Towards Incremental MTU Upgrade for the Internet

Junghan Yoon
Seoul National University

Youngmin Choi
KAIST

Juyoung Park
Seoul National University

Daehyeok Kim
University of Texas Austin

Changhoon Kim
Unaffiliated (now at Google)

KyoungSoo Park
Seoul National University

ABSTRACT

This paper proposes a systematic approach to incrementally enabling large MTUs in the Internet. We demonstrate that increasing the MTU size significantly enhances the performance of both middleboxes and end hosts. To bridge MTU mismatches at network borders, we introduce PacketExpress gateway (PXGW), an MTU-translating gateway that dynamically adjusts packet sizes for cross-traffic. PXGW merges and splits TCP payloads on the fly and tunnels UDP packets, ensuring seamless adaptation. Also, we propose F-PMTUD, a new path MTU discovery algorithm that determines the path MTU within a single round-trip without relying on ICMP. Our preliminary evaluation shows that the PXGW prototype achieves 1.45 Tbps of packet forwarding throughput using only 8 CPU cores. After dynamic conversion, 94% of transmitted TCP packets are 9000 B jumbo frames, indicating that most flows were effectively converted into large segments, thereby demonstrating the system's efficiency and scalability. We also find that large-MTU packets, made available via PXGW, enhance end-host performance by up to 2.5×.

CCS CONCEPTS

• **Networks** → **Network architectures**; **Routers**; **Middle boxes / network appliances**; **Public Internet**; **In-network processing**.

KEYWORDS

WAN, MTU, PMTUD, NIC

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

HotNets '25, November 17–18, 2025, College Park, MD, USA

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-2280-6/25/11
https://doi.org/10.1145/3772356.3772411

ACM Reference Format:

Junghan Yoon, Youngmin Choi, Juyoung Park, Daehyeok Kim, Changhoon Kim, and KyoungSoo Park. 2025. Towards Incremental MTU Upgrade for the Internet. In *The 24th ACM Workshop on Hot Topics in Networks (HotNets '25), November 17–18, 2025, College Park, MD, USA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3772356.3772411

1 INTRODUCTION

The origins of 1500-byte (1500 B) MTU on the Internet trace back to the first Ethernet specification in 1980 [41]. This document defines the payload size of an Ethernet frame to be between 46 and 1500 B, which comes down even to the latest standard [28]. The minimum size of 46 B was necessary for reliable collision detection in the shared medium [29], whereas the maximum size of 1500 B has long been a mystery in the community.

Recently, Bob Metcalfe clarified this magic number by stating that 1500 B was chosen to align with the disk sector size of Xerox-D systems [15] at that time, so that the OS can process data at sector boundaries.[1] He also explained a general strategy of the choice: increasing the payload size would lead to higher forwarding delays and greater susceptibility to packet losses due to channel errors, while a smaller size would reduce efficiency. As expected, the 1500 B limit had no intrinsic connection to Ethernet's fundamental operation. Nevertheless, it remains the de facto MTU across the entire Internet today!

Unfortunately, the legacy MTU of 1500 B is highly inefficient, especially for emerging applications that require a large network bandwidth. 8K/16K online video streaming [25, 44], cloud virtual reality (VR) applications [34, 45], and holoportation [38] often demand 100s of Mbps to over 1 Gbps of throughput per flow. Accommodating large physical bandwidth is viable as modern commodity NICs [4, 8, 35] can deliver 100s of Gbps while off-the-shelf switches can handle 10s of Tbps of traffic [10, 14, 21]. However, the key deterrent is the lack of processing cycles as CPU advancements have slowed down significantly [6, 43], making it too costly for middleboxes or end hosts to process such a large volume of

[1] Personal communication with Bob Metcalfe on 4/22/2024.

traffic, especially when transmitted in small legacy-MTU-sized packets. In the near future, the introduction of 800 Gbps or 1.6 Tbps NICs will require even hardware switches to handle over 67 or 133 million packets per second for a 1500 B MTU. Upgrading the MTU may soon become a necessity.

Employing a large MTU can substantially alleviate the CPU requirement for handling bandwidth-heavy applications. A large MTU increases the payload-to-header ratio and improves the packet-processing stack's efficiency (e.g., by reducing DMA overhead and minimizing the number of packets to process). Additionally, it enhances throughput by increasing the congestion window more rapidly. The benefit is more pronounced in middleboxes that perform multiple rule table lookups per packet, including carrier-grade NATs, firewalls, L4 load balancers, and a user plane function (UPF) in a 5G cellular network.

The performance benefits of larger MTUs are substantial and compelling. Our experiments show that a larger MTU size dramatically improves system performance. We observe that a 5G UPF achieves 5.6× higher throughput with 9 KB MTU compared to 1500 B MTU, reaching 208 Gbps on a single CPU core. End hosts also benefit significantly – even when accounting for modern optimizations like Large Receive Offload (LRO) or Generic Receive Offload (GRO), large MTUs still outperform legacy MTU with G/LRO by 5.4× in WAN environments. Compared to an alternative approach that employs parallel connections, a large MTU reduces the CPU usage by up to 2.88×.

However, adopting a large MTU across arbitrary Internet paths is challenging, as upgrading the MTU of every network on a flag day is virtually impossible. Instead, we envision a pragmatic approach that allows selective upgrades only for the networks that are willing to deploy a large MTU. For example, cellular access/core networks, data centers, and large enterprise networks, typically operated by a single administrative domain, can implement a large MTU and benefit from the increased efficiency within their networks. The key difference from the current practice of configuring a large MTU only for intra-network traffic is that our approach enables MTU upgrades even for inter-network traffic even when the external networks continue to use the legacy MTU of 1500 B.

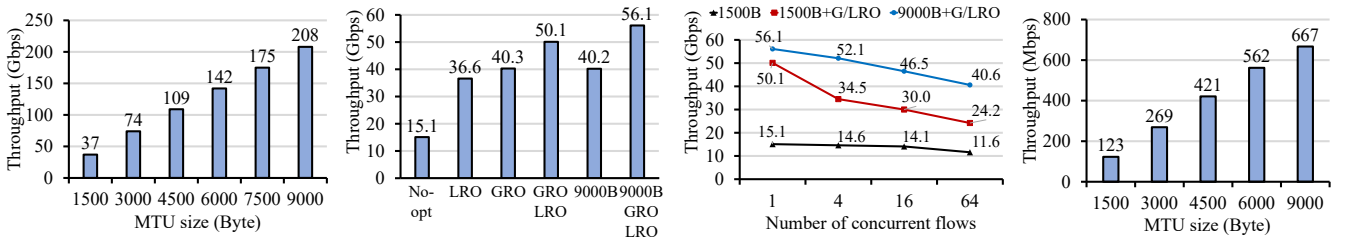
In this paper, we propose PacketExpress (PX), an in-network framework that dynamically adjusts the MTU of packets entering and exiting a beneficiary network of large MTU (called b-network in this paper) while maintaining compatibility with neighboring networks. The key idea is simple: PX deploys flow-aware, packet-level middleboxes, called *PX Gateway* (PXGW), at the border of the b-network. PXGW transparently translates the MTU of the packets between the b-network (A) and its neighboring networks (B). For example, if b-network A uses a large MTU (e.g., 9 KB) while neighboring network B retains a normal MTU (e.g., 1500 B), PXGW merges incoming normal-MTU packets from network B into large-MTU packets for network A. Similarly, PXGW splits large-MTU packets from network A into normal-MTU packets and then forwards them to network B. Within network A, all devices (e.g., switches, routers, middleboxes, and end hosts) are configured to operate with the larger MTU, benefiting from the larger payload size and fewer packets. This approach is transparent to neighboring networks, requiring no MTU upgrade outside the b-network.

Realizing PX entails several key technical challenges. The PXGW, located at network borders, must achieve very high throughput and low latency to handle Tbps traffic. Additionally, we must ensure transport-layer protocol conformance during MTU translation and require a more robust path MTU discovery scheme to fully harness the advantages of a large MTU over eligible path segments, as existing PMTUD [13] and PLPMTUD [31] suffer from issues such as ICMP black holes [26] and performance limitations. To address these challenges, we leverage NIC offload capabilities such as LRO, TCP segmentation offload (TSO), receive side scaling (RSS), and NIC hairpin, and introduce F-PMTUD, a new path MTU discovery algorithm that determines the path MTU within a single round-trip without relying on ICMP, along with UDP-caravan for handling UDP packet translation.

Our preliminary evaluation shows that a DPDK [22]-based PXGW prototype achieves 1.45 Tbps of packet forwarding performance with only 8 CPU cores. After dynamic conversion, 94% of the TCP packets become 9 KB jumbo packets, confirming the effectiveness of flow-level packet size conversion. We also find that large-MTU packets, which are made available via PXGW, enhance end host performance by 1.8× to 2.5×.

2 TRADE-OFFS OF THE MTU SIZE

The network environment has significantly evolved over the past four decades, allowing for a revisit of the legacy MTU choice, based on small forwarding latency and low bit error rates. The link bandwidth has increased by several orders of magnitude, significantly reducing forwarding latency. For example, in a 400 Gbps network, transmitting a 9 KB packet takes only 0.18 µs, and even a 64 KB packet takes 1.31 µs. Meanwhile, bit error rates (BER) have substantially improved due to advancements in modern NICs and switches. The IEEE standard mandates that the 64 B frame loss rate be less than 6.2×10^{-11} for 200/400 GbE-SR/VR [27], which means there is an average of one frame loss per every 1261 GB of data transferred, even with an MTU of 64 KB. While a larger MTU would result in more data being retransmitted in the case of an error, the impact remains minimal given the low probability. We discuss the pros and cons of MTU upgrades below.



(a) Impact of MTU size on the 5G UPF performance **(b) Impact of G/LRO (single flow)** **(c) Impact of concurrent flows** **(d) Impact of MTU size for WAN connection (single flow)**

Figure 1: Effectiveness of large MTU

# of Sessions	1 Conn. (MTU 9000 B)	6 Conn. (MTU 1500 B)
1	20.20%	19.52%
10	22.12%	34.53%
100	34.72%	100.00%

Table 1: Server-side CPU usage comparison: single TCP connection with 9 KB MTU vs. multiple parallel connections with 1500 B MTU. We use `axel` [1] to use parallel TCP.

2.1 Why is a Larger MTU Beneficial?

Higher TCP throughput. A larger MTU scales the congestion window size more rapidly, resulting in higher throughput. In the TCP slow start phase, the congestion window increases by one maximum segment size (MSS) per acknowledgment (ACK), and in the congestion avoidance phase, the window grows by one MSS per round-trip time (RTT) [2, 3]. Thus, a larger MSS would allow faster congestion window ramp-up at start while the TCP throughput in the steady state is proportional to the MSS, assuming the packet loss rate remains unchanged [32, 39]. The performance benefit persists even if one large-MTU packet is translated into multiple smaller packets, provided the network has sufficient bandwidth.

Improved CPU efficiency. A larger MTU reduces the number of packets, thereby improving packet I/O and packet processing performance. Larger packets enhance DMA efficiency and significantly diminish per-packet overhead. Also, a large MTU provides even greater benefits to middleboxes. We demonstrate this benefit with an open-sourced cellular UPF of the Open Mobile Evolved Core (OMEC) project [17]. The UPF leverages BESS [19], a modular software switch that runs on DPDK [22], which serves as the software-based datapath for UPF. The rule setup for the UPF is described in Section 5. We use iPerf to generate 800 concurrent TCP flows and configure the UPF to use a single CPU core.

Figure 1a shows that the 5G UPF throughput scales almost linearly with MTU size. The UPF achieves 208 Gbps with the 9 KB MTU, even on a single core, a speedup of 5.6× over the 1500 B-MTU. Larger MTUs are particularly advantageous to the UPF performance, as the UPF processes packets only based on the header information.

2.2 Potential Alternatives and their Pitfalls

Parallel connections. An alternative approach to increasing throughput is to employ parallel connections, which can scale the aggregate congestion window size similarly to adopting a large MTU. However, maintaining multiple connections can incur high overhead on the server and significantly increase the complexity of connection management, as similarly observed in the argument of HTTP/2 vs. HTTP/1.1 [5, 42]. Table 1 compares the CPU consumption when using a session that employs six parallel connections with the legacy MTU to collectively download the same large file vs. a single-connection session for downloading the large file with a 9 KB MTU. We note that both cases achieve similar network throughputs. However, as the number of sessions increases, the parallel-connection approach requires significantly more CPU cycles. At 100 streams, parallel connections consume 2.88× more CPU cycles on the server.

Packet coalescing at end hosts. The receiver-side end host can employ packet merging schemes like NIC-level LRO or kernel-level general receive offload (GRO). These improve CPU efficiency and might offset the benefits of in-network MTU translation. Figure 1b shows that one can achieve a similar throughput improvement by enabling GRO and LRO without increasing the MTU. With G/LRO, the single flow throughput goes up to 50.1 Gbps with 1500 B MTU. In fact, applying both G/LRO is more performant than the 9 KB MTU without the options. This raises a question: is a large MTU really necessary for endpoints?

We still argue that employing a large MTU is beneficial for endpoints. First, the effectiveness of G/LRO degrades rapidly, even with a small number of concurrent flows. Figure 1c shows that the aggregate throughput drops by 31% with only 4 concurrent flows. This is because the interleaved packets from multiple flows reduce the opportunity for packet aggregation. Concurrent flows also impact performance with larger MTUs, but the throughput degradation is much lower (e.g., 7% at 4 concurrent flows). Second, G/LRO's benefit is limited to saving the CPU cycles for receive (RX) packet processing, while a large MTU reduces the CPU cycles for both endpoints.

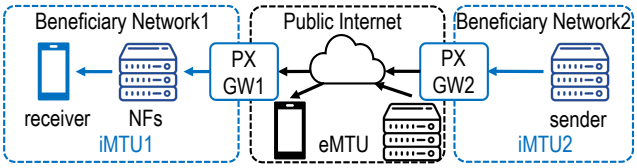


Figure 2: Overall architecture of PacketExpress

Also, a larger MTU would ramp up the sending rate faster as explained in Section 2.1. Figure 1d shows that 9 KB-MTU outperforms 1500 B-MTU with G/LRO by 5.4× in the WAN with 10 ms of end-to-end (E2E) delay and 0.01% of packet loss rate. Finally, LRO is not widely adopted by commodity on-board NICs [7, 20] of end hosts or mobile devices, and it is applicable only to TCP. While GRO may obviate the need for LRO, it consumes precious CPU cycles. Thus, we conclude that employing a large MTU remains beneficial for endpoints.

Concerns on congestion. A large MTU may exacerbate network congestion due to more aggressive sender behavior. While this is a valid concern, we make a few observations that alleviate it. First, the link bandwidth has scaled up by O(10,000) times from 10 Mbps when the legacy MTU was first suggested. Using an MTU a few times larger may have little impact on congestion. Also, modern datacenter and cellular core networks offer sufficient bandwidth that can accommodate bandwidth-hungry applications. Second, it is widely known that the network core is overprovisioned [9, 18, 33] and the access network bandwidth has been substantially improved. Moreover, the majority of flows in the WAN are short-lived, which implies that only a fraction of the flows require very high bandwidth. We admit this is only our estimate, and we leave a more thorough study on the possibility of congestion due to a large MTU to our future work.

3 CHALLENGES OF MTU UPGRADE

Challenges in UDP packet resizing. TCP packets can be dynamically merged or split thanks to TCP's byte-stream nature, but arbitrarily merging or splitting UDP packets hinders the applications from interpreting the payload. For example, if two QUIC datagrams are merged or a large datagram is split by the network, the receiver may fail to parse the packet, as QUIC encrypts both headers and payload and relies on strict datagram boundaries for interpretation. Unlike TCP, QUIC over UDP expects each datagram to be delivered intact and individually processed. This makes transparent packet merging or splitting infeasible for UDP-based protocols, where the application semantics and encryption schemes critically depend on preserving the original datagram boundaries.

High packet merging overhead. Packet segmentation is inherently scalable because each large packet can be split independently without flow tracking. In contrast, packet merging requires identifying flows and determining whether incoming packets are contiguous and mergeable, which inevitably introduces per-flow state. As the number of concurrent flows increases, searching for merge candidates across flows incurs significant CPU overhead, making it difficult to sustain high throughput at PXGW. Moreover, packets from small flows — typically unmergeable — consume CPU resources and interfere with the merging of large flows, thereby reducing overall efficiency. To build a scalable packet merging system, it is essential to leverage NIC hardware offloads to reduce CPU cycles and to adopt data structures that support fast lookup of adjacent packets under a large number of flows. Additionally, traffic classification techniques that separate merge-friendly large flows from small, sporadic flows will be necessary to mitigate interference and preserve throughput.

Limitation of classical PMTUD and PLPMTUD. When networks start to upgrade the MTU incrementally, finding the path MTU reliably becomes important as endpoints can leverage a large path MTU for efficient transmission on the path. However, the traditional PMTUD mechanism [13] is unreliable as it depends on ICMP messages to detect the smallest MTU. Unfortunately, many routers and middleboxes are configured to suppress ICMP messages, either due to misconfigurations or security concerns. This often results in "ICMP blackholes", where packets exceeding the path MTU are silently dropped, and no feedback is delivered to the sender. Moreover, it would require multiple rounds of time-consuming probing to figure out the path MTU even in a network that delivers ICMP messages. More recently, Packetization Layer PMTUD (PLPMTUD) [31] allows dynamic path MTU discovery using data segments as probes without dependency on ICMP. However, PLPMTUD relies on loss-based signals to infer MTU limitations, which introduces ambiguity in distinguishing between losses caused by congestion and those caused by MTU violations, making recovery and congestion control more complex. Similar to PMTUD, it follows a trial-and-error approach using data probes, multiple RTTs are needed to safely converge on a usable MTU. Due to these challenges, PLPMTUD has not seen wide deployment; only a few implementations, such as the Linux TCP stack, support it as an optional and non-default feature. We need a more efficient and robust PMTUD mechanism.

4 PACKETEXPRESS OVERVIEW

Figure 2 illustrates the overall architecture of a PX network that consists of two beneficiary networks (b-networks 1 and 2). Each b-network is configured with a large iMTU (an internal MTU that serves as the path MTU between any two nodes in the b-network), interconnected via the public Internet that operates with a smaller eMTU (an external MTU obtained from the next hop router). Note that this is just one plausible exmaple, and there can be many different scenarios

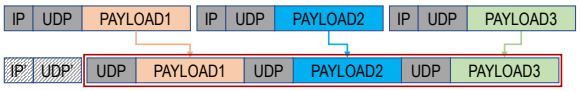


Figure 3: PX-caravan format

(e.g., only one endpoint belongs to a b-network). The PXGW between these domains dynamically performs translation between iMTU and eMTU. As long as each b-network is fully under operator control and can consistently enforce the use of iMTU within its domain, support for heterogeneous MTUs becomes practical through PXGW at the network edge. Note that it is recommended to deploy PXGW as close to a neighboring network as possible to allow more internal nodes to benefit from the larger MTU.

4.1 Protocol-conformant MTU translation

MTU translation for TCP packets. TCP is a byte-stream, order-preserving protocol, allowing in-network packet segmentation and reassembly on the fly. Thus, PXGW can perform MTU translation of TCP packets transparently without any modification of end hosts. Conceptually, this is analogous to applying TSO/GSO and LRO/GRO at end hosts. However, the MSS of a TCP connection is negotiated at handshake by the endpoints, so the sender can be constrained to transmit only small segments even if the internal path supports a larger MTU. To address this, PXGW needs to intervene during the MSS negotiation, effectively advertising a larger MSS on behalf of the downstream endpoint.

MTU translation for UDP packets. Since UDP packets do not allow dynamic segmentation and reassembly in general,[2] we introduce "PX-caravan" that tunnels multiple UDP packets of the same flow into one large packet. As shown in the Figure 3, the outer UDP/IP headers of a PX-caravan packet carry the entire length, while the inner UDP header of each packet carries its own length. PXGW merges the incoming UDP packets of the same flow or with the same destination, while it splits them when they leave the b-network. However, this requires the modification of the end hosts as they must deliver the individual packets from the PX-caravan packet to the application or tunnel multiple packets into a PX-caravan packet before forwarding in the b-network. The PXGW function designates the IP header's ToS field to indicate that the packet has been tunneled.

Scalable packet merging. PXGW exploits standard NIC offloads like LRO, TSO, and scatter-gather DMA for scalable packet merging and splitting. Basically, it leverages LRO to merge contiguous TCP packets in the same flow, and applies

[2] `UDP_SEGMENT` and `UDP_GRO` [24] are limited to in-host optimizations that bundle multiple UDP payloads into a single packet internally; they do not represent true datagram-level segmentation or reassembly.

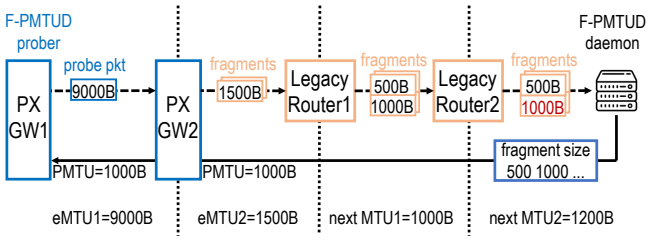


Figure 4: F-PMTUD with legacy routers

TSO to split the large packet content into multiple iMTU-bound packets before forwarding into the b-network. In addition, PXGW employs several optimization techniques: an enhanced algorithm for improved packet merging, a unified API for optimal utilization of multi-RX queues and RSS, delayed packet merging to maximize the number of iMTU-bound packets, merging after header-only DMA using NIC memory [40], and steering of small flows to prevent performance degradation using hairpin [37]. Due to the space constraint, we omit the details of these techniques.

4.2 Extending the Large-MTU Path

Incremental MTU upgrade offers a new opportunity to extend the benefits of a large MTU for selective end-to-end network paths or even path segments. We present two ideas to fully harness the advantages of a large MTU over eligible path segments.

Explicit iMTU advertisement. If a PX b-network directly neighbors other b-networks, it can extend the network path segment that employs a large MTU by explicitly exchanging the per-network iMTU information. Say a b-network employs the 9K iMTU and discovers that neighboring b-networks adopt the same or larger iMTU. Then, it can forward packets towards the neighboring domain without translating the MTU. Both networks can forward large-MTU TCP packets as is while they can use the PX-caravan packets without dynamic resizing. A key issue lies in how to disseminate the iMTU to neighboring networks. One can augment BGP announcements to carry the AS-level iMTU information, or one can come up with a new messaging protocol that runs on PXGW.

F-PMTUD. Another approach is to find the path MTU directly over an end-to-end path, leveraging a new PMTUD algorithm called F-PMTUD. The key idea behind F-PMTUD is to actively use IP packet fragmentation to accurately determine the PMTU. F-PMTUD consists of a PMTU prober and an F-PMTUD daemon running on the destination node. The prober sends a dummy UDP packet sized to the eMTU of the next hop (say eMTU1) to the destination node with a well-known port. Any PXGW along the path simply forwards the probe packet without merging it into a PX-caravan. If the PMTU is smaller than eMTU1, the probe packet is fragmented en route. The F-PMTUD daemon on the destination

node receives either the entire probe packet (if unfragmented) or a sequence of fragments. It then reports back the sizes of the packet or all received fragments to the prober, which determines the PMTU as either eMTU1 or the size of the largest fragment. This discovery process is fast as it takes only one RTT. Figure 4 shows an example scenario where a prober sends a 9 KB eMTU-sized probe packet to the destination daemon. As the packet traverses the network, routers with different MTU limits fragment it. Upon receiving all fragments, the daemon reports their sizes back to the prober, which determines the PMTU based on the largest fragment (1000 B in this case).

5 PRELIMINARY EVALUATION

We evaluate the feasibility of PX by answering the following questions. First, does the prototype PXGW achieve high throughput in dynamically converting packet size? Second, does PXGW improve the performance of end nodes? Finally, is it feasible to deploy F-PMTUD in the current Internet?

Setup. We use a machine with a Xeon Gold 6554S CPU and 4 ConnectX-7 400 GbE NICs [35] for PXGW. We use 4 client and 4 server machines, each equipped with an Xeon Gold 5512U CPU and a ConnectX-7 400 GbE NIC. We configure the MTU of 1500 B in the path between the servers and PXGW and 9 KB between the clients and PXGW. This setup enables both uplink and downlink flows, leading PXGW to split uplink packets and merge those for downlink. Using the testbed, we can evaluate the performance of PXGW by up to 1.6 Tbps of traffic. For the tests without MTU translation, we use 1500 B-MTU everywhere. We turn on TSO, LRO, GSO, and GRO on all endpoints. For the default test scenario, we use 800 iPerf bidirectional TCP flows to generate large TCP traffic from four servers to four clients and vice versa. This allows at most 2 Gbps per flow. To evaluate PX-caravan, we use 800 bidirectional iPerf UDP flows, with each flow generating at most 2 Gbps of traffic. PXGW is configured to merge consecutive UDP packets using the IP ID field to be compatible with `UDP_GRO`, so it produces PX-caravan packets in the format of Figure 3. We have modified the network stack of receiver end hosts to interpret the PX-caravan packets for UDP as `UDP_GRO` payload.

5.1 Performance of PXGW

We evaluate the performance of PXGW in terms of throughput and conversion yield. The conversion yield refers to the ratio of iMTU-sized packets after packet size conversion. We implement the TCP baseline with DPDK GRO library [16].

Throughput and conversion yield. Figure 5a presents the packet forwarding throughput and conversion yield of PXGW under 800 TCP flows. We evaluate two versions of PXGW: (1) "PX", which applies all techniques except header-only DMA, and (2) "PX + header-only". Header-only DMA is evaluated separately since it is experimental due to limited NIC store [40]. "PX" achieves 1.09 Tbps throughput and a 93% conversion yield using only 8 CPU cores, compared to the baseline's 167 Gbps and 76%. This shows that PXGW converts most packets into 9 KB segments while sustaining Tbps-scale throughput. With header-only DMA, the performance improves further to 1.45 Tbps and 94%, thanks to reduced memory bandwidth consumption. Figure 5b shows the results for 800 UDP flows. The peak throughput is slightly lower due to the absence of LRO and TSO benefits. Nevertheless, the conversion yield remains comparable to TCP, thanks to delayed merging. Enabling header-only DMA also improves the maximum throughput in UDP experiments.

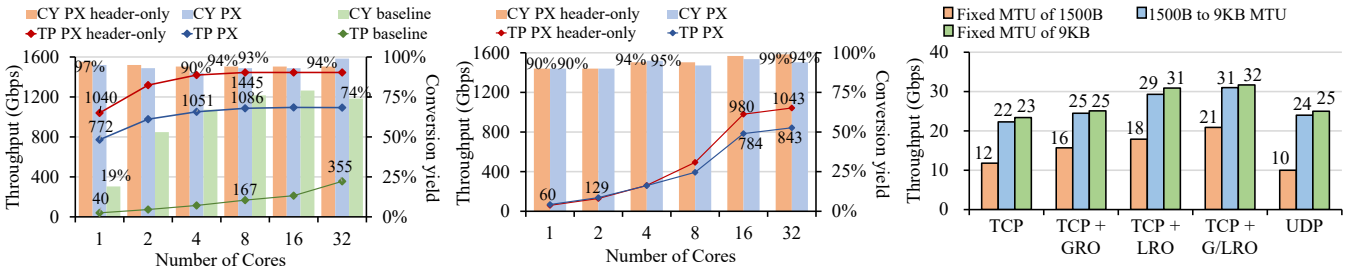
5.2 Benefits for Endpoints with PXGW

We evaluate the performance benefits for endpoints in a b-network. We use 9 KB of iMTU for the b-network and 1500 B eMTU for the external network.

Sender in a b-network. To simulate the WAN environment, we use a Linux software router and introduce 10 ms of delay and a 0.01% loss rate on the external network with `netem` [23]. We generate a single TCP flow with iPerf for a minute and measure the average TCP throughput. We observe that TCP throughput increases by 2.5× when PXGW employs the 9 KB iMTU even when the receiver is in a legacy network with the 1500 B eMTU. This is because the sender TCP stack increases the congestion window by 9 KB per RTT, 6× faster than with the legacy MTU of 1500 B. This confirms that incrementally upgrading only the sender network brings significant performance benefits.

Receiver in a b-network. To evaluate the impact on a receiver in a b-network, we run iPerf with 100 TCP flows and measure RX throughput with a single CPU core. Also, we incrementally enable LRO and GRO at the receiver to analyze the effects of these features. Figure 5c shows that the RX throughput improves by 1.5× ~ 1.8× when we use 9 KB-iMTU inside the b-network. Even when LRO and GRO are enabled, the performance gain from MTU translation is substantial as a larger MTU further improves the CPU efficiency. Obviously, the TCP receiver will benefit the most from PXGW in an environment where offload features such as LRO and GRO are unavailable, such as in mobile devices [3]. We also evaluate the performance of PX-caravan with the `UDP_GRO` option on. PX-caravan achieves a 2.4× better throughput than the baseline with the 1500 B MTU. Unlike the sender case, the receiver-side performance benefit is almost the same as adopting the 9 KB MTU in the end-to-end path.

[3] GRO might be available, but it consumes extra CPU cycles and power.



(a) TCP throughput (TP) / conversion yield (CY) **(b) UDP throughput (TP) / conversion yield (CY)** **(c) Throughput of an endpoint receiver**

Figure 5: Performance of PXGW and endpoints in a beneficiary network

5.3 Feasibility of F-PMTUD

Evaluating F-PMTUD across arbitrary network paths is challenging, so we test its feasibility on a small scale with CloudLab [36] while we validate fragmented packet delivery to popular sites from our campus. First, we measure the PMTU with 6 CloudLab nodes across the U.S., each probing all pairwise paths. F-PMTUD is compared against Scamper [30], a UDP-based PLPMTUD implementation. We confirm that both methods produce identical PMTU values on all paths, but F-PMTUD is significantly faster, as Scamper requires multiple RTTs to converge. For example, between the Utah and Massachusetts nodes, we observe that F-PMTUD is 368× faster than PLPMTUD.

We also test whether IP-fragmented packets that F-PMTUD depends on are successfully delivered over the WAN. From the top 1M domains on Cloudflare Radar [11], we obtain 389,428 live servers with unique IPs. We then send IP- fragmented HTTP requests and check the responses. We find that 99.98% of the servers respond, indicating widespread support for fragmented packets. The remaining 59 servers handle unfragmented HTTP requests, but they do not respond to fragmented requests with the identical content. 15 of them show that the last hop AS filters the fragments, but others do not respond to our probes. Although a strict apples-to-apples comparison is difficult due to differences in measurement dates and server sets, ICMP-based PMTUD was reported to succeed on only 51% of servers as of 2018 [12], with a continued decline. In contrast, F-PMTUD is expected to achieve near-universal success once deployed. A comprehensive evaluation on a consistent and up-to-date server set is left for future work.

6 CONCLUSION

The tremendous success of Ethernet has unintentionally imposed a limitation on the maximum packet size, which still impedes forwarding and processing efficiency in today's Internet. To address the problem, we have proposed PX, which presents an incremental path to MTU upgrade across the Internet without requiring coordination with all network entities. We have demonstrated that PXGW, even as a software-based platform, achieves a significant MTU translation throughput on the Tbps scale with 94% conversion to 9 KB packets. Our F-PMTUD is more robust and faster than existing PMTUD algorithms, and we have validated its feasibility with 99.98% of successful IP fragment delivery on the network paths.

Our proposal raises several open questions. What motivates ISPs to upgrade the MTU, and what are the costs involved? To what extent does a selective MTU upgrade lead to congestion? What is the best approach for exchanging iMTUs between neighboring networks? Where should we deploy the F-PMTUD daemon, and what are the incentives for doing so? Does a large MTU affect network congestion and how do we ensure fair bandwidth allocation in the mix of small and large-MTU senders? Despite these questions, we are convinced that now is the ideal time to upgrade the MTU on the Internet, and we urge the community to engage in this initiative.

ACKNOWLEDGMENTS

We are grateful to the ACM HotNets 2025 reviewers for their insightful comments and suggestions. We express our special gratitude to Bob Metcalfe, Geoff Thompson, Robert Garner, Dave Redell, and Alan Freier for their valuable discussions on the 1500-byte Ethernet frame size, which aided in identifying the authors of the initial Ethernet specification, the "DIX Bluebook" [41]. We extend our gratitude to Larry Peterson, Jim Kurose, and Matt Mathis for their expert insights on the MTU size and path MTU discovery. This work is in part supported by the ICT Research and Development Program of MSIP/IITP, Korea, under [RS-2024-00349594, Development of networked systems technologies leveraging SmartNIC], [2022-0-00531, Development of in-network computing techniques for efficient execution of AI applications], [RS-2024-00418784, Next-generation Cloud-native Cellular Network Center], and [RS-2025-02217106, Ethernet-based high-performance Smart Switch for Data Center]. Daehyeok Kim was supported in part by the U.S. National Science Foundation (NSF) Award 2403026. KyoungSoo Park is the corresponding author of this paper.

REFERENCES

[1] 2024. AXEL - Lightweight CLI download accelerator. https://github.com/axel-download-accelerator/axel. Last Accessed: 2025-07-03.

[2] M. Allman. 2003. TCP Congestion Control with Appropriate Byte Counting (ABC) (RFC-3465). https://tools.ietf.org/html/rfc3465. Last Accessed: 2025-07-03.

[3] M. Allman, V. Paxson, and W. Stevens. 1999. TCP Congestion Control (RFC-2581. https://tools.ietf.org/html/rfc2581. Last Accessed: 2025-07-06.

[4] AMD, Inc. 2024. AMD Pensando 2nd Generation (ELBA) DPU. https://www.amd.com/content/dam/amd/en/documents/pensando-technical-docs/product-briefs/pensando-elba-product-brief.pdf. Last Accessed: 2025-07-03.

[5] Mike Belshe, Roberto Peon, and Martin Thomson. 2015. Hypertext Transfer Protocol Version 2 (HTTP/2) (RFC-7540). https://tools.ietf.org/html/rfc7540. Last Accessed: 2025-07-03.

[6] Steve Blank. 2018. What the GlobalFoundries' Retreat Really Means. https://spectrum.ieee.org/nanoclast/semiconductors/devices/what-globalfoundries-retreat-really-means. Last Accessed: 2025-07-03.

[7] Broadcom, Inc. 2012. Broadcom BCM5720 Controller Technology. https://docs.broadcom.com/doc/5720-PB01-R. Last Accessed: 2025-07-03.

[8] Broadcom, Inc. 2024. Broadcom BCM57608. https://docs.broadcom.com/doc/BCM57608-PB. Last Accessed: 2025-07-03.

[9] Cisco. 2020. Best Practices in Core Network Capacity Planning White Paper. https://www.cisco.com/c/en/us/products/collateral/routers/wan-automation-engine/white_paper_c11-728551.html. Last Accessed: 2025-07-03.

[10] Cisco, Inc. 2022. Cisco Nexus 9800 Series. https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/nexus9800-series-switches-ds.html. Last Accessed: 2025-07-03.

[11] Cloudflare, Inc. 2024. Cloudflare Radar Domain Ranking. https://radar.cloudflare.com/domains. Last Accessed: 2025-07-03.

[12] Ana Custura, Gorry Fairhurst, and Iain Learmonth. 2018. Exploring Usable Path MTU in the Internet. In *Proceedings of the Network Traffic Measurement and Analysis Conference (TMA)*.

[13] Dr. Steve E. Deering and Jeffrey Mogul. 1990. Path MTU Discovery (RFC-1191). https://tools.ietf.org/html/rfc1191. Last Accessed: 2025-07-03.

[14] Dell, Inc. 2024. Dell PowerSwitch Z-series Spine, Core and Aggregation Switches. https://www.delltechnologies.com/asset/en-us/products/networking/technical-support/dell-powerswitch-z9664f-on-spec-sheet.pdf. Last Accessed: 2025-07-03.

[15] DigiBarn Computer Museum. [n. d.]. Alan O Freier on the Ď*Ḿachines, the Dolphin, Dorado, Dandelion and more. https://www.digibarn.com/friends/alanfreier/index.html. Last Accessed: 2025-07-03.

[16] DPDK. 2025. DPDK. https://doc.dpdk.org/guides/prog_guide/generic_receive_offload_lib.html. Last Accessed: 2025-07-03.

[17] Open Networking Foundation. 2022. UPF by Open Mobile Evolved Core (OMEC). https://github.com/omec-project/upf. Last Accessed: 2025-07-03.

[18] Fei Gui, Songtao Wang, Dan Li, Li Chen, Kaihui Gao, Congcong Min, and Yi Wang. 2024. RedTE: Mitigating Subsecond Traffic Bursts with Real-time and Distributed Traffic Engineering. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[19] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. 2015. *SoftNIC: A Software NIC to Augment Hardware*. Technical Report UCB/EECS-2015-155. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-155.html

[20] Intel, Inc. 2017. Intel Ethernet Connection I219. https://www.intel.com/content/www/us/en/content-details/333229/intel-ethernet-connection-i219-product-brief.html. Last Accessed: 2025-07-03.

[21] Intel, Inc. 2018. Intel® Intelligent Fabric Processors. https://www.intel.com/content/www/us/en/products/sku/218648/intel-tofino-2-12-8-tbps-20-stage-4-pipelines/specifications.html. Last Accessed: 2025-07-03.

[22] Intel, Inc. 2023. DPDK: Data Plane Development Kit. https://www.dpdk.org/. Last Accessed: 2025-07-03.

[23] Michael Kerrisk. 2011. tc-netem(8) — Linux manual page. https://www.man7.org/linux/man-pages/man8/tc-netem.8.html. Last Accessed: 2025-07-03.

[24] Michael Kerrisk. 2024. udp(7) — Linux manual page. https://www.man7.org/linux/man-pages/man7/udp.7.html. Last Accessed: 2025-07-03.

[25] Masahiko Kitamura, Daisuke Shirai, Kunitake Kaneko, Takahiro Murooka, Tomoko Sawabe, Tatsuya Fujii, and Atsushi Takahara. 2011. Beyond 4K: 8K 60p live video streaming to multiple sites. *Future Gener. Comput. Syst.* 27, 7 (July 2011), 952–959.

[26] K. Lahey. 2000. TCP Problems with Path MTU Discovery (RFC-2923). https://tools.ietf.org/html/rfc2923. Last Accessed: 2025-07-03.

[27] LAN/MAN Standards Committee of the IEEE Standard Society. 2022. *IEEE Standard for Ethernet, Amendment 3: Physical Layer Specifications and Management Parameters for 100 Gb/s, 200 Gb/s, and 400 Gb/s Operation over Optical Fiber Using 100 Gb/s Signaling (IEEE Std 802.3db-2022)*. IEEE.

[28] LAN/MAN Standards Committee of the IEEE Standard Society. 2022. *IEEE Standard for Ethernet (IEEE Std 802.3-2022)*. IEEE.

[29] Larry L. Peterson and Bruce S. Davie. 2021. *Computer Networks: A Systems Approach, 6th Edition*. Morgan Kaufmann.

[30] Matthew Luckie. 2010. Scamper: a scalable and extensible packet prober for active measurement of the internet. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC)*.

[31] Matt Mathis and John Heffner. 2007. Packetization Layer Path MTU Discovery (RFC-4821). https://tools.ietf.org/html/rfc4821. Last Accessed: 2025-07-03.

[32] Matt Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis J. Ott. 1997. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM SIGCOMM Computer Communication Review* 27, 3 (1997), 67–82.

[33] Michael Menth, Rüdiger Martin, and Joachim Charzinski. 2006. Capacity overprovisioning for networks with resilience requirements. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[34] NVIDIA, Inc. 2024. NVIDIA CloudXR Suite. https://www.nvidia.com/en-us/design-visualization/solutions/cloud-xr/. Last Accessed: 2025-07-03.

[35] NVIDIA, Inc. 2024. NVIDIA ConnectX-7. https://www.nvidia.com/content/dam/en-zz/Solutions/networking/infiniband-adapters/infiniband-connectx7-data-sheet.pdf. Last Accessed: 2025-07-03.

[36] The University of Utah. 2023. CloudLab. https://cloudlab.us. Last Accessed: 2025-07-03.

[37] Ori, Kam. 2019. DPDK Summit North America, Mountain View CA, November 12-13. https://www.dpdk.org/event/dpdk-summit-na-mountain-view/. Last Accessed: 2025-07-03.

[38] Sergio Orts-Escolano, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip L. Davidson, Sameh Khamis, Mingsong Dou, Vladimir Tankovich, Charles Loop, Qin Cai, Philip A. Chou, Sarah Mennicken, Julien Valentin, Vivek Pradeep, Shenlong Wang, Sing Bing Kang, Pushmeet Kohli, Yuliya Lutchyn, Cem Keskin, and Shahram Izadi. 2016. Holoportation: Virtual 3D Teleportation in Real-time. In *Proceedings of the Annual Symposium on User Interface Software and Technology (UIST)*.

[39] Jitendra Padhye, Victor, Firoiu Don, and Towsley Jim Kurose. 1998. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[40] Boris Pismenny, Liran Liss, Adam Morrison, and Dan Tsafrir. 2022. The benefits of general-purpose on-NIC memory. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[41] Rob Ryan (Intel, Inc.), Rich Seifert (DEC), and Dave Redell (Xerox, Inc.). 1980. *The Ethernet, A Local Area Network, Data Link Layer and Physical Layer Specifications (version 1.0)*.

[42] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. 2014. How Speedy is SPDY?. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*.

[43] Western Digital, Inc. 2016. CPU Bandwidth – The Worrisome 2020 Trend. https://blog.westerndigital.com/cpu-bandwidth-the-worrisome-2020-trend/. Last Accessed: 2025-07-03.

[44] Wenxiao Zhang, Feng Qian, Bo Han, and Pan Hui. 2021. DeepVista: 16K Panoramic Cinema on Your Mobile Device. In *Proceedings of the Web Conference*.

[45] Sihao Zhao, Hatem Abou-zeid, Ramy Atawia, Yoga Suhas Kuruba Manjunath, Akram Bin Sediq, and Xiao-Ping Zhang. 2021. Virtual Reality Gaming on the Cloud: A Reality Check. https://arxiv.org/abs/2109.10114. Last Accessed: 2025-07-03.