X-Plane: A High-Throughput Large-Capacity 5G UPF

Yunzhuo Liu[1,2*], Hao Nie[2,3*], Hui Cai[2], Bo Jiang[1†], Pengyu Zhang[2], Yirui Liu[2], Yidong Yao[2], Xionglie Wei[2], Biao Lyu[2,4], Chenren Xu[3], Shunmin Zhu[2,5], Xinbing Wang[1]

[1]Shanghai Jiao Tong University, [2]Alibaba Group, [3]Peking University, [4]Zhejiang University, [5]Tsinghua University

ABSTRACT

Cloud providers, such as AWS and Azure, have started providing 5G services on their cloud infrastructure. In this paper, we present the design and implementation of X-Plane, a system that uses commercial programmable ASICs and DRAM servers on today's cloud infrastructure to implement high-performance 5G User Plane Function (UPF). Building X-Plane is hard because we need to address the following challenges: consistency issues when concurrently accessing UPF state data, slow UE table lookup due to repetitive and numerous Packet Detection Rule (PDR) matching, and the need to handle out-of-order packets from disconnected UEs. X-Plane addresses these challenges by designing three novel technologies: concurrent state data access protocol, fast flow table and paging buffer for handling out-of-order packets. We demonstrate its feasibility and practicality with our implementation on a Tofino-based programmable ASIC. Our evaluation shows that X-Plane can support over ~490Gbps throughput per ASIC pipeline, over 10 million UEs, and finish packet processing within predictable ~4 us on average.

CCS CONCEPTS

• **Networks** → **Mobile networks**; **Programmable networks**.

KEYWORDS

5G UPF, RDMA, Programmable ASIC

∗Yunzhuo Liu and Hao Nie are equal contributors to this work and designated as co-first authors. †Bo Jiang is the corresponding author.

ACM MobiCom '23, October 2–6, 2023, Madrid, Spain

ACM ISBN 978-1-4503-9990-6/23/10…$15.00
https://doi.org/10.1145/3570361.3613267

ACM Reference Format:
Yunzhuo Liu[1,2*], Hao Nie[2,3*], Hui Cai[2], Bo Jiang[1†], Pengyu Zhang[2],, Yirui Liu[2], Yidong Yao[2], Xionglie Wei[2], Biao Lyu[2,4], Chenren Xu[3],, Shunmin Zhu[2,5], Xinbing Wang[1] . 2023. X-Plane: A High-Throughput Large-Capacity 5G UPF. In *The 29th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '23), October 2–6, 2023, Madrid, Spain.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3570361.3613267

1 INTRODUCTION

Cloud providers, such as AWS and Azure, have started running 5G network on their cloud infrastructure [1]. In this paper, we look at how to build 5G User Plane Function (UPF) on top of existing cloud infrastructure. The 5G UPF is a key module of 5G network, which is responsible for handling user plane traffic, such as traffic engineering, billing, and quality of service (QoS).

One way to run 5G UPF in the cloud is using software-based UPF, which is provided by vendors such as ZTE [9, 31] and Ericsson [7]. Such systems can be easily deployed on today's cloud infrastructure. However, this approach is expensive, as we need a large number of servers to support the amount of traffic in typical 5G scenarios. In addition, the OS scheduling introduces uncertainty in packet processing time, which can easily result in a high latency that does not meet the requirement of 5G application.

Considering the disadvantages of software-based UPF, a body of recent work [14, 18–20, 33] attempt to build 5G UPF using programmable ASICs. Such programmable ASICs can achieve very high speed in packet processing and are widely used in today's cloud infrastructure [23]. While this approach has demonstrated great potential, previous solutions still suffer the following limitations.

First, most of the existing systems based on programmable ASICs [14, 18, 19, 33] lack the support of key UPF functions, including traffic accounting, metering and data buffering. The reason behind is that the logic of these UPF functions is very complicated, and programmable ASICs are designed primarily for simple packet processing operations such as forwarding and dropping, rather than supporting complicated service-level logic.

Second, none of the existing systems can achieve high throughput, large capacity and low latency at the same time, despite the fact that they focus only on basic packet forwarding functionalities. For example, Kundel [19] only supports ~100Gbps traffic. HybridUPF [18] has a 350us latency (very high for UPF) on its slow path. Zhou et al. [33] can only support one thousand flows, several orders of magnitude smaller than the need of operators.

X-Plane targets at supporting key UPF functions, including counting, metering, PDR parsing, and UE paging, while achieving high throughput, large capacity and low latency at the same time. Achieving this goal is extremely hard because of the following reasons.

- **Limited capacity:** Programmable ASICs only have O(10M) on-device memory. Our empirical measurement with a Tofino-based programmable ASIC shows that the on-device memory can store only about O(1M) flow entries, much smaller than the need of 5G networks [31]. One way to address this problem is using the programmable ASIC plus external DRAM architecture, where the forwarding rules can be stored on the external DRAM. TEA [16] is an example of this architecture. X-Plane adopts the same architecture and focuses on supporting UPF functionalities on top of it.
- **High throughput VS counting/metering:** Since programmable ASICs have limited on-device memory and cannot store counters and meters of each UE, the only way to support counting and metering is using the programmable ASIC plus external DRAM architecture like TEA [16]. However, it is challenging to update the state of counters and meters at the external DRAM without tampering throughput. For example, the solution in [25] manages to support state updates only at the extremely high cost of over 87.5% degradation in throughput.
- **Low latency VS PDR parsing:** To identify the action to apply to an ingress packet, the UPF has to search through the UE table. However, this operation is sluggish because, even when the corresponding UE entry is located, the UPF still needs to scan the PDRs within the entry to identify the appropriate actions for the flow. This process may incur large latency as the number of PDR rules in a UE table entry can be substantial.
- **Handling out-of-order packets during UE paging:** Out-of-order packets occur when a UE disconnects and reconnects to the network. This happens because the UPF must buffer packets transmitted to the UE while it was disconnected from the network. When the UE reconnects, the buffered packets and newly arrived packets may become mixed up, leading to out-of-order problems.

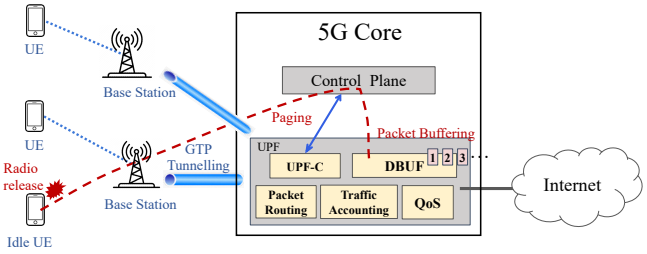


Figure 1: UPF is the user plane in a 5G core that routes the bi-directional traffic between a UE and the Internet.

X-Plane addresses above challenges by introducing the following novel technologies. First, we design a protocol that facilitates simultaneous access and update of stateful data in an external DRAM. Our design avoids adding locks on the external DRAM, which is critical for achieving high throughput. Second, X-Plane speeds up the sluggish UE table lookup by creating a fast table. The basic idea is to utilize the footprints of the first packet in each flow to determine the actions on subsequent packets in the same flow. This significantly reduces table lookup latency by eliminating repetitive and slow UE table lookup. Third, X-Plane designs a mechanism that buffers packets when a UE disconnects from networks and automatically sends the buffered packets to the UE when it re-connects to networks. More importantly, the order of packets is preserved meaning that X-Plane eliminates out-of-order packets that are present in many previous systems.

We implement and evaluate X-Plane on a server cluster built based on Tofino and CX6. Our empirical evaluation shows that X-Plane can achieve the following performance.

- **Performance:** X-Plane can support ~490 Gbps throughput per ASIC pipeline, over O(10M) flows, and finish packet processing within predictable ~4 us on average. As far as we know, X-Plane is the first system that supports high throughput, large capacity and low latency UPF on programmable ASIC.
- **UPF functionality:** X-Plane supports most of the key UPF functions, such as counting, metering, QoS, etc. In addition, when a UE disconnects from the network, X-Plane buffers data intended for this UE and pushes the data to the UE later when the UE reconnects to the network. X-Plane also ensures the order of packets even when the idle UE disconnects from the network.

In summary, we make the following two contributions. First, we prove it feasible to build a UPF with large capacity on top of memory resource limited programmable ASICs. Second, we show that it is possible to support UPF functionalities and states, such as counting and metering on a stateless

programmable ASIC. We open source X-Plane [30]. This work conforms to the IRB policies (if any) of our institution.

2 BACKGROUND

2.1 5G Core User Plane Function (UPF)

Figure 1 shows a 5G core User Plane Function (UPF), which is responsible for routing bi-directional IP traffic between the base stations and the Internet. The routing rules are determined by the 5G core control plane, which sends the rules to UPF-C via Packet Forwarding Control Protocol (PFCP) messages. UPF-C parses PFCP messages and configures the rules on UPF. In addition to IP packet routing, the UPF also performs several key functions, such as traffic usage accounting, QoS control and data buffer for the UEs in idle mode. These functions are essential for ensuring that the UPF traffic routing meets the requirements of the operators. We explain each of these key functions below.

- **Traffic accounting**: The UPF keeps track of data bytes transmitted and received by each user device. Operators use such data usage information for charging their customers, represented as User Ends (UEs).
- **QoS handling**: The UPF uses various mechanisms to enforce QoS policies, such as rate limiting and traffic shaping. These mechanisms ensure that UE data packets are delivered with the required level of performance while not causing network congestion and overload. The enforcement of QoS policies can be executed at the UE level, UE session level, and UE flow level.
- **Data buffer for disconnected UE**: The UPF needs to buffer the data from the Internet to a UE when the UE is disconnected from the network due to sleep or handover. Once the UE reconnects, the UPF sends the buffered data to the UE and resumes normal IP packet forwarding.

The above functions are defined in PDRs. As UEs attach, move, and detach, the 5G core control plane installs, changes, and removes PDRs by sending Packet Forwarding Control Protocol (PFCP) messages to UPF-C. UPF-C parses the PFCP messages and stores PDRs to a lookup table named UE table. To identify the action to apply to an ingress packet, the UPF searches through the UE table to identify the matched PDR.

2.2 Existing Programmable ASIC based UPF

The implementation of existing programmable ASIC based UPF systems[18–20, 33] rely heavily on the on-ship resources. They implement counting and metering using internal counters and meters or registers, which consume the on-chip Static Random-Access Memory (SRAM). For the matching of PDRs, existing systems store the PDRs in Ternary Content-Addressable Memory (TCAM), a type of resource scarcer than SRAM. TCAM enables fast match of PDRs by searching the entries in parallel. Relying on on-ship resources simplifies the design of UPF, but the SRAM and TCAM can only support a limited number of flows that does not satisfy the need of 5G UPF [31]. As a result, existing systems redirect a flow to CPU server for processing when the on-chip memory is not enough to store the corresponding PDRs or counters. Although some systems [18] propose optimized flow offloading policies, e.g., offloading high-bandwidth flows to be processed on ASIC, a large ratio of traffic, e.g., >50% [18], can still go to the CPU path and hinder performance.

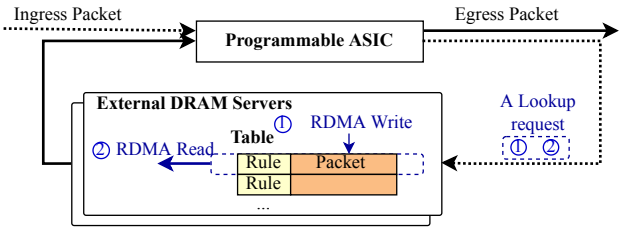


Figure 2: TEA uses external DRAM to extend programmable ASIC memory and stores rules with five-tuple keys

2.3 Programmable ASIC + External DRAM

Extending the memory of programmable ASIC with external DRAM has been explored by TEA [16]. TEA uses the external DRAM to store five-tuple rules and demonstrates how to perform a lookup of the external table. As shown in Figure 2, to achieve high lookup throughput and low lookup latency, TEA uses RDMA to access the rules in the DRAM server. Each lookup consists of two consecutive RDMA requests, i.e., an RDMA Write and an RDMA Read. An entry in the external table consists of the rule and a space reserved for temporally storing the packet. During the lookup, as the programmable ASIC lacks the ability to hold the packet, the packet is firstly stored to the reserved space by the first RDMA Write, then the followed RDMA Read fetches the stored packet back to the programmable ASIC together with the rule.

Such architecture enjoys the advantage that the memory can be easily enlarged according to needs by increasing the amount of DRAM on external server. Meanwhile, the packet processing is kept on the programmable ASIC, avoiding redirecting packets to, e.g., CPU, for processing when the programmable ASIC cannot store all rules. Thus it fully utilizes the merits of hardware-based packet processing. However, building a 5G UPF based on such architecture is not trivial. The design of TEA only shows how to perform table lookup of simple rules with five-tuple keys, and does not answer how the external DRAM can be further utilized to support functions like stateful data access, efficient PDR lookup and

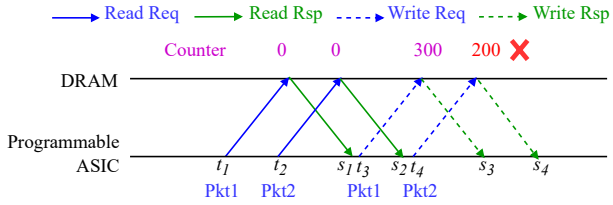


Figure 3: State data in-consistency issues when experiencing concurrent data access (read/write)

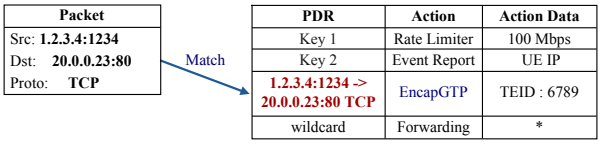


Figure 4: Search a matched PDR in the UE table

in-order buffering required by 5G UPF. Next, we will explain the challenges in detail.

3 CHALLENGES

3.1 Concurrent Access to Stateful Data

A 5G UPF needs to access (read/write) several types of data that mark the states of a UPF, such as counters, meters, etc. When reading and writing stateful data on external memory, multiple concurrent read and write can cause data inconsistency. The concurrent data accesses are caused by the delay in retrieving and writing back the state data due to the separation of the programmable ASIC and external DRAM. To illustrate how the concurrent accesses happen and cause inconsistency, we use counters in Figure 3 as an example.

As shown in Figure 3, *Pkt*1 with a size of 300 bytes arrives at t_1, and triggers the read of the counter value (initially 0). The value is updated to 300 and then written back to the external memory at t_3. During the delay between the counter is read and written back, *Pkt*2 arrives and triggers another read, causing concurrent access to the counter. As a result, the counter read triggered by *Pkt*2 obtains the old counter value 0 instead of the updated value 300, leading to inconsistency problem.

Essentially, the problem is due to that the operations on the state data in external DRAM are not atomic. Conventional solutions to similar problems rely on the use of locks to ensure atomicity. However, programmable ASICs lack the capability to store subsequent packets, e.g., *Pkt*2 in Figure 3, in local SRAM and cannot suspend packet processing while waiting for the lock release. Therefore, supporting concurrent access state data in external DRAM is an unaddressed problem and it is vital to the implementation of 5G UPF.

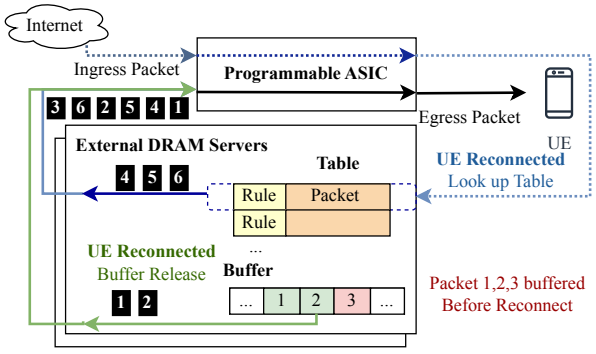


Figure 5: Out-of-order packets when a UE disconnects from and re-connects to networks

3.2 Slow Table Lookup

The second challenge we need to address is the slow UE table lookup in a UPF. To determine the appropriate action (forward/drop/usage reporting/QoS enforcement) to be taken on an ingress packet, the UPF needs to identify the PDR within the UE table. However, this can be a time-consuming process, as demonstrated in Figure 4. Consider an ingress packet with the 5-tuple (1.2.3.4, 21.22.23.24, 1234, 8080, TCP), which matches the third entry of the UE table. To determine the correct operation (forward/drop) for this packet, the UPF should further find the matched PDR. However, matching PDR can be a slow and bandwidth consuming process due to the following reason.

Each UE table entry can have a list of PDRs instead of just one. The UPF must check each PDR rule within this entry, following priority from high to low. Consequently, many rounds of table lookup to external DRAM are incurred in order to match a PDR. In our experiments, we show that one lookup incurs a latency about 3us. Although the latency of one lookup is relatively low, too many rounds of lookup can still lead to high latency. Also, recall the process in Figure 2, the packet needs to be stored into and fetched back from the external DRAM in each lookup, many rounds of lookup can cause high bandwidth consumption.

3.3 Out-of-Order Packets

The third challenge we need to address is handling out-of-order packets. The ability to maintain the order of packets is an essential characteristic of UPFs, ensuring that the order of ingress packets is the same as that of egress packets. When implementing UPF on programmable ASICs, out-of-order packets can occur in the UE paging scenario where a UE disconnects and reconnects to networks.

When a UE goes into idle mode and becomes disconnected from the network, the UPF must store the downlink packets intended for that UE in its buffer, as illustrated by (pkt_1, pkt_2, pkt_3) in Figure 5. Using the programmable ASIC +

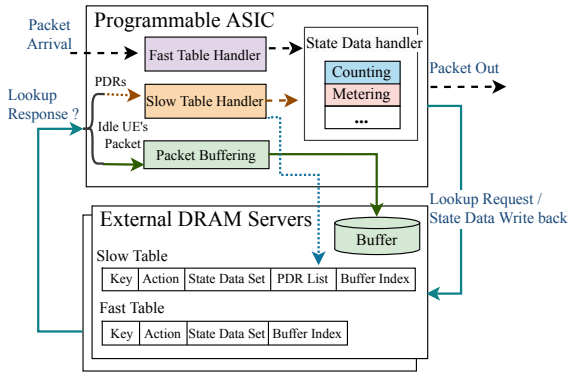


Figure 6: X-Plane system overview

external DRAM architecture, this buffer can only be implemented on the external DRAM. When the UE reconnects to the network, the UPF may simultaneously forward two types of packets to the UE: buffered packets (pkt_1, pkt_2, pkt_3) that were stored during the disconnection, and newly arrived packets (pkt_4, pkt_5, pkt_6) that are forwarded based on the lookup table in Figure 5. This interleaving of buffered packets with newly arrived packets at the ingress port of the programmable ASIC causes packet disordering.

One simple solution to this problem is connecting the output of the lookup table with the input of the buffer. This eliminates the out-of-order packets at the ingress port of the programmable ASICs. However, we face the following two issues when directly adopting this solution.

First, we need to find a consistent and coherent solution to update the pointers that mark the head and tail of the buffer. The update of the two pointers faces similar consistency issues described in section 3.1. Second, we need to design a mechanism that automatically releases all packets in the buffer. The pipeline in the programmable ASIC is driven by the arrival of ingress packets. When an ordinary data packet arrives, it passes through the pipeline and triggers operations designed for releasing buffered packets. However, when the UE re-connects to the network, the UPF might not receive new arrival packets. Therefore, even though knowing the UE's re-connection, due to the lack of packets at the ingress port, the programmable ASIC itself does not have a way to trigger the release of buffered data.

4 SYSTEM OVERVIEW

Figure 6 shows an overview of X-Plane which has the following three modules.

- **Concurrent state access:** X-Plane supports concurrent and consistent access (read and write) of UPF state data on the remote memory pool. This is done by combining a local state table on the programmable ASIC and a collapsed write back protocol. This module ensures the correctness of state data on the remote memory, compared to the case of inconsistent state data if directly adopting TEA [16].
- **Fast table:** X-Plane identifies the source of slow table lookup in a UPF system and addresses this problem by designing a fast table. Its basic idea is to remember the action applied to this flow learned during the PDR matching in the slow UE table. By doing this, X-Plane avoids repeated PDR matching for the same flow, significantly improves performance and reduces processing latency, particularly when we need to perform PDR matching for a large number of PDRs.
- **Paging buffer:** This module addresses the out-of-order packets in a UPF. It does so by building a FIFO buffer across three devices: programmable ASIC, RDMA QP and paging buffer on DRAM. This cross-device FIFO buffer guarantees the order of packets. It leverages the concurrent state access mechanism designed in section 5.1 to guarantee the in-order Read/Write operation on packets. X-Plane also designs a mechanism to generate internal signals to automatically deplete all the buffered data, which is not possible in previous systems like TEA.

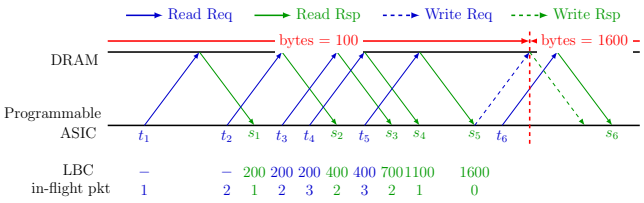


Figure 7: Local state table and collapsed writeback.

5 DESIGN

5.1 Concurrent State Data Access

To enforce state consistency during concurrent access, we maintain a *Local State Table (LST)* in the on-device memory of the programmable ASIC and perform collapsed writeback. The state of a flow is updated locally during the processing of packets that can potentially cause state inconsistency, and written back to the external memory only when all such packets have been processed.

More precisely, we divide a flow into busy periods, where a busy period is a consecutive period during which the flow has at least one in-flight packet, i.e., a packet whose read response has not yet come back to the programmable ASIC. When a new busy period starts, an entry is created for the corresponding flow in the LST, and the flow state is loaded from the external memory. The LST uses a local counter to keep track of the number of in-flight packets in the busy period. All subsequent state updates in the same busy period are performed on the LST. When the busy period ends, the flow state is then written back to the external memory.

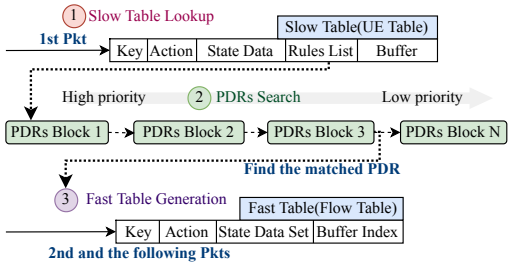


Figure 8: Fast flow table generation.

We illustrate this process with an example in Figure 7, where the state is taken to be the cumulative flow byte count. Suppose six consecutive packets of a flow arrive at time $t_1, \ldots, t_6$, with byte sizes 100, 200, 300, 400, 500, 600, respectively. Suppose there is no in-flight packet at time t_1. Then the first five packets form one busy period, while the sixth packet belongs to the next busy period. At time t_1, X-Plane creates an entry for the corresponding flow in the LST, and initializes the *in-flight-packet counter* to zero. The in-flight-packet counter is incremented by one every time X-Plane sends a read request, and decremented by one whenever a read response comes back. When the first read response comes back at time s_1, X-Plane initializes the *local byte counter (LBC)* in the LST to the byte count in the read response, which is 100 bytes in this example, and then increments it by the size of the first packet (100 bytes), so the LBC value is 200. When the second read response comes back at time s_2, X-Plane discards the stale byte count in the response, and increments the LBC by the size of the second packet (200 bytes), so the LBC value is now 400. The LBC is incremented in a similar way upon the next three read response arrivals. Now at time s_5, the busy period ends, and X-Plane sends a write request to the external memory, where the cumulative byte count of the flow is updated to the current LBC value, which is 1600 in this case. When the write response comes back, the LST entry for this flow will be removed if there is no in-flight packet; otherwise, it will be retained to serve the next busy period, which is the case in Figure 7. Since RDMA preserves packet order, the read request for the sixth packet is guaranteed to return the correct byte count at time s_6, which is consistent with the LBC value at the end of the previous busy period. Note that on the programmable ASIC, the accesses of different packets to local data are atomic. Therefore, the atomicity of the operations on LST is guaranteed.

Now we give a rough estimate of the storage needed for the LST. Note that at any time t, a flow has an entry in the LST if and only if it has at least one packet arrival within a window of length 2×RTT before t (see Figure 7). Thus the number of LST entries is upper bounded by the number of packet arrivals in this window. Suppose the throughput 1.6 Tbps, the RTT is 5 us, and the average packet size is 690 bytes [9, 29]. Then the number of LST entries is upper bounded by $1.6\text{Tbps} \times 5\text{us}/690\text{B} \approx 1450$. Each entry requires 13 bytes of memory for the 5-tuple flow identifier, 4 bytes for the LBC and 4 bytes for the in-flight-packet counter, so the amount of memory for the entire LST is $1450 \times (13\text{B} + 4\text{B} + 4\text{B}) \approx 31\text{KB}$. As detailed in Section 6, we will implement the LST by a hash table. To minimize collision, we will allocate much more than 31 KB of memory, but it is still within the capacity of the on-device memory, which is typically tens of megabytes.

5.2 Generating Fast Flow Table

Fast flow table speeds up the lookup by recording the footprints created by the first packet of each flow. Figure 8 illustrates how the table is generated.

① Upon the arrival of the first packet of a flow at the UPF, X-Plane initiates a search in the UE table to locate a matching entry. For uplink packets, the match keys include TEID, UE source IP and QoS flow identifier. For downlink packets, the match key is the UE destination IP. Once a match is found, it examines all the PDRs in the entry to identify the matched PDR.

② After matching the PDR in the UE table, X-Plane records the corresponding action as footprints, which are then used to create an exact-match entry in the flow table. The entry comprises a 5-tuple match key (source IP, destination IP, source port, destination port and protocol number), the actions to be applied to the packet, flow state data, and buffer information.

③ From the second packet of a flow onwards, the ingress packets do not undergo the UE table and PDR matching process. Instead, the UPF matches the 5-tuple of the ingress packet to an entry in the flow table to identify the corresponding action.

In practice, when a UE initiates a session, the 5G core control plane sends PFCP messages to UPF to configure the PDRs associated with the session. The UPF-C of X-Plane runs on the DRAM server, it receives and parses PFCP messages, and then generates and inserts PDRs into the slow table through RDMA. Upon receiving a PFCP session modification/deletion message, the UPF-C will modify/delete the rules in the slow table and delete the corresponding stale rules in the fast table. As most of the PDRs remain static during the lifetime of a UE session, such operations have relatively low frequency and will not cause high CPU consumption.

PDR Block: Due to the limited Packet Header Vector (PHV) resource of a programmable ASIC, such as Tofino, which only has about 160 bytes of PHV for header parsing, loading all PDRs of a UE table entry for matching is not feasible. To tackle this issue, X-Plane divides all the PDRs in a UE table entry into several PDR blocks, as shown in the green block of Figure 8. Each block contains a small number of PDRs that

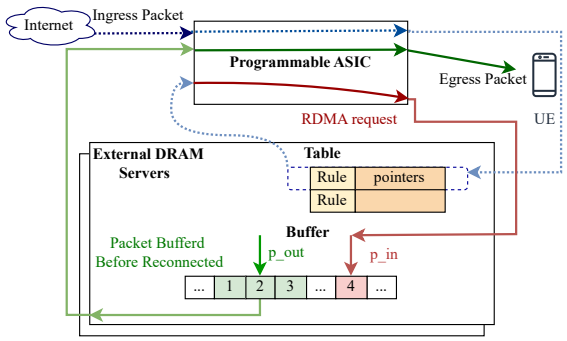


Figure 9: In-order packet buffer.

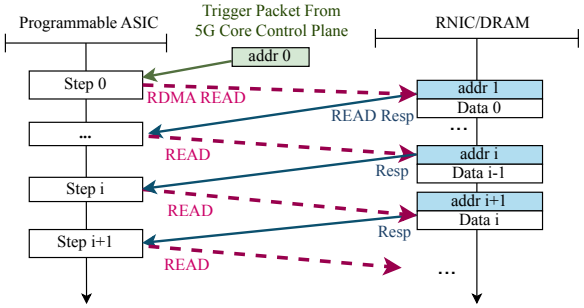


Figure 10: Buffer packets read loop.

can fit into the PHV. In our implementation, each PDR has a size of 28 bytes, and each PDR block contains five PDRs, which can fit into the 160 bytes space of the PHV. Note that the PDR size is configurable according to needs and it affects the maximum number of PDRs that can be fetched in a block. If a matched PDR is not found in the previous block, X-Plane loads the following block and continue this process until one matched PDR is found.

One thing to note is that X-Plane also stores the fast table in external DRAM and generates the fast table rules from within the data plane, rather than relying on the UPF-C to generate and insert fast table rules to programmable ASIC via the ASIC control API. X-Plane chooses this approach because the frequency of creating or updating flows is very high. We conduct an empirical measurement and find that we observe more than 1M new flows per second on a 1 Tbps traffic. This is way beyond the slow entry insertion speed (~ 100K per second) via the control API of today's programmable ASIC.

5.3 Addressing Out-of-Order Packets

As described in Section 3.3, to keep packets in order, we connect the output of the lookup table to the input of the buffer. The connecting is implemented with an extra RDMA request, as the red line illustrated in Figure 9. It means that until the corresponding buffer of a UE is completely released, its newly arrived packet will go through the buffer after the table lookup process. In this way, packets buffered during UE disconnection and newly arrival packets when the UE reconnects to networks will pass the same path, and thus X-Plane guarantees the order of packets.

As described in Section 3.3, to keep packets in order, we need to address two challenges: paging buffer state update and paging buffer release without external triggers. The paging buffer state is stored in external DRAM and includes a head pointer (p_out) and a tail pointer (p_in) of the buffer queue, as shown in Figure 9. The head/tail pointer is updated when releasing/storing packets from/to the buffer, and they are also read and compared to determine if the queue is empty when releasing or storing packets. As packet releasing and storing can happen simultaneously, the pointers can be accessed in parallel and suffer data in-consistency issue as explained by Figure 3. X-Plane uses the technology described in Section 5.1 to manage the pointers.

Instead of relying on external events, such as the arrival of ingress packets, X-Plane exploits internal events to trigger the release of the paging buffer. We utilize the response of an RDMA read to construct another request for releasing the next buffered packet. This process is described in Figure 10. When storing a packet, X-Plane adds the address of the next packet buffer to it. After the programmable ASIC sends an RDMA read and fetches a buffered packet, it parses the address and generates another RDMA read for releasing the next packet. Packets that pass through the buffer after reconnection may experience extra latency. In the worst case, they have to wait until all packets in the buffer are released. We will evaluate the latency in Section 7.7.

Another way to circumvent the two challenges is to use DRAM server CPU for buffer releasing. Such design can cause high CPU consumption when the paging ratio is high. Releasing buffer using ASIC allows us to reserve the CPUs for complicated UPF functionalities that cannot be offloaded (discussed in Section 8).

5.4 Putting Everything Together

Figure 11 shows X-Plane's workflow after we put everything together. We use the first and sequential packets of a flow to show how X-Plane works.

Slow Table Lookup (①-④). When the first packet of a flow from a UE arrives, X-Plane checks whether the 5-tuple of the packet exists in the flow table. Since it is the first packet of the flow, no matched entry is found in the flow table. Therefore, X-Plane turns back to the UE table, finds the matched entry and parses the PDRs in the entry to determine the action to be applied to the packet. Then, X-Plane writes the 5-tuple and action information into an entry in the flow table.

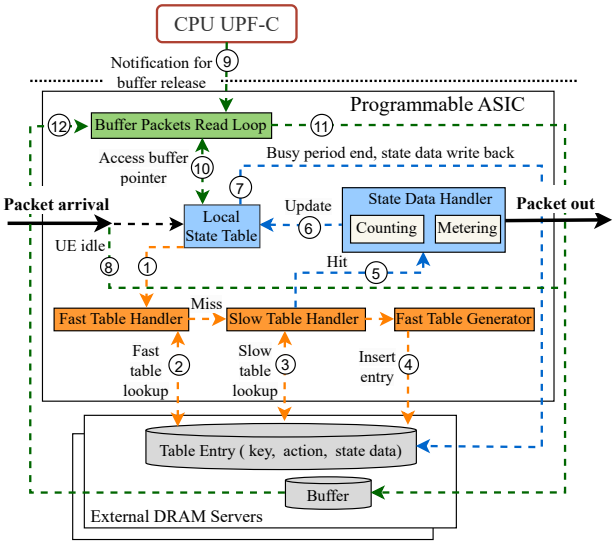


Figure 11: X-Plane's workflow.

Fast Table Lookup (①-②). When subsequent packets from the same flow arrive, since all of these packets have the same 5-tuple, X-Plane can find the matched entry in the flow table and therefore determine the actions to be taken on these packets based on the flow table information. Once the action is identified, X-Plane directly processes the packet without involving the slow UE table.

Counting and metering (⑤-⑦). After processing each packet, X-Plane updates the counters and meters and enforces the according forwarding policy. The update is carried out via the concurrent state data access protocol described in Section 5.1. In situations where there is a high influx of packets, X-Plane retains and updates the local state data on the programmable ASIC. Once the flow's peak traffic has been handled, the corresponding entry in the flow table is updated with the local state data.

Idle UE (⑧) and Buffered data release (⑨-⑫). X-Plane leverages a small SRAM space to keep track of the UE's idle state. If a UE is idle, the downlink traffic (Internet-to-UE) is buffered to the external DRAM server. Upon reconnection, X-Plane utilizes the Buffer Packet Read Loop module triggered by a notification packet from Control Plane, as described in Section 5.3, to deliver the buffered packets along with any newly arrived packets to the UE.

6 IMPLEMENTATION

X-Plane consists of ~5,000 lines of P4 code. We compile it to Intel Tofino based Semptian PS7350 programmable switch with P4 Studio 9.7.1. Our DRAM server runs an open-source RDMA agent [27] to set up connections with the ASIC switch and manage the table memory.

Local state table. As a P4 program cannot modify the tables on the data plane of the programmable ASIC at runtime, we implement the local state table with registers. Each column of the local state table, e.g., a key, state data or the in-flight number, is mapped to a register array. Each packet is matched with the registers using Tofino CRC hash and their keys are checked for collision using P4 conditional statements.

PDR lookup. In our PDR list, several PDRs are stored together in one entry and fetched by the programmable ASIC in one RDMA read. The programmable ASIC parses the PDRs as packet headers, and uses P4 conditional statements to match the keys. Limited by the ASIC PHV resources that are consumed to parse the header, we configure the number of PDRs in one entry to a maximum value of 5. We allocate 8 PDR entries for each UE, supporting a maximum of 40 PDRs.

Hash collision resolution and table size. Each entry in our UE/PDR and flow tables is allocated 2KB of memory. With a total number of 1M slow table entries, 8M PDR entries and 10M flow table entries, 38GB DRAM is required. To further reduce the hash collision rate, we allocate extra space for each table, i.e., 8M slow table entries, 64M PDR entries and 64M flow table entries, leading to a total of 272GB DRAM usage. Similarly, we allocate extra space for the LST tables, leading to a total SRAM usage of about 550KB. We empirically show that the ratio of hash collided packets in X-Plane is about 0.9%. When a collision happens, we redirect the packet to be processed by CPU on one of the DRAM servers. We show in our later experiments that CPU achieves a low latency within 20us when handling such small amount of traffic, and only a small extra CPU utilization is incurred.

7 EVALUATION

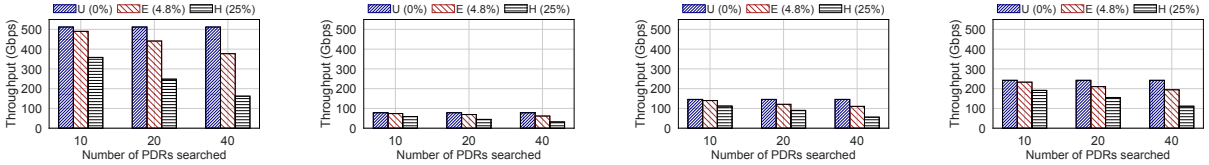
7.1 Setup and baseline

Testbed setup. Our testbed includes an Semptian PS7350 programmable switch with an Intel Tofino ASIC chip, two servers with Intel XEON 4314 CPUs (16 Cores) x2, and a server with Intel XEON 8380 CPUs (40 Cores) x2. All servers run CentOS 7.9 with a kernel version of 3.10.0, and we use the server with the 8380 CPUs as traffic generator and the other two as DRAM servers. The traffic generator has eight 100Gbps Mellanox CX-6 RDMA NIC and runs Trex 3.00 [5] based on DPDK 20.11 [10]. Each of the DRAM servers has four 100Gbps Mellanox CX-6 RDMA NIC and 16 × 32GB RAM. All servers are directly connected to the programmable switch. Note that X-Plane does not necessarily require two DRAM servers, the testbed is limited by the hardware we have at hand and we will further discuss the DRAM server requirements in Section 8.

Traffic workloads. We use synthetic traffic that contains 10 million flows from 1 million UEs. The UEs are generated

System	Counting	Metering	Packet Buffering	⋆Per Pipeline Flow Capacity	Per Pipeline Throughput	Fast-path Latency	Slow-path Latency	◇Fast-path Ratio	CPU Usage	Type
HybridUPF [18]	×	×	×	15M	10-100Gb/s	~1.3us	~350us	~40%	Medium	Academic
Kundel et al. [19]	×	×	×	N/A	100Gb/s	<1us	N/A	N/A	High	Academic
Zhou et al. [33]	×	×	×	1K	100Gb/s	<1us	~100us	N/A	High	Academic
Kaloom [14]	N/A	N/A	N/A	1M (4M)	375Gb/s (1.5Tb/s)	<1us	N/A	N/A	N/A	Commercial
P4UPF [20]	✓	×	✓	~117K	N/A	N/A	N/A	N/A	High	Academic
△ZTE-UPF (56 cores) [9]	✓	✓	✓	N/A	177Gb/s	N/A	~150us	N/A	High	Commercial
△ZTE-UPF (80 cores) [9]	✓	✓	✓	N/A	280Gb/s	N/A	~150us	N/A	High	Commercial
X-Plane	✓	✓	✓	10M	490Gb/s	~3.1us	~12us	~95%	Low	Academic

Table 1: Performance comparison of programmable ASIC-based UPF. ⋆ Kaloom uses two Tofino switches and contains at least 4 pipelines. We also report its total flow capacity and throughput in parentheses. ◇ Both the reported fast-path ratios are traffic-dependent, and they are both based on the same CAIDA trace [4]. Fast-path ratio of HybridUPF is a rough estimation from its performance data. △ZTE-UPF is a pure CPU-based UPF.



(a) Common 5G UPF test packet size (64-1518 bytes, an average of 690).
(b) 64-byte small packets.
(c) 128-byte small packets.
(d) 256-byte small packets.

Figure 12: X-Plane's throughput with packets of different size

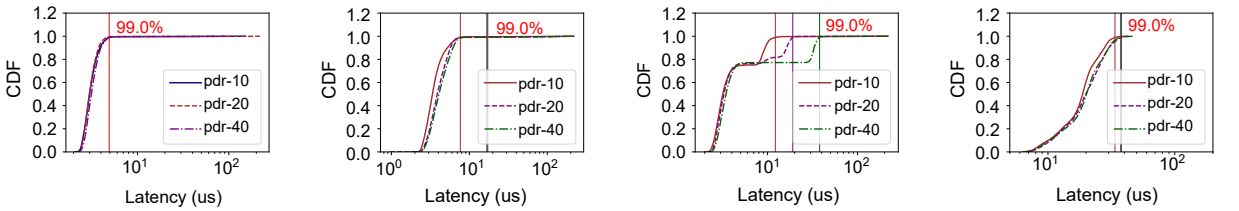
using 1 million random IPs. The UE IPs are used as the flow destination IPs to generate downstream traffic. We extend each UE IP to 10 IPv4 5-tuples by randomly selecting the flow source IP, source port and destination port from 1000 IPs and ports. The slow path packet ratio is an important factor that affects our system performance as it determines the amount of traffic that goes through the slow table path. We use a ratio of 4.8% obtained from the CAIDA [4] dataset. CAIDA is a widely used network traffic dataset that is collected from the backbone of the Internet. As the details of CAIDA trace are not publicly available, we use the revealed statistics from 2013 to 2019 and calculate the slow path ratio by dividing the average number of flows per second by the average number of packets per second. Note that such calculation is in fact an overestimation because flows in each second are usually not all new. We mark the packets with a probability of the slow path packet ratio, and our P4 program will direct the marked packets to the slow table path. We use a commonly adopted 5G UPF test traffic model [29] for packet size, which varies from 64 to 1518 bytes with an average of 690 bytes.

Baselines. We compare X-Plane with existing programmable ASIC-based UPF systems. We choose those proposed in existing work [6, 18–20, 33] that are either capable of reaching a throughput of 100Gb/s [18, 19, 33] or support UPF key features like counting or metering [20]. We also include a commercial UPF system, Kaloom[14]. As far as we know, these systems are all implemented with Intel Tofino. We report the per ASIC pipeline throughput of each system. Note that Kaloom uses two Tofino switches and contains at least 4 pipelines, we also report its total flow capacity and throughput. As none of the systems are open-sourced, we use the performance data reported in their papers. Likewise, the performance data of Kaloom is obtained from its product white papers. As far as we know, all the reported performance data are generated using similar settings as our evaluation, e.g., packet size distribution and the number of searched PDRs (< 10), and they represent the best-case performance of each system. Thus those reported data are eligible for the use of performance comparison with our system.

7.2 Overall Performance

Table 1 compares X-Plane with the baselines. Note that fast-path in baseline systems refers to processing traffic on programmable ASIC and slow-path refers to processing traffic on their CPUs. For X-Plane, fast-path refers to flow table path while slow path refers to UE table path, and in both paths the traffic is processed on programmable ASIC. We report the latency of the fast-path and slow-path respectively. From the table we make the following observations.

First, X-Plane supports all three key UPF features, including counting, metering, and packet buffering, while achieving the highest per pipeline throughput among all systems. The flow capacity of X-Plane is comparable to the highest 15M of HybridUPF [18], which however, is achieved without supporting the key UPF features. Kaloom achieves the closest per pipeline throughput (23% lower) to that of X-Plane, however, the flow capacity of X-Plane is 10X larger. As the tables of X-Plane are stored in external DRAM without relying on



(a) U (0%). (b) E (4.8%). (c) H (25%) (d) Hash collided packets (< 5Gb/s)

Figure 13: Latency distribution.

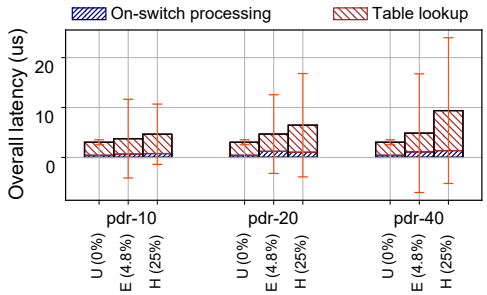


Figure 14: Latency breakdown. Note that the large error bars of U (0%) and E (4.8%) are mainly caused by the difference in fast-path and slow-path latency.

the local memory of the ASIC, we can further increase its flow capacity by adding more DRAM to the external server.

Second, X-Plane achieves a fast-path latency of 3.1 us, a slow-path latency of 12 us and an average latency of 3.6 us. In general, the fast-path latency is at a comparable level with those of other systems. It is about averagely 2 – 3 us higher due to the RDMA round trip to external DRAM. As the slow-path traffic of existing systems is processed on CPU, an latency of hundreds of microseconds is usually inevitable, e.g., about 350us for HybridUPF. In contrast, the slow path of X-Plane remains on hardware, leading to low slow-path latency. It shows that X-Plane further exploits the benefits of hardware processing compared with existing systems.

Third, X-Plane achieves the highest fast-path ratio. Existing hardware UPF systems redirect packets to CPU for processing when rule match misses on the hardware. HybridUPF optimizes the flow rules offloaded to hardware basing on heavy hitters to maximize fast-path throughput. We give a rough estimation from its reported performance data that about 40% of its traffic is processed on hardware. By comparison, X-Plane achieves a fast-path ratio of about 95%.

Finally, X-Plane achieves the lowest CPU usage. Note that we only give a rough judgment of the usage as low, medium and high. The low consumption of X-Plane is attributed to that its slow path is maintained on hardware and thus significantly reduces the CPU overhead. The optimized flow rule offload policy of HybridUPF increases its fast-path traffic compared with Zhou et al., Kundel et al. and P4UPF, thus lowering its CPU consumption. Note that we also include a pure CPU-based UPF in the table, i.e., ZTE-UPF. X-Plane outperforms ZTE-UPF in both throughput and latency. However, this is not really a fair comparison as first, the two systems differ hugely in the required CPU cores (for X-Plane, the total 64 cores of the DRAM servers are not necessary). Second, ZTE-UPF supports more complicated UPF functionalities like Deep Packet Inspection (DPI), putting it at a disadvantage when comparing throughput and latency. We will further discuss the support for more complicated UPF functionalities and CPU resource consumption in Section 8.

7.3 In-depth Performance Analysis

Throughput. We evaluate how the throughput of X-Plane is affected by three factors, including: (1) the slow path packet ratio, (2) the number of searched PDRs and (3) the average packet size. Besides the 4.8% slow path packet ratio that we obtained **E**mpirically from the CAIDA trace, labeled as "E (4.8%)", we also include two other settings. One is an ideal 0% slow path packet ratio that represents the performance **U**pper bound of X-Plane, labeled as "U (0%)". The other one is an example of ultra **H**igh slow path ratio, labeled as "H (25%)". Though such worse case is unlikely to happen in real deployments, it helps us understand the lower bound of our system's performance. We vary the number of searched PDRs from 10 to 40, and use packet sizes of 64, 128 and 256 bytes to evaluate the system throughput with small packets.

Results in Figure 12(a) show that the best performance of our system reaches a throughput of 490Gb/s (E (4.8%), PDR=10). Although the incoming ports provide a total bandwidth of 800Gb/s, 490Gb/s incoming traffic saturates the 800Gb/s links from the switch to backend DRAM servers. Such overhead is mainly caused by (1) RDMA overhead, including the RDMA header added to each packet to write/read to/from external DRAM to perform table lookup, and the additional RDMA requests to write back state data. (2) Slow path overhead, as a packet processed on the slow path can incur several rounds of write/read to/from external DRAM.

More rounds of PDR searching decrease the system throughput. Such degradation is more obvious with H (25%), as the PDR searching affects only the cost on the slow path. It is relatively milder with E (4.8%), where we observe a 23% throughput decrease as the PDR number increases from 10

to 40. However, even with H (25%) and a PDR number of 40, X-Plane achieves a throughput put of 162Gb/s, still outperforming all academic hardware UPF systems in Table. 1.

Figure 12(b)-12(d) show that X-Plane achieves a throughput of 78Gb/s, 146Gb/s and 242Gb/s respectively (E (4.8%), PDR=10) with smaller packets. When the packet size is 64 bytes, the maximum throughput in terms of Mpps is achieved, i.e., 156Mpps. Such throughput is limited by the RDMA read throughput of the CX6 NICs on DRAM servers. In our system, a 100Gb NIC port can achieve a maximum of near 40Mpps RDMA read throughput while 120Mpps for RDMA write when communicating with the ASIC switch. The RDMA read throughput is about 30% lower compared with the throughput between two CX6 NICs. Further improving the RDMA read performance between ASIC switch and DRAM server is expected to increase the overall performance of X-Plane.

Latency. Figure 14 shows the average latency of X-Plane. For U (0%), the average latency is about 3 us. We break down the latency as the on-switch processing time and the table lookup time. On-switching time refers to the time spent on ASIC switch to perform operations such as RDMA protocol stack processing, hash calculation and PDR match. Table lookup is the time spent on accessing data from the external DRAM. The results show that the table lookup time constitutes more than 83% of the total latency on average. This is because an RDMA round trip to external DRAM can take up to several microseconds, while traversing the pipeline in an AISC switch typically takes less than 1 microsecond.

Figure 13 demonstrates the distribution of the latency. Figure 13(a) shows that the distribution of U (0%) stays unchanged with different numbers of searched PDRs. Because no PDR lookup happens when the slow path packet ratio is zero. Over 99% of the latency of U (0%) are within 5us. For E (4.8%) in Figure 13(b), the 1% tail latency is determined by the slow path latency. As the PDR number increases from 10 to 40, over 99% of the latency remains under 20us, such latency is much lower than the hundred microsecond-level slow path latency in baseline systems, making it possible to satisfy the ultra-low 5G latency requirement, i.e., 1ms latency.

In addition, we demonstrate the latency of hash collided packets in Figure 13(d). As mentioned before, the collided packets are redirected to be processed by CPU on DRAM server. We show that these packets constitute less than 1% of the total traffic, i.e., < 5Gb/s. Although software UPF typically suffers latency up to hundreds of microseconds when running at normal traffic load, it achieves an average latency within 20us when handling such low load of traffic.

ASIC resource consumption. We obtain the resource consumption from the P4 compiler's output. As shown in Table 2, X-Plane consumes less than 30% of the available resources for the majority of resource types, including TCAM, SRAM, Hash Bits, VLIW and Match Crossbar. The TCAM consumption is zero as X-Plane does not rely on the ternary match capability of the switch. X-Plane utilizes the SRAM mostly for storing RDMA-related information like QP index, and for implementing the local state table. The total SRAM consumption is only 5.94%, and there remains enough space to implement other functionalities or optimization techniques like hot/hierarchical tables.

ASIC Resource	Usage (%)	Server Resource	Usage
TCAM	0	DRAM	264 GB
SRAM	5.94	CPU utilization	≈ 6.3%
Hash Bits	15.97	Memory Bandwidth	30%
VLIW	16.93		
Match Crossbar	20.38		
Gateway	51.56		

Table 2: Programmable ASIC and DRAM server resource consumption.

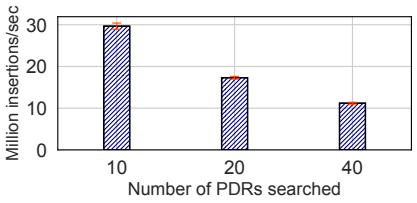


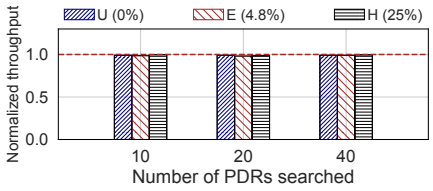
Figure 15: Flow table rule generation speed. X-Plane achieves millions of flow rule insertions per second.

DRAM server resource consumption. We report the server resource consumption in Table. 2. The total DRAM usage of a server is 264GB, including 136GB for implementing UE and flow table, and 128GB reserved for idle UE packet buffer. Two CPU cores are used to handle the hash collided packets and incur an average of 6.3% CPU utilization. The RDMA server procedure only incurs a short period of 6% CPU utilization during startup for setting up connections and initializing tables. The total memory bandwidth consumption is 30%. In general, X-Plane incurs a low CPU and memory bandwidth consumption on the DRAM servers.

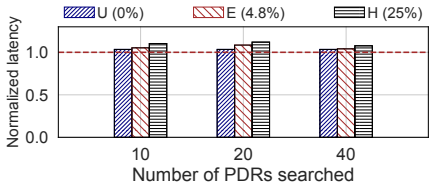
7.4 Resource Consumption

7.5 Flow Table Generation Speed

Next we evaluate the maximum speed of X-Plane to translate the PDR rules into flow rules and insert them into the fast flow table. The maximum speed is obtained by setting the slow path ratio to 100%. Results in Figure 15 show that X-Plane achieves an insertion rate of 29.7 million insertions per second when the number of searched PDRs is 10. As a flow rule can only be generated after a PDR is matched on the slow path, the insertion rate is limited by the slow path throughput. Thus we see that insertion rate decreases as the PDR number increases, and achieves 17.3 and 11.2 million insertions per second respectively with 20 and 40



(a) **Effect on throughput.** Throughput degradation is within 2% compared to when concurrency control is not applied.



(b) **Effect on latency.** Latency increase is within 12% compared to when concurrency control is not applied.

Figure 16: Concurrency control overhead.

PDRs. Basing on the analysis in existing work [26], flow rule insertions per second grows approximately by 4K per Gb/s of internet traffic. Thus about 2M insertions per second is required for 500Gb/s of traffic. The generation speed of X-Plane is sufficient to meet such requirement.

7.6 Concurrency Control Overhead

To evaluate the effect of our LST-based concurrency control mechanism on the performance of X-Plane, we normalize the system latency and throughput to those when the concurrency mechanism is removed. As shown in Figure 16, the throughput degradation is within 2%, and the latency increase is within 12%. Such results suggest that our LST-based concurrency control mechanism design can be implemented with only a small overhead. This can be explained as that the mechanism only incurs several extra hash calculations and register access for each packet processing on the ASIC switch, which does not significantly increase the consumption of ASIC clock cycles.

7.7 In-order Buffering Latency Overhead

Our FIFO buffer ensures the in-order release of buffered packets. We evaluate the extra latency of buffer design, by increasing the number of packets in the buffer from 2 to a maximum value of 128. Results in Figure 17 show that the extra latency increases linearly, from 10 us to a maximum value of 640 us. The linearity is due to that the buffered packets are released in a serial fashion and each release incurs a latency about 4-5 us.

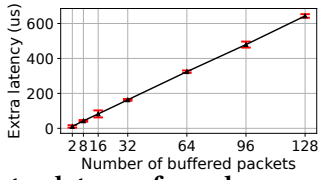


Figure 17: Extra latency for subsequent packets The extra latency increases linearly with the number of packets in the buffer.

7.8 Case Study: Rate Limiting

We show an example of implementing a UPF functionality on X-Plane. We apply rate limiting policies to three UEs. The bucket tokens used for rate limiting are stored in external DRAM and accessed from the ASIC switch using our concurrency control mechanism to ensure the correctness of the data. As shown in Figure 18, packets of three UEs arrive at time 0, and are forwarded with a data rate of 1Gb/s. Then around time 12 and 20, the mobile data usage of UE-0 and UE-1 reaches a predefined quota and the data rates of the two UEs are limited to a lower level, i.e., 256Mb/s and 512Mb/s respectively. The average throughput error is within 1%.

8 DISCUSSION

Offloading more complicated UPF functionalities. Note that X-Plane does not guarantee that all UPF functionalities can be offloaded to programmable ASIC, for example, Deep Packet Inspection (DPI) can have a long logic chain that exceeds the stage limit of ASIC pipeline, and it is challenging for the ASIC to perform scheduling that requires global UE/flow information. To support these functionalities, we can resort to CPUs on the DRAM servers. We envision that we can combine X-Plane and CPU-based UPF to get the best of both worlds by offloading all basic (and required) functionalities to X-Plane while keeping more complicated (and optional) functionalities to CPU.

DRAM server requirements. The current implementation of X-Plane only requires the DRAM server to provide (1) 800Gb/s RDMA and memory bandwidth and (2) a small amount of CPU cores (<4), which can be easily satisfied by a single off-the-shelf server. Combining X-Plane and CPU-based UPF to support more complicated UPF functionalities can increase the required number of cores on the DRAM server. We expect that such combination requires much fewer CPU cores to outperform a pure CPU-based UPF, but further investigation is needed to determine the required number of CPU cores and the level of improvement.

9 RELATED WORK

Other hardware offload UPF systems. Another major category of hardware UPF is smartNIC-based UPF systems [3, 12,

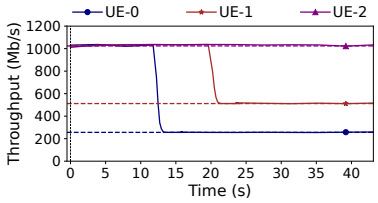


Figure 18: An UPF functionality example: rate limiting. The data rates of three UEs are eventually limited to 256Mb/s, 512Mb/s and 1Gb/s respectively.

18, 21, 22]. By comparison, smartNIC provides larger memory, typically several Gbs, and better programmability that can simplify the design of UPF. For example, Synergy [22] directly buffers/releases packets on/from smartNIC using on-chip CPU cores and can achieve low buffer releasing latency than X-Plane, i.e., < 10us. However, the memory of smartNIC is still not enough to provide buffer for all idle UEs. Based on our communication with service providers, the idle UE ratio is usually larger than 40% and a packet buffer of about 100KB is provided for each idle UE. Assuming 1M UEs, 300Gb memory is required, which exceeds the capacity of a smartNIC. Although we can stack many smartNICs together to increase the memory size, it causes a waste of computation and bandwidth resources on the smartNICs. In contrast, ASIC with external DRAM enables X-Plane to flexibly extend its memory capacity. Other works [2, 3] propose to also offload PFCP message parsing to hardware to cope with the rule insertion speed bottleneck while X-Plane does not suffer such problem as explained in Section 5.2.

RDMA-based memory extension for ASIC switch. Besides TEA [16] and its early design [17], Ribosome [26] uses RDMA-based external DRAM to extend ASIC switch memory. It sends packet headers to network function servers and only uses the RDMA-based memory for temporally storing the packet payload before the packet headers finish processing. It utilizes the external memory in a relatively simple way and does not answer the question that how it can be utilized to implement functions in 5G UPF.

Other ASIC switch memory extension work. Tiara [32] uses an FPGA based design to extend ASIC switch memory for implementing layer-4 load balancing. Its design relies on the fact that the load balancing rules can be split and only the connection-to-real server mapping needs to be maintained on ASIC switch. However, such rule splitting technique is not applicable to PDRs in 5G UPF.

PDR lookup optimization. L25GC [11] studies the performance of different PDR search algorithms and shows that optimized ones like *partition search* and *tuple space search* effectively reduce search overhead compared with simple linear search. The PDR search algorithm optimization is orthogonal to our design and can be further integrated to reduce the number of searched PDRs and improve the slow path performance of X-Plane.

Flow cache. The fast table of X-Plane is similar to the idea of microflow cache in Open vSwitch (OVS) [24] where exact-match rules corresponding to each connection are generated as the fast forwarding path. The difference is that X-Plane stores the fast table in external DRAM and utilizes hardware programmability to generate the rules directly from within the data plane for higher rule insertion speed.

External NF state data. Many network virtualization works have proposed to utilize remote storage to store NF state data [8, 13, 28]. Their concurrency problem happens in a scenario different from that of X-Plane when multiple of their nodes access the same state data. The problem can be solved with a common lock mechanism as the CPU-based NFs can suspend the packet processing and wait for the release of locks. CHC [15] ensures data consistency by using CPU on remote storage servers to operate on the state data. As state update is incurred by nearly every packet in 5G UPF, e.g., counter increase, applying such design causes high CPU consumption and degrades system throughput. The early work of TEA [16] uses RDMA atomic add operation to update counters in external DRAM. However, it achieves limited throughput as RDMA atomic operation has nearly 8X lower Mpps throughput than that of RDMA write/read [25]. In addition, as RDMA atomic operation does not support read, it fails when both read and write of the state data are required, like the read and update of the metering bucket token. In contrast, X-Plane implements concurrency control that supports both read and write while incurring near zero throughput penalty.

10 CONCLUSION

In this paper, we introduce the design, implementation and evaluation of X-Plane, a UPF that supports 490 Gb/s throughput, over 10 million flows and less than 4 us latency. X-Plane is built by combining programmable ASICs and external DRAM. It invents several novel technologies, such as concurrent state data access, fast flow table and our-of-order packets handler to enable high performance UPF functions, such as counting, meter, etc. We believe X-Plane opens opportunities to design high performance UPF on the cloud infrastructure.

ACKNOWLEDGMENTS

This work was in part supported by the National Natural Science Foundation of China (No. 62072302), Alibaba Innovative Research (No. 2022010307) and National Key Research and Development Program of China (No. 2020YFB1807803).

REFERENCES

[1] aws. 2020. 5g-network-evolution-with-aws. https://d1.awsstatic.com/whitepapers/5g-network-evolution-with-aws.pdf.

[2] Abhik Bose, Shailendra Kirtikar, Shivaji Chirumamilla, Rinku Shah, and Mythili Vutukuru. 2022. AccelUPF: Accelerating the 5G User Plane Using Programmable Hardware. In *SOSR '22*. 1–15.

[3] 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 *5th Asia-Pacific Workshop on Networking (APNet 2021)*. 57–64.

[4] CAIDA. 2019. Trace Statistics for CAIDA Passive OC48 and OC192 Traces. https://www.caida.org/catalog/datasets/trace_stats/.

[5] Cisco. 2023. TRex Realistic Traffic Generator. https://trex-tgn.cisco.com.

[6] Zhou Cong, Zhao Baokang, Wang Baosheng, and Yuan Yulei. 2021. CeUPF: Offloading 5G User Plane Function to Programmable Hardware Base on Co-existence Architecture. In *Proceedings of the 2021 ACM International Conference on Intelligent Computing and its Emerging Applications*. 34–39.

[7] ericsson. 2020. ericsson.com-cloud-native. https://www.ericsson.com/en/cloud-native.

[8] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. 2014. OpenNF: Enabling innovation in network function control. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 163–174.

[9] Intel. 2021. ZTE's High Performance 5G Core Network UPF Implementation Based on 3rd Generation Intel® Xeon® Scalable Processors. https://networkbuilders.intel.com/solutionslibrary/zte-s-high-performance-5g-core-network-upf-implementation-based-on-3rd-generation-intel-xeon-scalable-processors.

[10] Intel. 2023. Data Plane Development Kit. https://www.dpdk.org.

[11] Vivek Jain, Hao-Tse Chu, Shixiong Qi, Chia-An Lee, Hung-Cheng Chang, Cheng-Ying Hsieh, KK Ramakrishnan, and Jyh-Cheng Chen. 2022. L25GC: a low latency 5G core network based on high-performance NFV platforms. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 143–157.

[12] Vivek Jain, Sourav Panda, Shixiong Qi, and KK Ramakrishnan. 2022. Evolving to 6G: Improving the Cellular Core to lower control and data plane latency. In *2022 1st International Conference on 6G Networking (6GNet)*. IEEE, 1–8.

[13] Murad Kablan, Azzam Alsudais, Eric Keller, and Franck Le. 2017. Stateless Network Functions: Breaking the Tight Coupling of State and Processing. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 97–112.

[14] Kaloom. 2020. The Kaloom 5G User Plane Function (UPF). https://www.mbuzzeurope.com/wp-content/uploads/2020/02/Product-Brief-Kaloom-5G-UPF-v1.0.pdf.

[15] Junaid Khalid and Aditya Akella. 2019. Correctness and performance for stateful chained network functions. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 501–516.

[16] Daehyeok Kim, Zaoxing Liu, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, Vyas Sekar, and Srinivasan Seshan. 2020. Tea: Enabling state-intensive network functions on programmable switches. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 90–106.

[17] Daehyeok Kim, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, and Srinivasan Seshan. 2018. Generic external memory for switch data planes. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*. 1–7.

[18] Suneet Kumar Singh, Christian Esteve Rothenberg, Jonatan Langlet, Andreas Kassler, Peter Voros, Sandor Laki, and Gergely Pongracz. 2022. Hybrid P4 Programmable Pipelines for 5G gNodeB and User Plane Functions. *IEEE Transactions on Mobile Computing* (2022), 1–18.

[19] Ralf Kundel, Tobias Meuser, Timo Koppe, Rhaban Hark, and Ralf Steinmetz. 2022. User Plane Hardware Acceleration in Access Networks: Experiences in Offloading Network Functions in Real 5G Deployments.. In *HICSS*. 1–10.

[20] 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)*. 162–168.

[21] Napatech. 2022. 5G User Plane Function (UPF) Offload for Napatech Programmable SmartNICs with Link-Inline Software. https://www.napatech.com/support/resources/data-sheets/5g-user-plane-function-upf-offload/.

[22] Sourav Panda, K. K. Ramakrishnan, and Laxmi N. Bhuyan. 2022. Synergy: A SmartNIC Accelerated 5G Dataplane and Monitor for Mobility Prediction. In *2022 IEEE 30th International Conference on Network Protocols (ICNP)*. 1–12.

[23] Tian Pany and et al. 2022. Sailfish: Accelerating Cloud-Scale Multi-Tenant Multi-Service Gateways with Programmable Switches. In *In ACM SIGCOMM 2021, Conference (SIGCOMM'21)*. ACM, New York, NY, USA, 13 pages.

[24] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. 2015. The design and implementation of open vSwitch. In *12th USENIX symposium on networked systems design and implementation (NSDI 15)*. 117–130.

[25] Waleed Reda, Marco Canini, Dejan Kostic, and Simon Peter. 2022. RDMA is Turing complete, we just did not know it yet!. In *Proceedings of NSDI*, Vol. 22.

[26] Mariano Scazzariello, Tommaso Caiazzi, Hamid Ghasemirahni, Tom Barbette, Dejan Kostic, and Marco Chiesa. 2023. A High-Speed Stateful Packet Processing Approach for Tbps Programmable Switches. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*.

[27] Mariano Scazzariello, Tommaso Caiazzi, Hamid Ghasemirahni, Tom Barbette, Dejan Kostic, and Marco Chiesa. 2023. Ribosome RDMA Server Agent. https://github.com/Ribosome-Packet-Processor/Ribosome-RDMA-Server-Agent.

[28] Shinae Woo, Justine Sherry, Sangjin Han, Sue Moon, Sylvia Ratnasamy, and Scott Shenker. 2018. Elastic scaling of stateful network functions. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 299–312.

[29] Qiang Wu, Xiangping Bryce Zhai, Xi Liu, Chun-Ming Wu, Fangliang Lou, and Hongke Zhang. 2022. Performance Tuning via Lean Measurements for Acceleration of Network Functions Virtualization. *IEEE/ACM Transactions on Networking* (2022).

[30] X-Plane. 2023. x-plane-5g-upf. https://github.com/AlibabaResearch/x-plane-5g-upf.

[31] Guoliang Gu Yong Li, Jerry Zhang and et al. 2019. Implementation of ZTE's High-Performance 5G Core Network UPF.

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

[33] Cong Zhou, Baokang Zhao, and Baosheng Wang. 2022. A 100Gbps User Plane Function Prototype Based on Programmable Switch for 5G Network. In *6th Asia-Pacific Workshop on Networking (APNet 2022)*.