Synergy: A SmartNIC Accelerated 5G Dataplane and Monitor for Mobility Prediction

Sourav Panda, K. K. Ramakrishnan, Laxmi N. Bhuyan
University of California, Riverside, CA

Abstract—The 5G user plane function (UPF) is a critical interconnection point between the data network and cellular network infrastructure. It governs the packet processing performance of the 5G core network. UPFs also need to be flexible to support several key control plane operations. Existing UPFs typically run on general-purpose CPUs, but have limited performance because of the overheads of host-based forwarding.

We design Synergy, a novel 5G UPF running on SmartNICs that provides high throughput and low latency. It also supports monitoring functionality to gather critical data on user sessions for the prediction and optimization of handovers during user mobility. The SmartNIC UPF efficiently buffers data packets during handover and paging events by using a two-level flow-state access mechanism. This enables maintaining flow-state for a very large number of flows, thus providing very low latency for control and data planes and high throughput packet forwarding. Mobility prediction can reduce the handover delay by pre-populating state in the UPF and other core NFs. Synergy performs handover predictions based on an existing recurrent neural network model. Synergy's mobility predictor helps us achieve 2.32× lower average handover latency.

Buffering in the SmartNIC, rather than the host, during paging and handover events reduces packet loss rate by at least 2.04×. Compared to previous approaches to building programmable switch-based UPFs, Synergy speeds up control plane operations such as handovers because of the low P4-programming latency leveraging tight coupling between SmartNIC and host.

I. Introduction

The emergence of 5G promises high speed and low latency, enabling a wide range of innovative applications like Internet of Things (IoT), augmented/virtual reality, *etc.* At the crux of the 5G data plane in the packet processing core of the cellular network is the User-Plane Function (UPF) which serves as the interconnect point between the mobile infrastructure and the data network [1]. At the UPF, complex rules have to be followed for forwarding and tunneling. It processes packets belonging to different sessions with different priorities, including the need for shaping and policing the traffic. Additionally, the UPF must perform flow-state dependent processing, such as when a mobile device goes idle (to save battery energy) and the UPF has to be aware of the idle/active transitions of individual mobile devices (also called User Equipment, or UE). Similarly, when a UE is mobile, a handover is performed for the UE to have its radio network association change from one (source) base station to another (target) base station. For these situations, the UPF has to be aware of the state of the UE (hence the flow's state) which potentially requires the UPF to buffer packets until the UE is ready to receive data.

Implementing 5G core (5GC) NFs [2] on general-purpose CPU cores (we refer to as 'host'), including the UPF, can limit throughput and increase latency, especially when the number of CPU cores for the UPF is limited. Overheads, such as context switches, interrupts, PCIe transactions, data serialization and de-serialization, packet copy, *etc.* contribute to constraining the performance [3]. Since the 5GC supports a large number of UEs connected to multiple base stations, facilitating a wide range of critical applications and services [4], achieving high performance for the 5GC is key. Utilizing network acceleration to implement 5GC NFs can substantially improve throughput.

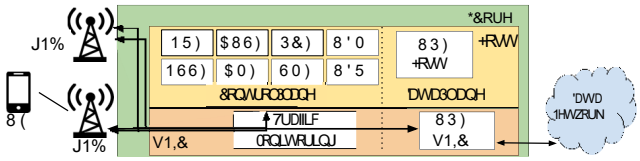


Fig. 1: Synergy 5GC Architecture

Another avenue for network acceleration is using programmable switches. While programmable switches (P4Switch) for the 5GC data plane packet processing show promise [5], they have two drawbacks. First, P4Switches do not have large buffers or the ability to hold packets as required by the 5GC's UPF [5]. To overcome this feature gap, [5] buffers packets in the host using a buffering microservice. Secondly, the limited amount of memory on a P4Switch (e.g., in the order of 100MB SRAM [6]) limits its ability to support flow state tracking, even though it can forward traffic at very high rates [3]. This impedes its ability to maintain flow state and conduct monitoring for a large number of flows.

In this work, we implement Synergy, a 5G UPF on a SmartNIC (sNIC), as shown in Fig. 1. Not only does it provide network acceleration to outperform host-based UPFs, but it can effectively carry out state tracking and buffering unlike programmable switches [3]. With the sNIC having memory of the order of GBs, packets can be buffered and flow state can be effectively retained on the sNIC. The P4 programmability [7] on the sNIC also enables handling various packet processing tasks. Furthermore, the CPU cores being just a PCIe transaction away provides for a tight coupling between the UPF on the sNIC and the other NFs of the 5G ecosystem running on host CPUs. Synergy is publicly available at [8].

Beyond implementing the core functions for a UPF on the sNIC to be compliant with the 3GPP specification [9], [10], we focus on two significant additional capabilities. The first is to support a responsive buffering capability in the UPF, since it impacts the idle-active and handover latency. Instead of buffering packets in the source 5G base station (gNB) during

978-1-6654-8234-9/22/$31.00 ©2022 IEEE

2022 IEEE 30th International Conference on Network Protocols (ICNP) | 978-1-6654-8234-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICNP55882.2022.9940261

handover (as in Sec. 9.2.3.2.2 in [11]), 'Smart buffering' of packets within the UPF has been proposed as a way of reducing the latency in L^{2}5GC [12] and CleanG [13]. This avoids the hairpin routing from source gNB to target gNB through the 5GC, and the associated latency. Synergy implements packet buffering in the sNIC UPF while ensuring packets are delivered in order. However, no change to the 3GPP control protocol messages are needed. Buffering at the source gNB (especially for small cells) may also be unattractive from a cost standpoint. Synergy is built on top of 3GPP compliant 5GC implementations L25GC [12] and Free5GC [2].

The sNIC can buffer most of the packets locally as opposed to the host so that it can rapidly respond to UE state changes and retain high packet throughput. We show that the packet loss rate during handovers reduces by 2.04× (§IV-D) when buffering within the sNIC instead of the buffering within the host. Compared to other sNIC-based flow state management approaches such as DeepMatch [14] and SmartWatch [3] that can also be used in UPF processing, Synergy achieves at least 1.40× lower packet loss rate because it reduces the flow state access latency (§IV-D). Our solution Synergy improves packet processing rate and latency during control-plane events such as handover and paging. We introduce a two-level flow caching mechanism that reduces flow state access times by at least 15% compared to UPF built over the flow management technique of SmartWatch [3] (§IV-B). Synergy increases its capacity by 44×, to support up to 12 million flows (§III-B2) compared to UPF built with the flow management technique of DeepMatch.

Mobility prediction helps in pre-populating and updating state on the 5GC NFs, thereby reducing the handover latency. In order to accommodate mobility predictions, we modify the sNIC packet processing pipeline to parse and monitor the control plane traffic in the sNIC. Control plane messages contain location [15] that can be monitored for mobility prediction. Synergy leverages intelligent algorithms for effectively predicting mobility events. 5G uses a control/user plane split (CUPS)-based architecture [16]. In this work, we propose running the control plane NFs on the host and the userplane on the sNIC. Since the sNIC and host are just separated by a PCIe transaction, it leads to very low programming latency. This allows us to push table modification more quickly as required for handovers and paging. Synergy parses control plane packets destined for control plane NFs running on the host and updates the flow state maintained in the sNIC. Feeding the monitored data to a mobility predictor helps achieve 2.32× lower average handover latency compared to not performing mobility prediction. Our paper makes the following contributions:

- We implement a 3GPP compliant 5G UPF on a highly parallel sNIC.
- Design a responsive buffering scheme on the sNIC to improve packet processing efficiency.
- Extend the monitoring functionality on the sNIC and use it towards mobility prediction.
- Evaluate our platform against real-world traces as well as simulation-generated traces.

II. Background

A. 5G Preliminaries

Majority of the cellular 'services' are provided by the core network, which is responsible for connecting UEs to the Data Network (typically the Internet or an IP network). The user accesses network services via a cellular base station (gNB) using a mobile device that we refer to as the User Equipment (UE). Some subcomponents of the core network (as shown in Fig. 1) are the Access and Mobility Function (AMF), Service Management Function (SMF), and User Plane Function (UPF). The AMF is responsible for authenticating the UE, connectivity, and mobility management. The Packet Forwarding Control Protocol (PFCP) is used by the SMF to configure the UPF data forwarding behaviour and user policies [9], [10]. The control plane must setup a unique tunnel endpoint identifier (TEID) to tunnel dataplane traffic between the gNB and UPF using GTP [5]. The datapath from the gNBs of the Radio Access Network (RAN) to the Data Network (Internet) is provided by the User Plane Function (UPF). It performs packet processing for user flows and includes support for UE mobility, buffering for idle UEs, traffic accounting, and QoS based on rules configured by the control plane [5].

Traffic Classification: Each uplink or downlink packet must be matched to a UE and its associated traffic class by the UPF based on a set of Packet Detection Rules (PDRs) [17]. A PDR may match the UE's IP address, the tunnel headers (uplink packets), the packet's five-tuple, or the domain name of the remote end-point. The matching PDR determines how the UPF then processes the packet. The control plane installs, changes, and removes PDRs when a UE attaches, moves to another gNB, goes idle or detaches [5].

Mobility and packet forwarding: As a UE moves, it may connect to a new gNB. The UPF applies a Forwarding Action Rule (FAR) identified by the PDR to place the appropriate tunnel header for DL packets forwarded to the right gNB. The FAR for DL traffic specifies the tunnel header field and gNB IP address. Generally, a FAR specifies a set of actions to apply to the packet, including tunneling, forwarding, buffering, and notifying the control plane. FARs are installed and removed when a UE attaches or detaches, respectively, and the DL FAR changes when the UE is handed off to a new gNB, goes idle, or is woken up [5], [18].

Buffering for idle UEs: Battery optimizations seek to have UEs go idle as soon, and for as long as possible. When a UE goes idle, the UPF buffers DL traffic destined to a UE until it wakes up to send or receive packets. When traffic first arrives, the UPF alerts the control plane, which then interacts with the gNB to wake up the UE. Once the UE wakes up, the UPF transmits the buffered DL traffic and resumes normal forwarding, ensuring packets are delivered in-order [5], [13].

Handover procedure: During a handover procedure, when a UE connects to a new gNB, the user typically experiences added delay and possibly data loss. The handover operation can take up to 130 milliseconds to complete [12]. This can severely affect data plane traffic. Further, UE handover operations may be more frequent because of the smaller cell sizes

and emerging applications such as connected vehicles [19]. These require completing handovers as quickly as possible. Along with the many control message exchanges [20], the 5G handover involves data packets being buffered at the source gNB. When the UE synchronizes with the target gNB these packets are re-routed to the target gNB, through the 5GC, using 'hairpin' routing. In Synergy we delegate this buffering task to the UPF, avoiding the hairpinning.

B. Requirements for Synergy

Requirement: Synergy aims to accelerate UPF performance while still being feature-rich, allowing the operator to dynamically update rules.

Network Acceleration: Achieving high performance for the UPF can be enabled by network acceleration, since hash computations, encapsulation, and state management are not impeded by interrupt processing, context switches, PCIe transaction delays, and expensive packet copies on the host. We design Synergy to achieve high performance, compared to a state-of-the-art host-based 5GC like Free5GC [2], with the dataplane (UPF) implemented using the DPDK [21] libraries.

UPF Programmability: The UPF must dynamically update FARs and PDRs along with their priorities (see §II-A). These translate to P4 table updates that need to be programmed into a switch or an sNIC if the UPF runs on one of these. Platforms based on programmable switches can handle on average 1200 new table rules per second [22]. Since rules will have to be pushed when the UE goes idle, encounters mobility, attaches or detaches the network, the 1200 rules/sec. will limit how many control plane events can occur in the 5GC (which may support multiple gNBs). P4Switch-based UPFs are being considered [5] because of the potential for higher dataplane throughput. Synergy can outperform them from the perspective of responsiveness and lower control plane latency.

Optimized UPF Buffering: Buffering is required for idle-active transitions. Furthermore, we also delegate the buffering task from the gNB to the UPF for mobility management. We advocate this implementation change to reduce computation and memory overhead on gNBs (especially small cells).

Requirement: Synergy aims to monitor the control plane traffic to predict future mobility events and reduce the control plane overheads incurred during handover.

Accommodate mobility predictions: Predicting mobility using off-the-shelf neural networks can speedup the handover process by prepopulating state at the 5GC. This speedup directly benefits the end-user experience. Since prepopulating state will also lead to more memory and computation overhead on the UPF, we incorporate a probabilistic data structure to minimize the throughput impact of mobility prediction.

Control Plane-UPF colocation: By co-locating control plane NFs on the same host as the UPF (e.g, on sNIC), we ensure control plane messages also traverse the same sNIC where the UPF is running. In doing so, we monitor control plane traffic, such as Location Request/Response to update the data structures stored in the sNIC and ultimately use it for mobility prediction.

C. Related Work

1) 5G UPF Designs: Several cellular dataplanes have been implemented that perform PDR matching and FAR actions. A P4Switch based UPF was introduced in [5] to improve data plane throughput. Free5GC [2] is an open-source 3GPP compliant kernel-based implementation, which consists of a UPF running on the host. Furthermore, DPDK-based software-UPF solutions [23] have also been introduced that seek to get rid of kernel overheads.

2) Monitoring at the Cellular Core: Monitoring of cellular networks has been explored before. NG-Scope [24] facilitates accurate and millisecond-granular capacity estimation updates for the cellular network. This allows the congestion controller and the upper layer applications to adjust their system parameters, such as send rate or video resolution, with the underlying network condition. NG-Scope performs network telemetry at UEs, but could potentially be supported at gNBs [24].

3) Buffering Optimizations: Prior work for buffering packets in the 5GC [2], [23], often leads to high packet loss compared to a sNIC-based UPF solution like Synergy, primarily due to the slower processing at the host (§IV-A). Even P4Switch-based solutions resort to buffering on the host because of the lack of support for it in the P4Switch [5].

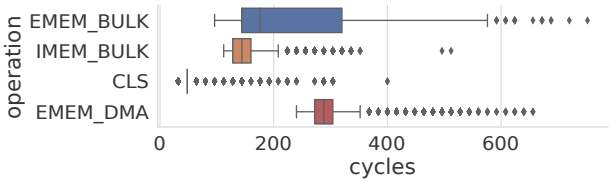


Fig. 2: Memory Latency sNIC

III. Design

A. sNIC Preliminaries

We design the 5G UPF to operate on a Netronome Agilio LX 2×40 GbE sNIC, which has 8GB DDR3 memory and 96 highly threaded flow processing cores, referred to as Micro-Engines (MEs). User code runs on up to 81 MEs distributed on seven islands and each ME has a private code store that can hold 8K instructions. MEs are 32-bit 1.2 GHz RISC-based cores that have 8 thread contexts [14]. Switching contexts takes 2 cycles and threads must explicitly yield execution as the thread scheduling is non-preemptive. The sNIC can be programmed in both Micro-C, which is an extended subset of C-89, and P4 [3]. P4 code parses the uplink and downlink traffic arriving at the UPF. P4-defined match-action tables, populated by the control plane at run-time, determine actions to apply to the packet based on the parsed headers [14].

Memory Hierarchy: MEs have access to large, shared global memories (8 GB) and small local memories (4 KB capacity) that are fast but require programmer management [14]. Table I shows the memory hierarchy on the 40 GbE sNIC [14], in terms of capacity and usage. Fig. 2 shows the memory latency for write operations. As expected, CLS being closer to the ME has lower write latency overhead compared to EMEM and IMEM bulk write operations (4 Bytes). We do not evaluate CTM because that is used to store packet payloads as they

are being processed. *Since EMEM is composed of SRAM and DRAM, where the SRAM acts as a cache to DRAM, EMEM writes would result in a longer tail than IMEM that solely consists of SRAM*. The EMEM DMA operation depicts the latency to copy MTU-sized packets (1500 bytes) from CTM to EMEM, which is crucial to packet buffering.

TABLE I: sNIC Memory Hierarchy

Memory	Capacity	Usage
Code Store (CS)	8 K Instrs.	Code instructions
Local Memory (LM)	4 KB	Registers
Cluster Local Scratch (CLS)	64 KB	Local to island. Shared across multiple MEs
Cluster Target Memory (CTM)	256 KB	Local to island. Shared across multiple MEs
Internal Memory (IMEM)	4 MB	Global Memory (SRAM)
External Memory (EMEM)	8 GB	Global Memory (SRAM + DRAM)

B. Leveraging sNIC Architecture

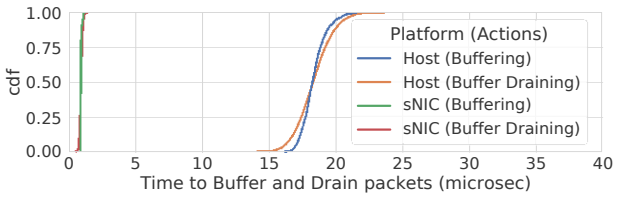


Fig. 3: Latency to Buffer/Drain Packets

1) Motivation to buffer packets in sNIC: We leverage the buffering mechanism we develop on the UPF to improve the performance of handover and paging. As explained in §II-C, the sNIC is more efficient for packet processing compared to host-based systems. However, others have taken the approach of just buffering the packets on the host while carrying out the rest of the features on a P4Switch [5]. The host has substantial memory compared to the sNIC to buffer packets. However, by buffering packets in the sNIC, we can store and drain packets much faster than would be possible when buffering on the host. We now carry out an experiment that compares the buffering and packet draining latency on the sNIC and host. This experiment uses a host-based DPDK implementation to efficiently buffer packets by holding on to the *rte_mbuf* [25]. For the sNIC implementation, we use a sNIC provided instruction (i.e., *pktdma_ctm_to_mu*) to buffer the packet by DMA'ing it to EMEM (DRAM) memory. For releasing the packet, in the host implementation, the packet is pushed to the NIC for transmission and its memory released. To drain the packet in sNIC, we use the sNIC instruction (i.e., *pktdma_mu_to_ctm*) to DMA packets from EMEM to send it over the network. Fig. 3 shows that it takes 20× the latency to individually buffer and drain packets in the host when compared to performing the same operations on the sNIC. The reason for this is that a host-buffered packet would involve DMA in and out of the host memory, traversing the PCIe bus. On the other hand, a sNIC-buffered packet only causes the packet to be DMA'd from one sNIC memory to another (e.g., CTM and EMEM). This allows us to achieve a much higher throughput for handover/paging with the sNIC, compared to processing and buffering in the host.

On the sNIC, we load a P4 match action table for PDR matching. However, the capacity of this table is only 64K entries [26], forcing us to resolve P4 table misses on the host. Other platforms such as T4P4S [27] and NetFPGA [28] that are based on other vendors have similar capacity constraints for individual P4 tables [26], especially when there is wildcard matching, as is required in the UPF. Since this limited-capacity P4 table will likely only store active flows, a DL packet for a UE device that is currently in an inactive state will most likely miss the P4 table and consequently be resolved on the host. In order to prevent this for most flows, we must have a data structure that is distinct from the P4 table of much larger capacity to maintain state for a large pool of flows, including inactive flows. This data structure to store flow state is declared and allocated in Micro-C. We refer to it as the 'flow table' while the P4 match action tables (e.g., not Micro-C) are referred to as 'P4 table'. In general, the packet will be processed by the P4 and Micro-C pipeline in the sNIC. This flow table can also help with monitoring control plane communication, which we discuss later. In order to construct the flow table, we design our own data structure and select appropriate update procedures implemented in Micro-C.

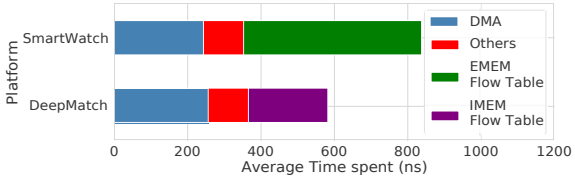


Fig. 4: Time spent to buffer packets in sNIC

2) Optimizing flow state access on sNIC: The UPF performs several state-dependent activities such as monitoring, handover, paging, and traffic shaping. Furthermore, to determine which memory location to store the packet in, and to transmit the packet, we need to track the flow state. For buffering, we draw inspiration from SmartWatch [3] and DeepMatch [14], in order to fully leverage the sNIC memory hierarchy. Since we seek to buffer packets for hundreds of UE sessions, each with at least thousands of packets within the sNIC, packets will have to be buffered in EMEM (§III-A) because of its large DRAM capacity. Next, we will evaluate the time spent to buffer packets when SmartWatch and DeepMatch maintain flow state to buffer packets in the sNIC.

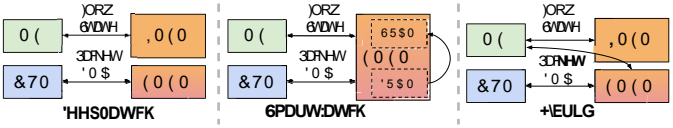


Fig. 5: Hybrid Memory Architecture

Fig. 4 shows the average delay caused by packet DMA, IMEM/EMEM flow table operations, and others (e.g., instructions, local memory) for SmartWatch and DeepMatch. It can be seen that EMEM flow table lookup contributes significantly toward SmartWatch's processing latency. This is because the

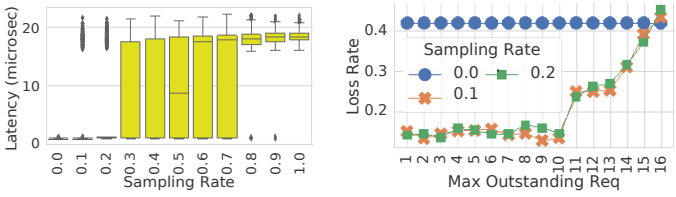
90th percentile bulk write operation on EMEM is 2.182× higher than IMEM due to the external DRAM as shown in Fig. 2. But, SmartWatch has 44× higher total capacity than DeepMatch (which only uses the IMEM). Even after allocating memory to buffer the contents of 262K 1500B packets in EMEM, we still were able to allocate a flow table of 12 million flow records on the EMEM (e.g., SmartWatch), compared to just 295K flow records in IMEM (e.g., DeepMatch).

For Synergy we propose a hybrid of these two approaches where the IMEM acts as a cache to the flow table stored in EMEM, operating under our programmatic control. We use an LRU replacement policy to evict flow records from the SRAM IMEM to the DRAM EMEM. We refer to this as our "Hybrid approach". The intuition is that, by having a cache hierarchy, we can reduce the average memory access time compared to SmartWatch, while increasing the memory capacity compared to DeepMatch. The intent is to have the working set of flows in the IMEM, so that they can be accessed faster, but retain the ability to fetch flow state from EMEM if there is a miss in the IMEM. We strive to minimize misses that require searching EMEM for flow state. Since our program explicitly places recently used flow records in IMEM that is entirely in SRAM, we do not have to rely on the sNIC's SRAM auxiliary cache's policy for the EMEM DRAM (as part of the EMEM design), since this may get polluted as we explain further below. Fig. 5 shows the memory architecture of DeepMatch, SmartWatch, and our Hybrid approach. DeepMatch stores the entire flow table in SRAM. SmartWatch stores the entire table in DRAM, but the sNIC controls what flow records are cached in SRAM. In the rest of the paper, we will refer to DeepMatch and SmartWatch's flow management techniques as "SRAM only" and "DRAM w/aux. cache", respectively. The Hybrid approach lets the programmer control the placement of flow records in SRAM and DRAM, which is adopted by Synergy for the sNIC UPF. Synergy is general and is applicable for many other sNIC architectures that provide onboard SRAM and DRAM [29]. While this flow replacement policy is novel, the lockless flow update scheme and how we DMA the state to the host have been leveraged from our previous work [3].

3) Utilizing the limited number of sNIC DMA engines: We use sampling to select whether the host NF or sNIC should buffer the packet. If it is in the sNIC, we find the appropriate memory location to buffer the packet in the sNIC EMEM. We utilize the *pktdma_ctm_to_mu* instruction provided by the Netronome sNIC to use the internal DMA to move the packet from one memory to another. On the other hand, if the packet is to be buffered on the host, we forward the packet to a dedicated virtual port to DMA to the host. A buffering service NF accesses the packet in the host memory. Buffering tasks are delegated from the sNIC to the host as needed. Even though a limited amount of buffering activity occurs on the host, the state tracking remains within the sNIC, including the trigger to release packets buffered in the host NF.

For buffering workloads that the sNIC sees a drop in throughput for large packet sizes due to a hardware limit of 16 outstanding DMA requests per CTM [14]. For packet sizes larger than 1024 bytes, we observe the EMEM DMA operation to take at least 2.15× more latency, on average, compared to processing 512 byte packet streams. Let the number of outstanding DMA requests in a CTM to the EMEM, per 1 sec measurement interval, be Θ_{ctm}. In Synergy, when $\Theta_{ctm} > 10$, we sample 10% of new buffering tasks so that they can be done on the host (i.e., for new handovers or if a device goes idle) instead of the sNIC. The sNIC maintains the required flow state, with the host just buffering the packet. With this, the DMA engines between the sNIC and host are used instead of the bottlenecked DMA engines between the CTM and EMEM. Fig. 6 provides justification for this parameter setting. As shown in Fig. 6(a), setting the sampling rate less than 0.3 ensures most packets are processed with low latency when buffered. A sampling rate of 0.3 results in 2.35× higher average latency compared to that at a sampling rate of 0.1 (at 0.1, more packets are processed in the sNIC instead of the host). Fig. 6(b) shows that setting the max. outstanding DMA requests above 10 causes the packet loss to increase by at least 1.87× because the queuing delay in the sNIC increases.

Design Summary: We introduced an efficient buffering mechanism in the sNIC UPF to be used during handover and paging. Furthermore, we have designed ways to reduce the time to access flow state which is important for speeding up state-dependent processing such as handovers, paging, tunneling, traffic shaping, and monitoring. In Synergy, monitoring is used for mobility prediction as described next.



((a)) Latency vs. Sampling Rate ((b)) Loss vs. Max outstanding DMA

Fig. 6: Parameter Setting for sNIC host load balancing

C. Integrating Mobility Prediction Models

1) Monitoring control plane events: The Location Reporting Control procedure is to allow the AMF to request the gNB node to report the UE's current location, or the UE's last known location with timestamp [15]. Since the AMF is colocated on the same host as the sNIC running the UPF in Synergy, those packets go through the sNIC before being delivered to the AMF. As location reports arrive at the sNIC, we parse the packet and update the same flow state that we had maintained for paging and handover purposes. This ensures that packets are parsed and the data structures updated with similar low processing and memory overhead as data plane packets. Since the prediction model uses location to make predictions, we use location reports. However, there are other models that use channel quality as features, such as the cause field within the "handover-required" message from source gNB to AMF (Section 4.9.1.3 in [30]), which is triggered by the source gNB based on measurement reports [31].

2) sNIC Design for Mobility Prediction: Handovers occur when the base station signal strength for a UE decreases.

This occurs often as a result of mobility. Time to complete control plane operations (e.g. mobility handoff, service establishment) directly impacts the delay experienced by end-user applications [32]. Furthermore, with 5G, the control traffic is expected to increase rapidly due to a shift to smaller cell sizes, which will likely cause more mobility handoffs [32]. Mobility prediction allows the network operator to pre-install the network state in the core networking elements, minimizing the delay experienced by end-user applications due to frequent handovers. Fortunately, vehicular mobility is a highly correlated process due to roadways, which can be effectively exploited by the gNBs-reported measurement of the radio signal strength from their connected mobile users [33].

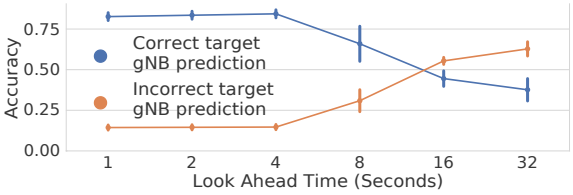


Fig. 7: Prediction Accuracy

We use a SUMO-based [34] vehicular mobility dataset [35] for subsequent experiments. For our experiments, we use the mobility predictor introduced in [36] since it is decentralized and shown to be highly accurate [33], [36]. Their technique uses neural networks to predict the next gNB by making use of the angle of arrival, which is derived from the vehicle and gNB coordinates. At each point in time, a vehicle (UE) connects with the gNB providing the best communication conditions, measured in terms of path loss [36].

As vehicles move, the UE attaches and detaches to various gNBs. We call the gNB the UE is attached to as the serving gNB. Throughout the duration when the UE is attached to a serving gNB, we collect UE location samples. All these collected location samples are used as a feature vector to determine the next serving gNB. The predictor uses a Gated Recurrent Unit-based Recurrent Neural Network (RNN) and returns a soft prediction, in the form of a probability vector for the next serving gNB. The prediction accuracy improves as the UE moves closer to get to the next gNB and as more samples are fed to the RNN model [37]. Fig. 7 shows the rate of correct and incorrect mobility predictions for the target gNB in the simulation, where the error bars represent the standard deviation observed across gNBs. Predictions made 1 to 5 sec prior to a handover have the highest accuracy. Correct predictions will make sure that the handover is accelerated due to the prepopulated state. Fewer mispredictions will ensure less compute and memory resources are wasted in the 5GC. If we make a prediction every 5 sec, the number of predictions is manageable and the accuracy is reasonably high. Unfortunately, in real-time, we do not know whether the UE is 5 sec or more seconds away from handover. Therefore, we will have to expire the predictions every 5 sec, increasing the memory and processing overhead as more flows are programmed. Therefore, we have a probabilistic data structure on the sNIC to ensure that we minimize these memory operations.

Fig. 8: Bloom Filter Architecture

3) sNIC Prediction Table Lookup Optimizations: We take the N2Handover codebase [38] and divide it into two phases corresponding to prediction and handover. The prediction phase is primarily concerned with pre-populating state to accelerate the end-to-end handover latency while the handover phase is carried out as the final step when the UE actually moves to the new gNB. When a DL packet arrives at a UPF, destined to a UE that does not have the required state to process the packet, the UPF in Synergy is responsible for buffering the packet. The packet is buffered until the state is propagated across the control-plane NFs and then all the buffered packets are drained and forwarded to the UE.

Next, we try to minimize the overhead related to checking for mobility predictions. As each EMEM access for prediction table lookup can take as much as 416 ns, this will severely degrade throughput. This is because predictions will likely reside in EMEM DRAM as they have not been accessed before, precluding the sNIC from caching the prediction in EMEM SRAM. Furthermore, we do not explicitly store the prediction in IMEM as the volume of predictions can be high. To solve this problem we use lightweight Bloom Filters [39] hosted on the CLS for prediction. We store the prediction in CLS because it is at least 3× to 10× faster than accessing IMEM and EMEM (see Fig 2), respectively. A Bloom filter is a data structure designed to determine, rapidly and memory-efficiently, whether an element, in this case a prediction, is present in a set. We use 8 parallel Bloom Filters, for predictions made every 625 millisec, that are arranged in memory such that one *mem_read8* (e.g., 1 Byte) can access all parallel Bloom Filters at the same hash index. Fig. 8 shows the code running in the ME to determine whether to check handover predictions for a UE when we miss on the EMEM and IMEM for its flow record. Although Bloom Filters have some false positives, our design seeks to minimize the penalty, and the space savings outweighs this drawback [40].

On replaying the SUMO trace [35] on our testbed, we see on average 22.34% packets every second, missing on the IMEM and EMEM flow table. Those packets are checked against the Bloom Filter before visiting the prediction table. The Bloom Filter is non-invertible, meaning we cannot retrieve the prediction from the Bloom Filter. There are three outcomes: 1) the prediction is found in the Bloom Filter, and then also found in the prediction table (True positive), 2) the prediction is found in the Bloom Filter, but not found in the prediction table (False positive), and 3) the prediction is not found in the Bloom Filter (True negative). In the case of true positives, the required state to process the packet in the sNIC is retrieved from the prediction table. As a consequence, the packet gets processed

in the sNIC, not suffering the long latency processing costs of the host. In the case of true negative, no further cycles are wasted searching the prediction table (416-1666 ns) once the Bloom Filter lookup completes. Lastly, false positives that occur for less than 1.5% of packets that visit the Bloom Filter result in wasted cycles searching the prediction table without retrieving the required state for processing the packet. In the event of true negatives and false positives, the packet will have to be forwarded to the host for processing.

D. Bringing it all together

1) Overall: In this section, we describe the Synergy architectural design for the 5G UPF within the sNIC (shown in Fig 9) to speed up the PDR lookup, allow for dynamic adaptation of the packet processing pipeline, and ensure that the mobility predictions do not impede processing throughput. A packet first arrives at the network-bound interface (NBI) ingress and is transferred to the CTM memory. The packet is served by one of the packet processing MEs in a run-to-completion manner. This processing includes both the P4 pipeline and additional Micro-C program(s). We adapt the Hybrid approach introduced in §III-B2 for updating flow state to achieve the highest throughput. The IMEM hosts the cache of the flow table while the EMEM memory hosts the flow table and prediction table. The per-island CLS hosts the Bloom Filter that is used for the very efficient probabilistic check if there are any pre-populated flow records based on mobility prediction. The EMEM also hosts the buffer where packets will be stored when a UE is idle or has an ongoing handoff. Once the packet is processed, it will be sent out of NBI egress.

2) Minimizing Handover Latency: Handover latency directly influences the end-user experience. To reduce it, we carry out mobility prediction and prepopulate the state within the control plane NFs and UPF. We evaluate the benefit of this in §IV-F. To achieve this, the host-sNIC interface incorporates several features: 1) ring buffers in the EMEM can export flow records to the host; 2) the host updates P4 table entries and data structures hosted in EMEM including the prediction table; 3) the sNIC can forward packets to the host using an SR-IOV virtual port. As described in §III-C1 we monitor control plane packets and update the flow state with the UE location information within the sNIC. This efficiently gets exported to the host via DMA. The tight coupling between the sNIC and host ensures that we provide the features to the prediction model running on the host quickly. The host programs the sNIC to update the P4 pipeline for handling packets of different sessions. By having the sNIC be just a PCI transaction away, we are able to push rules to the sNIC at lower latency. This allows us to reduce handover delays (see §IV-F) Lastly, for situations where Synergy cannot fully process the packet within the sNIC, the packet is forwarded to the host with minimal overhead instead of over the local data center network.

3) Speedup Flow Lookup to improve throughput: Slow flow lookups directly reduce the throughput of the UPF, despite the parallelism offered by the 80 MEs, each with 8 thread contexts. Slower flow lookups as a result of the higher memory access latency, are difficult to overcome even by switching to another thread's context causing the drop in throughput [14]. Furthermore, the capacity of the flow table has to be high otherwise flow table misses will also lead to long latencies because of host processing. We evaluate the benefit of the design options in §IV-A. Synergy is publicly available at [8].

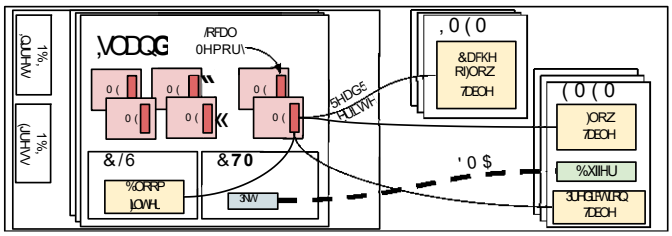


Fig. 9: Synergy Architecture

Fig. 10 shows the logical packet processing pipeline. When a packet arrives, we first find its flow state by computing a hash index and searching buckets at that hash index in IMEM and then EMEM memories. If we find the flow in IMEM (lower latency), we update the flow state and fetch the packet metadata as necessary. This may include state indicating the UE is idle or in an ongoing handover, requiring buffering. If we miss on the IMEM, we check the EMEM, where if we find the flow record, we swap the least recently used flow record in IMEM (LRU) with the flow that we hit in the EMEM. If the packet is to be buffered, we DMA the packet payload to EMEM. Next, we perform wildcard matching in the P4 table based on the Service Data Flow traffic filter (e.g., source-destination IP or port), UE IP, and TEID and then execute the corresponding action. If there is no P4 table hit, we check if there is a simple action to be carried out based on packet metadata (e.g., tunneling without wildcard matching). If yes, we carry out the action and clone the packet to the host so that it can subsequently update the P4 match action table; otherwise, we simply forward the packet to the host and let the UPF on the host process the packet. The P4 table also specifies meters to configure the average rate, burst size, and policing of the traffic. This is configured dynamically by the operator during runtime and executed by the target sNIC. If the traffic exceeds the configured rate and buffers overflow, we utilize the function 'netro *meter_drop_red* [41]' provided by the Netronome sNIC to drop traffic exceeding the limits.

4) Deployment Strategy: We define a 5GC instance as one complete set of control plane NFs and the UPF that can fully-process uplink and downlink packets. We ensure that the affinity [42] of the control plane NFs and UPF for a 5GC instance are accounted for in placing them on the same host server (node). There may be multiple 5GC instances within a data center to handle increased traffic loads. In Synergy, the load is balanced by assigning new UE sessions to the appropriate 5GC instance by an orchestrator. A UEs session is assigned to a 5GC instance for the period the UE maintains its attachment to the 5GC. Thus, the state does not have to be moved between 5GC instances. We believe that the number of

CPU cores available on the node supporting the 5GC instance is sufficient to handle the control plane load from the set of UEs generating the dataplane load handled by the sNIC. If the control plane load does go up, vertical scaling by adding CPU cores for a particular overloaded control plane NF can relieve the bottleneck at the NF. This deployment strategy is similar to that of L25GC [12].

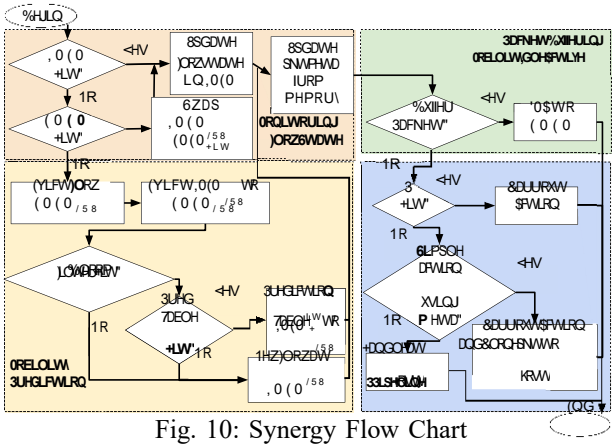


Fig. 10: Synergy Flow Chart

5) Security: As the node may support other third-party NFs, we seek to provide isolation between different groups of mutually trusting NFs. We use the notion of security domains introduced in NetVM [43]. Host NFs developed by the same vendor are allowed to share a private memory pool. But, that cannot be accessed by a different application on the same node. A DPDK primary process creates a private shared memory pool with an associated distinct file prefix, implemented as hugepages in the Linux file system. Each security domain uses the file prefix for the huge page it access to, which is provided to it by the primary DPDK process [44]. Location reports are encrypted to maintain user privacy. The crypto module of our target sNIC device [45] decrypts these location reports. The required keys are provided to the sNIC by the NF that is co-located on the same host.

6) Limitations: Due to Synergy's buffering and monitoring capabilities, the throughput achieved with Synergy is not as high as what is achievable with P4Switch [5]. Currently, the orchestrator in Synergy load balances UE sessions over 5GC instances. The load on the control plane NFs is likely dominated by the number of UE sessions. The load on the UPF potentially needs to consider the UE flow characteristics, but we anticipate a conservative allocation would help avoid the UPF from being overloaded. This is left as future work.

7) General applicability of sNIC design: Here we study the generality of our UPF implementation on the Netronome Agilio LX sNIC [46], and the potential for adoption with other sNICs, such as the Bluefield MBF1L516A ESNAT and LiquidIO OCTEON TX2DPU sNICs. All three sNICs have a multi-core architecture. Although the Bluefield and LiquidIO sNICs have fewer CPU cores (e.g., MEs) compared to the Netronome Agilio LX, they operate at a faster clock rate (at least 2.2 GHz instead of 1.2 GHz for Netronome Agilio LX). All their memory architectures have three levels. In the Netronome sNIC the three levels correspond to CLS, then EMEM SRAM along with IMEM, and the last-level being the EMEM DRAM. The last level cache and L1 cache have similar access times for all three architectures. The L2 access time is 25.6 ns in Bluefield vs. at least 50 ns with Netronome and LiquidIO. Atomic primitives are supported in all architectures along with programmability using GNU in Bluefield, GCC in Liquidio, and Micro C/P4 in Netronome [3], [29]. Given the similarity of the architectures and capabilities, Synergy should in principle be portable to any of the other sNICs.

IV. Evaluation

Testbed: We evaluate the effectiveness of Synergy on our local testbed consisting of Linux servers (kernel 4.4.0-142), each with 10 Intel Xeon 2.20GHz CPU cores, 256GB memory, and Netronome Agilio LX 2×40 GbE sNICs with 8GB DDR3 memory and 96 highly threaded flow processing cores.

Traces: We use two traces. A real-world trace [47] and a SUMO-based vehicular mobility dataset [35]. The real-world 5G trace dataset is collected from an Irish mobile operator. This dataset contains timestamps, coordinates, gNB id, bitrate, and channel quality indicator. The dataset is collected with a UE streaming videos and downloading files with the user in a vehicle driving on city streets. The second is a dataset with 700k vehicle trips across 247 gNBs. It provides the gNB, timestamp, and vehicle coordinates. At each point in time, the UE connects with the gNB providing the best communication conditions, measured in terms of path loss [36].

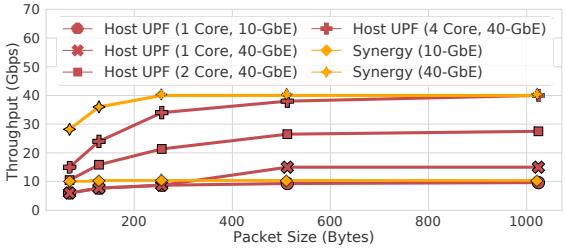


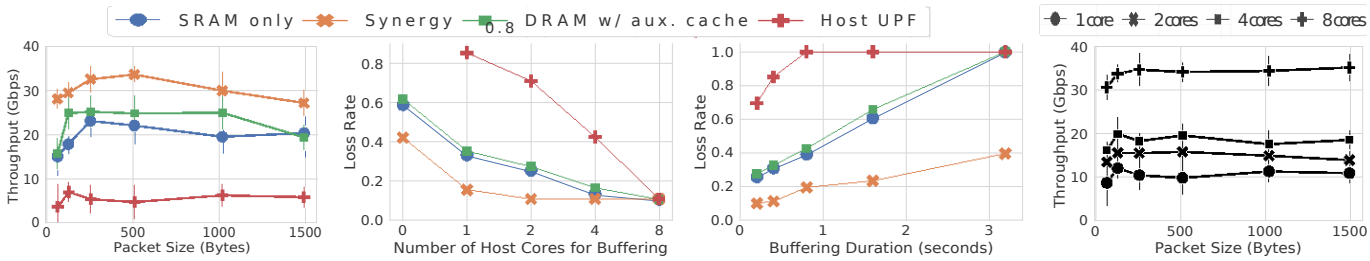
Fig. 11: Tunneling Throughput

A. Benefit of Network Acceleration

Fig 11 shows the throughput achieved for PDR matching and finding the associated FAR rules for DL packets (i.e., finding and inserting GTP [48] tunnel header) using the trace introduced in [47]. Here we compare the throughput between Synergy and a host UPF implemented in DPDK. We see the host achieves lower throughput with small packet sizes because of the overhead of processing higher number of packets. For large packet sizes the host can match Synergy's throughput only after dedicating four CPU cores to it. Otherwise, the throughput with one core is 3.85× lower on average than Synergy. Therefore, Synergy significantly reduces the number of CPU cores required for the primary 5G UPF tasks.

B. Interference of Buffering on sNIC Flow State Access

The faster the flow state can be retrieved and updated, the higher is the achievable packet processing rate. Here we show why Synergy achieves the lowest latency in comparison to "SRAM only" and "DRAM w/aux. cache" alternatives (see



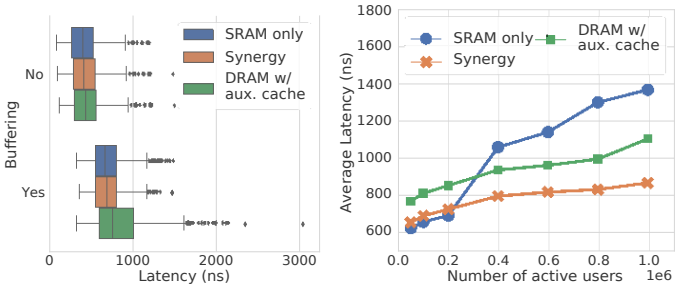
((a)) Standalone buffering in sNIC ((b)) Buffering w/ sNIC-host vs. cores ((c)) sNIC-host vs. buffering-duration ((d)) sNIC UPF, all buffering at host

Fig. 12: Throughput & loss rate w/ UPF buffering strategies. Sensitivity measured on pkt size, host cores, and buffering duration

III-B2). Recall "SRAM only" and "DRAM w/aux. cache" are derived from the SmartWatch and DeepMatch designs, respectively. For the host UPF, we use Free5GC [2], with UPF implemented on top of DPDK [21]. We first evaluate the packet latency observed for packet buffering with a workload having up to 200K flows, which is below the total SRAM size in all three alternatives. We try two variants, one with buffering (e.g., DMA to EMEM DRAM) enabled and the other with it disabled. As shown in Fig. 13(a), with buffering at the sNIC, Synergy has a similar performance as "SRAM only" because the small number of flows fit in IMEM. On the other hand, compared to "DRAM w/aux. cache", the 99 percentile latency is 1.38× lower in Synergy's approach when buffering is enabled. This is because in Synergy the DMA of packet payload from CTM to EMEM does not pollute the flow table stored in SRAM. For Synergy the programmer controls the flow records in IMEM SRAM while in "DRAM w/aux. cache", the sNIC control the flow records stored in the SRAM cache of EMEM, making it vulnerable to cache pollution. With buffering disabled, "DRAM w/aux. cache" and Synergy perform the same as there is no cache pollution caused by DMA operations. Next, we increase the number of active users by manipulating the trace. The average latency to buffer packets is shown in Fig. 13(b). The "SRAM only" approach has higher latency when the number of active users exceeds IMEM capacity as it has a lot of misses due to its limited capacity of flow records. Synergy achieves at least 15% lower access time compared to "DRAM w/aux. cache" due to more memory accessed from IMEM SRAM instead of the EMEM DRAM. Synergy's approach of programmatic flow record placement along with the large capacity DRAM ensures it is the most scalable among the three approaches.

C. Standalone host and sNIC buffering

In this section, we first evaluate the attainable throughput with Synergy. Fig. 12(a) shows that Synergy's throughput is, on average, 1.38× and 1.57× higher compared to "DRAM w/aux. cache" and "SRAM only", respectively. This is because of the lower memory access time for flow records (see §IV-B). We observe the throughput to first increase with packet size and then decrease beyond 1024 bytes per-packet size. Smaller packets are penalized more by the additional per-packet processing needed to update and maintain flow state [14]. On the other hand, when the packet size is larger than 1024 bytes we see a drop in throughput as occupancy of the DMA engines becomes the bottleneck (see III-B3).



((a)) Latency Distribution ((b)) Latency wrt users

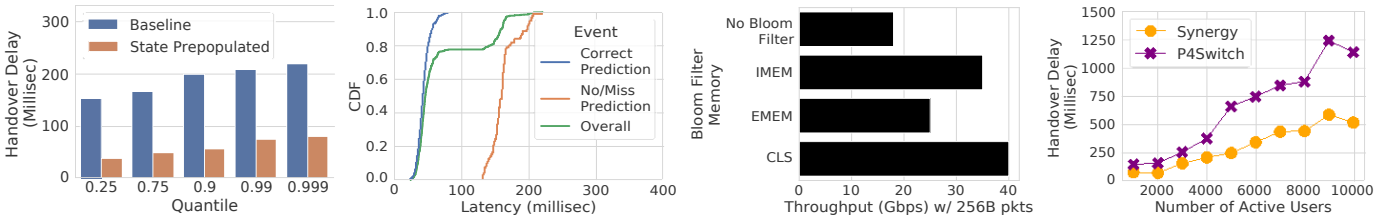
Fig. 13: Buffering Performance

Fig. 12(d) shows the throughput using the Synergy approach, but instead of buffering packets in the sNIC, they are buffered in a microservice in the host with variable number of CPU cores. A similar (although not using the sNIC) approach was adopted in a P4Switch-based UPF [5]. However, as seen in Fig. 12(d), even after performing the packet processing task on the sNIC and just buffering on the host, it requires 8 CPU cores to yield comparable throughput as Synergy (see Fig. 12(a)). The lack of network acceleration and having to DMA packets to host memory across the sNIC-host PCIe bus as opposed to between sNIC memories leads to poor throughput even with four host CPU cores. However, the throughput with packet size 1024 Bytes is not high enough on the sNIC (see Fig. 12(a)). Due to the limitations mentioned above, we introduced the sNIC host combination and load balancing the buffering task between them as described in §III-B3.

D. Buffering Performance on Synergy

Now we show the packet loss observed for buffering during handover events when using a real-world trace [47]. The sNIC runs a UPF, including the buffering function. We also implement a DPDK program to carry out some buffering tasks in the host as explained in §III-B3. Fig. 12(b) shows the packet loss rate during handovers while varying the number of host CPU cores, implementing just the buffering functionality, with a testbed consisting of a sNIC UPF (see §III-B3). With 2 CPU cores, Synergy achieves the same loss rate as "SRAM only" and "DRAM w/aux. cache" when they use 4 CPU cores. It does so because of lower flow state access latency (see §IV-B). Therefore, Synergy's packet processing pipeline requires about half the number of CPU cores as "SRAM only" and "DRAM w/aux. cache" or one-fourth the number of UPF host cores (e.g., PDR matching, buffering, etc) to get a similar loss rate.

Next, we show the packet loss rate vs. the buffering duration using the case of having two CPU cores running a DPDK



((a)) Speedup via state prepopulation ((b)) Synergy latency distribution ((c)) Throughput Analysis ((d)) sNIC/P4Switch P4-based UPF

Fig. 14: Impact of mobility prediction, fast programmability, and sNIC memory hierarchy on handover performance

program to buffer packets. The remaining UPF processing is done by the sNIC. Common control plane events such as handover and the paging cycle [49] take 135 ms and 1.28 seconds, respectively. The paging cycle is the duration after which the UE wakes up from idle mode to read the paging messages. Varying the buffering duration will cause different number of packets in the UPF to be buffered, increasing the overhead. Fig. 12(c) shows that Synergy is the most tolerant with respect to the duration of buffering because of lower access time and higher capacity, having the lowest loss rate.

E. Handover Performance

First, we show the control plane handover latency when the state is prepopulated by mobility prediction in all the control plane NFs vs. baseline latency for the handover procedure without any mobility prediction. This has been measured on Free5GC with DPDK [2] 5GC implementation. We observe that prepopulating state through mobility prediction provides an average speedup of 2.73×. Mispredictions are equivalent to no prediction, as the state will not be populated using the correct target gNB. This would cause the process to have to go through the entire control plane handover procedure. For this experiment, we used a vehicular mobility trace [35]. Fig. 14(b) demonstrates that by utilizing handover prediction, we can achieve 3.78× lower median handover latency for correctly predicted mobility events as opposed to mispredictions and no predictions (i.e., baseline). Considering all mobility events, including mispredictions and correction predictions (i.e., Overall curve in Fig. 14(b)), we see a 3.49× lower median handover latency compared to baseline.

Lastly, we evaluate the throughput achieved with and without the Bloom Filter optimizations running in the sNIC against a 256 byte packet stream. Fig. 14(c) shows the throughput for various Bloom Filter allocation strategies. When we do not allocate a Bloom Filter, the sNIC looks up the prediction table for each and every packet that misses on EMEM and IMEM flow table, causing the throughput to drop to less than 18 Gbps. However, with the Bloom Filter, we check the prediction table only when the prediction is found in the Bloom Filter. By allocating the Bloom Filter in EMEM or IMEM the throughput drops by at least 5 Gbps compared to line rate. This is because in either case, significant cycles are still spent looking up the Bloom Filter in IMEM and EMEM respectively (Fig. 2). Finally, by allocating the Bloom Filter in CLS, as is done in Synergy, we achieve line rate for this experiment. This is because the read-accesses of the Bloom Filter allocated in CLS are at least 3× to 10× faster than accessing IMEM and EMEM (Fig. 2). However, since the CLS memory is local to each island, we must replicate the Bloom Filter in each island as packets of a flow can be processed in any island. But this longer Bloom Filter update procedure overhead does not fall in the packet datapath and does not degrade throughput.

F. Programming Overheads

Finally, we evaluate the impact of the overhead of programming the sNIC on end-user handover experience. The longer the host takes to push rules into the sNIC in response to a handover event, the longer packets will have to be buffered before they can be forwarded. This contributes to handover delay. We compare Synergy against the P4Switch approach. Both P4Switch and sNIC-based UPF platforms have P4 tables and data structures that will have to be updated by the host. To emulate the P4Switch, we consider a limit of 1200 new flows per second, as in [22], using the same values for P4 table and rule updates as with the sNIC UPF. Fig. 14(d) shows the handover delay with respect to the number of active users in a SUMO-based vehicular mobility trace [35]. We observe that Synergy attains 2.11× lower handover delay on average. This is because the sNIC, according to our experiments, can yield up to 6.6× higher programming rate as the control plane NFs and the UPF are colocated on the same host.

V. Conclusion

Synergy is a SmartNIC-based UPF, a key 5GC component, that leverages the tight coupling of the SmartNIC and the host. It provides network acceleration, programmability, and monitoring for mobility prediction. Mobility and paging events have much better performance because a majority of packets are buffered within the SmartNIC, outperforming host and programmable switch-based approaches in terms of latency and packet loss. This is in part due to Synergy's two-level structure for flow table maintenance, which increases the scale while reducing latency. Synergy further reduces handover latency by pre-populating state in the control plane NFs. This is done by monitoring control plane traffic that flows to control plane NFs colocated on the same node. For mobility prediction, we maintain a Bloom Filter to judiciously access the prediction table, increasing the packet processing rate in the SmartNIC UPF. Efficient programming of SmartNIC flow tables allows us to manipulate actions and associated priorities rapidly, reducing the handover delay.

VI. Acknowledgement

We sincerely thank the US NSF for their generous support through grants CRI-1823270 and CSR-1763929. We thank our shepherd Dr. Arvind Narayanan and the anonymous reviewers for their suggestions and comments.

REFERENCES

[1] "Our high-performing core network." [Online]. Available: https://www.iplook.com/products/5gc-upf

[2] "free5gc." [Online]. Available: https://www.free5gc.org/

[3] S. Panda, Y. Feng, S. G. Kulkarni, K. K. Ramakrishnan, N. Duffield, and L. N. Bhuyan, "Smartwatch: Accurate traffic analysis and flow-state tracking for intrusion prevention using smartnics," in *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 60–75. [Online]. Available: https://doi.org/10.1145/3485983.3494861

[4] B. W. M. F. A. S. R. H. O. C. Mitziu Echeverria, Zeeshan Ahmed, "Phoenix: Device-centric cellular network protocol monitoring using runtime verification," in *The Network and Distributed System Security Symposium (NDSS)*. Springer, 2021.

[5] R. MacDavid, C. Cascone, P. Lin, B. Padmanabhan, A. ThakuR, L. Peterson, J. Rexford, and O. Sunay, *A P4-Based 5G User Plane Function*. New York, NY, USA: Association for Computing Machinery, 2021, p. 162–168. [Online]. Available: https://doi.org/10.1145/3482898.3483358

[6] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 15–28. [Online]. Available: https://doi.org/10.1145/3098822.3098824

[7] "P4 programming language." [Online]. Available: https://p4.org/

[8] "Synergy opensource code." [Online]. Available: https://github.com/spand009/Synergy

[9] "3gpp ts23.501 section 4.2: Architecture reference model." [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144

[10] "3gpp ts23.502: System architecture for the 5g system (5gs)." [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3145

[11] "5g nr 3gpp." [Online]. Available: https://www.etsi.org/deliver/etsi_ts/138300_138399/138300/15.08.00_60/ts_138300v150800p.pdf

[12] V. Jain, H.-T. Chu, S. Qi, C.-A. Lee, H.-C. Chang, C.-Y. Hsieh, K. Ramakrishnan, and J.-C. Chen, "L25gc: A low latency 5g core network based on high-performance nfv platforms," ser. SIGCOMM '22, 2022.

[13] A. Mohammadkhan, K. K. Ramakrishnan, and V. A. Jain, "Cleang—improving the architecture and protocols for future cellular networks with nfv," *IEEE/ACM Transactions on Networking*, vol. 28, no. 6, pp. 2559–2572, 2020.

[14] J. Hypolite, J. Sonchack, S. Hershkop, N. Dautenhahn, A. DeHon, and J. M. Smith, "Deepmatch: Practical deep packet inspection in the data plane using network processors," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*. New York, NY, USA: Association for Computing Machinery, 2020, p. 336–350. [Online]. Available: https://doi.org/10.1145/3386367.3431290

[15] "Ng application protocol." [Online]. Available: https://www.etsi.org/deliver/etsi_ts/138400_138499/138413/15.00.00_60/ts_138413v150000p.pdf

[16] J. Zhao, S. Ni, L. Yang, Z. Zhang, Y. Gong, and X. You, "Multiband cooperation for 5g hetnets: A promising network paradigm," *IEEE Vehicular Technology Magazine*, vol. 14, no. 4, pp. 85–93, 2019.

[17] 3GPP, "Packet detection rule specification," https://www.etsi.org/deliver/etsi_ts/129200_129299/129244/15.05.00_60/ts_129244v150500p.pdf, 2021, [ONLINE].

[18] ——, "LTE; 5G; Interface between the Control Plane and the User Plane nodes," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 29.244, 09 2019, version 15.7.0.

[19] "How do 5g small cells work and where are they located?" [Online]. Available: https://www.lifewire.com/5g-cell-towers-4584192

[20] "Control plane messages." [Online]. Available: https://www.etsi.org/deliver/etsi_ts/129200_129299/129244/15.05.00_60/ts_129244v150500p.pdf

[21] L. Foundation, "Data plane development kit (DPDK)," 2015. [Online]. Available: http://www.dpdk.org

[22] J. Xing, Q. Kang, and A. Chen, "NetWarden: Mitigating network covert channels while preserving performance," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 2039–2056. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/xing

[23] A. Bose, D. Maji, P. Agarwal, N. Unhale, R. Shah, and M. Vutukuru, "Leveraging programmable dataplanes for a high performance 5g user plane function," in *5th Asia-Pacific Workshop on Networking (APNet 2021)*, ser. APNet 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 57–64. [Online]. Available: https://doi.org/10.1145/3469393.3469400

[24] Y. Xie and K. Jamieson, "Ng-scope: Fine-grained telemetry for nextg cellular networks," *CoRR*, vol. abs/2201.05281, 2022. [Online]. Available: https://arxiv.org/abs/2201.05281

[25] "Rte mbuf." [Online]. Available: https://doc.dpdk.org/api/rte__mbuf_8h.html

[26] H. Harkous, M. He, M. Jarschel, R. Pries, E. Mansour, and W. Kellerer, "Performance study of p4 programmable devices: Flow scalability and rule update responsiveness," in *2021 IFIP Networking Conference (IFIP Networking)*, 2021, pp. 1–6.

[27] P. Vörös, D. Horpácsi, R. Kitlei, D. Leskó, M. Tejfel, and S. Laki, "T4p4s: A target-independent compiler for protocol-independent packet processors," in *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, 2018, pp. 1–8.

[28] "Netfpga-sume." [Online]. Available: https://www.xilinx.com/products/boards-and-kits/1-6ogkf5.html

[29] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta, "Offloading distributed applications onto smartnics using ipipe," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 318–333.

[30] "Handover required 3gpp." [Online]. Available: https://www.etsi.org/deliver/etsi_ts/123500_123599/123502/15.02.00_60/ts_123502v150200p.pdf

[31] "Measurement reports." [Online]. Available: https://www.etsi.org/deliver/etsi_ts/138100_138199/138133/15.02.00\60/ts_138133v150200p.pdf

[32] M. Ahmad, S. U. Jafri, A. Ikram, W. N. A. Qasmi, M. A. Nawazish, Z. A. Uzmi, and Z. A. Qazi, "A low latency and consistent cellular control plane," 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*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 648–661. [Online]. Available: https://doi.org/10.1145/3387514.3406218

[33] F. Meneghello, D. Cecchinato, and M. Rossi, "Mobility prediction via sequential learning for 5g mobile networks," in *2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2020, pp. 1–6.

[34] "Simulation of urban mobility." [Online]. Available: https://www.eclipse.org/sumo/

[35] "Vehicular mobility trace." [Online]. Available: http://kolntrace.project.citi-lab.fr/

[36] I. Labriji, F. Meneghello, D. Cecchinato, S. Sesia, E. Perraud, E. C. Strinati, and M. Rossi, "Mobility aware and dynamic migration of mec services for the internet of vehicles," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 570–584, 2021.

[37] M. Abdel-Nasser and K. Mahmoud, "Accurate photovoltaic power forecasting models using deep lstm-rnn," *Neural Comput. Appl.*, vol. 31, no. 7, p. 2727–2740, jul 2019. [Online]. Available: https://doi.org/10.1007/s00521-017-3225-z

[38] "Handover code free5gc." [Online]. Available: https://github.com/free5gc/free5gc/blob/main/test/registration_test.go#L1717

[39] L. L. Gremillion, "Designing a bloom filter for differential file access," *Commun. ACM*, vol. 25, no. 9, p. 600–604, sep 1982. [Online]. Available: https://doi.org/10.1145/358628.358632

[40] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485 – 509, 2003. [Online]. Available: https://doi.org/

[41] "Netronome metering." [Online]. Available: https://github.com/open-nfpsw/meters_lab/blob/master/meter_lab.p4#L114

[42] S. Panda, K. K. Ramakrishnan, and L. N. Bhuyan, "pmach: Power and migration aware container scheduling," in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, 2021, pp. 1–12.

[43] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: High performance and flexible networking using virtualization on commodity platforms," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Apr. 2014, pp. 445–458. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/hwang
[44] "Dpdk multi processor support." [Online]. Available: https://doc.dpdk.org/guides/prog_guide/multi_proc_support.html
[45] "Crypto module nfp." [Online]. Available: https://github.com/Netronome/nic-firmware
[46] "Agilio lx 2x40gbe smartnic." [Online]. Available: https://www.netronome.com/media/documents/PB_Agilio_LX_2x40GbE-7-20.pdf
[47] D. Raca, D. Leahy, C. J. Sreenan, and J. J. Quinlan, "Beyond throughput, the next generation: A 5g dataset with channel and context metrics," in *Proceedings of the 11th ACM Multimedia Systems Conference*, ser. MMSys '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 303–308. [Online]. Available: https://doi.org/10.1145/3339825.3394938
[48] "Gtp v1 header." [Online]. Available: https://www.etsi.org/deliver/etsi_ts/129200_129299/129281/15.03.00_60/ts_129281v150300p.pdf
[49] "Default paging cycle." [Online]. Available: http://www.techtrained.com/paging-procedure-lte/