Miresga: Accelerating Layer-7 Load Balancing with Programmable Switches

Xiaoyi Shi Lin He Jiasheng Zhou Yifan Yang Ying Liu

Institue for Network Sciences and Cyberspace, Tsinghua University

Background & Motivation

Modern online service providers utilize load balancing in cloud data centers to distribute traffic across large server clusters. Typically, these service providers deploy Layer-7 load balancers to correctly distribute traffic to server clusters running different services.

Figure 1 shows the architecture of a typical Layer-7 load balancer. The load balancer inspects the packet headers and payloads to determine the service type and the corresponding server cluster. Then, it forwards the packets to the selected server cluster.

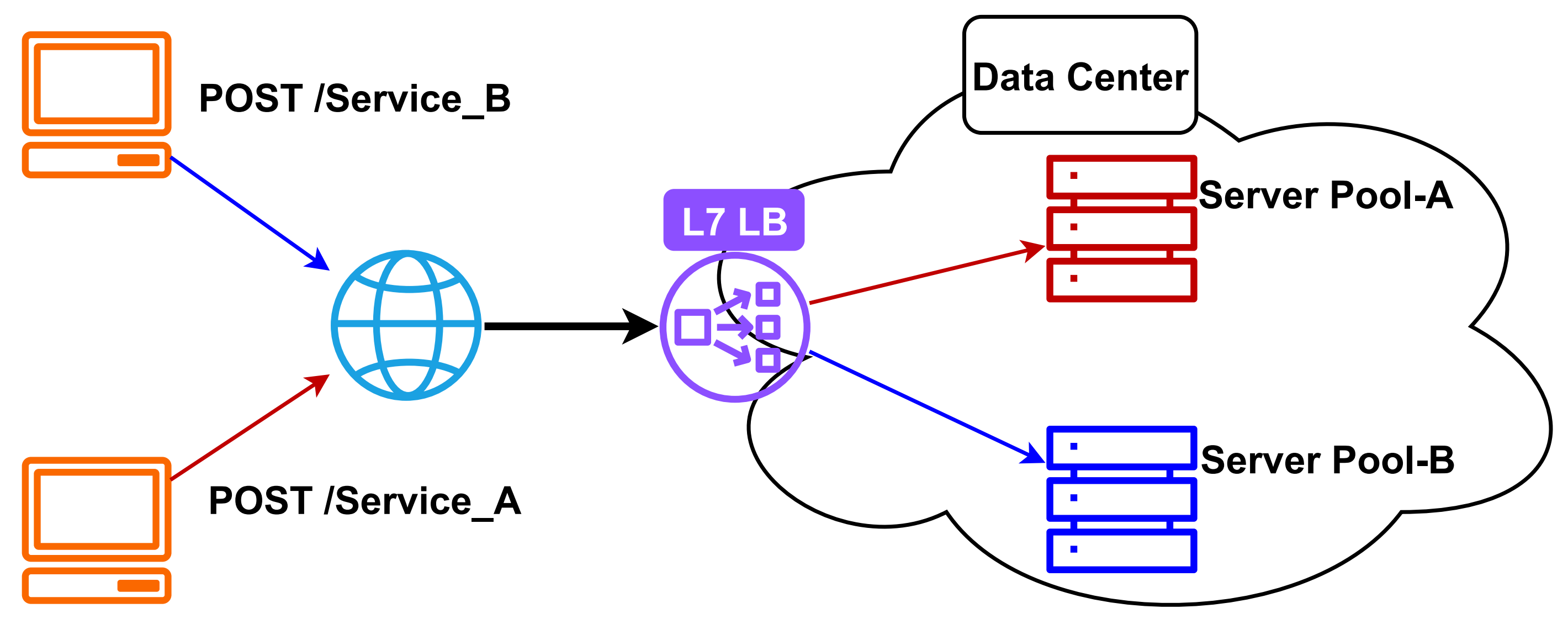


Figure 1. The architecture of a typical Layer-7 load balancer.

However, the increasing demand for high throughput and low latency in cloud data centers has outpaced the capabilities of traditional software Layer-7 load balancers, since the generic CPUs are not optimized for the high-speed packet processing required by Layer-7 load balancing.

The emergence of programmable network hardware, *e.g.*, the programmable switch, has enabled offloading some stateless operations onto the hardware, thereby significantly reducing the load on server CPUs. Programmable switches utilize programmable ASICs to achieve ~Tbps line-rate packet processing, enabling operators to modify packet headers based on customized rules.

So, our question is that **can we leverage programmable switches to perform part of the Layer-7 load balancing tasks to replace some server resources?**

System Overview

The core idea of is **Hardware-Software Co-design with Proper Layer-7 Load Balancing Partitioning** to fully leverage the high packet processing speed of programmable switches and the flexible computing capability of CPUs. This co-design is motivated by the strengths and limitations of programmable switches: while they excel in high-speed forwarding, their memory resources and L7 protocol processing capabilities are limited. In contrast, general-purpose servers offer abundant memory resources and strong L7 protocol processing capabilities but lack efficient forwarding performance.

Miresga partitions the Layer-7 load balancing tasks into three sub-tasks and uses three components, the programmable switch, the front-end server, and the back-end agent to handle them:

1. **Connection Establishment**: The programmable switch is responsible for establishing connections and handling SYN packets, while the front-end server is responsible for connection splicing.
2. **Application Protocol Parsing and Decision**: The front-end server is responsible for parsing the application protocol headers and making routing decisions.
3. **Header Rewriting**: The programmable switch is responsible for modifying the IP and port, while the back-end agent is responsible for modifying the seq/ack numbers.

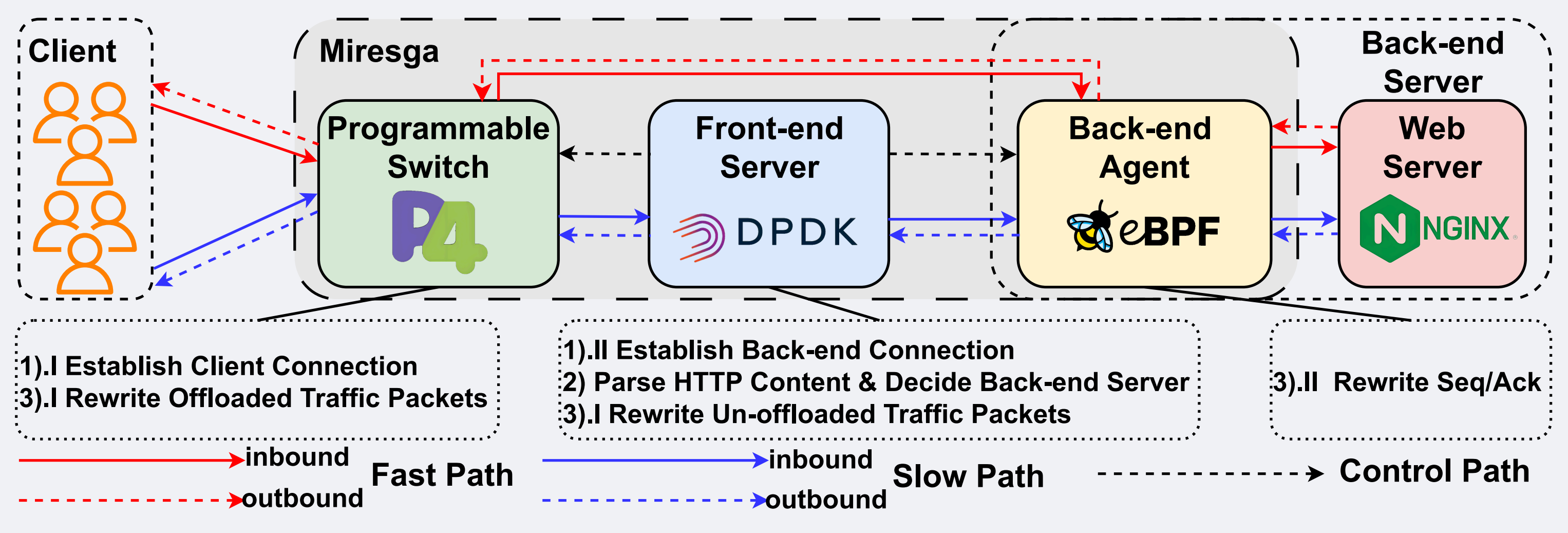

Client
Miresga
Programmable Switch
Front-end Server
DPDK
Back-end Agent
eBPF
Back-end Server
Web Server
NGINX
1).I Establish Client Connection
3).I Rewrite Offloaded Traffic Packets
1).II Establish Back-end Connection
2) Parse HTTP Content & Decide Back-end Server
3).I Rewrite Un-offloaded Traffic Packets
3).II Rewrite Seq/Ack
inbound
outbound
Fast Path
inbound
outbound
Slow Path
Control Path

Figure 2. The architecture of Miresga

Challenges & Design Details

Challenge 1: The finite memory resources inherent in programmable switches limit their ability to accommodate increased traffic volumes.

Firstly, Miresga compresses the stored table entries by merging the two entries corresponding to inbound and outbound traffic, and it employs a single index to obviate the need for storing the complete backend IP and port information. What's more, modifications to the sequence and acknowledgment numbers are delegated to the backend agent that leverages eBPF technology, as this component is not amenable to compression.

Challenge 2: The intricate kernel protocol stack contributes to the inefficiency of front-end servers.

Miresga employs DPDK technology to bypass the kernel protocol stack and constructs a lightweight protocol stack. Specifically, we consolidate the states of the two connections—between the front-end server and the client, and between the front-end server and the back-end server—into a single flow state. In this design, the front-end server is solely responsible for packet forwarding, while tasks such as retransmission and congestion control are delegated to the client and the back-end servers.

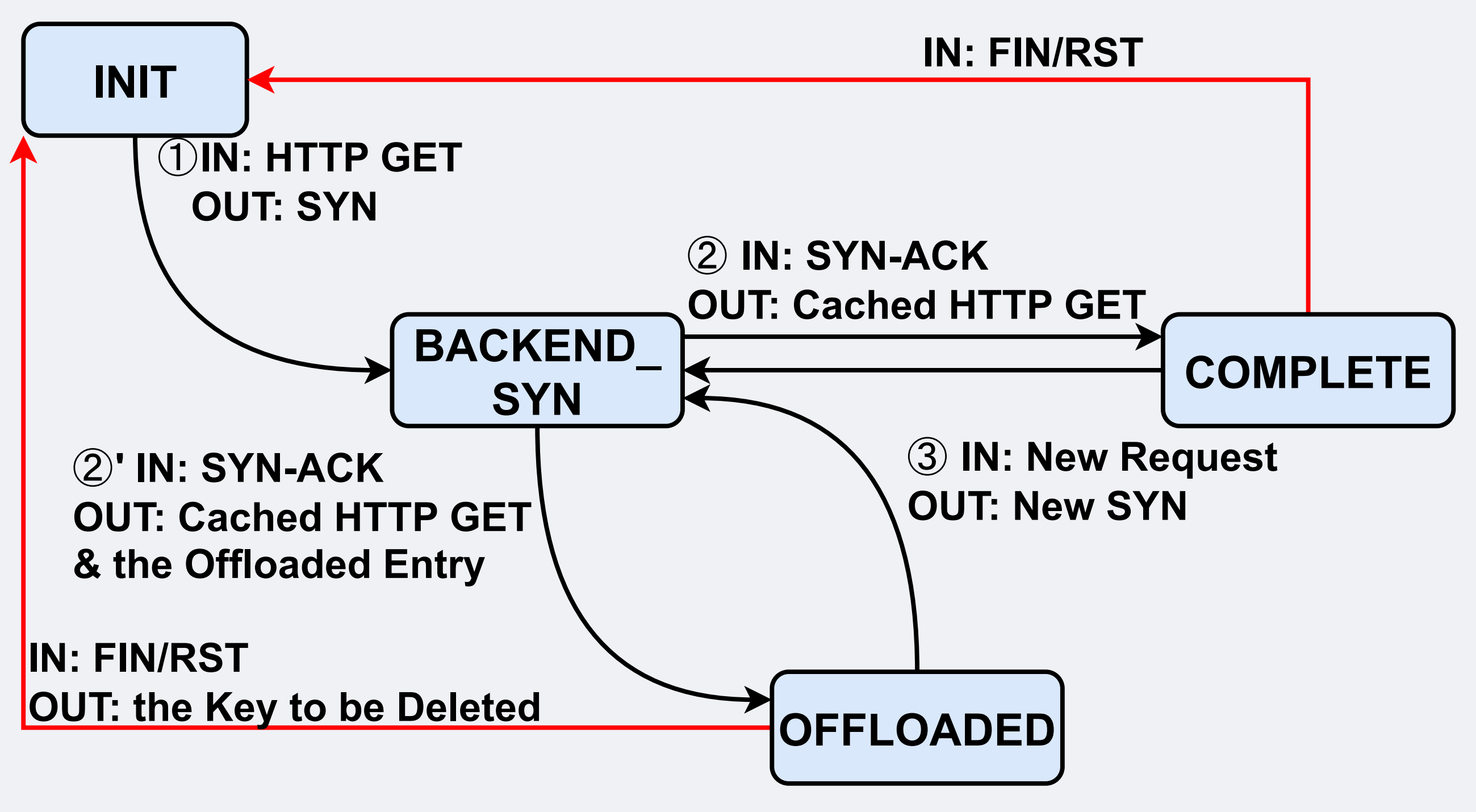


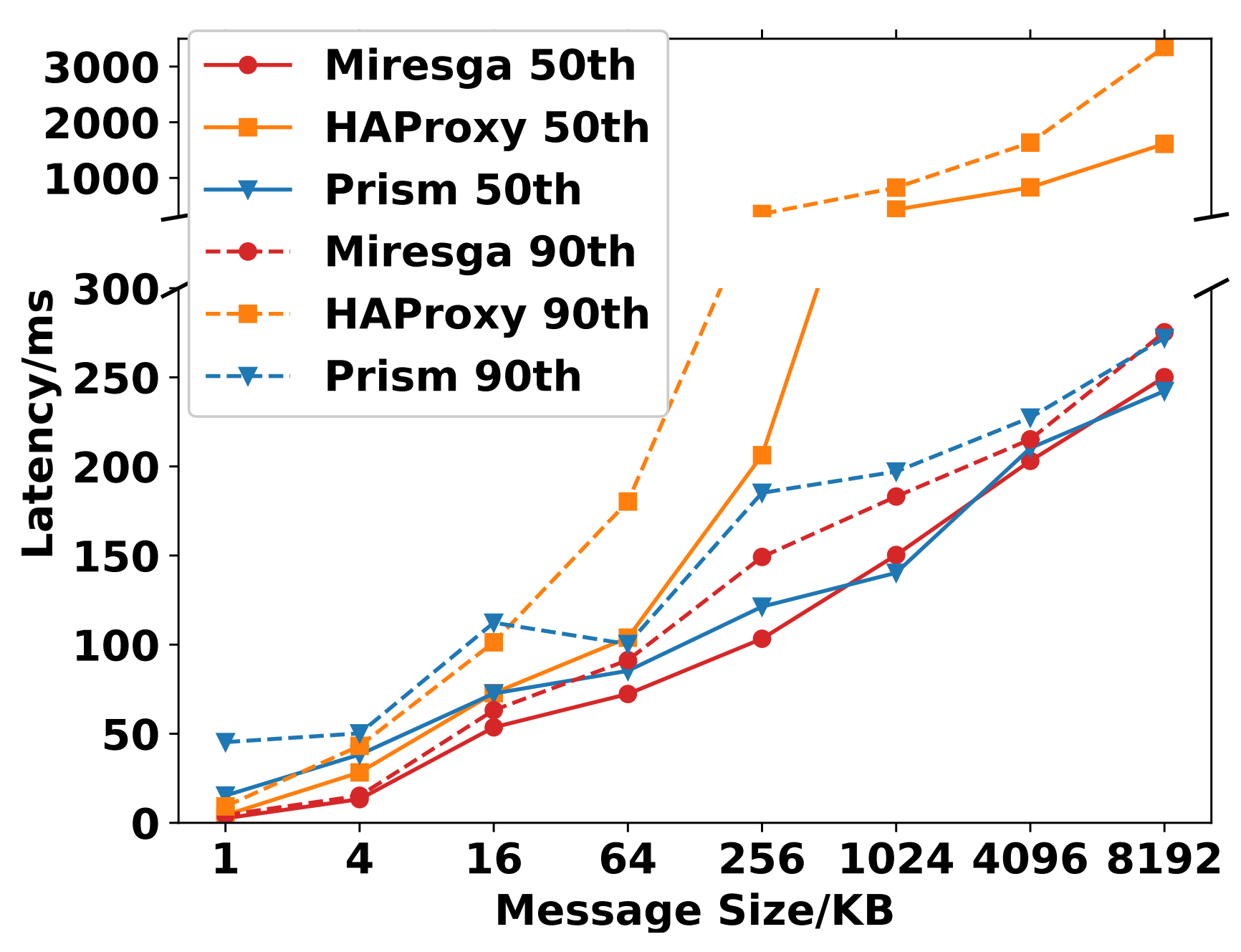
Figure 3. The FSM of the lightweight stack.

Challenge 3: State synchronization among the three components introduces additional overhead.

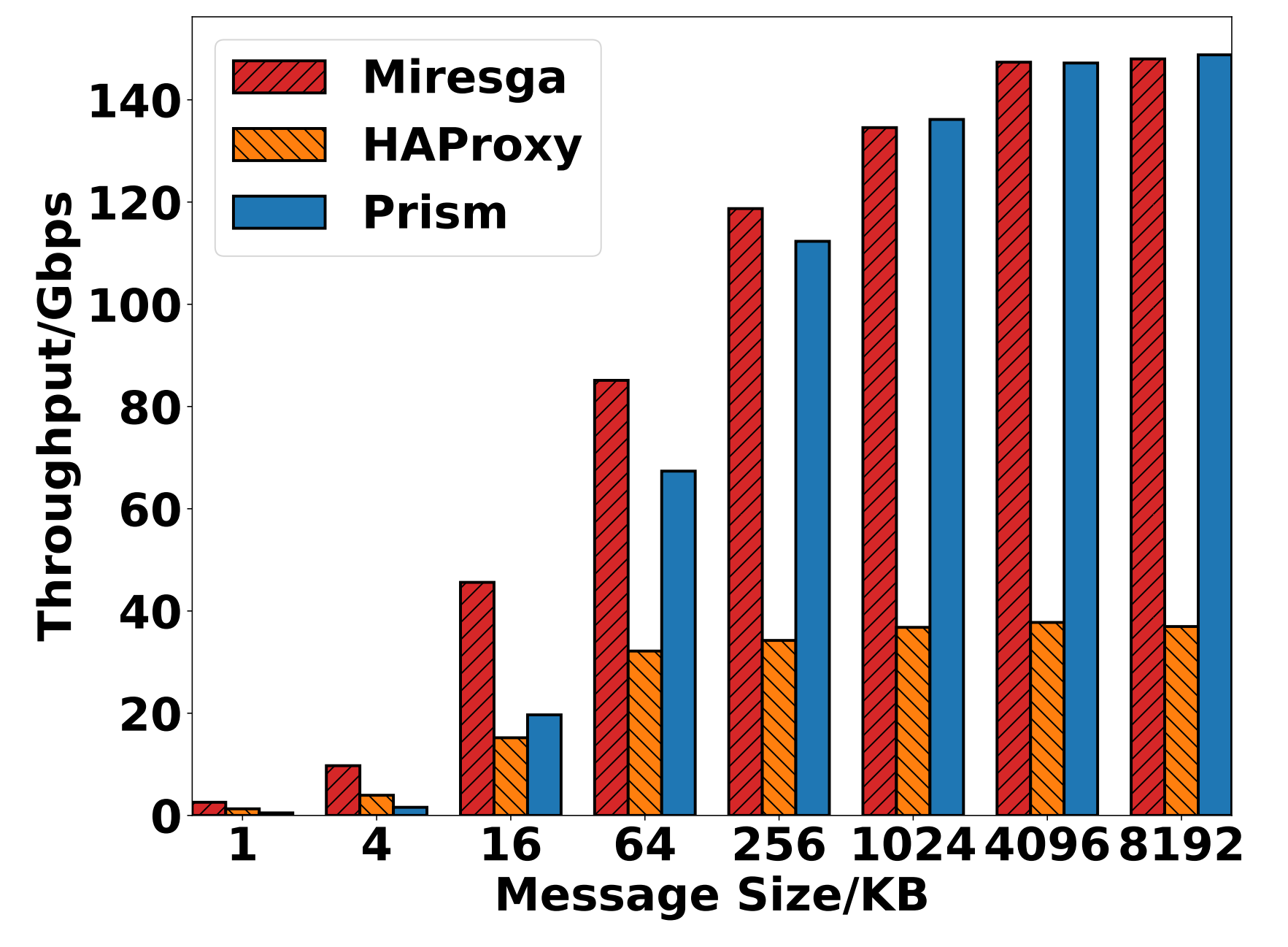
In Miresga, two types of states require synchronization: 1) the initial sequence numbers, and 2) the information for splicing connections to the offloaded flows. To synchronize the initial sequence number, Miresga fuses the initial sequence numbers into the regular packets to avoid additional delivery. For the latter, Miresga employs an asynchronous synchronization method that does not obstruct traffic flow during the synchronization process. Instead, the front-end server initially handles the traffic, and once synchronization is completed, processing is then transferred to the programmable switch.

Evaluation

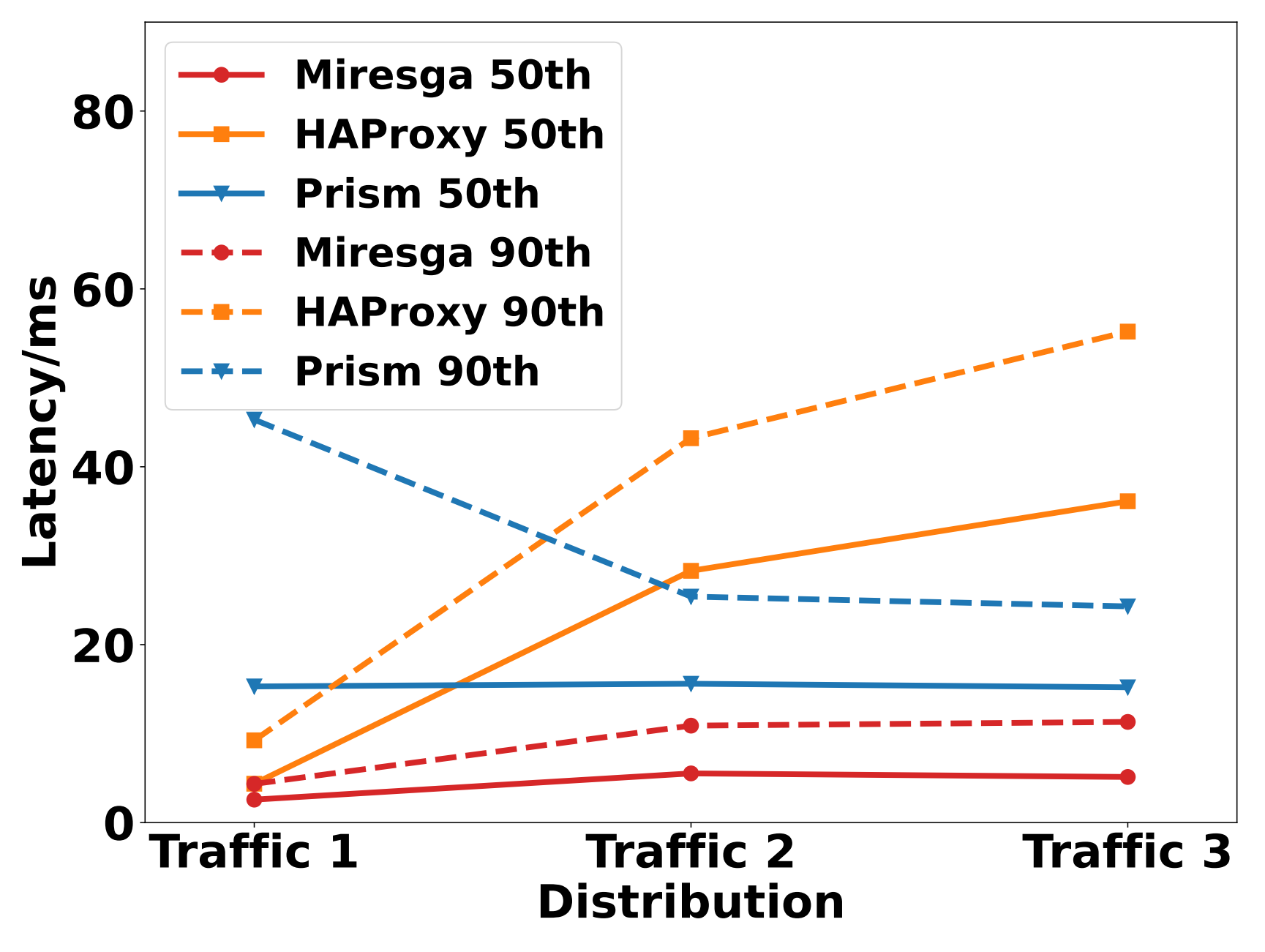
We primarily evaluated Miresga's end-to-end latency and throughput. The evaluation was conducted across two scenarios: (1) a fixed message size (Figure 4a and Figure 4b), and (2) message sizes generated according to a specific heavy-tailed distribution (Figure 4c and Figure 4d).



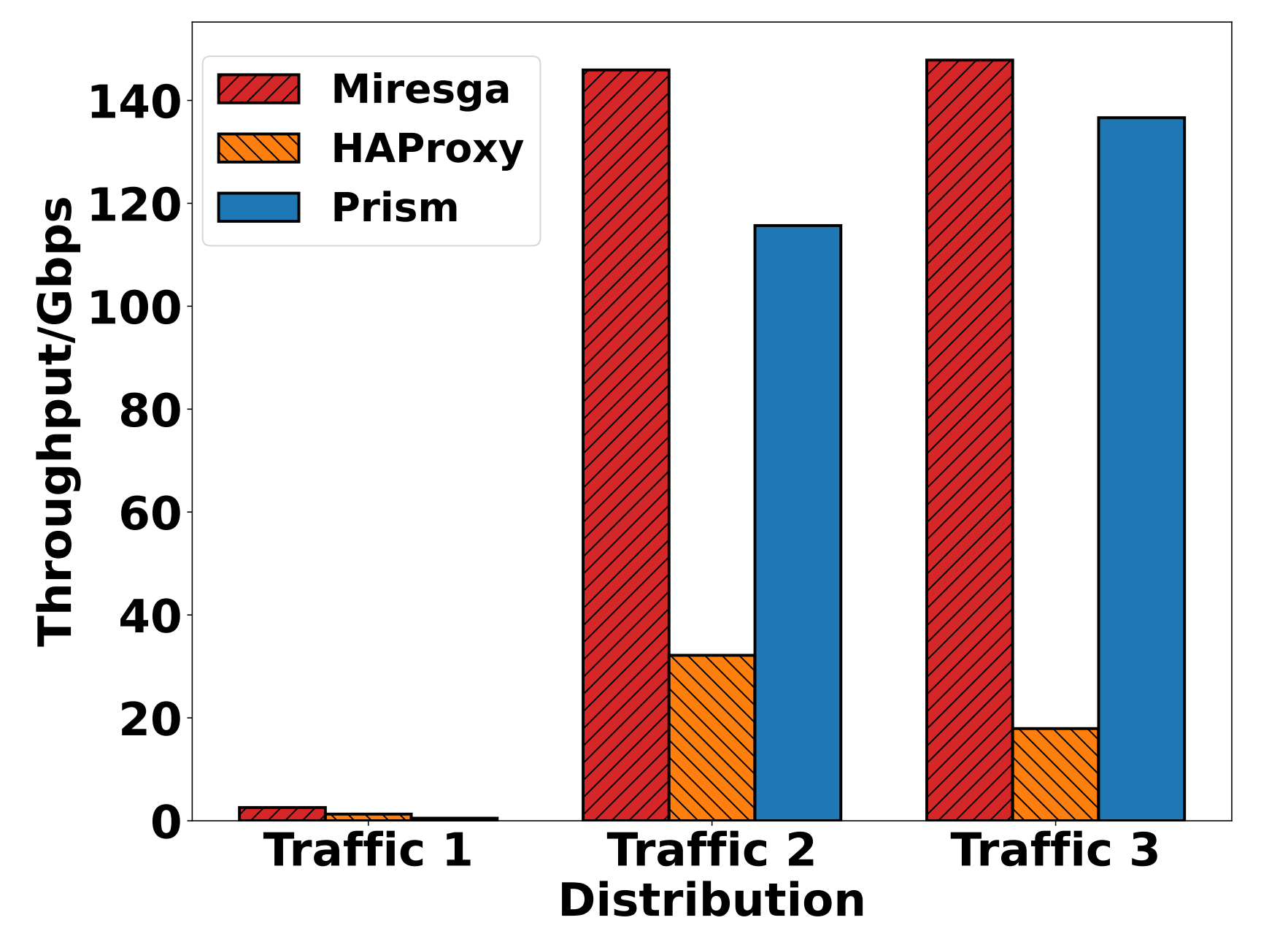
(a) Latency with different message sizes



(b) Throughput with different message sizes



(c) Latency with heavy-tailed traffic distributions



(d) Throughput with heavy-tailed traffic distributions

Figure 4. End-to-end throughput and latency comparison with different message sizes and using heavy-tailed traffic distributions. The distribution of the response size in Traffic 1 is 100% 1KB; Traffic 2 is 70% 1KB, 20% 10KB, 8% 1MB, and 2% 10MB. The distribution of the response size in Traffic 2 is 50% 1KB, 30% 10KB, 15% 1MB and 5% 10MB. We only measure the latency of 1KB message when using heavy-tailed traffic.

References

[1] Rohan Gandhi, Y Charlie Hu, and Ming Zhang. Yoda: A highly available layer-7 load balancer. In *Proceedings of the Eleventh European Conference on Computer Systems*, pages 1–16, 2016.

[2] Yutaro Hayakawa, Michio Honda, Douglas Santry, and Lars Eggert. Prism: Proxies without the pain. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 535–549, 2021.

[3] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 15–28, 2017.

[4] YoungGyoun Moon, SeungEon Lee, Muhammad Asim Jamshed, and KyoungSoo Park. Acceltcp: Accelerating network applications with stateful tcp offloading. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 77–92, 2020.

[5] Sophia Yoo, Xiaoqi Chen, and Jennifer Rexford. SmartCookie: Blocking Large-Scale SYN floods with a Split-Proxy defense on programmable data planes. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 217–234, Philadelphia, PA, August 2024. USENIX Association.

[6] Chaoliang Zeng, Layong Luo, Teng Zhang, Zilong Wang, Luyang Li, Wenchen Han, Nan Chen, Lebing Wan, Lichao Liu, Zhipeng Ding, et al. Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1345–1358, 2022.