Is Large MTU Beneficial to Cellular Core Networks?

Youngmin Choi*†, Junghan Yoon*†, YoungGyoun Moon+ and KyoungSoo Park†
†KAIST +Samsung Research
South Korea

ABSTRACT

The Maximum Transmission Unit (MTU) refers to the largest packet size that can be transferred on a particular layer-3 network. As the dominance of Ethernet prevails, the "de-facto" standard MTU of 1500B has become universal in the wide-area networks. Unfortunately, the current MTU size overly limits the transmission performance especially when the underlying link speed rapidly increases while the CPU advancement stagnates.

In this work, we investigate the potential impact of large MTU on fast-growing cellular core networks. First, we analyze the performance trend over the different MTU sizes on endpoint receivers as well as on User Plane Function (UPF) in a cellular core network that handles all data packets. Second, we present our dynamic MTU translation technique to transparently apply a large MTU inside a cellular core network without requiring update on other networks in the Internet. We observe that that large MTU is beneficial to both traffic endpoints and UPF, and our evaluation shows that dynamic packet merging scales the UPF performance by up to 4.9x, reaching 628 Gbps with only eight CPU cores.

CCS CONCEPTS

• **Networks → Routers**; **Wired access networks**; **Mobile networks**; **Middleboxes / network appliances**; **Network performance analysis**.

KEYWORDS

Cellular core networks, 5G, UPF, MTU, GRO

ACM Reference Format:

Youngmin Choi*†, Junghan Yoon*†, YoungGyoun Moon+ and KyoungSoo Park†. 2023. Is Large MTU Beneficial to Cellular Core Networks?. In *7th Asia-Pacific Workshop on Networking (APNET 2023), June 29–30, 2023, Hong Kong, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3600061.3600081

1 INTRODUCTION

Recent computing hardware trend shows that the network bandwidth continues to grow dramatically while the advancement of CPU has largely slowed down [7, 13]. Unfortunately, the widening gap between bandwidth and computation capacity poses a serious challenge to increasing the bandwidth of a cellular core network. As modern cellular core networks employ commodity-off-the-shelf (COTS) hardware, achieving high bandwidth would often depend on the throughput of a COTS server component like User Plane Function (UPF), which routes and forwards all data packets between the Internet * and the core networks.

One simple idea that would easily scale the network transmit (TX) /receive (RX) efficiency is adopting a large maximum transmission unit (MTU). A large MTU size would not only increase the payload-to-header ratio, but it would also improve the performance of direct memory access (DMA) between a network interface card (NIC) and host CPU. Especially, 5G UPF operates by looking up multiple rule tables for each packet, a larger MTU size would significantly reduce the packet processing load. For example, a jumbo frame (9000B-MTU) would reduce the processing load by six times over typical 1500B frames.

Unfortunately, due to the dominance of Ethernet as the layer-2 protocol, the 1500B-MTU has been fixated as the "de-facto" standard value in the Internet for many decades. However, there is no fundamental reason why we must stick to this ancient value, which is a byproduct of old 10 Mbps Ethernet for collision detection (CD) in a shared medium that supports carrier sense multiple access (CSMA) [14]. In contrast, virtually all end hosts are connected to *switched* Ethernet that largely obviates CSMA/CD these days. Nevertheless, modern networks are still tied up with this obscure value from an old technology.

There are a few reasons why network administrators are reluctant to upgrade the MTU size of their network. First, there might be legacy Ethernet devices in their network that cannot support an MTU size larger than 1500B. An IP packet is subject to fragmentation when it passes through a network of an MTU size smaller than its packet size, which would substantially lower the forwarding performance. However, this is unlikely to be a valid argument any more as modern Ethernet devices typically support a much larger MTU size than 1500B. Also, mechanisms like path MTU discovery [18] and the TCP maximum segment size (MSS) option at connection setup [19] will avoid IP packet fragmentation by picking the minimum path MTU. Second, a more plausible reason might be due to the perception that there is no real benefit in using a large MTU size. Traffic senders can reduce the DMA overhead by using a popular NIC offload feature like TCP segmentation offload (TSO). Also, traffic receivers may reduce the DMA overhead by turning on

*They contributed equally to this work.

APNET 2023, June 29–30, 2023, Hong Kong, China

ACM ISBN 979-8-4007-0782-7/23/06…$15.00
https://doi.org/10.1145/3600061.3600081

*Or Public Data Network (PDN) in 3GPP jargon.

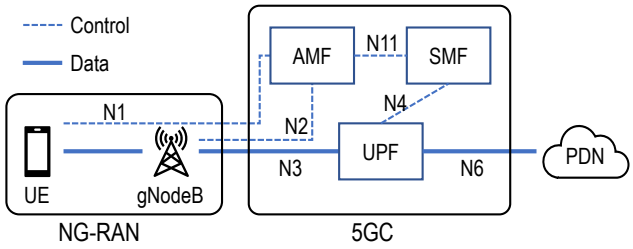


Figure 1: The architecture of cellular network

large receive offload (LRO) * and/or generic receive offload (GRO) *. In addition, today's in-network Ethernet switches are capable of transferring packets at tens of Tbps [11, 12, 20] even with the 1500B-MTU, there is little incentive to adopt a larger MTU size. However, our experiments show that the effectiveness of LRO/GRO rapidly drops even with only a few concurrent flows, so larger MTU can still provide practical benefit. Plus, an in-network function like cellular UPF would highly benefit from reducing the number of packets for transferring the same amount of data.

In this work, we analyze the benefit of large MTU in a cellular core network. Our experiments show that larger MTU is generally beneficial to traffic RX endpoints, and even when there are multiple flows, the performance degradation with larger MTU is much lower than that with 1500B-MTU. In addition, we find that the performance of UPF linearly grows to the MTU size. Then, we present how to transparently apply a large MTU without requiring update on the other networks in the Internet. The key idea is to exploit common NIC offload features (i.e., LRO/GRO) as well as scatter-gather DMA in the external UPF. Unfortunately, LRO/GRO alone becomes ineffective if multiple arriving flows are intermixed in the NIC RX queues. To overcome the problem, we employ the following two techniques. First, we increase the number of NIC RX queues to benefit from receive-side scaling (RSS) and minimize the intermixing of many flows into a particular RX queues. Second, we improve the efficiency of software GRO by implementing hashtable-based packet merging.

We implement our prototype UPF called dUPF, based on an open-sourced UPF implementation [15]. Our evaluation shows that dynamically increasing the incoming packet size improves the UPF processing throughput by 3.1x to 4.9x. We find that dUPF effectively increases the packet size – the average packet size is over 13KB and 67+% of the RX packets are larger than 4.5KB.

2 MOTIVATION

We present our motivation for applying a large MTU to a cellular core network.

2.1 Why Cellular Core Network?

There are a few reasons why we consider a cellular core network as the beneficiary of large MTU. First, a cellular core network is essentially a private network whose administrative policy is fully determined by a single operator. This means that the administrators can adopt and enforce whatever MTU size in their internal network without consulting with other networks if it benefits their network and/or subscribers. Second, there is a growing need to scale the bandwidth capacity of a cellular core network. Bandwidth-heavy mobile applications like high-quality video streaming, virtual reality (VR), and augmented reality (AR) applications drive up the capacity demand of a modern cellular network, and the network must be operated in a cost-effective manner. This is especially important as modern cellular core network runs on x86 COTS servers. Higher per-server performance would significantly reduce the operational expenditure (OpEx). We provide a brief background on the cellular core network as well as the bandwidth requirements of high-quality video applications below.

Application	BW requirement
60Hz 4K video streaming	25Mbps
60Hz 8K video streaming	100Mbps
120Hz 4K, 8K VR/AR (Cloud Rendering)	0.1 ~ 1Gbps
120Hz 4K, 8K VR/AR (VR headset)	0.1 ~ 10Gbps

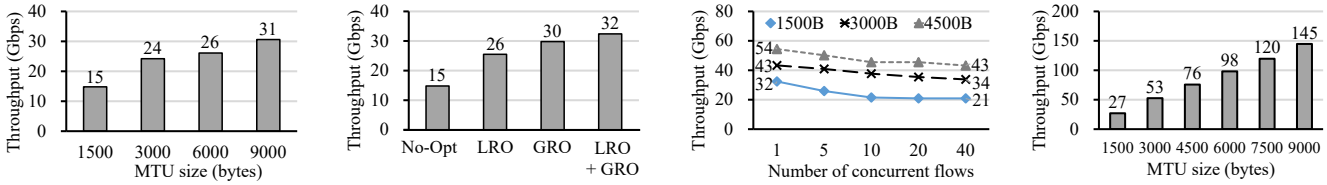
Table 1: Bandwidth requirements of video applications [1, 34]

Cellular core network architecture. Figure 1 shows a simplified architecture of the 5G cellular network. It consists of a radio access network (RAN) and a core network. RAN allows base stations (called gNodeBs in 3GPP jargon) to communicate wirelessly with user equipments (UEs) and a core network forwards the IP packets between a RAN and a public data network (PDN) (i.e., the Internet). The functionality of a core network is largely divided into control and user planes where the former handles all control tasks such as UE mobility management, authentication/authorization, packet routing policy, etc., while the latter performs IP data packet switching. The key component of the data plane is User Plane Function (UPF), which routes all IP packets from the PDN to UEs by GTP tunneling and forwards the GTP packets from UEs to the PDN after de-capsulating the GTP headers. Since UPF handles all IP data packets, the UPF performance often determines the bandwidth capacity of a core network. For each incoming packet, the UPF runs a series of match-and-actions with the rules set by the Session Management Function (SMF) through the N4 interface. These rules dictate how to identify packets (packet detection rules (PDRs)), how to forward them (forwarding action rules (FARs)), how to process them (buffering action rules (BARs)), how to mark them (QoS enforcement rules (QERs)), how to compute and report the usage of them (usage reporting rule (URRs))), and so on.

Bandwidth requirements of mobile applications. Mobile applications that require high-quality video streaming will demand much larger bandwidth in the near future. Table 1 refers to [1, 34], which shows the bandwidth requirements of some bandwidth-heavy applications in the mobile network. As shown in this table, VR/AR contents require up to 1 Gbps for cloud (or edge) rendering, and they even require up to 10 Gbps when using a VR headset that often has insufficient computing power for decoding the compressed data in real time. To support these applications, upcoming 6G core network must deliver a much larger per-user throughput than 5G

*LRO is a NIC hardware feature that coalesces contiguous TCP packets that arrive back to back into a single packet with the merged payload.

*GRO does the same task as LRO, but it runs as software in the kernel (or in the RX logic in DPDK [23]) for the RX'ed packets. Also, GRO can even merge the contiguous packets that did not arrive back to back.



(a) Impact of MTU size (single flow) **(b) Impact of GRO/LRO (single flow)** **(c) Impact of concurrent flows – GRO/LRO enabled w/ different MTUs** **(d) Impact of MTU size on UPF performance**

Figure 2: Effectiveness of large MTU and GRO/LRO. (a)-(c) show the performance at a receiver, and (d) shows UPF performance.

core network. Not only VR/AR streaming but also other candidate 6G applications like immersive XR, high-fidelity holograms, and digital replica are likely to demand 10x to 50x peak data rate than 5G [6, 32]. In this context, we expect our optimization scheme for heavy-bandwidth applications, called dynamic MTU translation, to be essential for the next generation core network.

2.2 Is Large MTU Beneficial?

Benefit of large MTU to endpoints. Figure 2a compares the RX throughputs over varying MTU sizes with a single TCP flow on a client. [*] We use iPerf [26] for benchmarking to minimize the application-level overhead. Both server and client use a single CPU core to avoid any other host network stack overheads resulting from multiple CPU cores [10]. We observe that the throughput improves by 2.1x when increasing the MTU from 1500 to 9000B – this confirms that larger MTU improves DMA efficiency and reduces TCP/IP packet processing overheads. However, we also find that one may achieve a similar throughput by enabling GRO and LRO without even increasing the MTU. Figure 2b shows that the single flow throughput is improved up to 32.4 Gbps by turning on LRO and GRO with 1500B-MTU. However, we find that the effectiveness of GRO/LRO rapidly degrades even with a small number of concurrent flows. As shown in Figure 2c, the aggregate throughput drops by 34% if we have 10 concurrent flows. As the number of flows grows, more packets from different flows get interleaved, which loses the chance of packet aggregation. While we do see a similar trend with larger MTUs, the throughput degradation with larger MTU is much lower (i.e., 13-16% at 10 concurrent flows) than that with 1500B-MTU. We conclude that large MTU is still beneficial to endpoints even with concurrent flows.

Support for larger MTU on UE? In general, for a wireless channel, the data transmission size directly affects the Bit Error Rate (BER), so it should be set carefully. However, in a cellular access network (e.g., NG-RAN in Figure 1), the link layer (L2) dynamically adjusts the Transport Block Size (TBS) [3] that the channel can transfer at a time as the capacity of channel (L1) is not fixed. There is a sub-layer named Radio Link Control (RLC) [4] inside the link layer to support segmentation and concatenation of data unit from the upper layer. Thus, the MTU size set by the UE kernel does not affect the BER, and can be set to a larger number than 1500B. Technically, the only limit for the MTU size is the buffer size of the underlying sub-layers which are implemented in hardware. The 3GPP spec for 5G standard [5] defines the maximum supported size of Service Data Unit (SDU) as 9000B for the Packet Data Convergence Protocol (PDCP) sub-layer which is an upper layer above the RLC sub-layer. Since there is one more upper layer named Service Data Adaptation Protocol (SDAP) on top of the PDCP sub-layer, which requires its own 1B header [2], UEs can use jumbo frame with size of 8999B, giving them the same benefits as wired endpoints.

Benefit of large MTU to UPF. We analyze the impact of larger MTU with an open-sourced UPF implementation of the Open Mobile Evolved Core (OMEC) project driven by Open Networking Foundation (ONF) [15]. It employs BESS [17], a modular software switch that runs on DPDK[23], that serves as the software-based datapath for UPF. The rule setup for UPF is described in section 4. Again, we use iPerf to generate TCP traffic and employ a single CPU core for UPF. We set up both server (in the PDN side) and client (in the UE side) to employ enough CPU cores to make sure that neither of them becomes a performance bottleneck. Figure 2d demonstrates that the UPF performance almost linearly scales to the size of MTU. At 9000B-MTU, the performance becomes 5.4x as large as that of 1500B-MTU. We generate 100 concurrent flows and ensure that each flow achieves from 270 Mbps to 1.5 Gbps depending on the MTU size, which meets the bandwidth requirement of our target applications in Section 2.1. Larger MTU is more beneficial to UPF performance as UPF typically processes the packets with only their header information. [*]

Takeaways. Large MTU is beneficial to the performance of both endpoints and UPF. The benefit of large MTU with an endpoint becomes relatively more significant with concurrent flows while the benefit to UPF is even larger as the packet processing throughput linearly scales to the MTU size due to header-based processing. Then, the key question is how to increase the MTU without coordinating with the PDN, which is the focus of the next section.

3 TRANSPARENT SCALING OF THE MTU SIZE

We have demonstrated that a large MTU size is beneficial to both UE and UPF. In this section, we present how to realize a large MTU size for UPF and UEs without updating the MTU size in other networks.

[*] For experiment setup details, please refer to Section 4.

[*] 3GPP 5G standard dictates that the UPF must handle deep packet inspection (DPI) rules if any, but one can configure a hierarchy of UPFs so that the first line runs header-based processing while the subsequent layers process DPI-related rules.

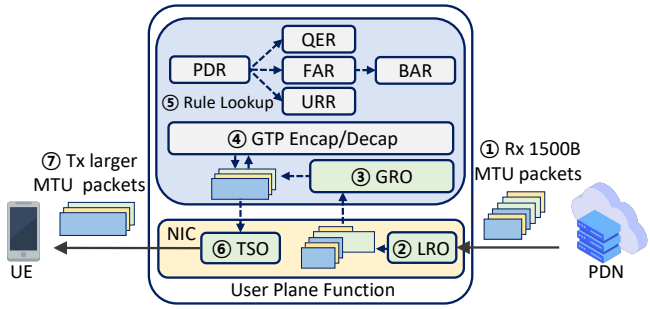


Figure 3: Overall Architecture of dUPF

3.1 Dynamic MTU Translation

Our key idea is to dynamically translate the MTU size at an PDU Session Anchor UPF (A-UPF). Figure 3 shows the overall operation. Since the A-UPF neighbors with both the PDN and the core network, we can configure it to use the 1500B-MTU for the PDN side while we set up a larger MTU size for internal use in the core network. For downstream traffic, the A-UPF reads and merges continuous TCP packets in the same flow into a single large packet, executes the UPF tasks with the merged packet, segments it to smaller packets to fit internal MTU, GTP-encapsulates [*] and forwards them over to the core network to reach the target UE. For upstream traffic, the A-UPF GTP-decapsulates each incoming packet, and segments it into 1500B-MTU packets (using TSO if NIC supports it), and forward them onto the PDN.

The A-UPF leverages LRO and GRO to merge the contiguous TCP packets coming from the PDN. We find that GRO is more effective than LRO in merging the packets when the number of concurrent flows increases. This is unsurprising as LRO requires that the contiguous packets arrive sequentially into an RX queue on a NIC. However, one downside with GRO is that it consumes CPU cycles, which could become significant if it deals with a large number of packets with many flows. We improve the behavior of the GRO implementation of DPDK below, which effectively reduces the CPU overhead.

One issue lies in determining the optimal MTU size. A larger MTU is generally more desirable as it reduces the per-packet overhead, but if it is too large, it could waste UPF LRO buffers and may affect the error detection capability of CRC-32 in Ethernet frames [29]. We currently limit it to 9000B-MTU as this size is popular in closed networks, but we plan to investigate on the optimal value in the future.

3.2 Deployment Issues and Optimizations

TCP MSS option and Path MTU discovery. To enable large MTU in the core network, A-UPF needs to deceive the UE into believing that the servers in the PDN also employ the same MTU size as in its own network. For this purpose, A-UPF must coordinate with the end-to-end MTU discovery process. First, it needs to correct the maximum segment size (MSS) in the TCP MSS option [19] exchanged at TCP connection setup. Specifically, it needs to update the MSS value sent from the the PDN side so that UEs receive a larger MSS value at TCP connection setup. Likewise, path MTU discovery [18] packets (with the "Don't fragmentation" bit on) from an UE must be split into 1500B-MTU packets before being forwarded to the PDN [*]. This would not generate ICMP packets with "Fragmentation Needed".

Hashtable-based GRO. DPDK provides a function that implements GRO of RX'ed packets. Unfortunately, the current implementation is inefficient on two fronts. First, it employs linear search to find the matching flow that an incoming packet belongs to. So, if there are k concurrent flows that are candidates for packet merging and n RX'ed packets, then it would run in O(kn) time. Depending on the size of k, the overhead could become significant. Second, it implements only one-sided merging, which loses the extra opportunity for merging for packets that arrive out of order. Say, packets arrive in the order of C, A, B, packet B is merged with only packet C in the current implementation, losing the opportunity to merge with packet A after the first merge.

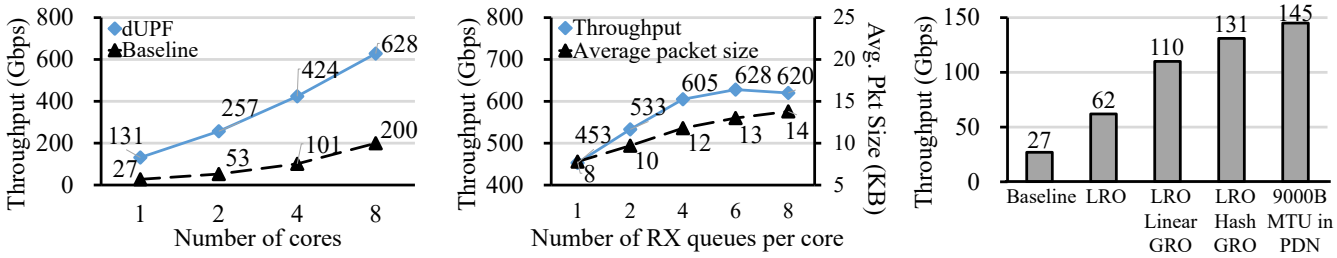
We first reduce the overhead of linear search by implementing GRO with a hash table. For each RX packet, our implementation looks up the flow in the hash table with the four tuple of the RX packet. Once the matching flow is found, it tries merging the packet into the flow. If the matching flow does not exist, it creates a flow from the packet and inserts it into the hash table. Since a hash table lookup takes O(1) time, our implementation runs in O(n) time for n RX packets. To avoid hash computation on CPU, we have the NIC tag the receive-side scaling (RSS) hash value (or Toeplitz hash value) of the packet along with it. Second, we fix the packet merging logic of DPDK GRO to consider merging packets on both sides, i.e., left and right sides of the incoming packet.

Increasing the number of RX queues to better exploit LRO/GRO. RSS on NIC ensures that the packets in the same flow are steered to the same RX queue. This allows all packets in the same flow to be processed by the same CPU core that handles the RX queue, which enables scalable processing without lock contention. Since RSS is a NIC hardware feature, if we increase the number of RX queues per CPU core, one can reduce the number of flows that are mapped to a particular RX queue without any CPU consumption. Fewer flows in a RX queue would increase the chance of LRO while they improve the efficiency of the GRO logic as well. While a larger number of RX queues increase the packet RX overhead as a CPU core needs to read from multiple RX queues, we find that more efficient LRO/GRO often outweighs the RX overhead.

Implementation. We implement our prototype, dUPF, on the UPF-EPC project [15] that employs BESS [17] as the data plane. Since BESS is based on DPDK, we implement our logic as a part of the DPDK RX code. We enable LRO to merge the packets from the N6 interface up to 9000B, and adjust the MSS option field after parsing a TCP SYN packet. We insert PDRs and sub-rules to global rule tables implemented as lock-free cuckoo hash tables. For the implementation of hashtable-based GRO, we use packet buffer chaining, and avoid any redundant memory copy for GRO. It requires 189 lines of C code for MTU translation, 463 lines for hashtable-based GRO, and 281 lines for BESS script for supporting multiple NICs and multiple RX queues.

[*]NICs often support TSO with GTP tunneling [25].

[*]This might generate partial ACKs in case of packet loss in the PDN, but it does not violate the TCP protocol.



(a) Throughput comparison over a different number of CPU cores

(b) Impact of the number of RX queues on dUPF performance

(c) Effectiveness of LRO + HashGRO

Figure 4: Performance evaluation with dUPF

4 PRELIMINARY EVALUATION

Setup. We employ a machine with a 16-core Intel Xeon Gold 6346 CPU with 36 MBs of LLC running at 3.10 GHz and four dual-port NVIDIA ConnectX-6 100GbE NIC [31] (capable of handling up to 800 Gbps traffic) for UPF. We employ eight machines for clients and servers where each machine is equipped with one ConnectX-6 NIC capable of handling 200 Gbps traffic. For UPF rule configuration, we install 4 million PDRs, 500K QERs, and 500K FARs. For QoS rules, we set 2 Gbps of per-flow maximum bit rate (MBR), and 100 Mbps of per-flow guaranteed bit rate (GBR). We use iPerf to generate large TCP traffic from servers to clients. We compare the performance of dUPF and a baseline UPF that uses 1500B-MTU without paket merging. For dUPF, we set up 1500B-MTU in the path between the servers and UPF and 9000B-MTU between the clients and UPF. For the baseline UPF, we use 1500B-MTU everywhere. We leverage the NIC HW offload feature [24] that enables GTP tunneling at dUPF and confirm that GTP tunneling with NIC HW offload does not degrade the performance. For client-side GTP tunneling support, we have tested with a gNodeB-like middlebox that only en/decapsulates GTP header and forwards packets, which does not reduce the throughput. However, we end up disabling GTP tunneling for our experiments as it would require employing many more middlebox machines than we have.

Overall throughput. Figure 4a compares the performance of dUPF and a baseline UPF over a different number of CPU cores. We employ 100 concurrent flows per each CPU core employed. Overall, the performance of both systems scale well with the number of CPU cores while dUPF outperforms the baseline by 3.1x to 4.9x and its throughput reaches 628 Gbps. The average packet size for dUPF grows up to 8.7x larger than that of the baseline thanks to dynamic packet merging. This significantly reduces the total number of packets to be processed by dUPF and improves the performance as most of UPF processing operates on packet header level. However, the dUPF performance gets saturated at 8 CPU cores, and it does not further improve as we employ more CPU cores. This seems to be due to memory bandwidth bottleneck. We observe that the memory bandwidth of the UPF machine measured by MLC [22] is 150GB/s while the average memory bandwidth utilization with dUPF is 108GB/s at 8 CPU cores, which gets close to the maximum measured memory bandwidth.

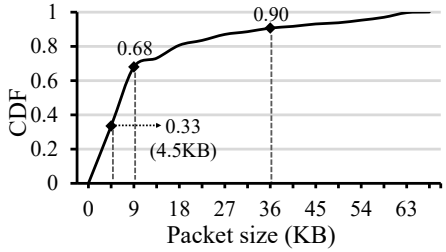


Figure 5: Packet size distribution at dUPF

Performance over the number of RX queues per CPU core. Figure 4b shows the dUPF throughput over the number of RX queues employed by each CPU core. For this experiment, we use eight CPU cores that handle the traffic from eight 100 Gbps NIC ports. We observe that the throughput scales well as the effectiveness of both LRO and GRO improves with a fewer number of concurrent flows per queue. However, the performance plateaus at 6 RX queues per core, and the average packet size increases by 66% over using 1 RX queue per core. While the average packet size grows further with 8 RX queues per core, the performance does not improve more as CPU cache utilization degrades.

Effectiveness of LRO and GRO vs. static large MTU. We investigate the impact of each feature on the performance of dUPF with a single CPU core. As shown in figure 4c, merging packets with only LRO improves the UPF performance by 2.3x over a baseline that disables LRO/GRO. Using both LRO and GRO with linear search to merge RX packets improves the performance by 4.1x. So, software GRO is substantially more effective than LRO alone. Our hashtable-based GRO outperforms LRO+linear-search-GRO by 19%. Employing static 9000B-MTU for both N3 and N6 interfaces would perform the best, but this would require upgrading the MTU on all networks. The performance with our technique is only 10% worse than that with the static MTU.

Packet size distribution. Figure 5 shows packet size distribution with 8 cores and 6 RX queues per each core at dUPF. We observe that our techniques effectively enlarge the RX packet size – 66.5 % of packets are larger than 4500B with the average packet size of 13 KB. We also measure packet size distribution at the client side. As shown in figure 6, dUPF successfully increases the packet size that gets forwarded to the client side – 75 % of packets are larger than 4500B when the MTU is set up to 9 KB.

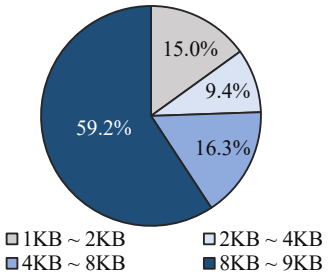


Figure 6: Packet size distribution at client

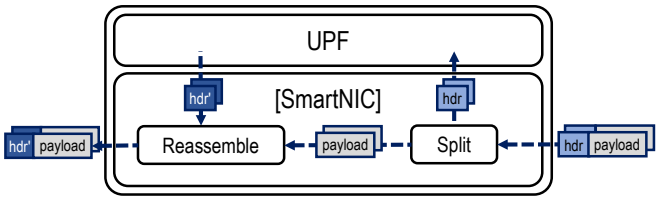


Figure 7: Split-process-reassemble scheme

5 FUTURE DIRECTION

We plan to alleviate the memory bandwidth bottleneck and to perform automatic parameter tuning in the future.

Split-process-reassemble. Figure 7 shows our **split-process-reassemble** strategy. The key idea is to split the header and the payload of each RX packet at SmartNIC and to move only the packet header to main memory for UPF processing. The packet header must carry the pointer to its payload at SmartNIC memory so that it can be scatter-gather DMA'ed with its payload when it is forwarded. After the packet header is finished with UPF processing, it is tagged with the resulting action and delivered to the SmartNIC so that SmartNIC carries out the action before forwarding. This idea is similar to PayloadPark [16], but we apply it in a different context with SmartNIC. We expect three benefits with this strategy. First, it can reduce the DMA overhead substantially as only packet headers are delivered to main memory. Second, it will improve CPU cache utilization as a collection of packet headers would consume a much smaller memory footprint. Third, its effectiveness can be maximized with our dynamic MTU translation technique presented in this paper.

Automatic parameter tuning. We observe that our UPF performance is sensitive to the choice of parameters such as the number of RX queues per CPU core, RX batch size, LRO buffer size, and so on. These values affect the CPU cache utilization due to DDIO of modern CPU, which has impact on the overall performance. We plan to develop a tool that automatically finds the optimal values for specific NIC hardware and CPU types.

6 RELATED WORK

Effectiveness of Large MTU. Murray et al. [29] investigate the pros and cons of employing a jumbo frame (9000B-MTU) in the public Internet. Similarly to our work, they report that large MTU improves the TCP transmission throughput with lower CPU usage, but our work presents how to dynamically support large MTU in a celullar core network without upgrading other networks. Salyers et al. [33] present a layer-2 mechanism for efficient forwarding in the Internet core network. In their scheme, an ingress router of the Internet core network merges incoming frames destined to the same egress router into a large frame. When the large frame reaches the egress router, the egress router splits it into original small frames. Their simulation result reports a better forwarding throughput, but it does not carefully reflect the increased computational overhead at the routers, which may become a bottleneck. There was a proposal in the 5G Advanced standardization context [27] to concatenate multiple PDCP SDUs to alleviate the L2 processing load in NR. Our work would be also beneficial for reducing the NR data processing load at base stations.

Cellular core network acceleration. There have been several works that improving the data plane performance of a cellular core network with programmable NICs or switches. Bose et. al. [8] exploit the dynamic device personalization (DDP) [21] feature in Intel X710 NIC for flow steering, and offload both data plane and control plane functions of UPF to Agilio on-path SmartNIC [30]. They report that SmartNIC-based offloading presents a great potential for performance improvement but that it also limits the flexibility of the operations by the operator such as dynamic scaling, handling skewed traffic across users, etc. MacDavid et al. [28] present a P4-based UPF. In their prototype, the data plane of UPF is implemented with P4 [9] switches, and the control plane is implemented with microservices which translate PFCP messages* to P4 runtime commands. We believe UPF task offloading to programmable hardware is promising in the future, but the current prototypes seem to be hampered by limited memory size and processing capacity. A hybrid architecture that harnesses both CPU and programmable hardware would be a reasonable alternative.

7 CONCLUSION

In this paper, we argue that the de-facto MTU size of 1500B in the current Internet overly limits the packet processing performance at middleboxes and clients. We have investigated a cellular core network as a special case and we have shown that a larger MTU size is generally beneficial to the performance of both UEs and UPFs in 5G core network. To enable large MTU transparently in a cellular core network without requiring an upgrade elsewhere, we have presented the dynamic MTU translation technique that leverages standard NIC offload features such as GRO, LRO, TSO and RSS as well as manipulating SYN MSS options and Path MTU discovery packets. We have also shown that our hashtable-based GRO implementation and enlarging the number of RX queues per core significantly improve the UPF performance. Our preliminary evaluation demonstrates that our solution, dUPF, improves the packet processing throughput by 3.1x to 4.9x over a baseline UPF and that 67% of the packets arriving at the client side have a size larger than 4500B.

*PFCP (Packet Forwarding Control Protocol) messages consist of the rules required by lookup on UPF, and they are sent by SMF via N4 interface.

ACKNOWLEDGMENTS

We appreciate the insightful feedback and suggestions from APNet 2023 reviewers. This work is in part supported by the Samsung Research program, under [high performance 6G acceleration techniques] and by the ICT Research and Development Program of MSIP/IITP, Korea, under [2018-0-00693, Development of an ultra low-latency user-level transfer protocol].

REFERENCES

[1] 3GPP. 2016. TS 22.261 Service Requirements for the 5G System (5GS). https://www.3gpp.org/DynaReport/22261.htm. Last Accessed: 2023-03-17.
[2] 3GPP. 2017. TS 37.324 Evolved Universal Terrestrial Radio Access (E-UTRA) and NR; Service Data Adaptation Protocol (SDAP) specification. https://www.3gpp.org/DynaReport/37324.htm. Last Accessed: 2023-03-17.
[3] 3GPP. 2017. TS 38.214 NR; Physical layer procedures for data. https://www.3gpp.org/DynaReport/38214.htm. Last Accessed: 2023-03-17.
[4] 3GPP. 2017. TS 38.322 NR; Radio Link Control (RLC) protocol specification. https://www.3gpp.org/DynaReport/38322.htm. Last Accessed: 2023-03-17.
[5] 3GPP. 2017. TS 38.323 NR; Packet Data Convergence Protocol (PDCP) specification. https://www.3gpp.org/DynaReport/38323.htm. Last Accessed: 2023-03-17.
[6] Next G Alliance. 2022. 6G Applications and Use Cases. https://www.nextgalliance.org/wp-content/uploads/dlm_uploads/2022/05/Next_G_Alliance_6G_Applications_and_Use_Cases_7-1.pdf. Last Accessed: 2023-03-17.
[7] Steve Blank. 2018. What the GlobalFoundries' Retreat Really Means. https://spectrum.ieee.org/nanoclast/semiconductors/devices/what-globalfoundries-retreat-really-means. Last Accessed: 2023-03-17.
[8] Abhik Bose, Diptyaroop Maji, Prateek Agarwal, Nilesh Unhale, Rinku Shah, and Mythili Vutukuru. 2021. Leveraging Programmable Dataplanes for a High Performance 5G User Plane Function. In *Proceedings of the Asia-Pacific Workshop on Networking (APNet)*. https://doi.org/10.1145/3469393.3469400
[9] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Computer Communication Review* 44, 3 (jul 2014), 87–95. https://doi.org/10.1145/2656877.2656890
[10] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. 2021. Understanding Host Network Stack Overheads. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. https://doi.org/10.1145/3452296.3472888
[11] Cisco. 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: 2023-03-17.
[12] Dell. 2022. 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: 2023-03-17.
[13] Western Digital. 2016. CPU Bandwidth – The Worrisome 2020 Trend. https://blog.westerndigital.com/cpu-bandwidth-the-worrisome-2020-trend/. Last Accessed: 2023-03-17.
[14] IEEE Standards for Local Area Networks. 1985. Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. *ANSI/IEEE Std 802.3-1985* (1985), 13–130. https://doi.org/10.1109/IEEESTD.1985.82837
[15] Open Networking Foundation. 2022. UPF by Open Mobile Evolved Core (OMEC). https://github.com/omec-project/upf. Last Accessed: 2023-03-17.
[16] Swati Goswami, Nodir Kodirov, Craig Mustard, Ivan Beschastnikh, and Margo Seltzer. 2020. Parking Packet Payload with P4. In *Proceedings of the International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*.
[17] 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
[18] IETF. 1990. RFC 1191 - Path MTU Discovery. https://tools.ietf.org/html/rfc1191. Last Accessed: 2023-03-17.
[19] IETF. 2012. RFC 6691 - TCP Options and Maximum Segment Size (MSS). https://tools.ietf.org/html/rfc6691. Last Accessed: 2023-03-17.
[20] Intel. 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: 2023-03-17.
[21] Intel. 2019. Intel® Dynamic Device Personalization (DDP) Brief. https://www.intel.com/content/www/us/en/architecture-and-technology/ethernet/dynamic-device-personalization-brief.html. Last Accessed: 2023-03-17.
[22] Intel. 2021. Intel® Memory Latency Checker v3.10. https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html. Last Accessed: 2023-03-17.
[23] Intel. 2023. DPDK: Data Plane Development Kit. https://www.dpdk.org/. Last Accessed: 2023-03-17.
[24] Intel. 2023. DPDK Generic Flow API Documentation. https://doc.dpdk.org/guides/prog_guide/rte_flow.html. Last Accessed: 2023-03-17.
[25] Intel. 2023. DPDK NVIDIA MLX5 Ethernet Driver Documentation. https://doc.dpdk.org/guides/nics/mlx5.html. Last Accessed: 2023-03-17.
[26] Dugan Jon, Estabrook John, Ferbuson Jim, Gallatin Andrew, Gates Mark, Gibbs Kevin, Hemminger Stephen, Jones Nathan, Qin Feng, Renker Gerrit, Tirumala Ajay, and Warshavsky Alex. 2021. iPerf - The ultimate speed test tool for TCP, UDP and SCTP. https://iperf.fr/. Last Accessed: 2023-03-17.
[27] Donggun Kim, Sangkyu Baek, Jaehyuk Jang, and Daegyun Kim. 2021. High-Speed Packetization for 5G Advanced. In *2021 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 1–6.
[28] Robert MacDavid, Carmelo Cascone, Pingping Lin, Badhrinath Padmanabhan, Ajay ThakuR, Larry Peterson, Jennifer Rexford, and Oguz Sunay. 2021. A P4-Based 5G User Plane Function. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*. https://doi.org/10.1145/3482898.3483358
[29] David Murray, Terry Koziniec, Kevin Lee, and Michael Dixon. 2012. Large MTUs and Internet Performance. In *Proceedings of the IEEE International Conference on High Performance Switching and Routing*. https://doi.org/10.1109/HPSR.2012.6260832
[30] Netronome. 2020. Agilio LX SmartNICs. https://www.netronome.com/media/documents/PB_Agilio_LX_2x40GbE-7-20.pdf. Last Accessed: 2023-03-17.
[31] NVIDIA. 2022. NVIDIA Connectx-6 Dx NIC. https://nvdam.widen.net/s/qpszhmhpzt/networking-overal-dpu-datasheet-connectx-6-dx-smartnic-1991450. Last Accessed: 2023-03-17.
[32] Samsung Research. 2020. The Next Hyper-Connected Experience for All. https://cdn.codeground.org/nsr/downloads/researchareas/20201201_6G_Vision_web.pdf. Last Accessed: 2023-03-17.
[33] David Salyers, Yingxin Jiang, Aaron Striegel, and Christian Poellabauer. 2007. JumboGen: Dynamic Jumbo Frame Generation for Network Performance Scalability. *SIGCOMM Computer Communication Review* 37, 5 (oct 2007), 53–64. https://doi.org/10.1145/1290168.1290174
[34] VdoCipher. 2021. What is Video Bandwidth? 720p, 1080p, GB Transfer Explained. https://www.vdocipher.com/blog/video-bandwidth-explanation/. Last Accessed: 2023-03-17.