TCP is Harmful to In-Network Computing: Designing a Message Transport Protocol (MTP)

Brent E. Stephens
University of Utah

Darius Grassi
UIC

Hamidreza Almasi
UIC

Tao Ji
University of Texas

Balajee Vamanan
UIC

Aditya Akella
University of Texas

ABSTRACT

This paper presents the motivation and design of MTP, a new offload-friendly message transport protocol. Existing transport protocols like TCP, MPTCP, and UDP/Quic all have key limitations when used in a network that may potentially offload computation from end-servers into NICs, switches, and other network devices. To enable important new in-network computing use cases and correct congestion control in the face of ever changing network paths and application replicas, MTP introduces a new message transport protocol design and pathlet congestion control, a new approach where end-hosts explicitly communicate messaging information to network devices and network devices explicitly communicate network path and congestion information back to end-hosts.

CCS CONCEPTS

• **Networks** → **Transport protocols**;

ACM Reference Format:
Brent E. Stephens, Darius Grassi, Hamidreza Almasi, Tao Ji, Balajee Vamanan, and Aditya Akella. 2021. TCP is Harmful to In-Network Computing: Designing a Message Transport Protocol (MTP). In *Proceedings of The 20th ACM Workshop on Hot Topics in Networks (HotNets'21).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3484266.3487382

1 INTRODUCTION

Existing transport protocols like TCP [35] are no longer a good fit for the needs of today's data center networks. There is an increasing need for in-network computing, *i.e.*, a network that provides offloads. Also, these multipath networks need fine-grained load-balancing, scheduling, and performance isolation. However, TCP and its stream abstraction are largely incompatible with both of these needs.

The contributions of this paper are as follows: First, we identify the emerging needs of today's networks that are incompatible or hindered by the design of TCP. Second, we present the preliminary design of MTP, a new clean-slate transport protocol that is designed explicitly to meet the needs of in-network computing. Third, we present results from simulations that demonstrate the potential benefits of using MTP.

In more detail, the following are the requirements that we have identified are needed to support in-network computing on multipath networks that are not met by TCP:

- *Data Mutation:* Offloads like compression and serialization change the size of data in a stream. This is incompatible with the TCP stream abstraction.
- *Low Buffering and Computation Requirements:* The TCP stream abstraction significantly complicates message reassembly and application-level message parsing. Further, some types of middleboxes may need large packet buffers.
- *Inter-Message Independence:* Many application-level requests are sent across the same TCP flow. This prevents in-network caches from interposing on requests and bypassing backends, and this prevents load-balancing different requests in a flow across different servers for scalability.
- *Multi-Resource and Multi-Algorithm Congestion Control:* There is a need to be able to simultaneously use any of many new congestion control algorithms [3, 5, 6, 13, 17, 19, 30, 38] to share many different types of resources [4]. Further, load-balancing in multipath networks [2, 10, 14] breaks congestion control in TCP when paths change because the feedback and ideal window for the old path is not guaranteed to be the same for the new path.
- *Multi-Entity Isolation:* TCP provides per-flow fairness. This is not a good isolation policy. The service received by an entity should not be based communication patterns.

We believe that these many limitations of TCP are fundamental, and that it is not possible to modify or extend TCP

ACM ISBN 978-1-4503-7020-2...$15.00
https://doi.org/10.1145/3484266.3487382

so as to meet all of these requirements. As such, there is a clear need for a new transport protocol designed from the ground-up to meet these new requests.

To overcome these limitations of TCP, this paper presents the preliminary design of MTP, a new message transport protocol. Building and deploying a clean-slate replacement for TCP is a significant undertaking that needs to be completed before the full benefits of in-network computing can be achieved in practice, and this new design is an important first step towards accomplishing this goal.

There are two key aspects of MTP that are together able to meet all of our requirements: a message transport protocol and a new pathlet approach to congestion control. In the message transport, messages are the granularity of retransmission, scheduling, and load-balancing, and this allows for reordering and mutation of messages. Pathlet congestion control enables end-hosts to simultaneously track congestion state of many different network resources with each resource using its own type of congestion feedback. The network can dynamically exert control over multipath routing and load balancing. End-hosts ensure that each pathlet is not congested by evolving congestion windows and/or computing rates. Further, MTP performs congestion control at the coarser granularity of traffic classes, not flows; flows that use the same pathlet share congestion information. Thus, our design provides stronger isolation among entities.

Our preliminary evaluation demonstrates that MTP enables key emerging in-network computing applications. Using ns-3 simulations, we show that MTP enables efficient multi-path congestion control and achieves significant throughput improvement over existing state-of-the-art. By providing visibility into message sizes, MTP empowers load balancers to achieve optimal bandwidth utilization while avoiding costly packet reordering. Lastly, MTP equips operators to define and enforce network policies at a coarser granularity (*e.g.*, tenants, traffic classes) than at flow granularity, which is known to cause unfairness among applications.

2 MOTIVATION

In-network computing has become commonplace to meet the needs of emerging applications. This section discusses key properties needed to support in-network computing and why transport protocols like TCP fail to meet these requirements.

2.1 The Need for In-Network Computing

Figure 1 shows an example computing cluster where in-network computing is used to accelerate a document lookup in a dynamic website like Facebook or Twitter. There are three different types of functionality in this figure:

First, there are **computational offloads** inside the network. In (1) in this figure, there is an application-aware cache that

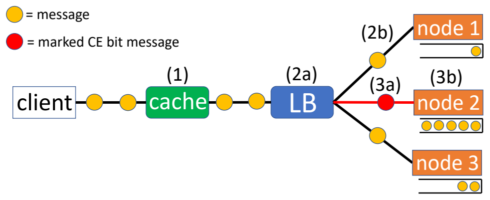


Figure 1: A network that benefits from in-network computing

is used to store hot documents and directly answer popular queries, bypassing the backend [16].

Second, the network performs **load balancing and scheduling** in the network and across different types of resources. In this example, (2a) shows an application-aware (L7) load-balancer that balances requests across different backend storage replicas [24], and (2b) shows packets being load-balanced across parallel paths in a multipath topology [2, 9, 10, 32].

Third, the network computes and communicates **congestion control** feedback, and this also happens at both network and application layers. In (3a), a request traverses a congested link and has the CE bit in its IP header set [8]. In (3b), a request lands at a congested replica, and information about processing latency and queue occupancies are communicated back to the load-balancer [42].

Further, there are many more approaches that utilize in-network intelligence. For example, recent work has offloaded computation to the network for key value stores [16, 24, 25], machine learning [23, 37], and intrusion detection systems (IDSes) [45]. Other work has shown the benefits of load-balancing and scheduling [2, 9, 10, 12, 18, 29, 32, 39, 40, 44, 47]. There are many new congestion control algorithms [3, 5, 6, 13, 17, 19, 30, 38], and recent work has shown the benefits of application-level congestion-control [4, 22, 46].

2.2 Transport-Level Requirements

To seamlessly support in-network computing, the transport protocol must provide the following features:
Data Mutation: Useful **offloads** that mutate packets and change message lengths include compression, message serialization [36, 43], and request preprocessing [20, 33].
Low Buffering and Computation Requirements: High-throughput devices like switches can implement **offloads**, but switches have limited state and computational power. To enable devices with limited compute, state, and buffering capacity, offloads should be able to process messages with bounded state and buffering requirements. Thus, it is important to provide message attributes in each packet.
Inter-Message Independence: Independent messages should be able to be sent to different offloads and end-hosts, and the network should be able to dynamically alter paths to perform load-balancing and scheduling without packet reordering. This is necessary for the in-network cache **offload** and multipath **load-balancing** in Figure 1. It is also necessary

Transport (RPF = requests per flow)	Data Mutation	Low Buffering & Computation Requirements	Inter-Message Independence	Multi-Resource & Multi-Algorithm Congestion Control	Multi-Entity Isolation
TCP Pass-Through (many RPF)	✗	✓	✗	✓	✗
TCP Pass-Through (one RPF)	✗	✓	✗	✗	✓
TCP Termination (many RPF)	✓	✗	✗	✓	✗
TCP Termination (one RPF)	✓	✗	✓	✗	✓
DCTCP	✗	✗	✗	✗	✗
UDP	✓	✓	✓	✗	✗
QUIC	✗	✓	✓	–	✗
MPTCP	✗	✗	✓	✓	✗
Swift	✗	✓	✗	✗	✗
RDMA RC	✗	✓	✗	✗	✗
RDMA UC	✗	✓	✗	✗	✗
RDMA UD	✓	✓	✓	✗	✗

Table 1: Comparison of features available in current transport protocol approaches

for an in-network KVS offload like NetCache [16] and for an in-network ML accelerator like ATP [23].

Multi-Resource and Multi-Algorithm Congestion Control: There many different types of resources (pathlets) on a path across the network to an application, and congestion control enables these resources to be shared without explicit scheduling. However, **congestion control** feedback should be actionable by end-hosts even if the resources being used change dynamically. For example, the in-network cache and the different backends in Figure 1 likely have different throughputs, and this is also the case for an in-network KVS. To avoid either under or over utilization depending on which resource is being used, end-hosts should be able to identify which resource/pathlet congestion control feedback is for. Additionally, different resources may want to provide different forms of congestion control feedback.

Multi-Entity Isolation: Different messages have differing utility, and the number of messages and flows generated by an application is not necessarily proportional to utility. For **scheduling**, **load-balancing**, and **congestion control**, it should be possible for devices to identify the provenance of every message and apply per-entity policies.

2.3 Limitations of TCP Variants

TCP can be used in many different ways. The number of open connections and the distribution of requests can vary. Some application-level (L7) load-balancers terminate TCP connections, while network-level middleboxes are often passthrough devices [21]. However, after evaluating four different TCP configurations, we find that there is no configuration of TCP that can meet all of our requirements (Table 1).

Pass-through with many messages per flow: This approach is how TCP is typically used, and it has limitations with respect to supporting mutation, load-balancing and scheduling, and multi-path and per-entity congestion control. For example, TCP's stream abstraction is incompatible with message mutation. If a packet changes in length, then sequence numbers will be incorrect. L7 parsing is difficult because application-level messages may occur anywhere in a packet. Different

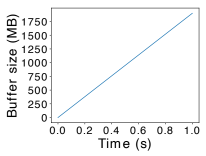


Figure 2: Buffer size at proxy with unlimited receive window

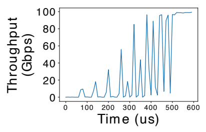


Figure 3: One request per-flow leads to congestion control issues

replicas cannot process different messages from the same stream. These properties prevent in-network caches and KVS.

This approach also causes problems with in-network load-balancing. TCP's performance is hurt by reordering [2, 10]. TCP does not distinguish between different paths, so end-hosts will incorrectly react to congestion events on the previous pathlets. TCP also does not provide multi-bit feedback.

Termination with many messages per flow: Some network devices like L7 load balancers may terminate TCP connections. In this configuration, the device accepts incoming connections from clients and then opens a second TCP connection to the server. Although this enables mutation, this has key limitations with respect to buffering and computation requirements. On devices like switches and FPGAs, it may not be feasible to perform TCP processing, and this approach also suffers from either high buffering and HOL-blocking.

We performed experiments to show that TCP termination has a limiting trade-off between buffering requirements and head-of-line (HOL) blocking latency (Figure 2). In this experiment, there is a proxy with a 100 Gbps connection to the client and a 40 Gbps connection to the server. Connections are terminated at the proxy and new traffic is generated between the proxy and sink. As expected, we observed significant buffer buildup at the proxy node over time due to the rate mismatch. We repeated the same experiment but we limited the size of the TCP receive window advertised by the proxy. In this case, we observed HOL-blocking.

One message per flow (Both Passthrough and Termination): Sending a single message per flow is not a viable approach because it can interfere with congestion control, especially for small messages. Figure 3 shows this noisy behavior when 4 hosts in a dumbbell topology with 100Gbps links generate 16KB messages with a new connection for each message.

2.4 Limitations of RDMA

RDMA provides a message transport like MTP, but it falls short of our requirements for in-network computing. RDMA provides three transport service modes: reliable connection (RC), unreliable connection (UC) and unreliable datagram (UD) [28]. RC and UC support large messages that span multiple packets while the size of a UD message is limited by the packet length. Similar to TCP, RDMA RC and UC use the packet sequence number for flow control, so it is non-trivial to support data mutations that extend a message to more fragments. Unlike TCP, RDMA RC and UC do not co-locate parts of two messages in one packet, which simplifies the buffering and computation needed to perform L7 parsing. However, both RC and UC mandate in-order packet delivery within a connection [27], which effectively disables the use of multiple paths because out-of-order packets are considered a sign of packet loss, even if they belong to different messages and are not actually lost. RDMA is not designed with support for multi-bit feedback or multi-entity isolation.

3 MTP DESIGN

This section presents the design of MTP, a new message transport protocol that is designed to be compatible with in-network computing. There are two key aspects of MTP that that allow it to meet the requirements in Section 2.2. First, MTP is message-oriented in that it groups packets together into messages instead of streams. This enables data mutation, low buffering and computation requirements, load-balancing, and scheduling. Second, MTP introduces pathlet congestion control, a new approach that enables multi-algorithm and multi-resource congestion control.

3.1 MTP Overview

3.1.1 Packet Header Overview. Figure 4 provides an illustration of the fields in an MTP header. These fields facilitate the message transport and pathlet congestion control.

First, the packet header starts with a source port and destination port. As in TCP and UDP, this information is used to route messages to the appropriate applications on a server.

The header also contains message-level information. This starts with a `Msg ID` field, which is a unique ID amongst all outstanding messages from the end-host. There is also a priority that is assigned by the application that is an integer describing the relative priority of parallel messages. Next, there are fields that describe the message length in bytes and packets, the offset and length of the current packet in bytes, and the current packet number. These fields provide the information needed for retransmission.

The rest of the information in the packet header is used for pathlet congestion control. There is a list of paths that the source is requesting the network to exclude from use with this packet. After that, there are two lists of `Path ID, Feedback` pairs that are used to evolve the congestion windows. The first list `Path Feedback` is initially empty when a packet is generated and is modified by the network devices the packet goes through. When the destination receives a packet with a path feedback list, it then copies this list to the `ACK Path Feedback` list in the header of the acknowledgement reply sent from the destination to the source. Finally, the packet header contains SACK and NACK lists used to selectively acknowledge and negatively acknowledge the different packets in messages.

3.1.2 Message Transport. In MTP, every message is independent, and connections are not established before messages are sent. End-hosts place information about the ID and size of the message in every packet of the message. This enables devices to be able to easily parse messages and know in advance how much buffering is needed to process a message. Devices are allowed to drop packets on overload, although network devices may also provide lossless forwarding. Acknowledgements and retransmissions are performed with respect to message IDs and packet numbers, not bytes, and pathlet information is used for reproducible retransmission.

The devices inside the network are allowed to mutate messages by changing data, packet lengths, and the number of packets in a message. However, this introduces challenges with respect to consistency. If the packets in a message are split across different paths and network devices, then the message may be corrupted and impossible to reassemble.

To ensure message-level consistency, network devices in MTP must process messages atomically. In other words, the network is not allowed to reorder packets from a message, and a message cannot be split across different paths and replicas.

There are two use use cases for generating messages that we envision for MTP: (1) Facilitating remote procedure calls (RPCs), and (2) sending bulk data. To facilitate RPCs, all of the packets for the RPC are placed in the same message, and each RPC generated by the application is placed in a separate message. This allows for different requests to be load-balanced and scheduled while still enabling in-network caches that need to see an entire request to generate a response. To support applications generating blobs of data, MTP can generate new messages for each packet. This enables multiplexing and parallelization at the network layer and operates similar to TCP. A layer beneath the application in a library or OS service is responsible for reassembling the blob and reliably handling any packet loss and reordering of messages.

3.1.3 Pathlet Congestion Control. MTP allows different congestion control algorithms to coexist and enables high-level policies to be easily enforced. To do this, the network communicates pathlet and congestion information via headers to end-hosts, and end-hosts communicate back to the

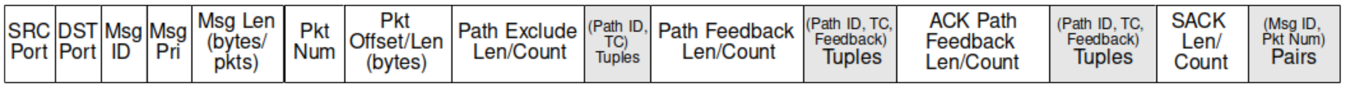


Figure 4: The MTP Header Format

network which pathlets the network should not use because the end-host has received feedback that they are congested.

Congestion feedback is provided at the pathlet granularity, where pathlets are identified by a unique ID. It is the responsibility of the network to determine how to group resources and assign pathlets, and there are trade-offs associated with different granularities. Representing the entire network as a single pathlet mimics TCP. At the other extreme, every single resource can be identified as a separate pathlet.

Instead of maintaining per-flow congestion windows, MTP end-hosts perform per-pathlet congestion control and maintain per-pathlet congestion windows. The feedback for each pathlet is identified by a Type-Length-Value. This allows for algorithms like RCP [6] and DCTCP [3] to coexist.

Pathlets may be differently congested, and it is possible for an end-host to be congested on some pathlets and not others. To handle this scenario, MTP has end-hosts provide feedback to the network about the pathlets that should not be used.

3.2 Requirements

MTP is able to meet all of the requirements in Section 2.2:
Data Mutation: MTP's message transport is designed to enable data mutation without corrupting packets.
Low Buffering and Computation: MTP allows devices to make per-message decisions about buffering, and each message is easily parsable since messages start at packet 0.
Inter-Message Independence: Messages in MTP are independent and may be sent on separate paths and processed by different network devices.
Multi-Resource and Multi-Algorithm Congestion Control: Through pathlet congestion control feedback, different segments can provide different types of network feedback.
Multi-Entity Isolation: For each MTP message, applications assign a priority, and network pathlets assign a TC. Because end-hosts evolve congestion windows for each pathlet and TC pair, this enables per-entity resource allocation.

4 DISCUSSION

There are still some questions to answer about MTP:
Interaction with TCP: MTP can coexist with legacy TCP devices. In this scenario, the MTP header can be included as a new TCP option, and MTP devices can bridge TCP islands.
Pathlet ID Choice: Pathlet IDs enable flexibility in how MTP is used. Using a single pathlet mimics TCP, and using a different pathlet for every resource precise feedback but with higher overheads. How to best define pathlets is an open question.
Packet Header Overheads: Packet headers in MTP can theoretically grow to be larger than today's TCP headers. To overcome this challenge, we plan to look into ways to reduce the size of MTP packet headers. For example, feedback can be aggregated, and feedback can be selectively returned.
Security Considerations: Message data in MTP can be encrypted. Encrypting messages makes it harder to accelerate application-specific computation, but this is still possible by sharing keys to trusted devices or with privacy preserving cryptography [41]. Also, this is a general problem with in-network computing that MTP does not make worse.
Managing Complexity: The use of pathlets makes the complexity of MTP flexible. We expect that the complexity of supporting our motivating use cases can still be low and manageable. To support this claim, we note that it is simple to use MTP to support simple use cases:

TCP Congestion Control: If the network is a single pathlet, MTP can behave as existing congestion control algorithms like TCP [35], DCTCP [3], and DCQCN [48].

NDP [13]: By design, implementing NDP in MTP is simple. End-hosts learn about available paths from the network, and switches generate NACKs to implement packet trimming.

*ML Training (*e.g. *ATP [23]):* In-network aggregation of gradients is challenging for congestion control because aggregation-levels can change over time. MTP can improve the precision of congestion control in ATP by making aggregation levels and pathlets explicit.

KVS [16] and Middlebox [21] Offloads: MTP uses pathlets to improve the resource sharing problems that arise with in-network KVSes and middleboxes. Using separate pathlets for an in-network offload allows for a fast in-network KVS to be fully utilized without overloading slower downstream replicas, and it enables upstream NFs avoid wasting resources.

5 A STUDY OF POTENTIAL BENEFITS

We demonstrate MTP's benefits in emerging network scenarios using an implementation in ns-3 simulator [1].

5.1 Multi-path congestion control

When the network frequently changes paths (*e.g.*, dynamic load balancers), TCP windows will be inaccurate. This can lead to under and over utilization, and TCP can experience longer convergence times. In some cases, TCP may *not* converge at all. Because MTP isolates congestion information for each pathlet, it does not suffer from this problem.

To quantify this benefit, we consider a scenario where there are two paths—a fast path (100 Gbps) and a slow path

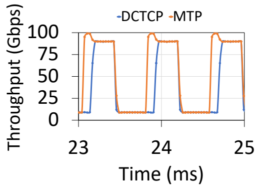


Figure 5: Multi-path CC

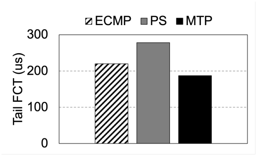
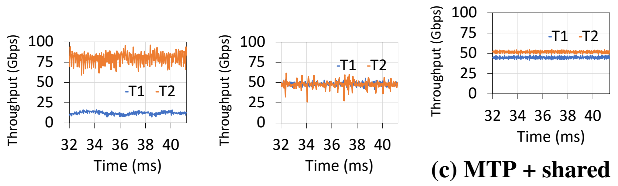


Figure 6: Load-and-request-aware load balancing



(a) Shared queue **(b) Separate queue** **(c) MTP + shared queue**

Figure 7: Per-entity isolation

(10 Gbps)—between a sender and a receiver. The first hop switch periodically alternates between the two paths (*e.g.*, an optical switch) every 384 μs. The links have a delay of 1 μs; the switch buffer size and ECN threshold are set at 128 packets and 20 packets, respectively. We start a long-lasting flow and measure the flow throughput every 32 μs. We repeat the experiment for DCTCP and MTP, and we compare their throughput in Figure 5. We observe that MTP (orange) converges faster than DCTCP (blue) and achieves 33% higher goodput on average. This experiment shows the potential benefit of using MTP with optical networks and application-level load balancers that frequently switch paths.

5.2 Load- and request-aware load balancers

MTP provides more visibility to load balancers about message attributes (*e.g.*, size) and allows dynamic path changes based on load. To show this potential benefit, we compare ECMP and packet spraying approaches with an MTP-enabled in-network load balancer that considers both network load and request size. In this experiment, we have a sender communicate with a receiver along two 100 $Gbps$ paths with one of them having an additional delay of $1\mu s$. The workload consists of a mix of message sizes (10 KB–1GB). The message size distribution is skewed toward short messages as per existing studies [3]. Figure 6 shows tail flow completion times (99^{th} percentile). While ECMP suffers higher delays due to unbalanced path delays, packet spraying incurs higher packet reordering. In contrast, MTP-based load balancer achieves near-perfect load balancing without reordering.

5.3 Per-entity isolation

By enabling operators to define and enforce policies at the level of traffic classes (TC) as opposed to flows or messages, MTP avoids fairness problems of TCP (*e.g.*, applications with more flows get a higher share). We demonstrate MTP's benefit using a simple policy of fairly sharing bandwidth between two tenants. We perform an experiment where two tenants send data to receivers across a common 100 $Gbps$/10 μs link via a common switch. The second tenant generates 8x the number of messages as the first tenant. We compare three systems: the baseline uses DCTCP and a single common shared queue in the switch, the second system uses two separate queues for each tenant, and an MTP-enabled third system enforces a fair-share policy between tenants at the switch ingress without requiring separate queues. Figure 7 shows the aggregate throughput achieved by the two tenants. We see that with a shared queue, Tenant 2 achieves roughly 8x throughput than Tenant 1 (i.e., 80 Gbps vs. 10 Gbps). Both separate queue and MTP-enabled shared queue achieve nearly *equal sharing* of capacity. While providing separate queues for entities is expensive, MTP provides enough information to switches to enable policy enforcement without requiring separate queues.

6 RELATED WORK

MTP is inspired by both existing message-based protocols [7, 15, 26] and pathlet routing [11]. Ports and Nelson [34] point out some similar problems with existing transport protocols. There are many interesting congestion control algorithms [3, 5, 6, 13, 19, 30, 30]), and MTP is designed to be able to implement these algorithms. MPTCP splits a stream into multiple substreams [31], but its congestion response will likely suffer when in-network load balancing schemes switch paths. Swift [22] uses delay measurement to identify bottlenecks in both the network and end-hosts. MTP is also designed to be able to implement Swift.

7 CONCLUSIONS

Today's networks look very different than networks did when TCP was created in 1981 [35], and TCP has become a burden. We believe it is now time to create a new, clean-slate transport protocol to support in-network computing. This paper presents the preliminary design of a MTP, a message transport protocol intended to replace TCP for in-network computing applications. By providing visibility of message attributes to the transport layer and by disaggregating congestion feedback from distinct parts of the network path, MTP seamlessly supports features required by many applications of in-network computing. We present results from simulations that demonstrate the potential benefits of using MTP with modern load balancers and enforcing stronger isolation policies. Encouraged by positive findings, we plan to implement MTP in our testbed and study its behavior with real applications.

Acknowledgements: We thank our shepherd, Paolo Costa, and the anonymous reviewers for their feedback. This work was funded by NSF Awards CNS-2008273, CNS-1942686, CNS-1850053, CNS-1565277, and CNS-1717039 and by gifts from Google and Facebook.

REFERENCES

[1] The ns-3 discrete-event network simulator. http://www.nsnam.org.

[2] ALIZADEH, M., EDSALL, T., DHARMAPURIKAR, S., VAIDYANATHAN, R., CHU, K., FINGERHUT, A., LAM, V. T., MATUS, F., PAN, R., YADAV, N., AND VARGHESE, G. CONGA: Distributed Congestion-Aware Load Balancing for Datacenters. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2014), SIGCOMM, Association for Computing Machinery.

[3] ALIZADEH, M., GREENBERG, A., MALTZ, D. A., PADHYE, J., PATEL, P., PRABHAKAR, B., SENGUPTA, S., AND SRIDHARAN, M. Data Center TCP (DCTCP). In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2010), SIGCOMM, Association for Computing Machinery.

[4] CHO, I., SAEED, A., FRIED, J., PARK, S. J., ALIZADEH, M., AND BELAY, A. Overload control for µs-scale rpcs with breakwater. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (2020), USENIX Association.

[5] DONG, M., LI, Q., ZARCHY, D., GODFREY, P. B., AND SCHAPIRA, M. PCC: Re-architecting congestion control for consistent high performance. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2015), USENIX Association.

[6] DUKKIPATI, N. *Rate Control Protocol (Rcp): Congestion Control to Make Flows Complete Quickly*. PhD thesis, Stanford, CA, USA, 2008. AAI3292347.

[7] DUNNING, D., REGNIER, G., MCALPINE, G., CAMERON, D., SHUBERT, B., BERRY, F., MERRITT, A., GRONKE, E., AND DODD, C. The virtual interface architecture. *IEEE Micro 18*, 2 (1998), 66–76.

[8] FLOYD, S., RAMAKRISHNAN, D. K. K., AND BLACK, D. L. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, Sept. 2001.

[9] GANDHI, R., LIU, H. H., HU, Y. C., LU, G., PADHYE, J., YUAN, L., AND ZHANG, M. Duet: Cloud scale load balancing with hardware and software. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2014), SIGCOMM, Association for Computing Machinery.

[10] GHORBANI, S., YANG, Z., GODFREY, P. B., GANJALI, Y., AND FIROOZSHAHIAN, A. DRILL: Micro Load Balancing for Low-Latency Data Center Networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), SIGCOMM, Association for Computing Machinery.

[11] GODFREY, P. B., GANICHEV, I., SHENKER, S., AND STOICA, I. Pathlet routing. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2009), SIGCOMM, Association for Computing Machinery.

[12] GROSVENOR, M. P., SCHWARZKOPF, M., GOG, I., WATSON, R. N. M., MOORE, A. W., HAND, S., AND CROWCROFT, J. Queues don't matter when you can JUMP them! In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2015), USENIX Association.

[13] HANDLEY, M., RAICIU, C., AGACHE, A., VOINESCU, A., MOORE, A. W., ANTICHI, G., AND WÓJCIK, M. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), SIGCOMM, Association for Computing Machinery.

[14] HE, K., ROZNER, E., AGARWAL, K., FELTER, W., CARTER, J., AND AKELLA, A. Presto: Edge-based load balancing for fast datacenter networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2015), SIGCOMM, Association for Computing Machinery.

[15] Infiniband trade association, 2017.

[16] JIN, X., LI, X., ZHANG, H., SOULÉ, R., LEE, J., FOSTER, N., KIM, C., AND STOICA, I. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of the Symposium on Operating Systems Principles* (2017), SOSP, Association for Computing Machinery.

[17] JOSE, L., YAN, L., ALIZADEH, M., VARGHESE, G., MCKEOWN, N., AND KATTI, S. High speed networks need proactive congestion control. In *Proceedings of the ACM Workshop on Hot Topics in Networks* (2015), HotNets, Association for Computing Machinery.

[18] KAFFES, K., CHONG, T., HUMPHRIES, J. T., BELAY, A., MAZIÈRES, D., AND KOZYRAKIS, C. Shinjuku: Preemptive scheduling for µsecond-scale tail latency. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2019), USENIX Association.

[19] KATABI, D., HANDLEY, M., AND ROHRS, C. Congestion control for high bandwidth-delay product networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2002), SIGCOMM, Association for Computing Machinery.

[20] KAUFMANN, A., PETER, S., SHARMA, N. K., ANDERSON, T., AND KRISHNAMURTHY, A. High Performance Packet Processing with FlexNIC. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems* (2016), ASPLOS, Association for Computing Machinery.

[21] KULKARNI, S. G., ZHANG, W., HWANG, J., RAJAGOPALAN, S., RAMAKRISHNAN, K. K., WOOD, T., ARUMAITHURAI, M., AND FU, X. NFVnice: Dynamic Backpressure and Scheduling for NFV Service Chains. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), SIGCOMM, Association for Computing Machinery.

[22] KUMAR, G., DUKKIPATI, N., JANG, K., WASSEL, H. M. G., WU, X., MONTAZERI, B., WANG, Y., SPRINGBORN, K., ALFELD, C., RYAN, M., WETHERALL, D., AND VAHDAT, A. Swift: Delay is simple and effective for congestion control in the datacenter. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2020), SIGCOMM, Association for Computing Machinery.

[23] LAO, C., LE, Y., MAHAJAN, K., CHEN, Y., WU, W., AKELLA, A., AND SWIFT, M. ATP: In-network aggregation for multi-tenant learning. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2021), USENIX Association.

[24] LI, J., NELSON, J., MICHAEL, E., JIN, X., AND PORTS, D. R. K. Pegasus: Tolerating skewed workloads in distributed storage with in-network coherence directories. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (2020), USENIX Association.

[25] LI, X., SETHI, R., KAMINSKY, M., ANDERSEN, D. G., AND FREEDMAN, M. J. Be Fast, Cheap and in Control with SwitchKV. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2016), USENIX Association.

[26] LIU, J., WU, J., AND PANDA, D. K. High Performance RDMA-based MPI Implementation over InfiniBand. *Int. J. Parallel Program. 32*, 3 (June 2004), 167–198.

[27] LU, Y., CHEN, G., LI, B., TAN, K., XIONG, Y., CHENG, P., ZHANG, J., CHEN, E., AND MOSCIBRODA, T. Multi-path transport for RDMA in datacenters. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2018), USENIX Association.

[28] MELLANOX TECHNOLOGIES. RDMA Aware Networks Programming User Manual, 2015.

[29] MIAO, R., ZENG, H., KIM, C., LEE, J., AND YU, M. 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* (2017), SIGCOMM, Association for Computing Machinery.

[30] MONTAZERI, B., LI, Y., ALIZADEH, M., AND OUSTERHOUT, J. Homa: A receiver-driven low-latency transport protocol using network priorities. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2018), SIGCOMM, Association for Computing Machinery.

[31] PAASCH, C., BARRE, S., ET AL. Multipath TCP implementation in the Linux kernel. Available from http://www.multipath-tcp.org.

[32] PATEL, P., BANSAL, D., YUAN, L., MURTHY, A., GREENBERG, A., MALTZ, D. A., KERN, R., KUMAR, H., ZIKOS, M., WU, H., KIM, C., AND KARRI, N. Ananta: Cloud scale load balancing. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2013), SIGCOMM, Association for Computing Machinery.

[33] PHOTHILIMTHANA, P. M., LIU, M., KAUFMANN, A., PETER, S., BODIK, R., AND ANDERSON, T. Floem: A Programming System for NIC-Accelerated Network Applications. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation* (2018), OSDI, USENIX Association.

[34] PORTS, D. R. K., AND NELSON, J. When should the network be the computer? In *Proceedings of the Workshop on Hot Topics in Operating Systems* (2019), HotOS, Association for Computing Machinery.

[35] POSTEL, J. Transmission control protocol. RFC 793, September 1981.

[36] RAGHAVAN, D., LEVIS, P., ZAHARIA, M., AND ZHANG, I. Breakfast of Champions: Towards Zero-Copy Serialization with NIC Scatter-Gather. In *Proceedings of the Workshop on Hot Topics in Operating Systems* (2021), HotOS, Association for Computing Machinery.

[37] SAPIO, A., CANINI, M., HO, C.-Y., NELSON, J., KALNIS, P., KIM, C., KRISHNAMURTHY, A., MOSHREF, M., PORTS, D., AND RICHTARIK, P. Scaling distributed machine learning with in-network aggregation. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2021), USENIX Association.

[38] SHARMA, N. K., KAUFMANN, A., ANDERSON, T., KRISHNAMURTHY, A., NELSON, J., AND PETER, S. Evaluating the power of flexible packet processing for network resource allocation. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2017), USENIX Association.

[39] SHARMA, N. K., LIU, M., ATREYA, K., AND KRISHNAMURTHY, A. Approximating fair queueing on reconfigurable switches. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2018), USENIX Association.

[40] SHARMA, N. K., ZHAO, C., LIU, M., KANNAN, P. G., KIM, C., KRISHNAMURTHY, A., AND SIVARAMAN, A. Programmable calendar queues for high-speed packet scheduling. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2020), USENIX Association.

[41] SHERRY, J., LAN, C., POPA, R. A., AND RATNASAMY, S. BlindBox: Deep Packet Inspection over Encrypted Traffic. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2015), SIGCOMM, Association for Computing Machinery.

[42] SURESH, L., CANINI, M., SCHMID, S., AND FELDMANN, A. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2015), USENIX Association.

[43] WOLNIKOWSKI, A., IBANEZ, S., STONE, J., KIM, C., MANOHAR, R., AND SOULÉ, R. Zerializer: Towards Zero-Copy Serialization. In *Proceedings of the Workshop on Hot Topics in Operating Systems* (2021), HotOS, Association for Computing Machinery.

[44] YU, Z., WU, J., BRAVERMAN, V., STOICA, I., AND JIN, X. Twenty Years After: Hierarchical Core-Stateless Fair Queueing. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2021), USENIX Association.

[45] ZHAO, Z., SADOK, H., ATRE, N., HOE, J. C., SEKAR, V., AND SHERRY, J. Achieving 100Gbps intrusion prevention on a single server. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (2020), USENIX Association.

[46] ZHOU, H., CHEN, M., LIN, Q., WANG, Y., SHE, X., LIU, S., GU, R., OOI, B. C., AND YANG, J. Overload Control for Scaling WeChat Microservices. In *Proceedings of the ACM Symposium on Cloud Computing* (2018), SoCC, Association for Computing Machinery.

[47] ZHU, H., KAFFES, K., CHEN, Z., LIU, Z., KOZYRAKIS, C., STOICA, I., AND JIN, X. RackSched: A microsecond-scale scheduler for rack-scale computers. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (2020), USENIX Association.

[48] ZHU, Y., ERAN, H., FIRESTONE, D., GUO, C., LIPSHTEYN, M., LIRON, Y., PADHYE, J., RAINDEL, S., YAHIA, M. H., AND ZHANG, M. Congestion control for large-scale RDMA deployments. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2015), SIGCOMM, Association for Computing Machinery.