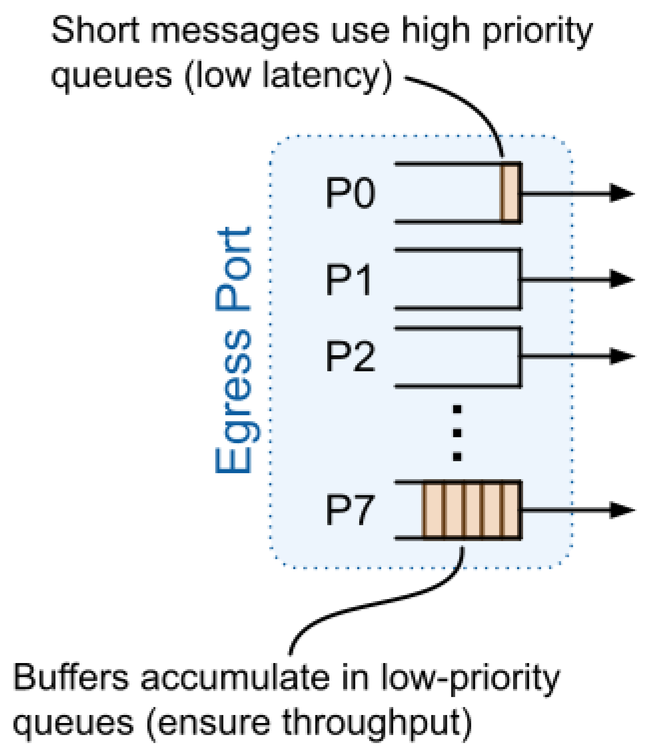
- **Receiver can delay grants to:**
  - Reduce congestion in TOR
  - Prioritize shorter messages
- **Message sizes allow receivers to predict the future:**
  - Faster, more accurate response to congestion

Courtesy of John Ousterhout, Stanford University

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

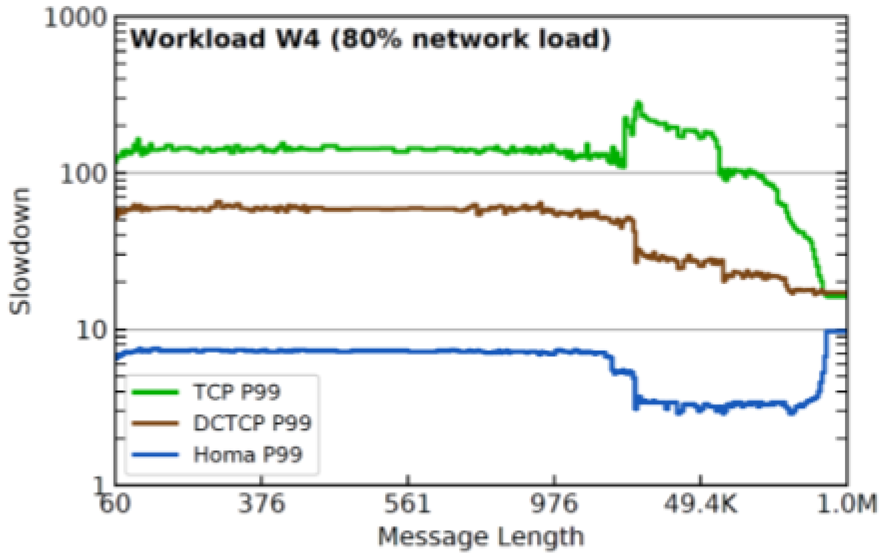
Overcommitment



Courtesy of John Ousterhout, Stanford University

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

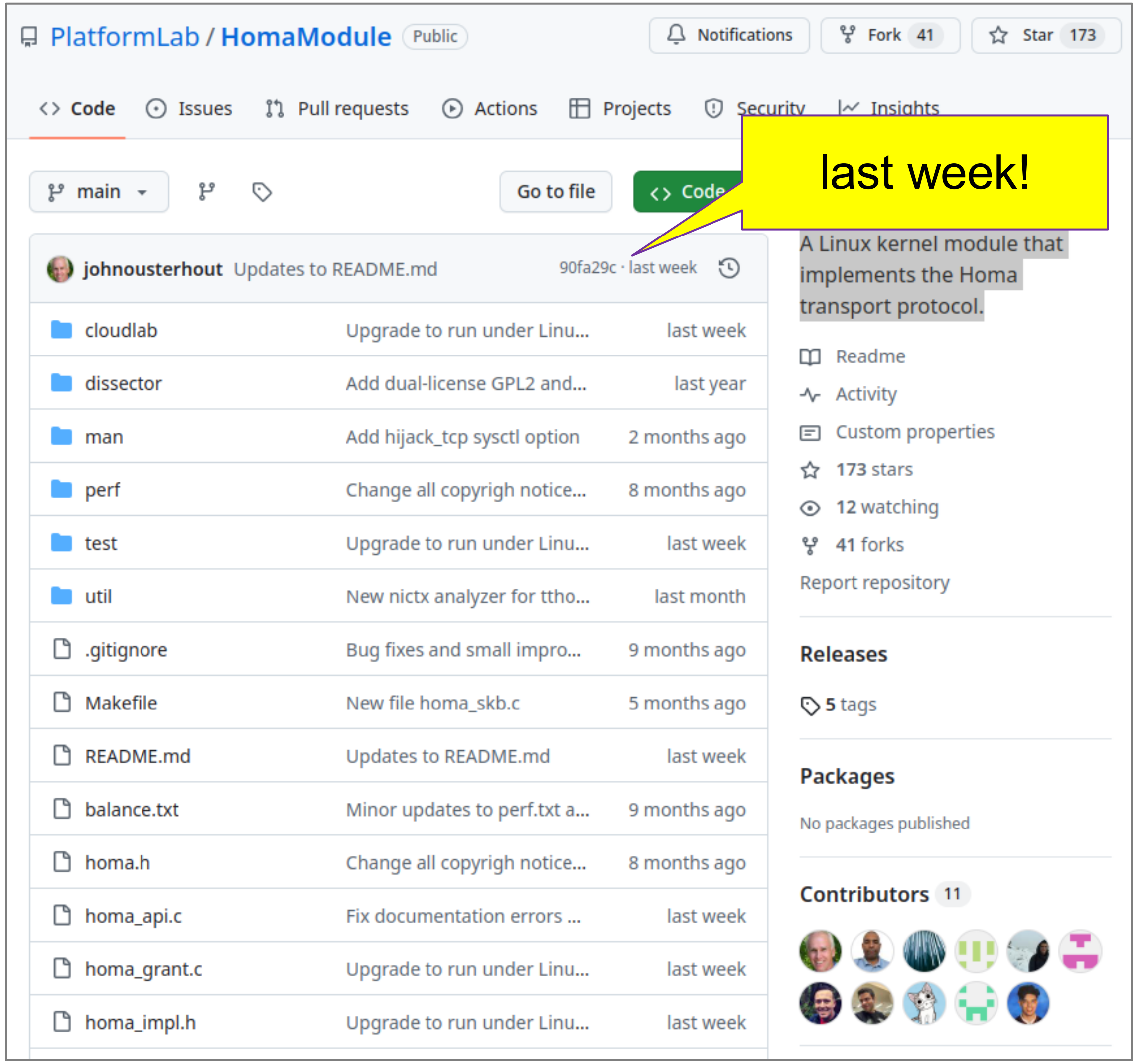


Courtesy of John Ousterhout, Stanford University

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**

HomaModule - Implemented as a Linux Kernel Module



Uses A-Priori Knowledge

- Link Rate between NIC and ToR switch
- NIC Queue Length (SRPT), i.e. "estimated time until Tx buffer is empty"
- Coexistence w/ other protocols
  Interaction with Tx pacer in Linux netdev NAPI
- Distance between machines
  Handling non-uniform RTT
- Priority Queues in Switches

Part 2

Can Homa coexist peacefully with TCP?

Experimental Test Setup (at Cloudlab xl170)

HW Setup

25 GigE network
4 or 12 nodes, each

Intel E5-2640v4

Mellanox ConnectX-4

SW Setup

HomaModule v2023-12-20

util/cp_vs_tcp tests in parallel

with Linux iperf

NW Traffic

util/cp_node for the
Homa vs TCP tests

RTT and Slowdown

plus additive TCP “background noise” via iperf (any-to-any)

Experimental Test Setup

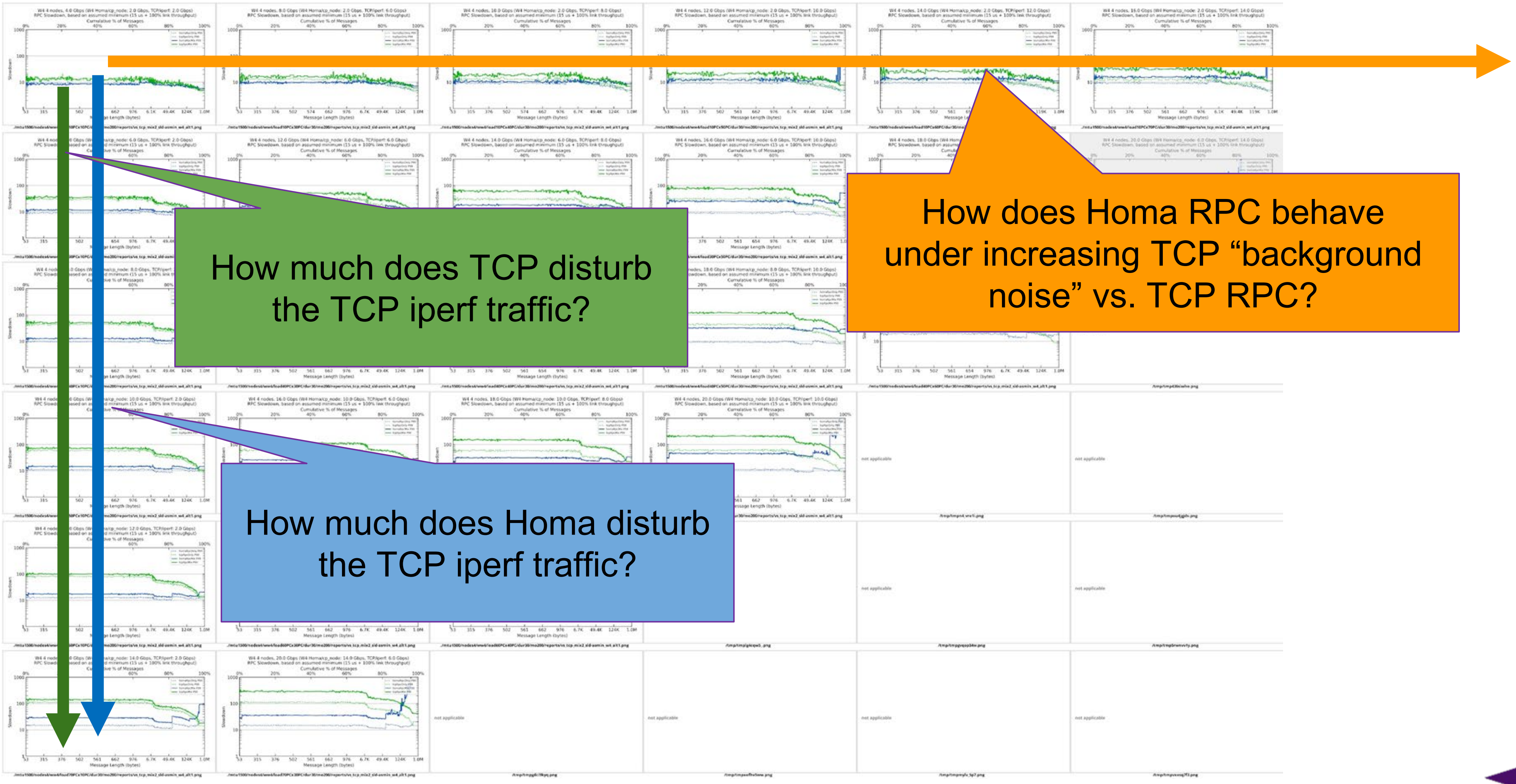
		(TCP) Stream target load *(using iperf)*					
	%	10	30	40	50	60	70
Homa vs TCP RPC target load *(using cp_node)*	10	W2, W3, W4,	W2, W3, W4,	W2, W3, W4,	W2, W	W2, W3, W4,	W2, W3, W4,
	30			W4, W5			5
	40	W3, W4, W5	W3, W4, W5	W3, W4, W5	W3, W4, W5	W3, W4, W5	N/A
	50	W3, W4, W5	W3, W4, W5	W3, W4, W5	W3, W4, W5	N/A	N/A
	60	W		W5	N/A	N/A	N/A
	70	W			N/A	N/A	N/A

How much does TCP RPC disturb the TCP iperf traffic?

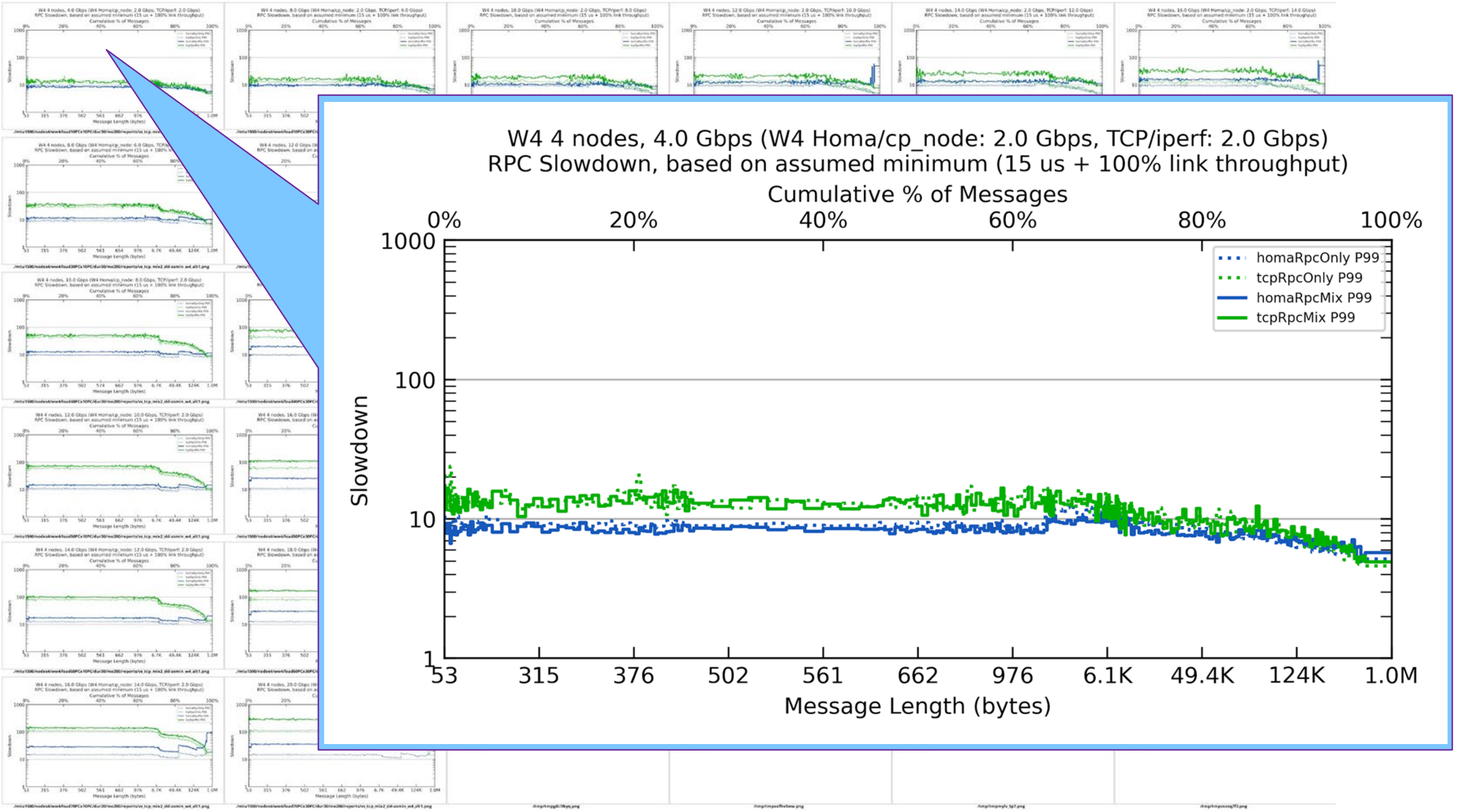
How does Homa RPC behave under increasing TCP "background noise" vs. TCP RPC?

How much does Homa RPC disturb the TCP iperf traffic?

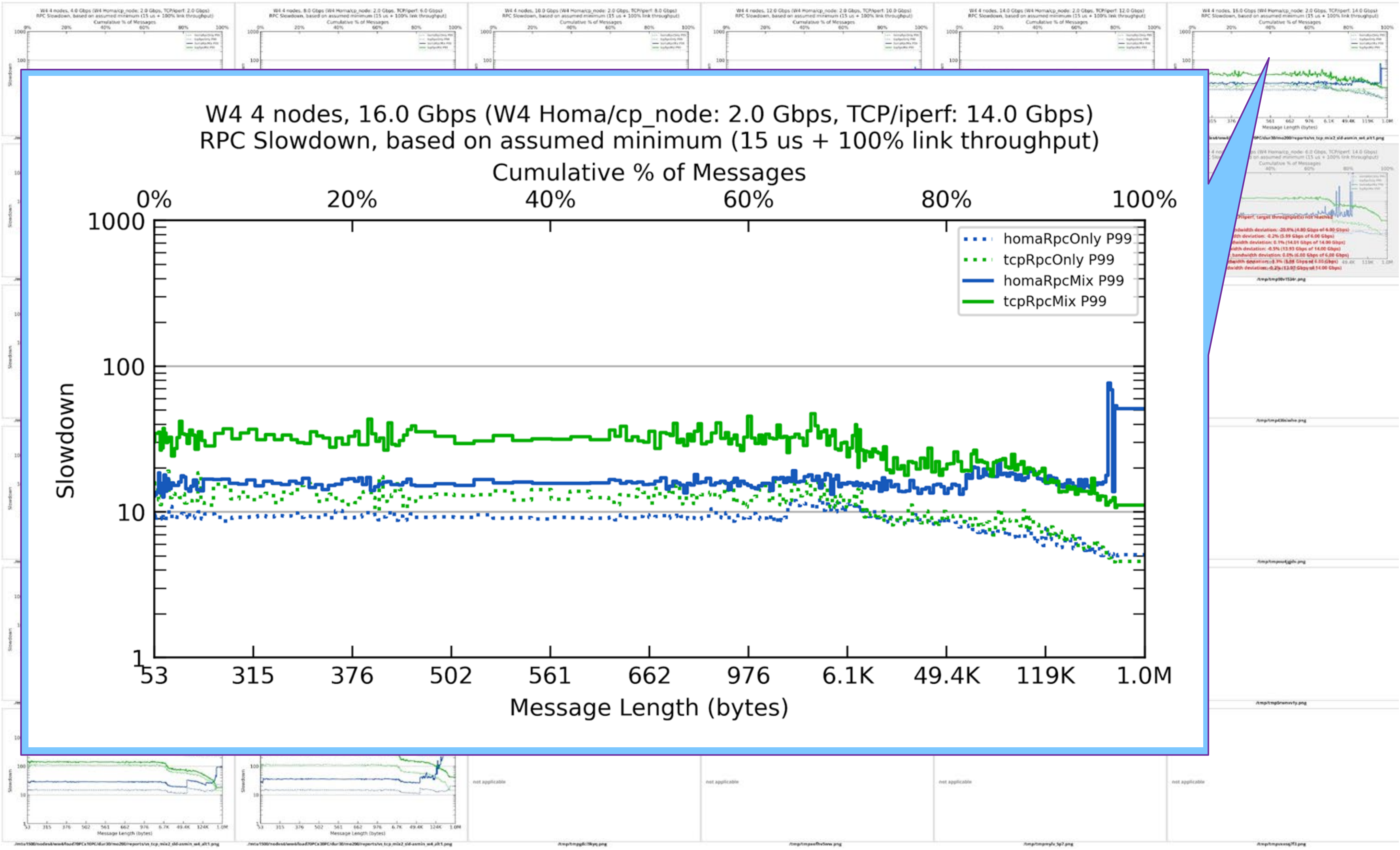
Slowdown Results Workload W4



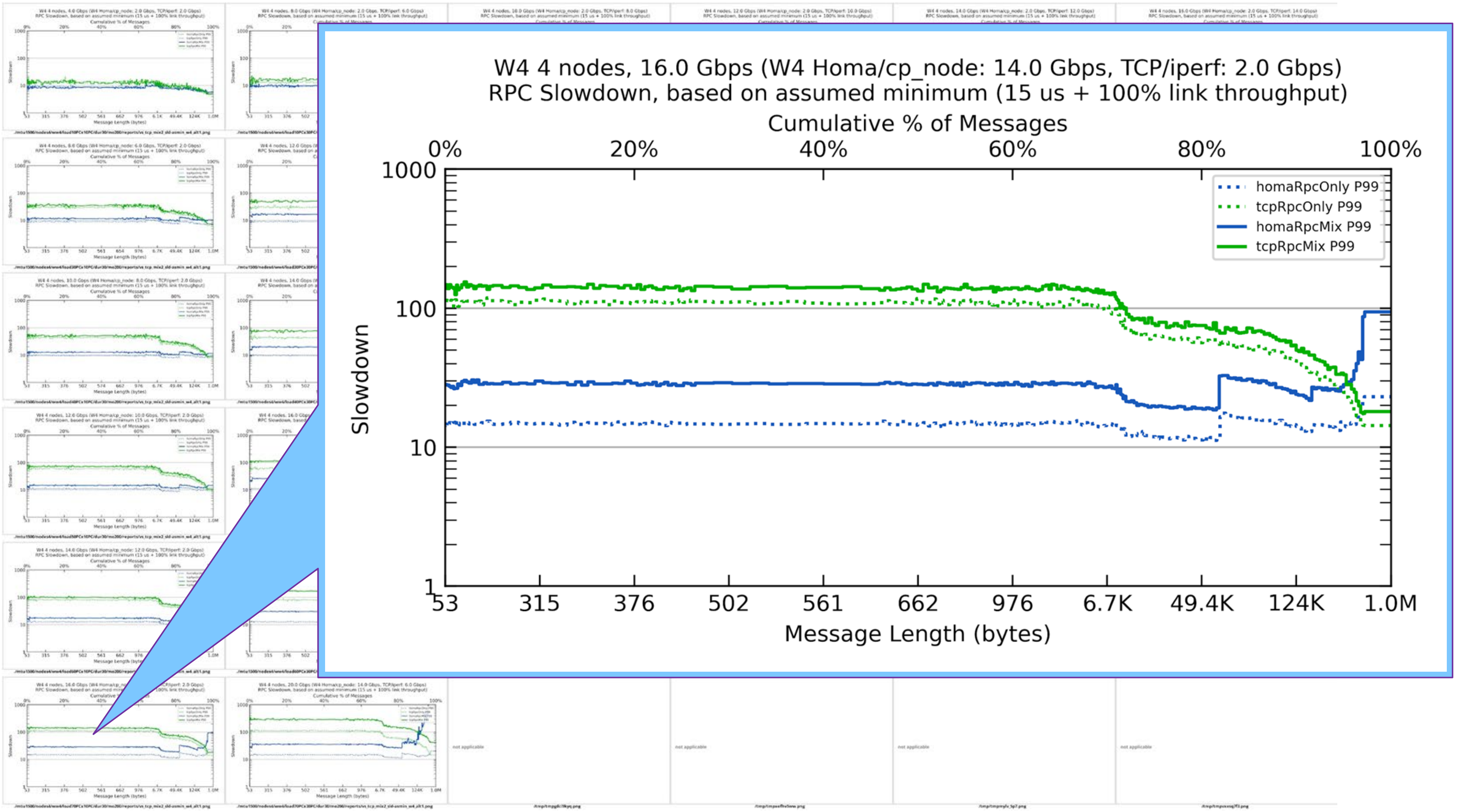
Slowdown Results Workload W4



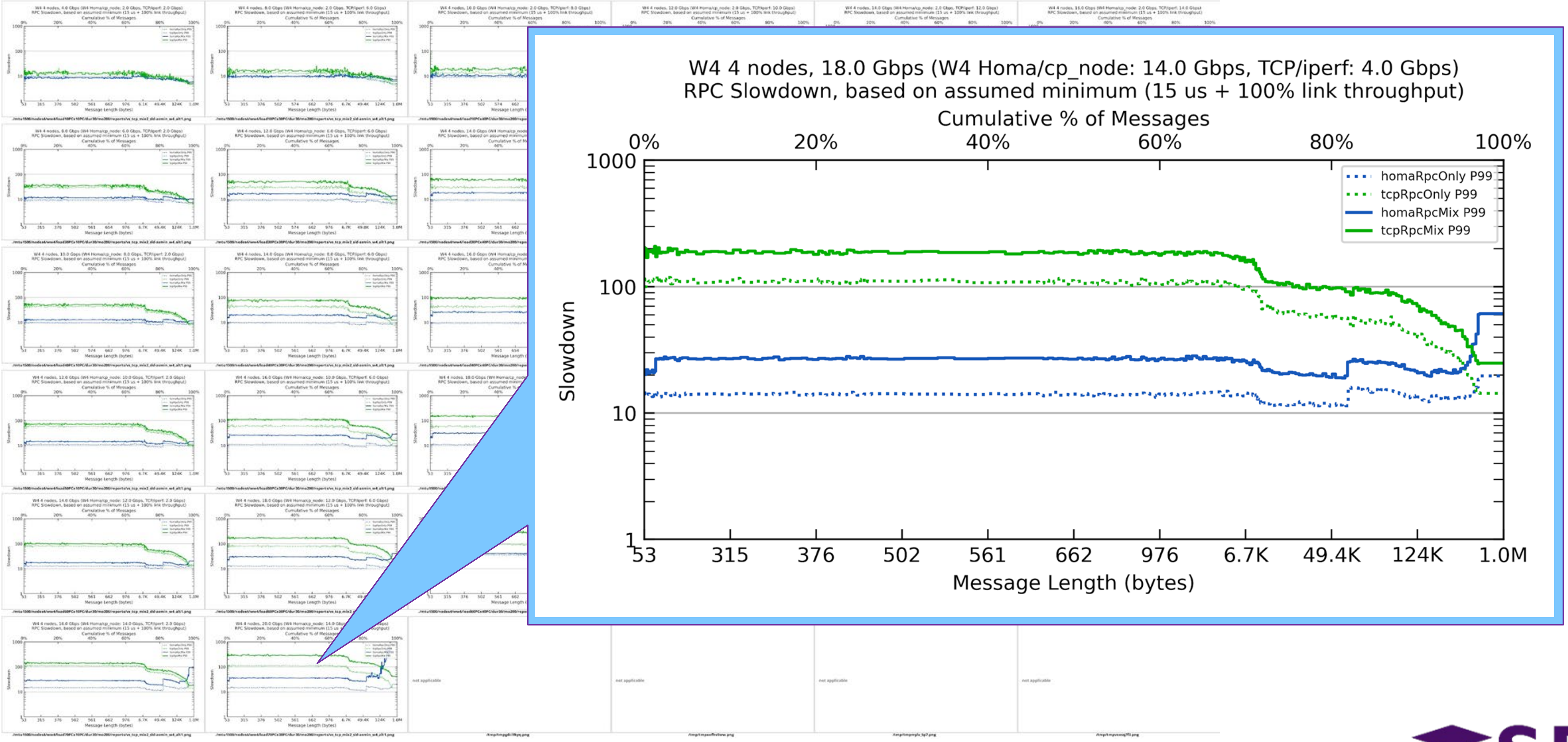
Slowdown Results Workload W4



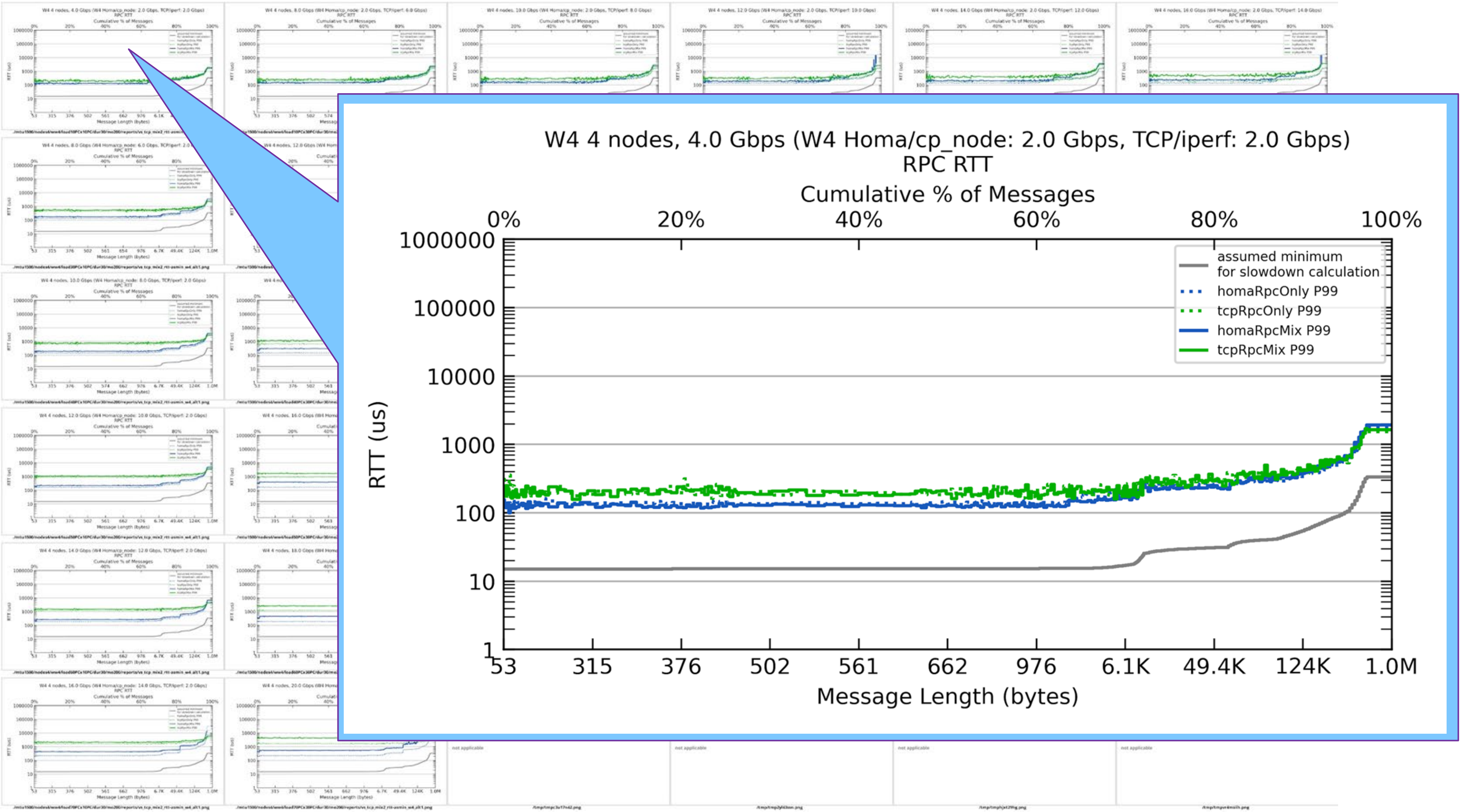
Slowdown Results Workload W4



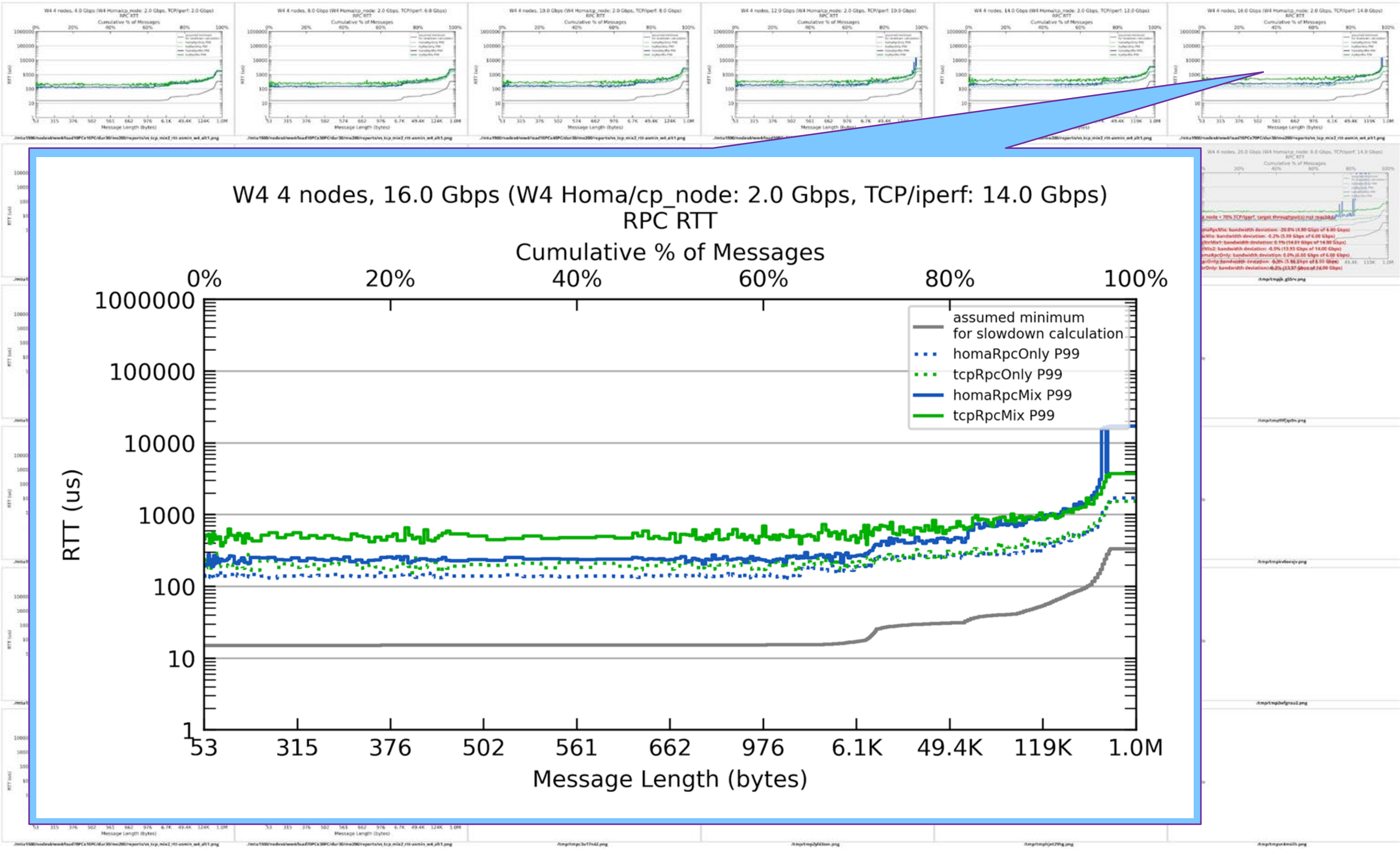
Slowdown Results Workload W4



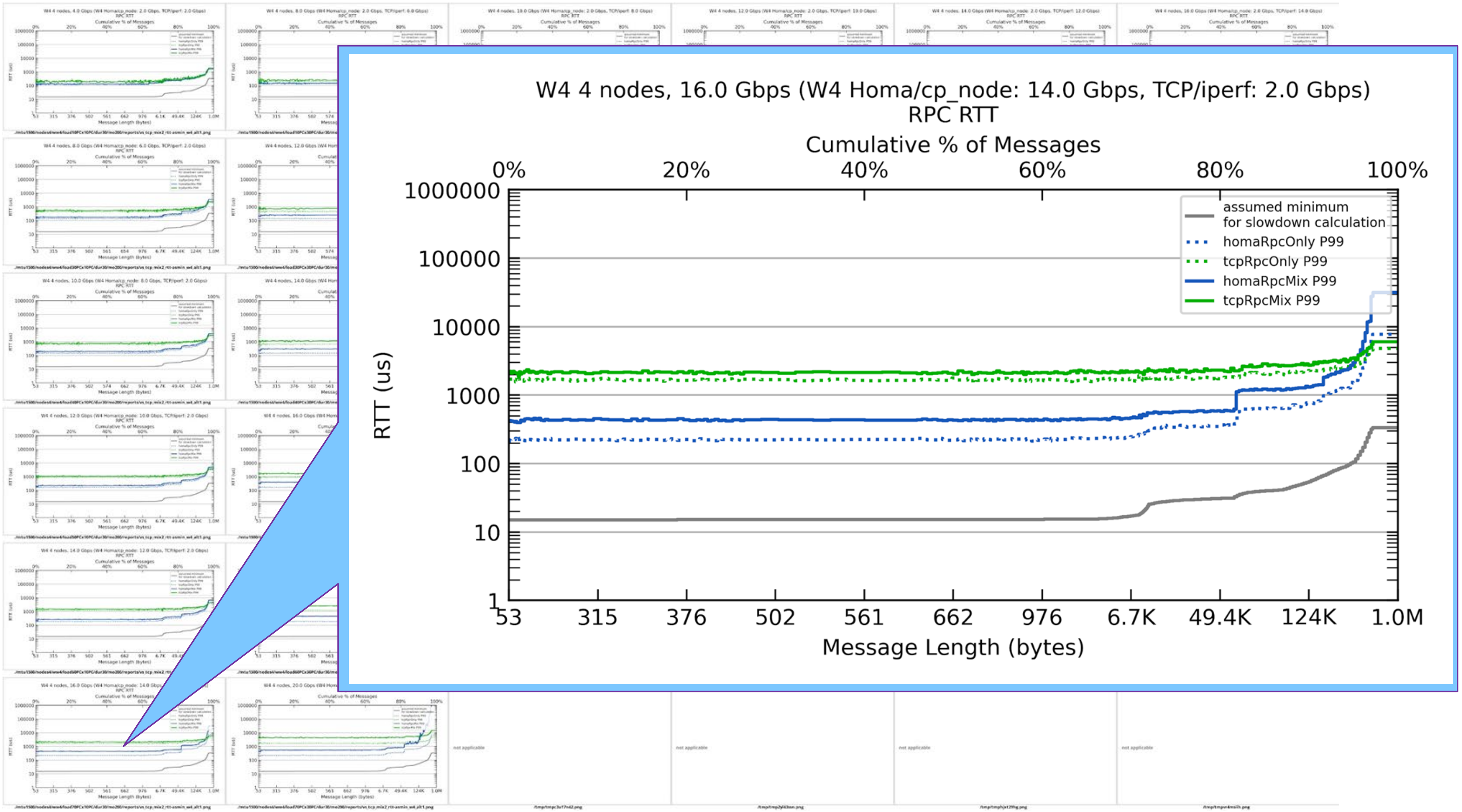
RTT Results Workload 4



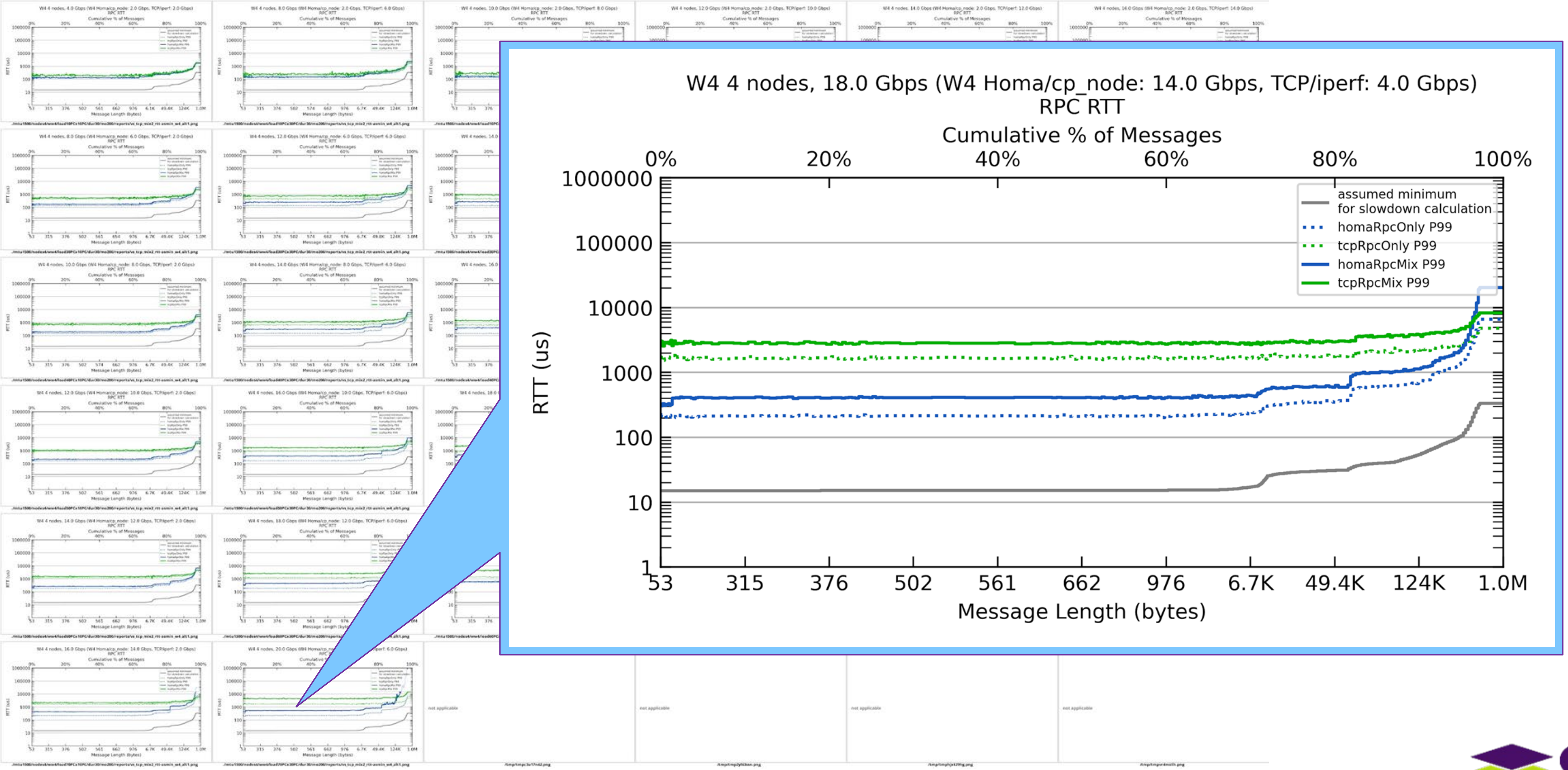
RTT Results Workload 4



RTT Results Workload 4



RTT Results Workload 4



Part 3

RRRRP - Homa, FPGA Accelerated

HomaModule - A Linux Kernel Implementation of Homa

A layer just above IP, parallel to TCP and UDP

Uses GRO (Generic Receive Offloading)

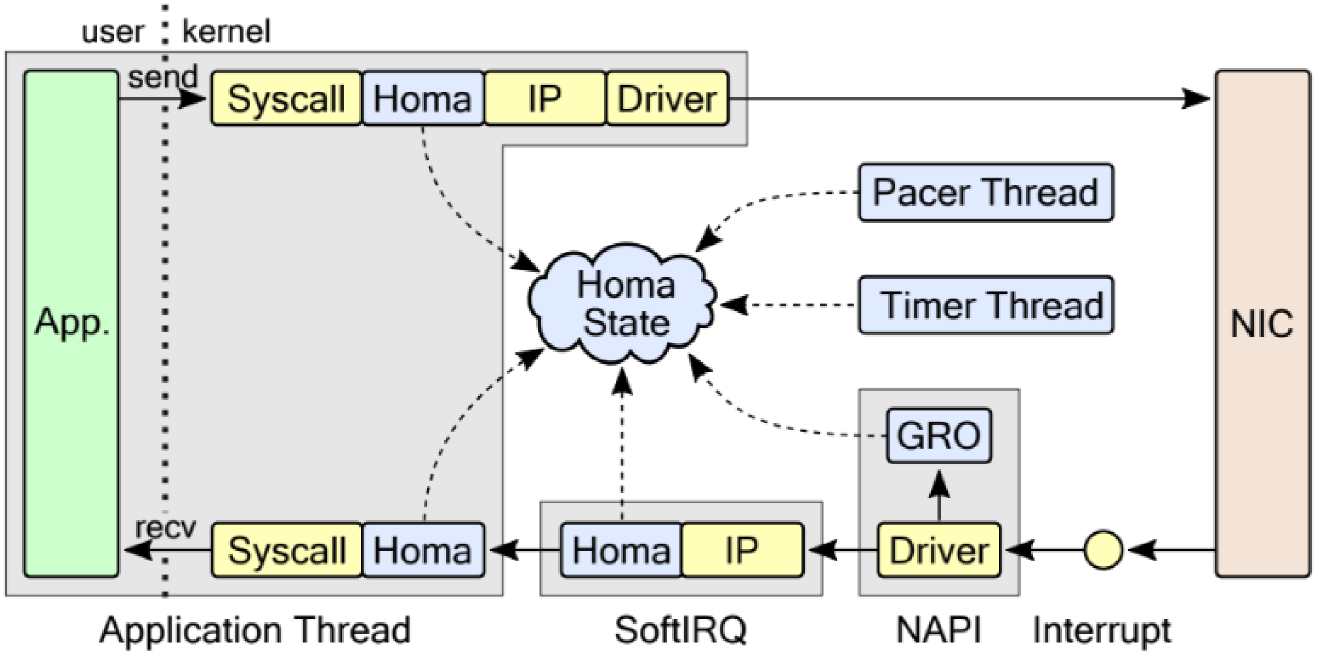


Figure 2: Structure of Homa/Linux. Homa components are shown in blue; existing Linux kernel modules are in yellow. Gray areas represent different cores. Only the primary sending and receiving paths are shown; other Homa elements such as the pacer thread and timer thread also transmit packets.

RRRRP - Two Acceleration Approaches

Offload Engines

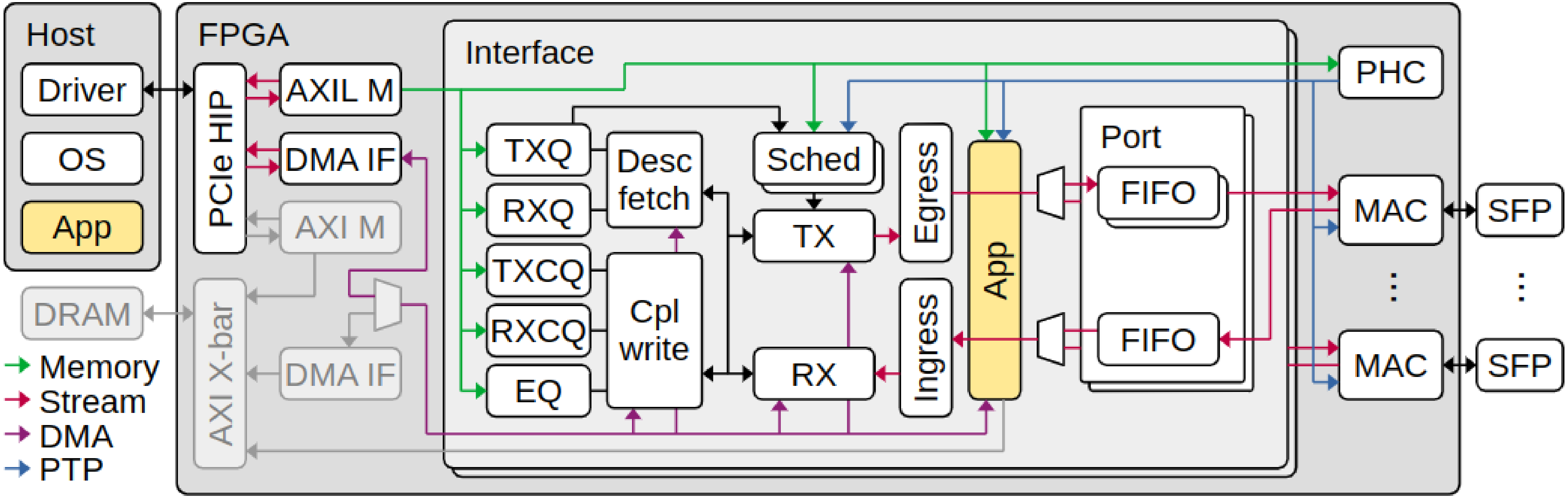
- Runs as SW on CPU
- Offload engines in FPGA
- Uses “golden” reference implementation (i.e. HomaModule)
- Low to medium eng. work
- Instant benefits

Full Acceleration

- Entire stack runs in FPGA
- No SW on CPU
- Major eng. work (mostly in testing correctness)
- Integrated with MLE NPAP, a TCP/UDP/IP FPGA Stack
- Maybe later

RRRRP - Offload Approach with Corundum.io

Open source FPGA NIC ported to many FPGA cards (http://www.corundum.io)
Good PCIe subsystem which supports many Rx/Tx FPGA queues.



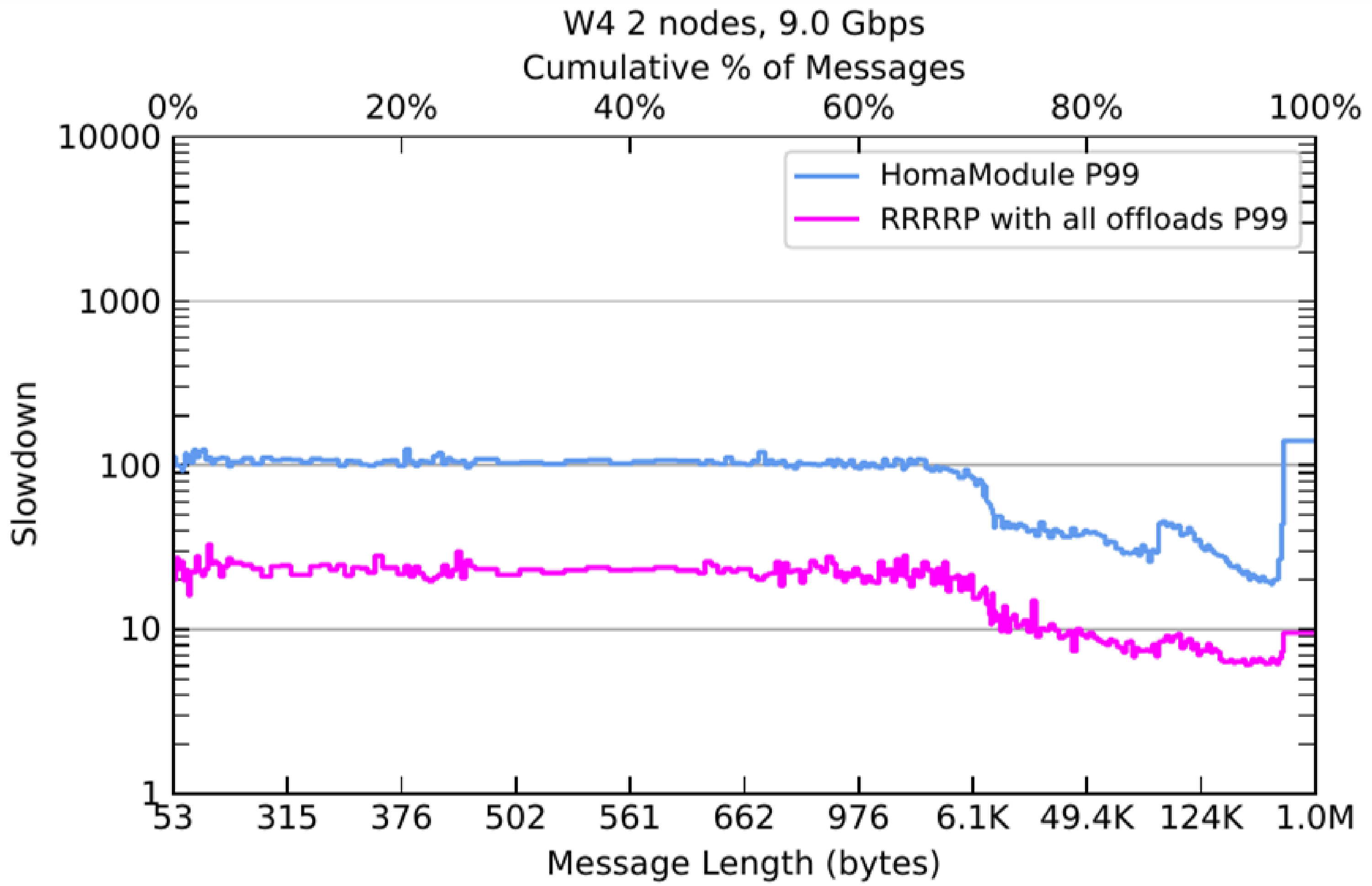
 |

Offload Engines

TCP (MLX 5)	HomaModule	RRRRP
Checksum Offload	N/A	N/A
Receive Side Scaling	Receive Flow Steering	Receive Side Scaling
Large Segmentation Offload	Generic Segmentation Offload	Large Segmentation Offload
Large Receive Offload	Generic Receive Offload	Large Receive Offload

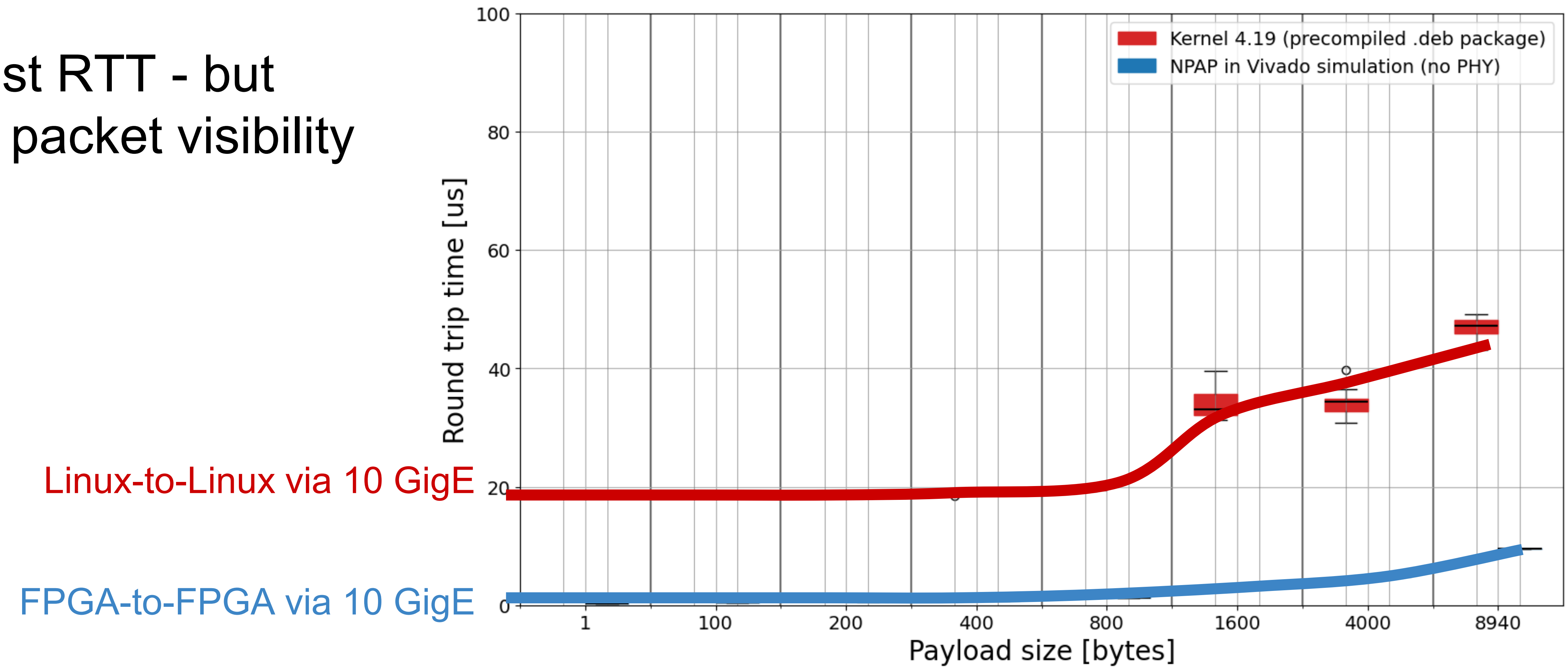
HomaModule vs RRRRP - Slowdown for Workload W4

- Similar results for other workloads
- Current implementation runs on 10 GigE NIC
- Next: 25/50/100 GigE



Motivation for Homa Full Acceleration

Lowest RTT - but lacks packet visibility



Part 4

Call for Collaboration

⇒ Run on larger FPGA cluster to check scalability

⇒ Try other workloads

⇒ Look at applications
Networked storage systems?
AI clusters?

Please take a moment to rate this session.

Your feedback is important to us.

