Opening Up Kernel-Bypass TCP Stacks

Shinichi Awamoto and **Michio Honda**
School of Informatics, University of Edinburgh

7th July, 2025 @ USENIX ATC

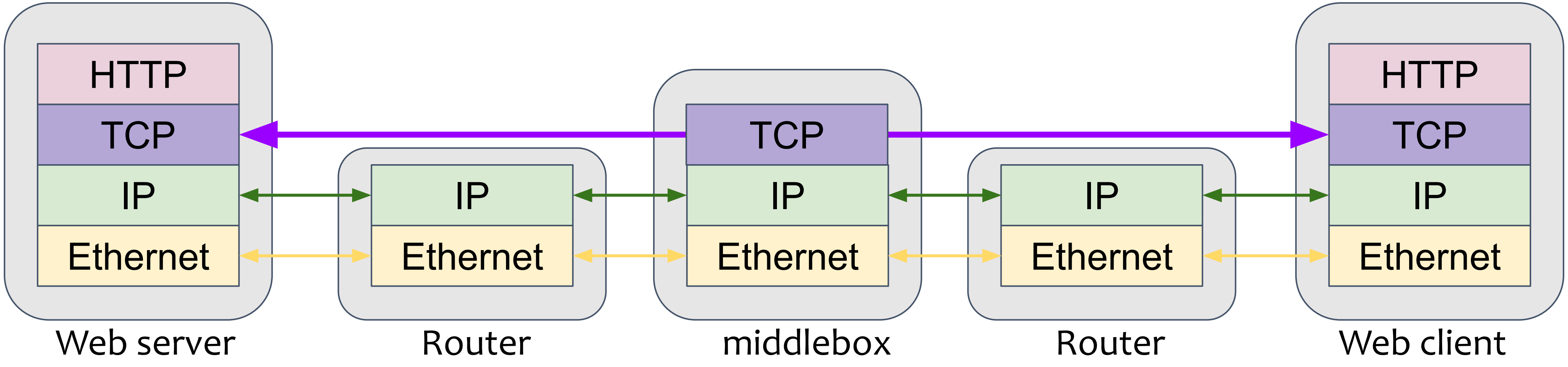
THE UNIVERSITY *of* EDINBURGH
informatics

Motivation

- Kernel bypass TCP stacks offer high performance
  - Clean-slate design
  - Benefit from fast packet I/O like DPDK
  - Optimization for specific workloads like RPCs
- These are great, but can we use those stacks in practice?

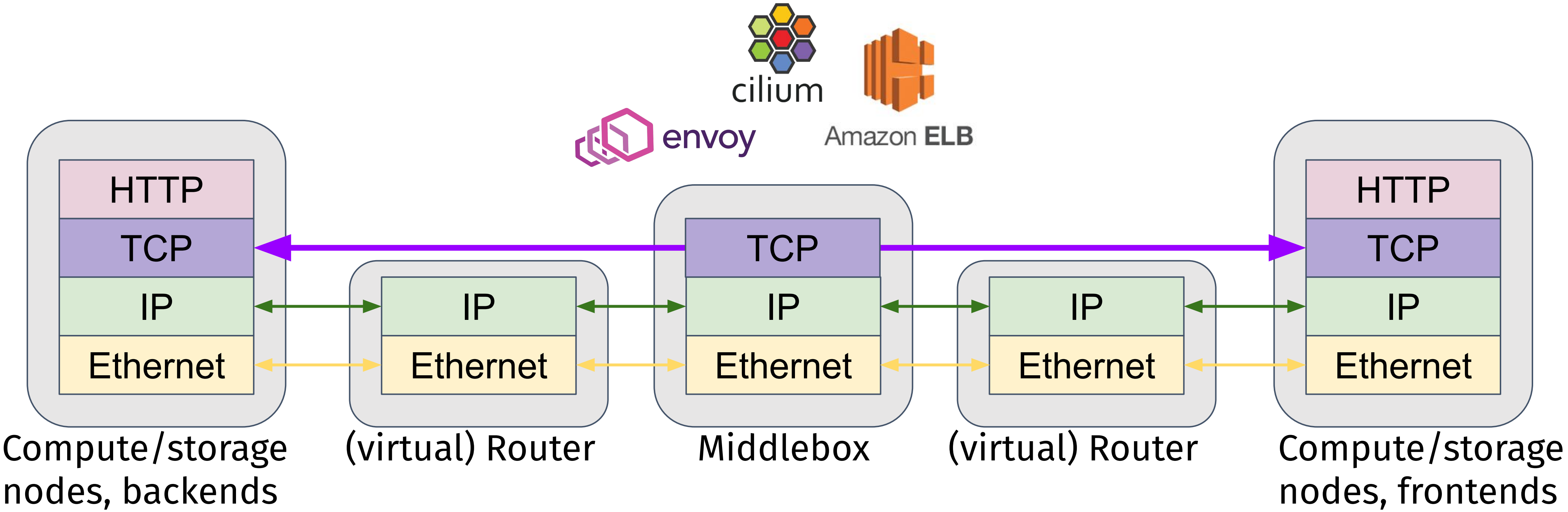
Everybody "loves" TCP

- Compatibility of apps, peers and middleboxes



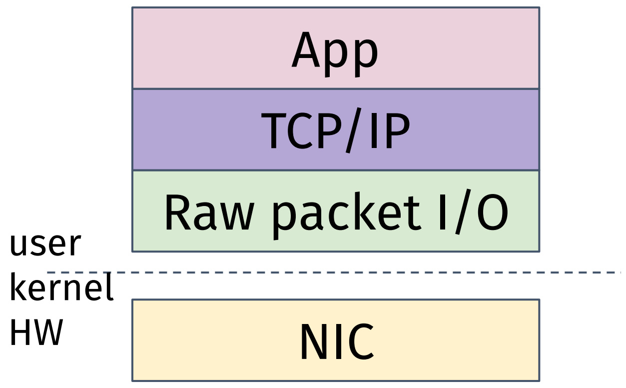
Everybody “loves” TCP

- Cloud/datacenter apps also use TCP (and middleboxes)



Kernel-bypass TCP stacks

- Criticisms of kernel TCP:
  - Small messages (e.g., RPCs)
  - Large number of connections (e.g., C10K/M)
  - Multiple CPU cores
- TCP/IP on top of fast packet I/O
  - mTCP [NSDI'14], F-Stack, IX [OSDI'14], TAS [EuroSys'17], Demikernel [SOSP'21], **to name a few**



Building a practical stack in reality

- TCP has many extensions
  - With and without RFCs (e.g., [1])
- Kernel TCP stack has long been evolved
  - e.g., 5–25% LoC modification each year [2]
- Creating a practical stack needs community support
  - At least that for many, not hyperscalers

What stack could we pick or build?

[1] Cheng et. al., "Making Linux TCP Fast", Netdev 2016
[2] Pismenny et. al., "Autonomous NIC offload", ASPLOS'21

Problem:

We don’t know how proposed stacks perform in various workloads and compare to each other

Limited comparison and workload when a new stack is proposed, likely due to difficulty of running existing ones.

“Hacking into these problems will take an unexpected amount of engineering effort with rather limited community support. Hence, building a new user-space TCP from scratch can be actually more time-saving.” - Deploying User-space TCP at Cloud Scale with Luna, USENIX ATC’23

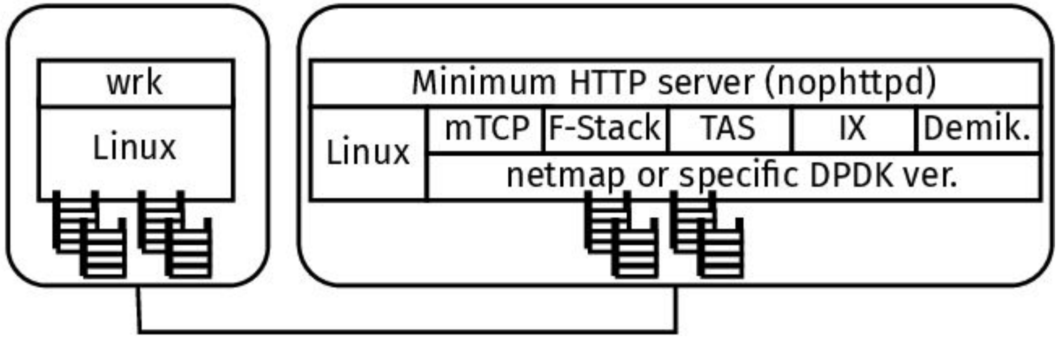
Contributions

- This paper addresses those problems:
  - We compare 6 existing stacks with exactly same application, hardware and workloads
  - We provide third-party experience of using or fixing existing stacks

There is no single stack that always performs the best

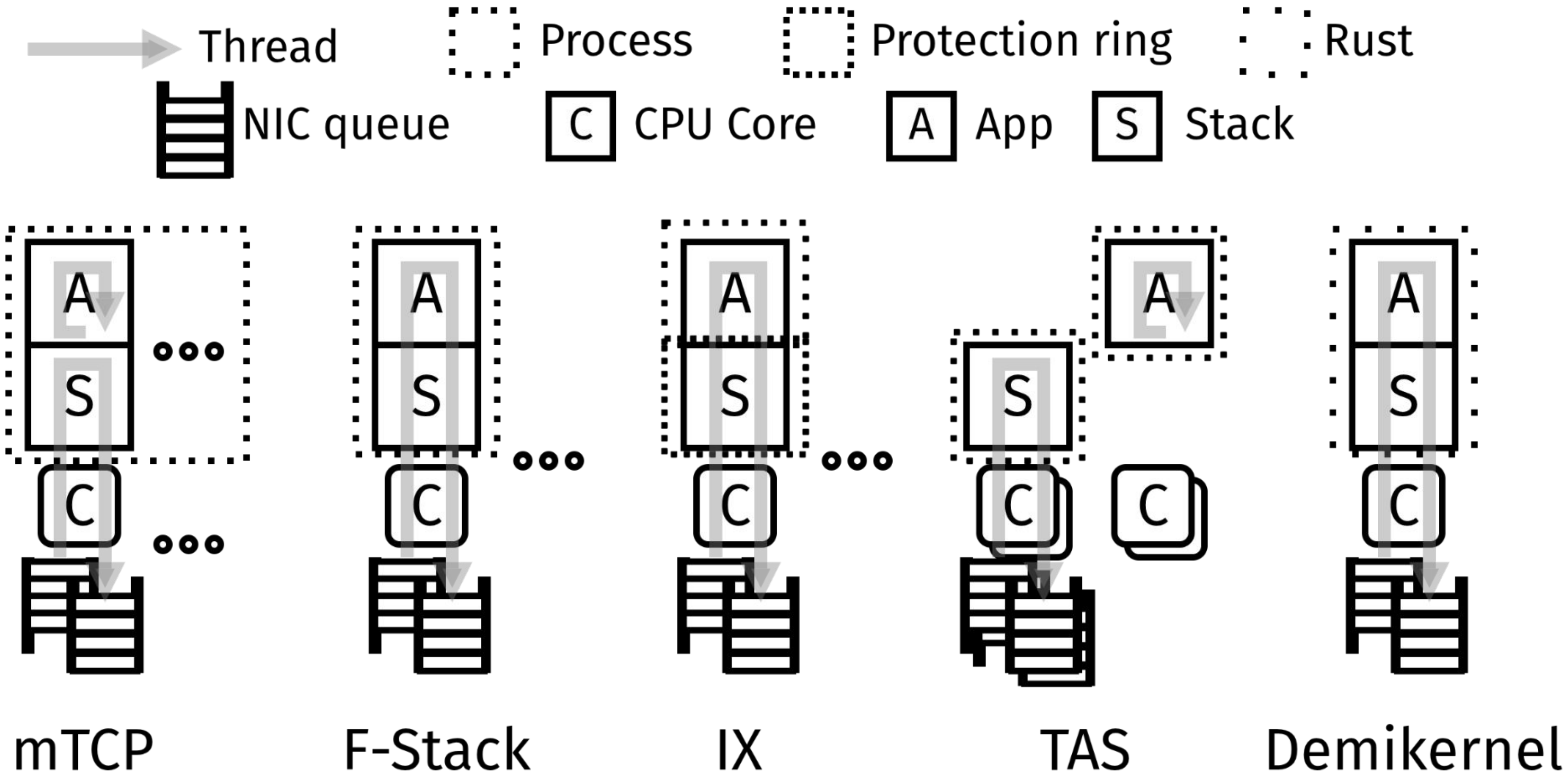
Methodology

- Server: minimum HTTP server
  - Optimized for individual stacks
- Client: ordinary wrk/Linux
  - slight modification for multicore scalability



Stack selection

- Based on the architecture



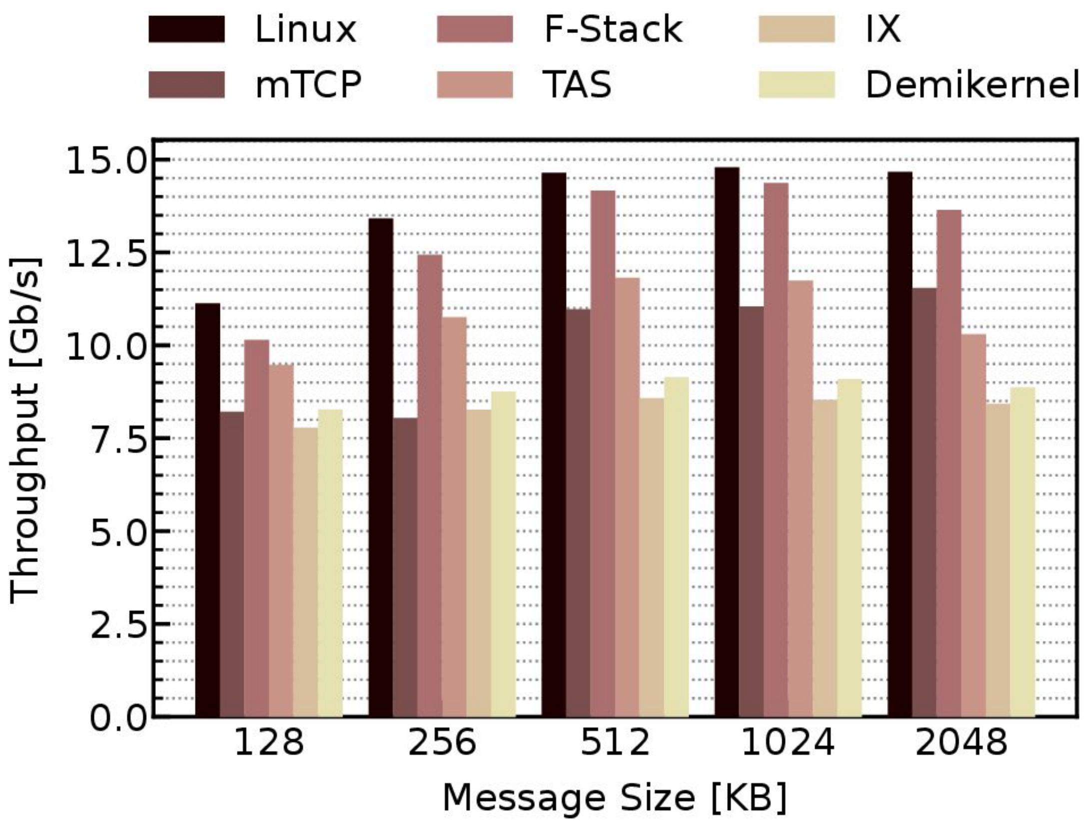
Stack selection

- Based on the architecture

	Architecture	API	TCP impl.	Use by author(s)	Use w/o author(s)
mTCP [32]	App-stack thread pair on the same core	Socket-like (no semantics)	Custom	Up to 8 KB data and 8 cores [32, 31, 50, 38]	Up to 8 KB data or 24 cores [3, 41, 7, 36, 81]
F-Stack [16]	App-level processing in the stack thread	Event callback to the stack	FreeBSD	–	Up to 8 KB data [57, 81], 8 cores [10] or 64 conns. [58]
IX [7]	App-level processing in the stack thread	Packet-level TX/RX buffers	`lwIP` [19]	Up to 8 KB data [38]	Up to 64 B data [36], 4 KB w/ 8 cores [81] or low data rate [41]
TAS [36]	Dedicated threads for TCP data path	Socket-like	Custom	Up to 2 KB data and 24 cores [36, 72, 75]	Up to 0.3 MReqs with high overhead apps [41]
Demikernel [80]	App-level processing in the stack thread in Rust	Packet-level TX/RX buffers	Custom	Up to 16 conns. [64] and 256 KB data [80, 15]	Up to 64 conns. [58]

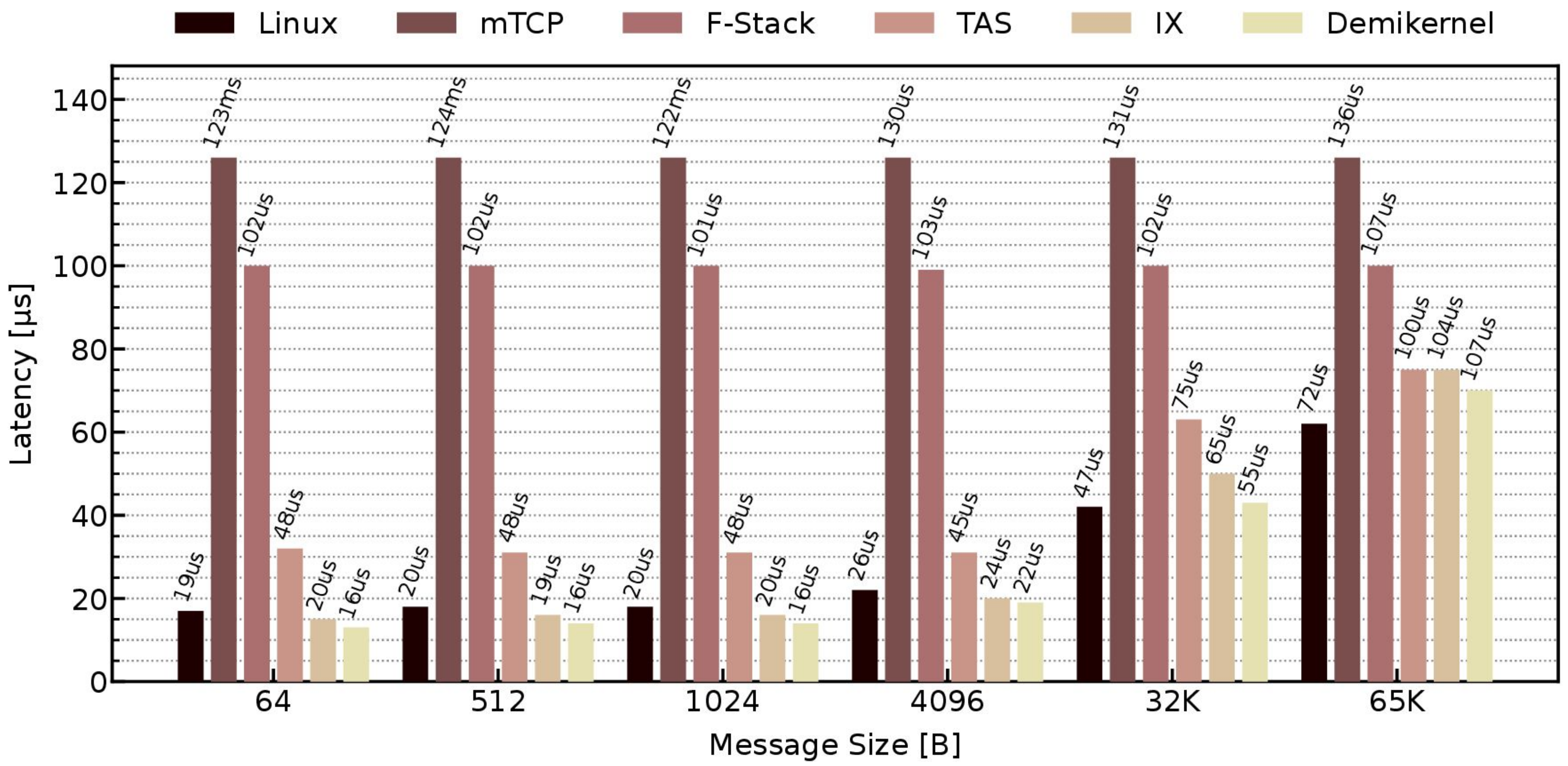
Large send

- Linux performs the best
- Why large send matters?
  - Data-driven workloads
  - Terabit Ethernet



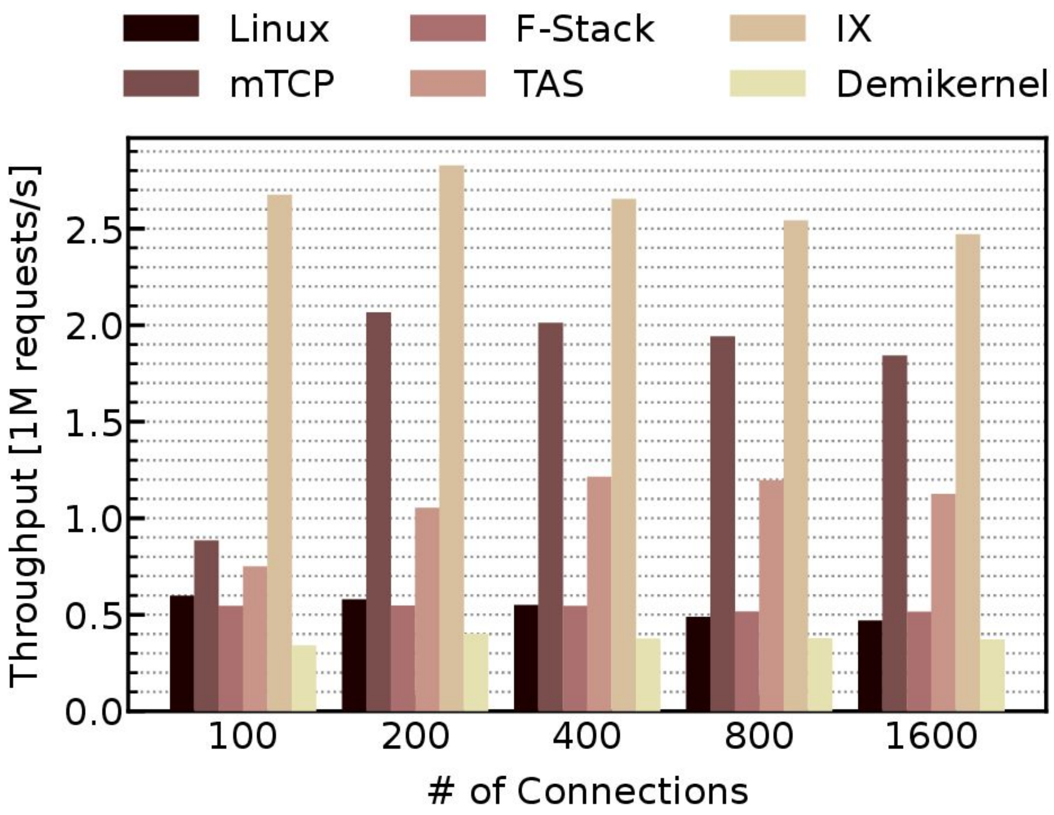
Idle-time small-message latency

- Demikernel does the best job for very small messages



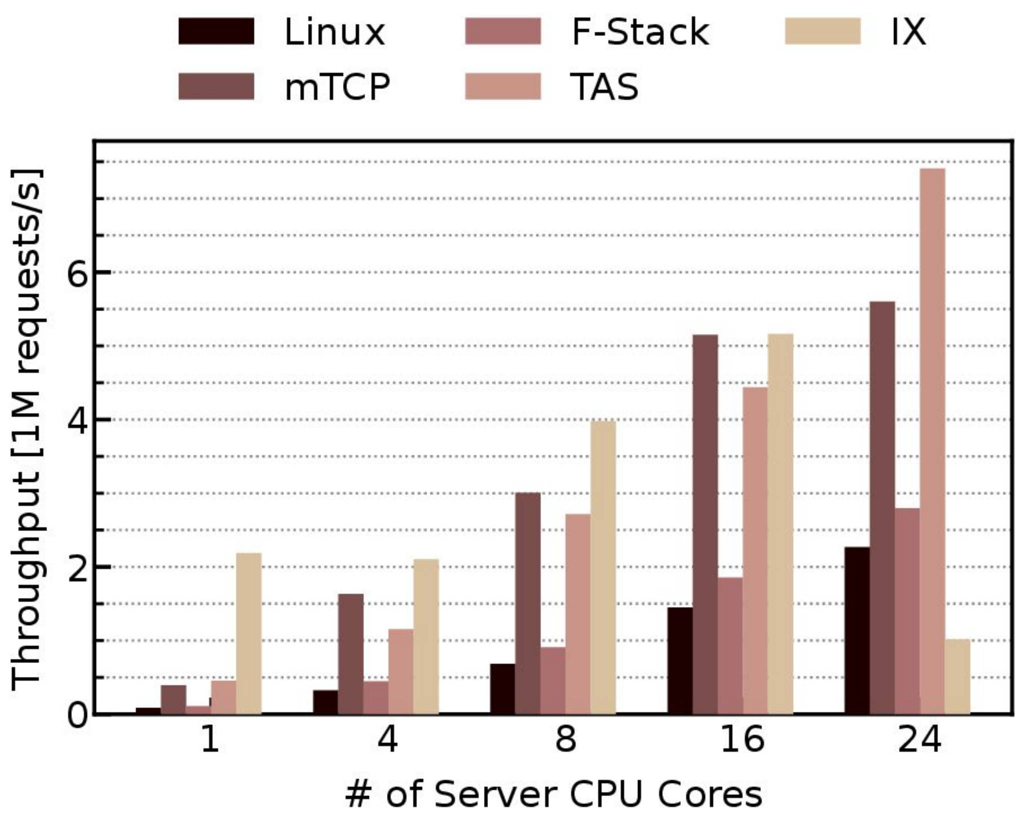
Connection scalability

- IX performs the best



Multicore scalability

- TAS performs the best



Discussion

- How to profile/analyze kernel-bypass stacks?
  - Perf-like tools require code knowledge
  - NSight [NSDI'22] would be useful
- Should we enhance kernel or kernel-bypass stack?
  - Low connection scalability with Linux and F-Stack is prohibiting
  - Run-to-completion would be unsuitable for efficient ack-clocking

More discussion in the paper

Conclusion

- No stacks serve all the workloads well

Best bulk transfer

IX: A Protected Dataplane Operating System for High Throughput and Low Latency

Adam Belay[1] George Prekas[2] Ana Klimovic[1] Samuel Grossman[1]
Christos Kozyrakis[1] Edouard Bugnion[2]
[1]Stanford University [2]EPFL

Abstract

The conventional wisdom is that aggressive networking

The conventional wisdom is that there is a basic mismatch between these requirements and existing networking stacks in commodity operating systems. Conse-

Best RPC throughput

The Demikernel Datapath OS Architecture for Microsecond-scale Datacenter Systems

Irene Zhang, Amanda Raybuck, Pratyush Patel, Kirk Olynyk, Jacob Nelson,
Omar S. Navarro Leija, Ashlie Martinez, Jing Liu, Anna Kornfeld Simpson, Sujay Jayakar,
Pedro Henrique Penna, Max Demoulin, Piali Choudhury, Anirudh Badam
Microsoft Research, University of Texas at Austin, University of Washington,
University of Wisconsin Madison, University of Pennsylvania, Zerowatt, Inc.

Abstract

Datacenter systems and I/O devices now run at single-digit

like Redis [80], can achieve *single-digit microsecond latencies*. To avoid becoming a bottleneck, datapath systems software must operate at *sub-microsecond – or nanosecond – la-*

Best idle-latency

TAS: TCP Acceleration as an OS Service

Antoine Kaufmann, MPI-SWS
Tim Stamler, The University of Texas at Austin
Simon Peter, The University of Texas at Austin
Naveen Kr. Sharma, University of Washington
Arvind Krishnamurthy, University of Washington
Thomas Anderson, University of Washington

ACM Reference Format:
Antoine Kaufmann, Tim Stamler, Simon Peter, Naveen Kr. Sharma, Arvind Krishnamurthy, and Thomas Anderson. 2019. TAS: TCP Acceleration as an OS Service. In *Fourteenth EuroSys Conference 2019 (EuroSys '19), March 25–28, 2019, Dresden, Germany.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3302424.3303985

Abstract

As datacenter network speeds rise, an increasing fraction of

1 Introduction

As network speeds rise, while CPU speeds stay stagnant, TCP packet processing efficiency is becoming ever more important. Many data center applications require low-latency and high-throughput network access to deliver remote procedure calls (RPCs). At the same time, they rely on the lossless, in-order delivery properties provided by TCP. To provide this convenience, software TCP stacks consume an increasing fraction of CPU resources to process network packets.

Best multicore scalability

Stack modifications, tools and configuration used in this paper: https://github.com/uoenoplab/stackbench