An Implementation of the Homa Transport Protocol in RAMCloud

Yilong Li, Behnam Montazeri, John Ousterhout
Introduction

- **Homa**: receiver-driven low-latency transport protocol using network priorities
  - Key ideas and simulation results presented before
- **HomaTransport in RAMCloud**: a working implementation
  - Unusual features: message-oriented, connectionless, no ACKs, etc.
- **Excellent performance**
  - Extreme network condition: 80% network load on 10 Gbps network
  - Slowdown of 99%-tile latency of almost all message sizes within 2-3.5x
  - 99%-tile round-trip latencies for small messages < 15 μs
  - Nearly 100x faster than best published result
Outline

- Homa Overview
- Implementation Features
- Evaluation
Homa Overview

- **Goal:** low latency at high network load
  - Focus on tail (e.g., 99th percentile) message latency
  - Implement shortest-remaining-processing-time (SRPT) policy
- **Key Idea 1:** Divide outgoing messages into unscheduled and scheduled portions

---

Unscheduled Packets
- Transmit blindly
- Priority computed by receiver in advance

Scheduled Packets
- Wait for receiver’s grant
- Priority granted on the fly

Sender
Goal: low latency at high network load
- Focus on tail (e.g., 99th percentile) message latency
- Implement shortest-remaining-processing-time (SRPT) policy

Key Idea 2: Dynamic priority allocation
- Receiver can change the priorities granted to incoming messages on the fly
- ... based on the exact set of incoming messages
- **Goal: low latency at high network load**
  - Focus on tail (e.g., 99th percentile) message latency
  - Implement shortest-remaining-processing-time (SRPT) policy

- **Key Idea 3: Controlled overcommitment**
  - Receiver grants to a smaller number of senders simultaneously
  - … to avoid wasting downlink bandwidth
  - … when our most favored sender doesn’t send back granted data in time

---

**Homa Overview**

- Receiver
- G → P0
- G → P1
- G → P2
Implementation in RAMCloud

- **RAMCloud**: low-latency key-value store
  - Modular transport architecture
  - Optimized software stack: 1~2 µs to send/receive an RPC
- **RAMCloud::HomaTransport**
  - Kernel bypass via DPDK
  - Use polling to detect incoming packets
  - ~4000 lines of C++ code (including comments)
Homa: Structurally Different From TCP

- **Message-oriented, not stream-oriented**
  - Independent delivery of messages: no head-of-line-blocking
  - RPC interface
    - Natural fit for datacenter applications
    - Socket-like byte stream interface on top of Homa

- **Connectionless**
  - No setup phase required before sending an RPC
  - No state kept after RPC completes
Retransmission

- No explicit ACKs
  - RPC response as the acknowledgment for the request
  - Optimize for small RPCs: reduce half of the packets required
    - The simplest RPC requires only 1 DATA packet in each direction

- Receiver-driven approach to detect lost packets
  - Receiver timeouts on messages that have been silent for a long period
    - … and request retransmission for the missing bytes
  - What if all unscheduled packets of a request are lost?
    - Client eventually timeouts on the response message
    - … and request retransmission of the initial bytes of the response
    - Server doesn’t recognize this RPC, assumes that the request must be lost
    - … and request retransmission of the initial bytes of the request
Sender-Side Queue Limiting

- **Sender implements SRPT by default**
  - Need to preempt long messages for short ones
  - Keep the transmit queue in NIC short to avoid queueing delay

- **QueueEstimator**
  - Keep a running estimate of the transmit queue length

- **Limiting queue length**
  - Too large: increase queueing delay for short messages
  - Too small: risk of TX queue running dry
  - Enqueue a packet only if queueLength ≤ one full-size packet
● 8 clients and 8 servers
● Each client generates a series of echo RPCs to random servers
  ○ Client sends a message of a given size
  ○ Server replies with the same message
● RPC message size chosen randomly to match the given workload
● RPC inter-arrival times follow poisson distribution
  ○ Average inter-arrival time configured to generate a given network load
Workloads

- **Workload: distribution of message sizes**
  - W3: aggregated RPC workload from Google datacenter applications
  - W4: Facebook Hadoop workload
  - W5: web search workload used for DCTCP
Workloads

- **Workload: distribution of message sizes**
  - W3: aggregated RPC workload from Google datacenter applications
  - W4: Facebook Hadoop workload
  - W5: web search workload used for DCTCP
● Hardware configurations
  ○ CloudLab m510 cluster: 8-Core Xeon D1548 @ 2.0 GHz, 10 Gbps network
  ○ Local Infiniband cluster: 4-Core Xeon X3470 @ 2.93 GHz, 24 Gbps network
  ○ All nodes in a cluster are connected to a single switch
Measurements are taken on 10 Gbps network at 80% network load unless stated otherwise.
Measurements are taken on 10 Gbps network at 80% network load unless stated otherwise.
Measurements are taken on 10 Gbps network at 80% network load unless stated otherwise.

Slowdown = Actual RPC Completion Time / Best-case RPC Completion Time (On Unloaded Network)
Homa Absolute Performance

Best-case RPC time
- 100 bytes: 4.7 µs
- 1000 bytes: 8.8 µs
Best-case RPC time
- 100 bytes: 4.7 µs
- 1000 bytes: 8.8 µs

99%-tile latency of 100B echo RPC is less than 15 µs in all three workloads!
HOL Blocking = 100x Tail Latency

Note: InfRC measurements are taken on a 24 Gbps Infiniband network using the absolute same workload, so the actual network load is only 33%.
HOL Blocking = 100x Tail Latency

Head-of-line blocking
HOL Blocking = 100x Tail Latency

No HOL blocking; but not scalable
Homa vs. TCP

Workload: W3

Workload: W4

Workload: W5

99% Slowdown (Log Scale)

Message Size (Bytes)

Message Size (Bytes)

Message Size (Bytes)
Basic vs. Homa: 5 - 15x higher tail latency for most RPCs
Priority & Controlled Overcommitment

Basic vs. Homa: 5 - 15x higher tail latency for most RPCs

Queueing Delay at Receiver’s Downlink
Basic vs. Homa: 5 - 15x higher tail latency for most RPCs

SRPT tends to produce run-to-completion behavior
How Many Priorities Does Homa Need?

4 priority levels is almost as good as 8.
Results Overview

- Homa vs. Other RAMCloud transports at 80% load (except InfRC)
We designed and implemented Homa, a new transport protocol for datacenter networks
  - Provide very low latency for short messages
  - Support high network utilization

Our Homa implementation has several unusual features
  - Message-oriented, connectionless, no explicit ACKs, etc.

Implementation measurements show excellent performance numbers across various workloads
Questions?