A Case for Replacing TCP (in the Datacenter)

John Ousterhout
Stanford University
Introduction

● TCP has been tremendously successful
● But every aspect of its design is wrong for the datacenter
● Talk outline
  ▪ Problems with TCP
  ▪ What a good datacenter protocol should look like (Homa)
  ▪ How to get past TCP
● Key theme: load balancing
  ▪ Essential both in hardware and in software
  ▪ TCP interferes with load balancing
Datacenter Transport Requirements

- **High performance:**
  - Low tail latency (<10 µs RTT for short messages?)
  - High throughput
    - Large amounts of data in a single message/stream
    - Large numbers of small messages

- **Implied requirements:**
  - Congestion control
    - At the edge
    - In the core
  - Efficient load balancing across many server cores
  - Transport implementation in special-purpose NIC hardware
Key Properties of TCP

- Stream-oriented
- Connection-oriented
- Fair scheduling (bandwidth sharing)
- Sender-driven congestion control
- Assumes in-order packet delivery

Each of these is wrong for the datacenter!
TCP Data Model: Byte Stream

- Applications care about **messages**, but TCP doesn’t preserve boundaries
- Extra complexity/overhead for message re-assembly
- Disastrous for load-balancing:
  - Unsafe for multiple threads to read from stream
  - Can’t offload dispatching to NIC
  - Must introduce dispatcher thread to parse stream into messages
- Poor tail latency (short messages stuck behind long ones)

![Diagram showing messages sent and blocks received](image-url)
Stream-Level Reliability is 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
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 to set up connection
  - 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
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
- TCP isn’t actually fair!
  - Severe bias against short messages
TCP: Sender-Driven Congestion Control

- Senders have no direct knowledge of congestion

- Congestion signals based on buffer occupancy:
  - Packets dropped if queues overflow
  - Congestion notifications based on queue length

- Results:
  - Significant buffer occupancy when system is loaded
  - Queuing causes delays, especially for short messages
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 virtually certain even at modest loads
  - Dominant cause of core congestion in datacenter networks?

- **Software: packets of a flow must traverse the same cores**
  - Uneven core loading, hot spots
  - Dominant source of software-induced tail latency
TCP Is Beyond Repair

- Too many problems
- Problems are 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 all of the TCP problems
  - Design elements are synergistic
Homa: Message-Based

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

- **Fundamental unit is an RPC**
  - Request message
  - Response message
  - RPCs are independent (no ordering guarantees)

- **No long-lived connection state**
  - ~200 bytes of state for each peer

- **No connection setup overhead**
  - Use one socket to communicate with many peers

- **Homa ensures end-to-end RPC reliability**
Homa: SRPT

- Prioritize shorter messages
  - Use switch priority queues
- Run-to-completion improves performance for every message length!
- Starvation risk for longest messages?
  - Use 5-10% of bandwidth for oldest message
Homa: Receiver-Driven Flow Control

- **Receiver has knowledge of all incoming messages:**
  - Reduce congestion in TOR
  - Prioritize shorter messages with grants
Homa: Packet Spraying

- Homa does not depend on in-order packet delivery
- Hypothesis: packet spraying will eliminate core congestion (unless core is systemically overloaded)
How to Replace TCP?

- TCP is deeply entrenched
  - Homa doesn’t support the TCP API

- Solution: add Homa support to RPC frameworks
  - Most datacenter traffic uses a framework
  - Only a small number of popular frameworks (gRPC and Thrift?)

- Progress so far:
  - Linux kernel driver for Homa [ATC ’21]
  - Working on gRPC support
    - C++ support is functional
    - Java support under development
gRPC Experiences

- **gRPC code base complex**
  - Separate code bases for C++ and Java
  - Transport API not clearly defined or documented
  - C++ code makes extreme use of callbacks/closures; very difficult to follow code

- **gRPC is slow!**

- **Round-trip latency for short RPCs:**

<table>
<thead>
<tr>
<th></th>
<th>Network</th>
<th>Client</th>
<th>Server</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>gRPC/TCP</td>
<td>30 µs</td>
<td>30 µs</td>
<td>30 µs</td>
<td>90 µs</td>
</tr>
<tr>
<td>gRPC/Homa</td>
<td>20 µs</td>
<td>16 µs</td>
<td>19 µs</td>
<td>55 µs</td>
</tr>
</tbody>
</table>

Will eventually need a more efficient RPC framework
Infiniband Isn’t the Answer Either

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
  - Poor congestion control
- Unreliable datagrams are … unreliable
  - And, poor congestion control
Summary

● TCP: bad

● Homa: good

● Time to change

(If you know of something better than Homa, or things about Homa that are problematic, please tell me)