UNIVERSITY OF CALIFORNIA
RIVERSIDE

Network Monitoring with SmartNICs in Data Centers and 5G Cellular Networks

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Sourav Panda

March 2023

Acknowledgments

I would like to take this opportunity to thank everyone who helped me throughout this journey and complete this dissertation. I would never have been able to finish it without the guidance of my committee members, help from friends, and support from my family.

First, I would like to express my deepest gratitude to my advisor, Prof. K. K. Ramakrishnan and Prof. Laxmi Bhuyan for their excellent guidance and support during the course of my PhD. The completion of this dissertation would not have been possible without his mentorship and invaluable insights. I would like to thank all the members of my dissertation committee: Prof. Nael B. Abu-Ghazaleh and Prof. Jiasi Chen for their insightful comments. I would also like to thank Prof. Yingbo Hua for their helpful directions as members of my PhD candidacy committee.

I would like to thank all the researchers that I was fortunate enough to collaborate with in various projects during my PhD: Dr Ofir Weisse (Google) and Prof. Sameer G. Kulkarni (IIT Gandhinagar). I would like to thank my family. Without their unwavering support, this journey would not have been possible.

This dissertation includes content published in the following proceedings (those marked with ‘**’ are described thoroughly while those marked with ‘*’ are mentioned briefly in this dissertation):

1. Sourav Panda, Yixiao Feng, Sameer G Kulkarni, K. K. Ramakrishnan, Nick Duffield, Laxmi Bhuyan. SmartWatch: Accurate Traffic Analysis and Flow-state Tracking for Intrusion Prevention using SmartNICs. The 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT), December 2021.**

2. Sourav Panda, K. K. Ramakrishnan, Laxmi N. Bhuyan. pMACH: Power and Migration Aware Container scHeduling. The 29th IEEE International Conference on Network Protocols (ICNP), November 2021.**

3. Sourav Panda, K. K. Ramakrishnan, Laxmi N. Bhuyan, Synergy: A SmartNIC Accelerated 5G Dataplane and Monitor for Mobility Prediction. The 30th IEEE International Conference on Network Protocols (ICNP), November 2022.**

4. Weiwu Pang, Sourav Panda, Jehangir Amjad, Christophe Diot, Ramesh Govindan. 2022. CloudCluster: Unearthing the Functional Structure of a Cloud Service. In Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI), April 2022.*

5. Vivek Jain, Sourav Panda, Shixiong Qi and K. K. Ramakrishnan. Evolving to 6G: Improving the Cellular Core to lower control and data plane latency. 1st International Conference on 6G Networking (6GNet), July 2022.*

6. Yixiao Feng, Sourav Panda, Sameer G. Kulkarni, K. K. Ramakrishnan, and Nick Duffield. 2020. A SmartNIC Accelerated Monitoring Platform for In-band Network Telemetry. In Proceedings of the 26th International Symposium on Local and Metropolitan Area Networks (LANMAN '20).*

7. Ali Mohammadkhan, Sourav Panda, Sameer G Kulkarni, K. K. Ramakrishnan, Laxmi N. Bhuyan. P4NFV: P4 Enabled NFV Systems with SmartNICs. The 5th IEEE Conference on Network Function Virtualization & Software Defined Networks, 2019.*

To my parents for all the support.

ABSTRACT OF THE DISSERTATION

Network Monitoring with SmartNICs in Data Centers and 5G Cellular Networks

by

Sourav Panda

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, March 2023
Professor K. K. Ramakrishnan, Co-Chairperson
Professor Prof. Laxmi Bhuyan, Co-Chairperson

The performance and security of modern data centers and 5G networks are critical for meeting application's service-level objectives and quality-of-experience. Careful scheduling and resource allocation are needed to minimize power consumption, enable efficient cellular handovers, and avoid performance degradation. Network telemetry can help identify potential security breaches and protect against cyber attacks by detecting anomalies in traffic patterns, suspicious activities, or unauthorized accesses. Monitoring the network helps with both performance optimization and security. However, the resource utilization (e.g., CPU and memory of a monitoring host) for monitoring is a challenge. This dissertation presents designs that use heterogeneous computing infrastructure, including SmartNICs and network switches, to optimize the performance and security of data centers and 5G cellular networks. First, we design SmartWatch, a monitoring platform deployed on SmartNICs to scalably and accurately detect network traffic anomalies with little processing overhead. We then build on the SmartWatch traffic monitoring design to optimize performance and manage power consumption in data centers with pMACH, and capture traffic characteris-

tics in 5G cellular networks and predict mobility patterns, in Synergy. Finally, we develop 5GDMON, a distributed monitoring system to analyze traffic from multiple vantage points in an open radio access network (O-RAN) infrastructure of the evolving software-based 5G network.

The first chapter addresses the challenge of detecting low and slow attacks in networks. Traditional traffic queries deployed on network switches are limited by hardware constraints, leading to undetected attacks during high traffic volumes. SmartWatch proposes a flow-state tracking and flow logging system that leverages SmartNICs to detect stealthy attacks in real-time. SmartWatch's yields 2.39 times better detection rate compared to existing platforms deployed on programmable switches. SmartWatch can detect covert timing channels and perform website fingerprinting more efficiently compared to standalone programmable switch solutions, relieving switch memory and control-plane processor resources. Compared to host-based approaches, SmartWatch can reduce the packet processing latency by 72.32%. Our subsequent work, namely pMACH, Synergy, and 5DG-Mon, builds upon the SmartNIC's packet processing pipeline proposed in SmartWatch.

The second chapter proposes pMACH, a distributed container scheduling system that optimizes power consumption and task completion time in data centers. pMACH leverages affinity between application components for placement-decisions to minimize communication overheads and latency. pMACH extends SmartWatch's monitoring capabilities in the data center to capture cloud-application communication-patterns and uses it towards making better task placement decisions. It proposes in-network monitoring using SmartNICs to measure communications and perform scheduling in a hierarchical, parallelized framework. Both testbed measurements and large-scale trace-driven simulations show that pMACH saves at least 13.44% more power compared to previous scheduling systems. It speeds task completion, reducing the 95th percentile by a factor of 1.76-2.11 compared to existing container scheduling schemes. Compared to other static graph-based approaches, our incremental partitioning technique reduces migrations per epoch by 82%.

The third chapter focuses on 5G user plane function (UPF), a critical interconnection point between the data network and cellular network infrastructure. UPFs typically run on general-purpose CPUs but are limited in performance due to host-based forwarding overheads. We design Synergy, a novel 5G UPF running on SmartNICs, that provides high throughput and low latency while supporting monitoring functionality for handover prediction and optimization during user mobility. Synergy extends SmartWatch's monitoring capabilities to capture and predict vehicular mobility patterns. This is then used to prepopulate state, even before the vehicle moves to the next base station, reducing 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. The subsequent paper, 5GDMon proposes a distributed monitoring system that analyzes traffic at multiple vantage points in the 5G ORAN infrastructure. In other words, 5GDMon is the distributed implementation of Synergy designed to detect network-wide anomalies.

The fourth chapter proposes 5GDMon, a distributed cellular monitoring solution that summarizes traffic characteristics monitored in the distributed radio access network (RAN). The summaries are communicated to data analysis engines running in the core of the network and support zooming into traffic subsets. SmartNICs are used for fast and efficient monitoring in the RAN, with query computation distributed to multiple UPFs using graph partitioning to balance the load. Here we leverage SmartWatch to collect the packet matrix while pMACH is used to load balance the query processing tasks. Deploying 5GDMon results in 37.99% fewer infected devices during controlled Mirai Botnet attack experiments and 1.35× higher resource fairness against adversarial heavy hitter attacks. 5GDMon achieves 3.92× lower error in detecting mobile proxies and up to 36% higher accuracy in detecting Tunnel Endpoint Identifier brute-forcing attempts.

In conclusion, our four chapters utilize SmartNICs to optimize performance and security in the data center and cellular networks. By using SmartNICs, we allow for CPU cores to be dedicated towards mission-critical tasks while ensuring performance is not compromised by adversaries or poor scheduling decisions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Traffic monitoring provides critical insights into performance and security, enabling network and system administrators to take targeted actions that can improve the overall efficiency, availability, and security of the network. Monitoring use cases include:

- Performance Optimization: Monitoring provides visibility into the performance of network infrastructure and applications, allowing administrators to identify and resolve issues quickly. This can help to optimize network performance and improve the user experience.

- Security: Monitoring can help identify potential security breaches and protect against cyber attacks by detecting anomalies in traffic patterns, suspicious activities, or unauthorized access attempts.

The difficulty with network traffic monitoring is that it is resource intensive, particularly because it consumes significant CPU and memory resources. Monitoring platforms must process packets arriving at high bandwidth links, making it CPU intensive. Furthermore, the need to process millions of packet flows, makes monitors memory intensive. Lastly, monitoring by itself does not generate the data center or the cellular network any revenue. In other words, operators face huge opportunity costs by dedicating general-purpose servers towards monitoring. Therefore, this thesis envisions a monitoring solution that enhances performance and security using heterogeneous computing capabilities.

Smart Network Interface Cards (SmartNICs) can provide a number of benefits for data centers and cellular networks. First, SmartNICs offload some of the networking and security processing from the server's CPU, which can improve server performance and reduce CPU overhead, leading to higher application performance and lower latencies. In data centers, SmartNICs can also help enable software-defined networking (SDN) and network function virtualization (NFV), which can improve the flexibility and scalability of network infrastructure. In addition, SmartNICs can also accelerate advanced networking and security capabilities such as network telemetry, encryption and compression, which better helps monitor the network. In cellular networks, SmartNICs can help manage the limited radio resources more efficiently by offloading some of the resource management and monitoring tasks from the base station and other data planes. This helps improve the quality of experience for users and enables more effective use of the radio resources. This thesis includes four contributions that optimize performance and security.

In this thesis, we first start with designing SmartWatch, a monitoring platform deployed on SmartNICs to scalably and accurately detect network traffic anomalies with little processing overhead. We then build on the SmartWatch traffic monitoring design contributions to optimize performance and manage power consumption in data centers, capture traffic characteristics in 5G cellular networks and predict mobility patterns. Finally, we develop a distributed monitoring system to analyze traffic from multiple vantage points in an open radio access network (O-RAN) infrastructure of the evolving software-based 5G network. pMACH extends SmartWatch's monitoring capabilities in the data center to capture cloud-application communication-patterns and uses it towards making better task placement decisions. In doing so, pMACH optimizes performance and minimizes power consumption. The latter two works are based on applying SmartWatch and pMACH's resource allocation and monitoring capabilities towards cellular networks. Synergy extends pMACH's monitoring capabilities to capture and predict vehicular mobility patterns. This is then used to prepopulate state, even before the vehicle moves to the next base station, reducing handover latency. Lastly, 5GDMon proposes a distributed monitoring system that analyzes traffic at multiple vantage points in the ORAN infrastructure. In other words, 5GDMon is the distributed implementation of Synergy inorder to detect network-wide anomalies. Here we leverage SmartWatch to collect the packet matrix while pMACH is used to load balance the query processing tasks.

Chapter 1: SmartWatch, Distributed Data Center Monitoring:

Network-borne attacks that aim to disrupt mission-critical systems and applications are a persistent problem for both network and data center operators. To counter threats, operators must deploy infrastructure to monitor systems and network traffic, driving analyses to detect anomalies and specific attacks in a timely manner. The infrastructure may directly protect against some attacks or provide alerts that trigger intervention, either automated or human, to block or ameliorate the impact of those attacks. SmartWatch, a network monitoring platform architecture, comprising a commodity-class host and sNIC, working cooperatively with programmable switches. SmartWatch makes the following contributions:

- To detect or prevent stealthy attacks such as low-rate port scans, SmartWatch performs lossless state-tracking and flow logging. SmartWatch yields 2.39 times better detection rate compared to existing programmable switch based platforms. Compared to host-based approaches, SmartWatch reduces packet processing latency by 72.32%.

- SmartWatch works cooperatively with P4 switches for network-based monitoring. SmartWatch uses iterative-refinement between the programmable switch and sNIC to judiciously use switch resources while operating at Terabit line rates. Switches direct the right traffic subset to the sNIC for processing.

- SmartWatch reduces the memory requirements of programmable switches for monitoring, thus allowing it to be used for common data center operations. This is done by running coarse grained queries in the switch. Fine grained processing is done by the sNIC-host subsystem.

Three components (sNIC, host, P4Switch) cooperatively perform monitoring. The sNIC offloads processing, reducing load on the host, and the P4Switch directly forwards the bulk of benign flows to their destination, preventing any unnecessary performance impact. The first stage relies on obtaining coarse-grained flow information. Flow subsets with attack indicators at a coarser granularity are forwarded for fine-grained processing, starting with the next monitoring interval. The first detection stage is performed on the P4Switches, where the focus is on processing high volumes of traffic. We are cognizant of the impact on latency critical user traffic, so that our in-line monitoring application minimizes the number of operations per packet and is lightweight. Following the broad approach of Sonata[194], Nitro Sketch[244], and BeauCoup[158], this first stage then steers traffic subsets that satisfy aggregate-traffic queries for finer grained analysis. The next stage is implemented on the sNIC-host subsystem of SmartWatch where processor and memory intensive operations are performed on the packet. The ability to add more monitoring functions as needed, ensuring efficiency and low latency, are all important. SmartWatch is responsible for configuring P4Switches, including specifying queries to be run by the switch. A more specific query would direct more targeted, and thereby less, traffic to SmartWatch from the P4Switch. We effectively create a control-loop where the sNIC-Host subsystem receives different amounts of traffic based on the degree of specificity of the query that SmartWatch installs on the P4Switch. The P4Switch queries are implemented using P4 tables where stateful operations are performed using P4 registers. At the end of a P4Switch monitoring interval, processing of the switch queries will determine some traffic subsets deemed as being suspicious (e.g., a threshold is crossed). Subsequent packets of these traffic subsets are forwarded to the sNIC-host subsystem. Over time, flows that get classified as benign by SmartWatch (e.g., after successful authentication) no longer have to be forwarded for fine-grained inspection by the sNIC-host subsystem. A P4 table in the P4Switch whitelists benign flows, thus reducing the overall traffic forwarded to the sNIC-host subsystem. This also reduces any performance impact on those flows.

We leverage the sNIC's memory to design a FlowCache that consist of a hash table and ring buffers. An incoming packet/flow processed by each sNIC packet processing entity updates the FlowCache using a hash of the 5-tuple. We use the sNIC memory (DRAM) to support 25 million flow entries. We propose a two-level cache on the sNIC (e.g., like a CPU's L1-L2 cache) and empirically select an eviction policy by examining the performance with a number of CAIDA traces[21]. The ring buffers accommodate evictions from the hash table and are used to periodically flush snapshots of the hash table to the host. Beyond providing a large reservoir of memory for tracking flow state and logging flow information, the host is also responsible for processing packets that the sNIC alone cannot process. Therefore, we need to minimize the cycles spent on the host for logging flows (e.g., handling flow records exported from sNIC to host). We develop a reconfigurable FlowCache that trades-off between sNIC packet processing throughput and the host's processing requirement. We do this adaptively by changing the eviction rate on the sNIC in response to the packet arrival rate.

The latter three contribution build on the SmartNIC monitoring capability of SmartWatch. pMACH optimizes the performance and reduces energy consumption in data centers. The SmartNIC's role is to collect the container affinity graph in pMACH. Synergy reduces handover delays by carrying-out mobility prediction. The SmartNIC's role is to collect the vehicular mobility patterns and use it for predictions. Lastly, 5GDMon extends Synergy to a distributed context, detecting anomalies by collecting packet matrices in the ORAN using SmartNICs.

Chapter 2: pMACH, Graph based Container Placement.

Striking the right balance between conflicting scheduling requirements such as overprovisioning to satisfy an application's service level agreements (SLA) vs. tightly packing servers to save power in a data center (DC) is challenging. Tightly packing containers is necessary to achieve high server utilization and power saving [316, 198, 240, 283] by turning off idle servers. DCs operate at $\sim 20\%$ server utilization [253, 220, 247] in order to meet application SLAs. This results in high DC power consumption as more servers remain powered on.

While there exists some prior work to minimize both power and task completion time [345], they are not incremental, leading to a significant number of container migrations. They ignore the cost of container migrations when adapting to workload changes or when the workload is consolidated to a smaller number of servers to reduce power consumption. Container migration (e.g., CRIU[65]) also results in downtime [37], and frequent migrations can adversely impact task completion times and are likely to result in SLA violations [314]. Thus, it is desirable to have a DC scheduler that simultaneously reduces power, task completion time, and container migrations and is also scalable to DC scales. The challenges are several - the need to operate servers efficiently [322], support fluctuating workloads [292], account for application container affinity [170], and account for migration overheads [37].

A DC cluster of several thousand servers, switches and links is typically broken up into smaller identical units. These units are called pods, comprising of several hundred servers along with the top-of-the-rack and aggregation switches. The DC network provides high-performance connectivity between all pods in the DC. We propose pMACH a Two-Tier distributed scheduling framework to adaptively 'right size' the DC by first considering a pod-level partitioning of containers, and then repartitioning the container sub-graph within a pod. pMACH schedules groups of containers (pMACH is generic, and may be used for scheduling VMs as well) of a partition on a server. It minimizes container migrations by adopting an incremental partitioning technique. pMACH's main focus is on achieving scalability using a Two-Tier partitioning algorithm, and executing the algorithm in an entirely distributed manner, unlike a centralized approach.

pMACH significantly reduces task completion time as containers that frequently communicate with each other are placed together in the DC topology. Power saving is achieved by having a minimal number of servers, so that unused servers can be turned off. Container migrations are reduced by accounting for dirty vertices (vertices that are moved from their original group to another group in the graph), thereby minimizing downtime. We consider three mechanisms to perform hierarchical partitioning of the container graph, namely, ParMetis Base partitioning, ParMetis Adaptive partitioning[**?**], and Tabu Search.

To obtain the container graph, we use a sNIC to collect the communication graph and provide it to the appropriate ParMetis graph partitioning worker nodes. This helps us save crucial CPU cycles. We use an efficient data stream summarization [330] to derive the edge weights with reasonable accuracy to allow frequently communicating container pairs to be placed together, to minimize task completion time.

Both testbed measurements and large-scale trace-driven simulations show that pMACH saves 13.44% more power compared to other scheduling systems. It speeds task completion, reducing the 95th percentile by a factor of 1.76-2.11 compared to existing container scheduling schemes. Compared to the static graph-based approach[345], our incremental partitioning technique reduces the migrations per epoch by 82%.

pMACH acts as the basis for 5GDMon, our forth contribution, to load balance the query computation over several traffic analyzers. In the second half of the thesis, we shall apply the SmartNIC's monitoring capabilities to the cellular environment. Our next contribution, Synergy, offloads a critical 5G userplane component to the SmartNIC.

Chapter 3: Synergy, Faster Handovers using Mobility Prediction. 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, 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[92]. 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).

Implementing 5G core (5GC) NFs [36] 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, contribute to constraining the performance[274]. Since the 5GC supports a large number of UEs connected to multiple base stations, facilitating a wide range of critical applications and services[259], achieving high performance for the 5GC is key. Utilizing network acceleration to implement 5GC NFs can substantially improve throughput.

In this work, we implement Synergy, a 5G UPF on a SmartNIC (sNIC). 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[274]. 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 [94] 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 [108].

Beyond implementing the core functions for a UPF on the sNIC to be compliant with the 3GPP specification [2, 3], 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 handover (as in Sec. 9.2.3.2.2 in [6]), 'Smart buffering' of packets within the UPF has been proposed as a way of reducing the latency in $L^2$5GC [210] and CleanG [260]. 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[210] and Free5GC[36].

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× when buffering within the sNIC instead of the buffering within the host. Compared to other sNIC-based flow state management approaches such as DeepMatch[208] and SmartWatch[274] 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 . 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[274] (§5.3.1).

Synergy increases its capacity by 44×, to support up to 12 million flows (§5.2) 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[81] that can be monitored for mobility prediction. 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. Feeding the monitored data to a mobility predictor helps achieve 2.32× lower average handover latency compared to not performing mobility prediction.

The following contribution mitigates the challenges of making Synergy a distributed solution for the purposes of detecting network wide anomalies. It leverages the contributions made in pMACH and SmartWatch to analyze and collect traffic matrices respectively.

Chapter 4: 5DGMon, Monitoring Distributed Cellular Networks. Cellular Networks have become predominantly IP-based data communication infrastructures. As such, cellular networks are also increasingly vulnerable to attacks, just as any other data communication network. Cellular networks have limited resources, especially radio resources, that must be managed carefully to provide the best quality of experience for as many active users as possible. Resource management and ensuring the cellular network's infrastructure and users are protected against attacks requires monitoring network traffic as in traditional IP networks.

The O-RAN software framework splits the RAN processing into several sub-components, with a Central Unit (CU), a Distributed Unit (DU), and a Radio Unit (RU)[201] together performing the processing that a traditional monolithic 5G base station (gNB) would perform. The RAN intelligent controllers (RICs) are tasked with streaming telemetry from the RAN so that they can provide intelligence to a Service Management Orchestrator (SMO) to deploy control actions and policies for resource allocation and management of the traffic by the CU, DU, and RU of the O-RAN environment. Network control functions manage the RAN by utilizing applications (called xApps) along with the RICs. A number of O-RAN RU and DU units may be managed by an SMO and RIC complex. A number of O-RAN complexes may be backhauled to a 5G cellular core network which is the main interface to the rest of the data network (including the Internet). Thus, by its nature, the overall O-RAN-based cellular infrastructure is widely distributed, with a number of vantage points for monitoring traffic and exercising control for varying subsets of the traffic carried by the overall cellular network. The traffic observed at the cellular core is the aggregation of all the traffic at the different O-RAN subnetworks. Traffic monitoring, closely coupled to the cellular network architecture can help in resource management, identifying anomalies, and combating attacks. Given the network's distributed nature, monitoring needs to be performed at multiple vantage points (e.g., close to each of gNB and the cellular core (see Fig. 6.2).

In this chapter, we argue that cellular attack detection requires: aggregation, refinement, and filtering. This is because of the limited radio resources and the heterogeneity of cellular cells. The purpose of traffic aggregation is to group monitored data that have a shared characteristic[231], thus minimizing the memory overhead to maintain statistics for the group. Refinement lets us zoom into traffic subsets, thereby adaptively allocating more memory resources only to those traffic subsets that would return higher detection accuracy[263, 195]. Lastly, filtering ensures that we only transmit to a monitor only those traffic subsets that help to evaluate a query result[197].

Monitoring in a heterogeneous cellular network: The cellular network includes a range of gNB sizes, such as Macro Cells, Micro Cells, etc. The traffic handled by different gNBs can also greatly vary, meaning that any monitor must be able to analyze the traffic adaptively. Jaal's traffic summaries consume considerable memory. However, accuracy drops significantly if we configure Jaal's parameters to reduce memory consumption. This is because of a lack of traffic refinement. As Dream[263], we must allocate more memory resources to those selected traffic subsets that improve overall detection accuracy. Jaal does not adapt to the heterogeneous traffic intensities as seen in the different cell sizes.

Detection at Periphery: Cellular resources are scarce, and therefore we want to be able to detect attacks as rapidly as possible at the periphery. In Dream, if one of the monitored prefixes is "interesting" from the perspective of a specific task, it divides that prefix to monitor into traffic subsets and uses more memory to monitor it. However, the refinement proposed in Dream is slow in the cellular context, mainly because it only considers IP prefixes and ignores other fields, such as flags, necessary to isolate benign vs. malicious traffic. For example, during a SYN Flood attack[195], using the SYN flag will better help discriminate benign and malicious traffic subsets rather than simply using IP prefixes.

Low Communication Overhead: Cellular monitoring overhead must be low while maintaining reasonable accuracy. CMY mainly configures thresholds and does not aggregate traffic based on subsets having similar traits. Therefore, the number of messages sent using CMY can be very high (400k messages for just 20 sites per epoch)[197]. To overcome this problem, we first used sampling, as is done in NitroSketch[245]. However, this only increased the convergence time with perceptible accuracy degradation. Elastic Sketch and Defeat suffer from the inherent problem of sketches, which involves trading off memory vs. accuracy and causes overestimation due to hash collisions. Furthermore, despite being invertible for five-tuple (e.g., can recover flows from sketch data structure itself [312]), dedicated sketches will have to be deployed as several detectors require data beyond just the five-tuple (e.g., SYN Flag). This results in higher memory requirement.

Chapter 2

Related Work

2.1 Resource Provisioning in Data Centers

A data center scheduler runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines [317]. This work focuses on scheduling for light-weight container instances. The broad goals for a power and migration aware DC scheduling approach are:

- **Task Completion Time:** There is a need to honor container resource requirements and place frequently communicating container pairs together so that the task completion time (latency) is reduced.
- **Power Consumption:** It is desirable to consolidate containers to fewer servers and operate them at peak energy efficiency.
- **Downtime:** It is important to minimize the downtime impact of container migrations.

E-PVM and RC-Informed are instances of the vector bin packing that model static resource allocation problems, where there is a set of servers with known capacities and a set of services with known demands [278]. Firstly, E-PVM distributes containers to the least occupied servers, leaving sufficient head room for spikes, but resulting in undesirably higher power consumption [135]. Alternatively, RC-Informed [165] predicts the workload for the scheduler to safely oversubscribe resources and tightly pack containers, thereby consuming less energy compared to E-PVM [345]. The problem with E-PVM and RC-Informed is that they do not consider container pair affinities and nor do they take advantage of peak energy efficiency. Compared to E-PVM, 6.6% to 18.8% power can be saved by alternatives that pack containers more tightly [345]. Furthermore, workload prediction can be imperfect and RC-Informed is shown to predict a new VM's CPU utilization with only 81% accuracy [165]. Under-prediction will cause the target peak utilization to be exceeded, and with oversubscribing of resources at 125%, it can result in violating latency requirements. Thus, it is desirable to have a lower utilization level for each processor and still save energy.

Another approach is to represent containers and the communication between them as a graph and use partitioning to allocate containers to different nodes[345]. The approach considers a container graph with resource demands as vertex weights and inter-container communication as edge weights. By running the graph partitioning algorithm accounting for edge cut and partition aggregate utilization, containers with high communications are grouped together and the load of the container group gets balanced. Goldilocks[345] is based on periodic partitioning of the container graph by Metis [219] and mapping it to

DC resources. According to the formulation in [345], Metis K-Way partitioning places frequently communicating containers together by minimizing edge cut (e.g., communication between servers). It respects server capacities by balancing container resource demands across servers.

Tabu Search is a widely used meta-heuristic for graph partitioning as shown in [239, 290, 209] and allows us to provide a custom cost formulation that can account for the cost of migrations. Local search methods have the tendency to be stuck in suboptimal regions. Tabu Search enhances the performance of these techniques by prohibiting already visited solutions or others through user-provided rules[344]. As shown in the next section, Tabu search reduces the number of migrations considerably, which is an important criterion.

The shortfall of Goldilocks is that the partitioning at every epoch is not incremental, causing a lot of container migrations, and it is not parallelizable, making it slow. In reality, there are only small changes in the workload between epochs. Incremental partitioning [297] reduces vertex migrations while reducing the edge cut and load imbalance. ParMetis is a Message Passing Interface (MPI [189]) based graph partitioning technique that distributes the graph's vertices across processing cores, to reduce the partitioning time. Instead of the centralized partitioning and scheduling in Goldilocks, we envision a distributed architecture to do both functions. Thanks to advances made in the graph partitioning algorithms, edge-cut minimization and load balance can also be carried out in parallel (e.g., multi-core or multi-server) by using ParMetis [297]. Alternatively, we could use Tabu Search also, but it is not scalable.

2.2 Monitoring Data Centers

Terabit-Scale Traffic Programmable switches (P4Switch) can run telemetry queries written in P4. Examples include Intel/Barefoot Tofino switches[54]. The P4Switch enables examining traffic at intensities much higher than would be possible with SmartNIC (sNIC) or Host alone [304]. A P4Switch's forwarding ASICs are able to quickly forward and perform simple computations on packets at line-rate, thus enabling the analysis of billions of packets at the Tbps rates[142]. But, they have accompanying memory and processing constraints, that limit the ability to do all the monitoring on the P4switch alone.

Table 2.1: Slow Attacks requiring flow-state tracking

Attack	**Challenges**
SSH Brute Forcing	SSH connections are encrypted Detector requires conn-attempt outcome[211], which is determined heuristically using protocol state transitions and traffic volume[50].
Stealthy Port Scan	Detector probes whether conn-attempt from a remote node elicits suitable response from local nodes[215].
Forged TCP RST (In-sequence)	Detector identifies race conditions between RST packets and in-flight data packets[320].

Network traffic monitoring has been widely explored [268, 235, 160, 174, 244, 331, 312, 246, 227, 332, 180, 301, 287]. Based on these, we first summarize the key requirements for network monitoring and then explain how fine-grained analysis is critical for monitoring:

Stateful Packet Processing: Low and slow attacks such as *in-sequence forged TCP RSTs* require state to be maintained to detect them. The relative timing between a potentially forged TCP RST packet and in-flight data packets must be measured [320]. Neither switch queries [194, 158] nor Sketches[227] can effectively detect such attacks due to hardware constraints and the need for non-volumetric, stateful detection.

Flow Logging: Flow logging maintains an accurate count of every packet of the flow received and stores the connection's 5-tuple, packet count, timestamp, and required-state depending on the specific attack being monitored. Volume based attacks such as DDOS [173] can be detected using Sketches[312, 244, 331] and using switch-based detection with the appropriate queries[158, 194]. They have been shown to identify attack indicators and can trigger alerts quite well for heavy hitters and other volumetric attacks. However, it can be challenging with monitoring applications that require high fidelity traffic matrices to carry out statistical analysis on network traffic [287, 300].

As an example, we motivate the need of a combination of fine-grained and coarse-grained traffic analysis by studying monitoring for the Slowloris Attack [105]. This attack keeps a very large number of connections to a target web server open, rendering it difficult for legitimate web requests to be served. We study the difference between a coarse and fine grained detector as outlined below.

Coarsed Grained Detector: This detector identifies end-points that use many TCP connections, each with low traffic volume (e.g., $\frac{\#\ Conns\ Est.}{\#\ Bytes\ Sent} > Threshold$) [194]. Also, instead of tracking this traffic volume for each and every host IP, it tracks it at a coarser-grained IP prefix level, such as the first 16 bits of the host IP address. Tracking aggregate activity consumes less memory, making it suitable to run on a P4Switch.

Fine Grain Detector: A widely used IDS, such as Zeek[116], measure HTTP request duration and identifies "stalling" flows (e.g., $duration > 10$ seconds). This is a memory and compute intensive activity, and such a fine-grained indicator is necessary to accurately detect the existence of the attack, the victims, and the attacker [118]. A host or sNIC is suitable to deploy such a detector.

Since most connections are benign, a compute and memory intensive monitoring capability in the data path will unduly penalize user traffic. For this, it is desirable to minimize the number of packets that are forwarded to a sNIC-Host system for monitoring beyond the processing performed on the packet already in the switch. Between the host and sNIC, processing packets in the sNIC avoids expensive host processing, data copies and PCIe transactions that impact end-to-end latency[241]. We desire a monitoring platform that can detect low and slow attacks that hide in the presence of large volumes of traffic. Here, we explain the challenges faced by other platforms.

P4Switch Switches have been widely explored for monitoring [194, 158, 329, 142]. Here, we study hardware constraints that limits its ability to conduct fine-grained analysis.

Memory Constraints: The SRAM memory enables P4 programs to retain state across packets and to hold the exact-match tables [142]. Tracking traffic flows at fine granularity on P4Switches requires sufficient memory. As noted in [224], given the limited on-chip SRAM on a typical P4Switch ASIC (of the order of 100MB SRAM[257], even though it can forward traffic at very high rates), it may require multiple such switches, which adds to cost. To accommodate such P4Switch memory constraints, Sonata[194] carries out dynamic query refinement, so that it only focuses on the subsets of traffic that actually satisfy a query, while ignoring the rest of the traffic, just like the coarse-grained detector above.

Flexibility accessing state: Accessing all available registers on a P4Switch can be a complex task since registers in one stage cannot be accessed at a different stage [142]. Next, the number of match-action pipeline stages limits the number of sequential processing steps $(10-20)$. Thus, the amount of P4Switch computation is bounded[338]. To maintain line rate, programmable switches allow only a small constant number of memory accesses per packet. This makes it infeasible to update multiple data structures for each packet[158]. Therefore, it is preferable for switches to perform coarse-grained analysis like in [158, 244] as they minimize per-packet memory operations.

sNIC Programmable NICs have also been used for monitoring in [304, 342, 182]. sNICs do not have explicit stages and have more memory compared to a P4Switch, meaning they have fewer constraints. However, a pure sNIC solution can only scale to Gigabit traffic[342]. But, they outperform host-based solutions[304, 342]. Unlike , other sNIC solutions generally do not focus on stealthy and slow attacks. Except for Pigasus [342], but they focus on a different approach, pattern matching, for intrusion detection.

2.3 Resource Provisioning in Cellular Networks

Several cellular dataplanes have been implemented that perform Packet Detection Rule (PDR) matching. A P4Switch based UPF was introduced in [249] to improve data plane throughput. Free5GC[36] is an open-source 3GPP compliant kernel-based implementation, which consists of a UPF running on the host. Furthermore, DPDK-based software-UPF solutions[148] have also been introduced to get rid of kernel overheads.

Monitoring of cellular networks has been explored before. NG-Scope[328] 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[328]. Lastly, prior work for buffering packets in the 5GC [148, 36], often leads to high packet loss compared to a sNIC-based UPF solution, primarily due to the slower processing at the host. Even P4Switch-based solutions resort to buffering on the host because of the lack of support for it in the P4Switch [249].

2.4 Monitoring Cellular Networks

Analyzing the presence of attacks across the entire network with a monitor per gNB will require significant communication between monitors. Therefore, we seek to utilize packet summaries (like Jaal)[138] to concisely transmit traffic matrices to traffic analyzers (i.e., query processors) in the 5GC. Furthermore, we assume that the queries of interest (e.g., to detect Mirai Botnet, heavy hitters, superspreaders, etc.) are known apriori, allowing us to filter traffic flows and zoom into traffic subsets of interest that have a higher chance of satisfying the query. Zooming into traffic subsets has been explored in Dream[263] and Sonata[195].

To achieve greater accuracy and faster detection times, we must use packet header fields beyond the IP prefix, contrary to Dream[263]. We envision a distributed monitoring system that can zoom into traffic subsets as easily as Dream, while considering substantially more packet header fields than just IP addresses.

The purpose of traffic aggregation is to group monitored data that have common characteristics [231]. It helps the operator detect network-wide anomalies while more concisely transmitting information among network monitors. First, we explore existing aggregation techniques and then select the suitable method for the large scale required for monitoring a cellular network spanning a large number of gNBs. Dream [263] and Sonata [195] introduce IP-prefix-based traffic aggregation at varying granularities (e.g., subnet /8, /16, etc). Defeat [236] utilizes random aggregation using sketches to summarize traffic. Lastly, Jaal [138] and CloudCluster[277] use clustering-based mechanisms such as K-Means and hierarchical clustering to summarize traffic.

First, we depict an example of prefix-based traffic aggregation and how it reduces communication overhead. The radio resource management in cellular networks typically allocates more radio resources to source UEs generating more traffic [155]. This makes it prone to misuse as an attacker can acquire an unfair share of the resources that otherwise should be allocated to legitimate users. A prefix-based detector will first identify source traffic aggregates at the gNB that exceed a threshold at a /16 prefix granularity. By analyzing traffic at the /16 granularity, fewer monitoring classes have to be tracked, leading to less overhead on the monitoring devices[195]. The monitoring class in this example is the first 16 bits of the source IP address, meaning 192.168.120.23 and 192.168.345.23 would belong to the same class and be aggregated together [274]. This reduces memory and communication overhead between the monitoring peers [195]. Of these, some source traffic aggregates that crossed the threshold will be further analyzed at a /24 granularity in the subsequent time interval. The remaining traffic that did not satisfy the query at the /16 analysis is thus filtered away. This is referred to as refinement, allowing us to dive deeper (analyze a larger number of monitoring classes) while removing the traffic from the analysis that does not satisfy the query. This is repeated until the exact UE is pin-pointed (e.g., /32). Prefix-based mechanisms allow the operator to change the granularity of analysis which is not possible with existing random or clustering-based traffic aggregation approaches commonly used for monitoring. This allows queries to zoom into traffic subsets of interest. The major problem with prefix-based approaches is that it does not consider fields other than IP addresses, such as SYN (or RST) flags, which is necessary to discriminate benign vs. malicious traffic during SYN (or RST) floods [195]. In the cellular context, this means that as we carry out refinement across epochs, the number of epochs required to correctly isolate attack traffic will increase, causing the attack to infiltrate inside the cellular network as opposed to remaining at the periphery. The goal of this paper is to detect attacks earlier at the periphery and to preserve scarce resources. Therefore we must be able to zoom into malicious traffic subsets as quickly as possible by also considering fields other than IP prefixes.

Clustering-based techniques bin traffic flows with similar characteristics together. In order to scale to cellular network-wide monitoring and utilize more fields (possibly in excess of 10 packet header fields, e.g., IP address, Ports, Flags, etc.), we must rely on cluster labels that aggregate traffic having similar header field values. Here, the subset of the packet's header fields becomes the feature vector for clustering. Then a clustering mechanism like K-Means assigns each packet a cluster label. Next, the cluster's centroid is computed and deemed representative of all the packets within a cluster [138]. Since the number of clusters is configurable, the number of 'representative packets' that have to be communicated become configurable (e.g., one per cluster). This is also referred to as traffic summaries [138]. Thus, the operator can configure the amount of monitor-to-monitor communication and trade that off against accuracy. This is because more aggressive summarization will lead to a lower detection rate. As traffic characteristics vary depending on the gNB the data is collected from (e.g., macro cell, small cell, etc.[66]), it is difficult to ascertain how many monitoring classes (e.g., bins) to use to summarize the traffic. Therefore, the problem with Jaal[138], which uses K-Means++[58], is that the number of clusters is unknown. Furthermore, due to the statically configured monitor-to-monitor communication vs. accuracy trade-off, the entire traffic is analyzed at the same granularity (e.g., no refinement, unlike prefix-based traffic aggregation). This prevents us from zooming-into traffic subsets without increasing the memory overhead significantly.

Thus, we need to utilize a clustering mechanism that considers packet header fields and still allows zooming into traffic subsets. This makes hierarchical clustering a natural choice to detect network-wide anomalies. We use hierarchical clustering to summarize traffic and transmit representative packets between monitors. For this, a dendrogram is constructed to derive the hierarchical relationship between packets. Next, the dendrogram is sliced horizontally, resulting in child branches below that become clusters. The representative packets are computed using the Ward method [115], which reduces the growth in the total intra-cluster sum of squared error among packet features[267].

As we zoom into the traffic subset of interest, we can also filter away traffic-subset reports to the traffic analyzers that do not serve to satisfy any query to further reduce the amount of data transmitted between monitoring peers. This differs from traffic aggregation (i.e., summarization), which groups traffic with similar traits. Queries typically have a threshold associated with them (e.g., $\#conns > T_g$) and are selected based on the likely use cases of the particular queries[158]. As introduced in CMY [197], in a distributed setting, if there are n monitors and the network operator is to compute the answer to a query whose (global) threshold is T_g (e.g., $\#SSH\ conns > T_g$), then messages can be filtered in the following manner, assuming traffic does not flow from one monitor to another. The local threshold at monitors is set to $T_l = \frac{T_g}{n}$. When T_l is crossed (e.g., $\#SSH\ conns > T_l$ at the local monitor $\mathcal{M}$), a report ($R_{\mathcal{M}}$) is submitted from the monitor $\mathcal{M}$ to the traffic analyzer with the actual counts observed for the query, otherwise no report is transmitted. This reduces the amount of communication between the monitor and analyzer. If threshold T_l is not crossed locally, the analyzer assumes the observed value ($A_{\mathcal{M}}$) for the query at the monitor $\mathcal{M}$ was $T_l - 1$. Let the estimate for the query computed at the traffic analyzer (across all monitors) be $\mathcal{E}$, where $\mathcal{E}$ is the sum of assumed and reported values, ($\mathcal{E} = \sum_{\mathcal{M} \in monitors}(A_{\mathcal{M}} + R_{\mathcal{M}})$). When $\mathcal{E} > T_g$ (would be rare if a majority of the traffic is benign), the traffic analyzer explicitly requests $R_{\mathcal{M}}$ from all monitors regardless of whether the local threshold T_l was crossed or not, to compute the query result.

There are several queries we seek to deploy in a cellular environment, especially with similar monitoring classes. This means that the fields inspected between two queries are not disjoint. In our distributed design, we cannot use a centralized monitor as in [330, 197, 263, 236, 138]. A centralized monitor would tremendously increase the overhead for processing the traffic summaries. Furthermore, the traffic analyzer can be a single point of failure. To prevent this, a subset of UPFs also act as traffic analyzers in . The queries have to be distributed among the traffic analyzers. Then the traffic pruning mechanism will ensure traffic summaries are only sent to the correct traffic analyzer, and only if it crosses the local threshold unless the query estimate exceeds global threshold (see §2.4). Carrying out monitoring in a distributed manner brings about an inherent problem. Some query pairs are disjoint in terms of the fields they inspect, while others are not. If two queries intersect (e.g., have similar fields) and are deployed on two different traffic analyzers, then the monitor will have to emit the traffic summary to both analyzers.

Chapter 3

SmartWatch: Distributed Data Center Monitoring

3.1 Introduction

Network-borne attacks that aim to disrupt mission-critical systems and applications are a persistent problem for both network and datacenter (DC) operators. To counter threats, operators must deploy infrastructure to monitor systems and network traffic, driving analyses to detect anomalies and specific attacks in a timely manner. The infrastructure may directly protect against some attacks or provide alerts that trigger intervention, either automated or human, to block or ameliorate the impact of those attacks.

The main challenge addressed by our work is on designing a cost-efficient traffic monitoring and analysis infrastructure that can detect a range of network attacks within a high-rate traffic stream. Systems must detect not only relatively crude volumetric attacks that overwhelm the network through sustained network activity (e.g., SYN flooding), but also more sophisticated attacks that probe for system weaknesses (SSH Brute forcing, port scan [215]), or attacks that exploit protocol dynamics (low-rate TCP attacks [230]) or deliver packet payloads that exploit application vulnerabilities incorrect implementations [262]. We seek to design a framework that can:

- detect low and slow attacks using state tracking;
- accurately track flow-state changes caused by each and every packet to ensure attacks cannot bypass the monitor;
- scale to monitor Terabit scale traffic by cooperative monitoring using multiple components including programmable switches, sNICs and hosts.
- provide flow-logging for rapid volumetric attack detection as well as comprehensive inspection of all flows offline;
- support a large number of monitoring features running simultaneously on the platform.

A number of different algorithms and monitoring platforms have been designed in the past to address this. Programmable switches have been used for telemetry queries at Terabit scale [194, 158] while SmartNICs (sNIC) enable end-hosts to scale to more modest, 40/100 Gbps, rates[304, 342]. Since sNICs support more operations [78] than switches, we use them for stateful packet processing, to complement the coarse-grained query processing of programmable switches. sNICs have become critical components of DC operations due to the performance boost they provide for the tasks offloaded from server CPUs[184]. We create a network monitoring system that leverages the sNIC capabilities of programmability, co-designed the scalability of programmable switches and flexibility of host-based CPU processing. Programmable switches are used to forward just the 'right' traffic subset for fine-grained sNIC-based monitoring.

This paper proposes SmartWatch, a network monitoring platform architecture, comprising a commodity-class host and sNIC, working cooperatively with programmable switches. SmartWatch makes the following contributions:

- To detect or prevent stealthy attacks such as low-rate port scans, SmartWatch performs lossless state-tracking and flow logging. SmartWatch yields 2.39 times better detection rate compared to existing programmable switch based platforms. Compared to host-based approaches, SmartWatch reduces packet processing latency by 72.32%.
- SmartWatch works cooperatively with P4 switches for network-based monitoring. SmartWatch uses iterative-refinement between the programmable switch and sNIC to judiciously use switch resources while operating at Terabit line rates. Switches direct the right traffic subset to the sNIC for processing. Less than 16% of packets processed by the sNIC go to the host.
- SmartWatch reduces the memory requirements of programmable switches for monitoring, thus allowing it to be used for common data center operations. This is done by running coarse grained queries in the switch. Fine grained processing is done by the sNIC-host subsystem.

We build a monitoring framework to detect a wide range of anomalies, caused both by volumetric attacks as well as specific slower targeted attacks which are increasingly successful [1] in the middle of a high traffic volume. For this goal, SmartWatch leverages the following key capabilities.

Cooperative Monitoring

The 3 components (sNIC, host, P4Switch) cooperatively perform monitoring. The sNIC offloads processing, reducing load on the host, and the P4Switch directly forwards the bulk of benign flows to their destination, preventing any unnecessary performance impact.

Two-Stage detector: The first stage relies on obtaining coarse-grained flow information. Flow subsets with attack indicators at a coarser granularity are forwarded for fine-grained processing, starting with the next monitoring interval. The first detection stage is performed on the P4Switches, where the focus is on processing high volumes of traffic. We are cognizant of the impact on latency critical user traffic, so that our in-line monitoring application minimizes the number of operations per packet and is lightweight. Following the broad approach of Sonata[194], Nitro Sketch[244], and BeauCoup[158], this first stage then steers traffic subsets that satisfy aggregate-traffic queries for finer grained analysis. The next stage is implemented on the sNIC-host subsystem of SmartWatch where processor and memory intensive operations are performed on the packet. The ability to add more monitoring functions as needed, ensuring efficiency and low latency, are all important.

Control Loop: SmartWatch is responsible for configuring P4Switches, including specifying queries to be run by the switch. A more specific query would direct more targeted, and thereby less, traffic to SmartWatch from the P4Switch. We effectively create a control-loop (Fig. 3.1) where the sNIC-Host subsystem receives different amounts of traffic based on the degree of specificity of the query that SmartWatch installs on the P4Switch. The P4Switch queries are implemented using P4 tables where stateful operations are performed using P4 registers. At the end of a P4Switch monitoring interval, processing of the switch queries will determine some traffic subsets deemed as being suspicious (e.g., a threshold is crossed). Subsequent packets of these traffic subsets are forwarded to the sNIC-host subsystem. Over time, flows that get classified as benign by SmartWatch (e.g., after successful authentication) no longer have to be forwarded for fine-grained inspection by the sNIC-host subsystem. A P4 table in the P4Switch whitelists benign flows, thus reducing the overall traffic forwarded to the sNIC-host subsystem. This also reduces any performance impact on those flows.

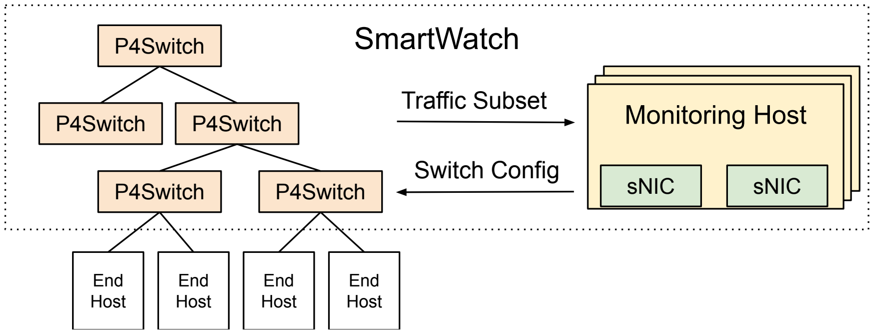


Figure 3.1: SmartWatch Control Loop.

Lossless flow monitoring

SmartWatch is a flexible network monitor for tracking flow state and logging flows with the SmartNICs (sNIC). This enables us to detect both stealthy as well as large-scale, volumetric network attacks. SmartWatch innovates by designing novel data structures in a P4-programmable Netronome sNIC. With the 40 Gbps Netronome sNIC, we are able to perform loss-less flow-state tracking and logging at 'near line rate' - a maximum 43 Mpps achieved with 64 Byte packets. This is consistent with [261] where even without any extra processing, line rate is only achievable using packets larger than 128B (This paper uses the same NIC). The bottleneck is most likely in the packet scatter-gather functionality across the micro-engines [261]. Similar limitations were also observed with the Intel XL710 NIC-based systems [244, 56]. Higher packet rates can be supported using a 100 Gbps sNIC (which we plan to do) or by sampling as in [158, 244]. However, such sampling techniques would not be able to support flow-state tracking.

Partitioning of Functions (Host vs. sNIC): The sNIC acts as an accelerator and helps track flow-state significantly faster than performing these functions on a host (e.g., TurboFlow [304] vs. Krononat [137]). On the other hand, the host has a much larger memory reservoir [224]. sNIC operations are also limited as there are no recursive functions or floating point operations available in the packet processing pipeline [78]. To ensure efficient state-tracking at line rate, we consider an sNIC cycle-budget for each and every packet. A violation of the cycle-budget potentially leads to dropping of packets at higher arrival rates. To achieve flow state-tracking at high packet arrival rates, SmartWatch designs a novel in-memory data structure on the sNIC with a flow eviction policy. This frees up a significant fraction of the time and cycle-budget on the sNIC for operations required by monitoring tasks.

sNIC FlowCache: We leverage the sNIC's memory to design a FlowCache that consist of a hash table and ring buffers. An incoming packet/flow processed by each sNIC packet processing entity updates the FlowCache using a hash of the 5-tuple. We use the sNIC memory (DRAM) to support 25 million flow entries. We propose a two-level cache on the sNIC (e.g., like a CPU's L1-L2 cache) and empirically select an eviction policy by examining the performance with a number of CAIDA traces[21]. The ring buffers accommodate evictions from the hash table and are used to periodically flush snapshots of the hash table to the host.

Reconfigurable FlowCache: Beyond providing a large reservoir of memory for tracking flow state and logging flow information, the host is also responsible for processing packets that the sNIC alone cannot process. Therefore, we need to minimize the cycles spent on the host for logging flows (e.g., handling flow records exported from sNIC to host). We develop a reconfigurable FlowCache that trades-off between sNIC packet processing throughput and the host's processing requirement. We do this adaptively by changing the eviction rate on the sNIC in response to the packet arrival rate.

To the best of our knowledge, this work is the first of its kind to cooperatively monitor traffic using a combination of programmable devices that span the range of memory and compute capabilities.

3.2 SmartWatch Architecture

In our cooperative monitoring scheme a P4Switch helps to identify attack indicators at a coarse granularity, effectively utilizing its limited memory and simple packet processing capabilities. Attacks listed in Table 3.1 may only be detected at a coarse granularity by aggregate traffic queries using a framework typified by SONATA [194], ConQuest [157], and Beaucoup [158].

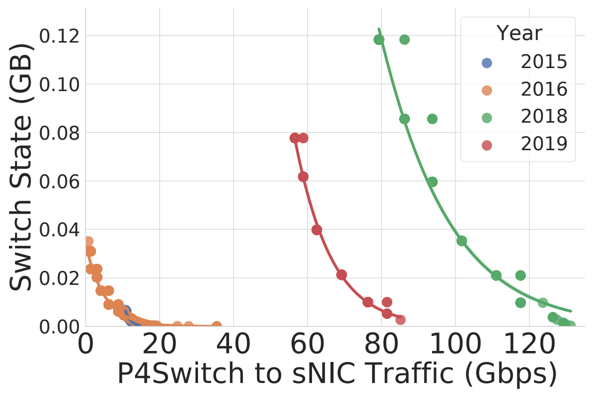


Figure 3.2: SSH Bruteforcing

Selective 'Bump-in-the-wire' processing: In general, packets processed in the P4Switch are directly forwarded to their intended destinations without any involvement of the P4-capable sNIC or the host. The P4 pipeline in the P4Switch passively monitors the traffic and computes the outcome of switch queries such as "is the number of ssh connections above a threshold?". If the threshold is crossed, subsequent packets of this traffic subset are steered to the sNIC (e.g., such as excessive SSH traffic destined to the same destination IP prefix). Therefore, only traffic that requires further inspection is forwarded to the sNIC-

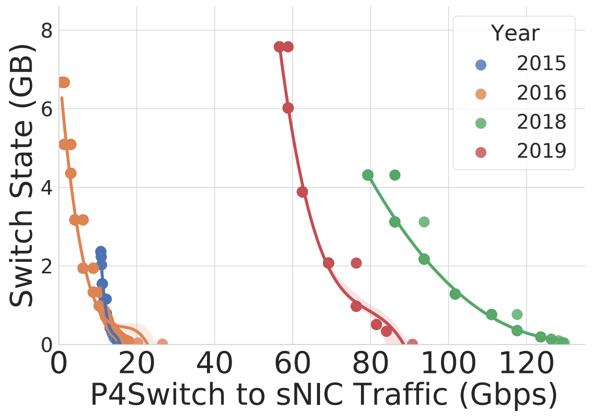


Figure 3.3: Port Scan

host subsystem, and are subjected to the higher latency due to this 'bump-in-the-wire'-like processing. At the sNIC-host subsystem we identify the SSH connection attempt outcomes (e.g., failure or success) and determine if there is a SSH-guess attempt. SmartWatch will then program the P4Switch to avoid benign flows (e.g., successful SSH authentication) from being subjected to the additional latency of sNIC's processing.

When installing rules in the P4Switch that whitelists benign flows, SmartWatch needs to be wary of the amount of state used in the P4Switch. To solve this problem, we borrow the 'hoverboard' intuition from Andromeda [167]. Selecting heavy flows that are benign (e.g., top-k) as opposed to mice flows helps reduce the amount of redirected traffic to SmartWatch, with relatively few rules installed in the P4Switch. Figs. 3.2 and 3.3 shows the P4Switch state vs. the traffic volume directed from P4Switch to the sNIC for the SSH bruteforcing and port scan attacks, respectively. We use CAIDA traces from different years [21] for this experiment. We see that there exists a knee, beyond which whitelisting flows does not reduce P4Switch state further. We describe these attacks in greater detail later in this paper. Furthermore, the sNIC FlowCache data structure is responsible for identifying the top-k heavy benign flows, which we describe in this section.

Switch Query Refinement: We borrow the iterative refinement approach from Sonata[194] to selectively steer flows from the P4Switch to sNIC or host. Let us consider an example where we have to track the number of SSH connections per destination IP. Treating these as key-value pairs, each destination IP is a key and the number of SSH connections is the associated value. Now, instead of tracking each individual destination IP (dIP), we aggregate them to a less-specific subset, such as based on their 16-bit prefix (dIP/16). This coarse grained analysis requires less state on the P4Switch due to fewer key-value pairs. Steering traffic that satisfies a query at the coarser dIP/16 granularity instead a dIP/32 granularity will cause much more traffic to be redirected, since dIP/32 is more specific. Iterative-Refinement zooms into traffic by filtering just the correct flows to allow multiple queries to run on the P4Switch despite its limited memory. If we compare Sonata's iterative-refinement to SmartWatch, in Sonata[194], the P4Switch memory is reused for traffic subsets at a more specific granularity (e.g., /16 instead of /8). The rest of the traffic is not examined. Instead of reusing the switch memory and only analyzing a narrow window of traffic in the P4Switch as in Sonata[194], in our design we send the narrow window of traffic to the sNIC-host subsystem, but have the switch continue to examine the coarser subset of traffic. SmartWatch reuses Sonata's[194] interface to load switch queries on the P4Switch.

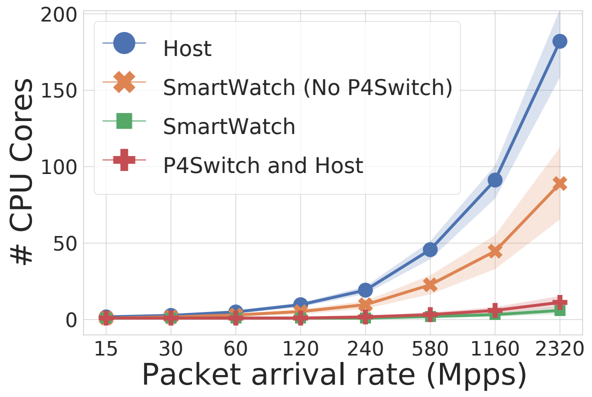


Figure 3.4: #CPU cores required

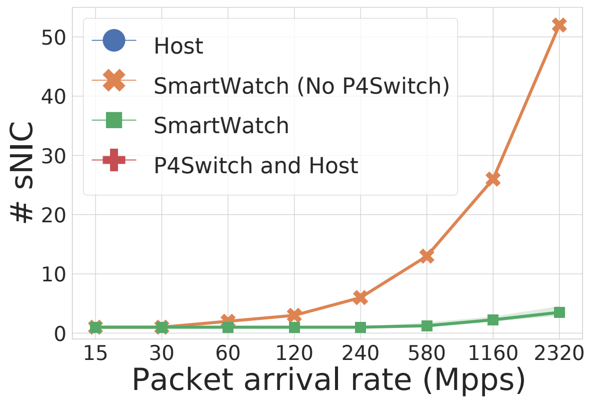


Figure 3.5: #sNIC required

Resource Usage: We simulated four different scenarios, including 1) standalone host based monitoring system, 2) SmartWatch without a P4Switch, 3) SmartWatch, and 4) host with P4Switch. The P4Switch runs the iterative query refinement algorithm derived from [194]. In this experiment, we speedup the CAIDA 2018 trace to emulate different packet arrival rates. Our findings are in Figure 3.4 and 3.5. The y-axis is the amount of resources (i.e., CPU cores and sNIC respectively) required to sustain different packet arrival rates (x-axis). There is only one P4Switch in this simulation experiment. We observe that the P4Switch helps SmartWatch reduce the number of sNIC and CPU cores by at least 14 times by forwarding the bulk of the traffic through to the destination when the packet arrival rate is 2320 Mpps. The number of required sNIC and CPU cores are 4 and 6, respectively. This makes SmartWatch practical to scale to Terabit level traffic. The detection rate of our approach is also better, as shown in Section 3.4.3.

3.2.1 sNIC FlowCache Design

SmartWatch utilizes both the host-CPU and sNIC's packet processing engines, referred to as Micro-Engines (ME), to track flows in a loss-free manner and minimize communication overhead between host and the sNIC. The sNIC has a P4 match-action table sequence and a FlowCache data structure, which is designed using C functions. An incoming packet is first scheduled to a packet-processing micro-engine, (PME), by a 'global' load-balancer where packets are serviced in a "run-to-completion" manner (e.g., non-preemptive). The P4 match action tables provide specific packet processing/forwarding rules for the incoming flow. Packets are also processed by the FlowCache for monitoring tasks.

sNIC FlowCache: We use the sNIC's memory to design a FlowCache that consist of a hash table and ring buffers. The cache is in contiguous memory and is allocated at compile time. An incoming packet processed by each PME updates the FlowCache using a hash of the 5-tuple. We use the sNIC memory (DRAM) to support 25 million flow entries. The ring buffers accommodate evictions from the hash table and are used to periodically flush snapshots of the hash table to the host. We dedicate 8 ring buffers. Having these 8 ring buffers mitigates the access contention for ring buffers among the 80 PMEs.

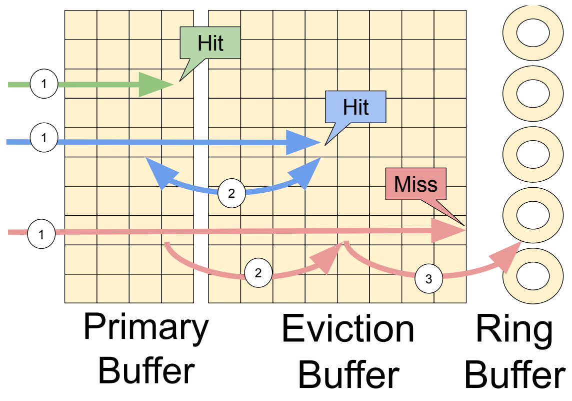


Figure 3.6: Adaptive Data structure

Data Structure: We use a large hash table with an array of buckets to cache flow records within the sNIC. To keep the per-packet latency low, we restrict the entries for a hash index to at most 12 buckets, ensuring the sNIC can maintain high throughput and minimize packet drops. As observed in other programmable dataplanes[224], Cuckoo hashing is not suitable for caching flow records in the sNIC because it can often require multiple memory accesses. In a Cuckoo hash table, a hash collision will cause a hash entry to be moved to its secondary location, causing a write operation. On the other hand, in our proposed mechanism while there may be multiple read operations, there is just one write operation. Empirically, with a limit of 12 recursive insertions (with Cuckoo hash) vs. 12 buckets (with FlowCache), the 99.9 percentile latency for a CAIDA DC trace[21] was observed to be 2.43 times lower with FlowCache. This is because sNIC write operations are relatively expensive compared to reads. Unlike a write, for a read the calling thread yields so that another thread can continue its work while the memory is being read [113].

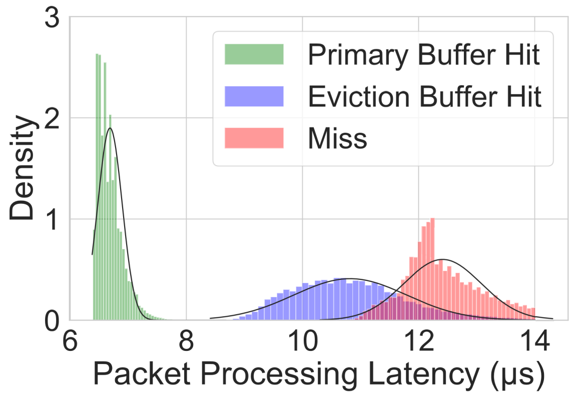


Figure 3.7: FlowCache latency dist.

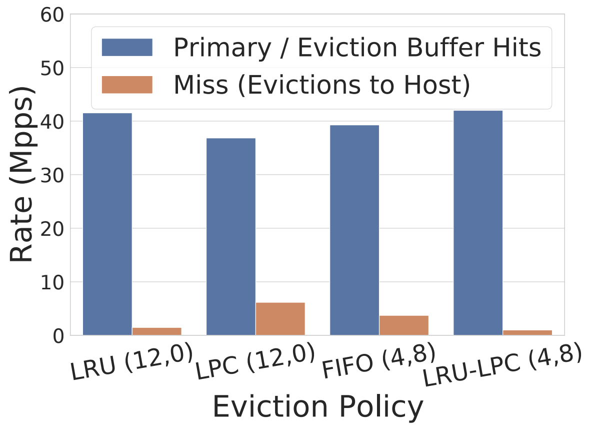


Figure 3.8: Hit and Miss counts

Partitioning & Eviction Policies: The key insights from previous work with typical Internet data center (DC) traffic characteristics and the CAIDA packet traces are: 1) a few large flows account for a majority of the packets, 2) numerous small flows frequently compete for a hash entry, and 3) packets of elephant flows arrive over several bursts. We

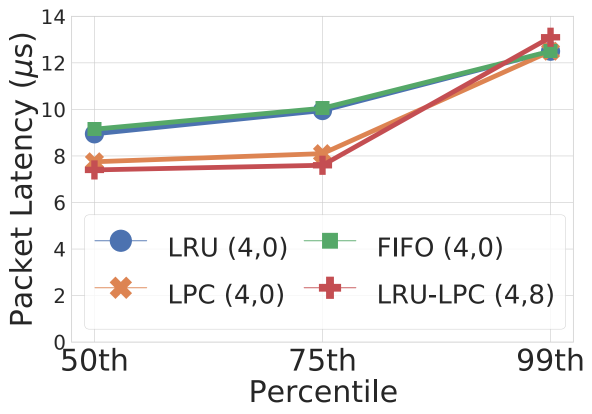


Figure 3.9: Latency profile

experiment with a number of widely used eviction policies: Least Recently Used (LRU), Least Packet Count (LPC) and First-In-First-Out (FIFO) on the hash table with 2^{21} rows $\times$ 12 buckets per row. We further devise a split of the hash table into two buffers, namely a *Primary* P and an *Eviction* E . Note: we use the notation (x, y) to designate a configuration with x buckets in P and y buckets in E per row, respectively. We measure the hit and miss rate when the sNIC is subjected to 43 Mpps (64 Byte packets) CAIDA 2018 trace [21]. Note that all four policies have the same memory footprint. Misses cause evictions of flow records to the host, and therefore we seek to minimize them. Among the four policies, LRU has the highest hits, but LPC has lower latency. In order to effectively reap the benefits of LRU (handle a continuous train of packets from a flow) and at the same time benefit from the low latency with LPC (large number of hits coming a small set of elephant flows), we installed a hybrid LRU-LPC policy in P and E respectively, which provides the highest hit rate and lowest latency (median and 75%ile).

Pinning Flow Records: Low and slow attacks require per-packet flow state updates to accurately detect malicious flows. We seek to avoid host involvement to keep packet processing latency small, by dynamically pinning flow records in the sNIC FlowCache. This prevents the eviction of a flow record which potentially would result in inaccurate tracking of a suspect flow's state. In an event where all flow records of a row are pinned and one flow must be evicted, the packet being processed is sent to the host, which we strive to minimize. If a we cannot find a flow entry for a packet (Miss) either because it was evicted or not-pinned, then we create a new flow entry. We will not retrieve packet counters from the host as this tremendously increases packet latency.

Table 3.1: SmartWatch Resource Summ. (No P4Switch)

Attack	**sNIC Cycles(%)**	**Host Processed(%)**
Heavy Hitter, Heavy Changes, Cardinality, Flow Size Estimation and Slowloris	80.32	0
Zeek SSH Bruteforcing	1.79	1.24
Zeek Expiring SSL certificate	1.98	1.35
Zeek FTP Bruteforcing	1.85	1.19
Zeek Kerberos Ticket Monitoring	1.99	2.9
In-Sequence Forged TCP RST	1.94	0.95
TCP Incomplete Flows	2.01	8.3
Stealthy Port Scan	1.99	0
DNS Amplification	1.93	0
Micro-bursts	2.08	0
EarlyBird Detection Worms	2.06	0

Reduce Host Packet Processing with FlowCache We benchmark 15 attack detectors *simultaneously* running in SmartWatch. Table 3.1 (Host Processed col.) shows the percentage of trace packets (CAIDA 2018) processed by the host. Most are processed by the sNIC. Thus, there is a dramatic reduction in host overhead. Depending on the flow-state, select packets are forwarded to the host. The average packet processing latency reduces to just 28% compared to when everything runs on the host. PCIe transactions and packet copies contribute to the host-based processing being slower [241].

3.2.2 sNIC Reconfigurable FlowCache

Increasing the number of buckets accommodates more flows with fewer evictions from the sNIC to the host. But, it lowers throughput due to higher processing latency. To adapt to fluctuating packet arrival rates, we dynamically mutate between the General and Lite mode, with minimal overhead. General Mode has 12 buckets per row in (4,8) configuration. This captures most of the large flows and operates in a loss-free mode for arrival rates upto 30Mpps. But, it is insufficient to keep up with the maximum achievable line-rate (43Mpps) for the 40Gbps NIC. But, it also has the benefit of a lower eviction rate (reduced transfers to host). The Lite Mode on the other hand supports higher packet rates. Fig. 3.10, shows that both (2,0) and (1,0) meet near line-rate for 64B packets. So, we select (2,0) as Lite mode. Unfortunately, the Lite mode results in a higher eviction rate because of a lack of buckets to resolve collisions, despite having the same memory footprint. Figure 3.13 shows that the CPU time [97] for the host thread responsible for snapshotting flow records increases by 2.08 times when we employ the Lite (2,0) mode compared to the General mode of the sNIC FlowCache. This is because of the 47% increase in eviction rate.

Map Offline Tasks to Micro-engines. There are a total of 80 sNIC MEs usable for i) packet processing (PMEs) or ii) custom processing (CMEs), separated from the packet processing pipeline. Fig. 3.11 shows the variation of throughput as we change the number of PMEs. We are able to allocate a total of 3 MEs for custom processing (as CMEs) without any degradation in the maximum packet processing throughput. We use CMEs for the task of switching between the two modes. Later, we also show their utility for other monitoring tasks such as reporting flows that cause micro-bursts.

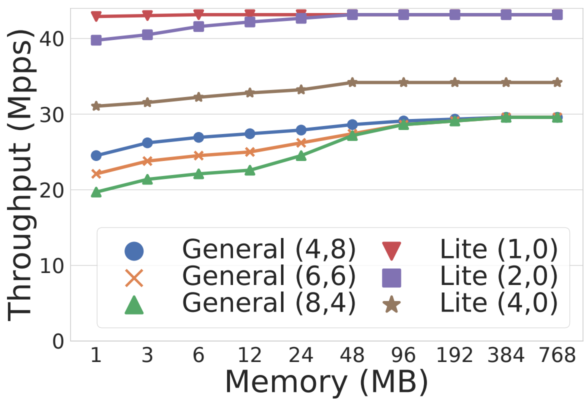


Figure 3.10: Tput vs. MEM

Correct State-Tracking without Flow Duplicates. For the *General Mode*, packets have to be checked against bucket [0, 12) across P (e.g. [0,4)) and E (e.g. [4,12)). For the *Lite Mode*, only two buckets are checked at an offset determined by the higher order bits of the hash digest. Clearly, the candidate buckets for the *Lite Mode* are a subset of the *General Mode*. Thus, there is no overhead to switch from the *Lite Mode* to *General Mode*,

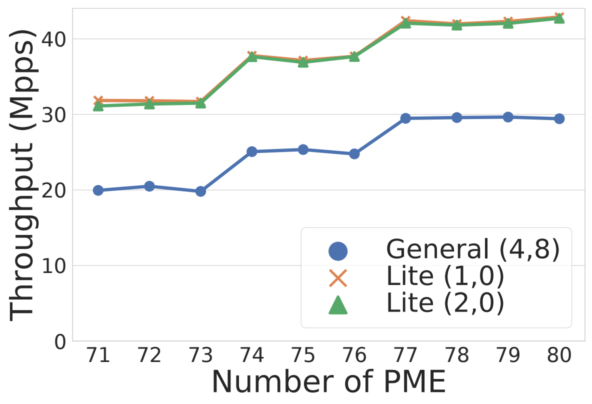


Figure 3.11: Tput vs.# PME

and there is less flow-matching penalty in the *Lite Mode*. On the contrary, flow entries will have to be reordered when transitioning from *General* to *Lite Mode*. The contiguous memory and the logical partitioning between P and E allows us to move the logical boundary between them and resize the number of rows in the hash table. We use a global variable `mode` for the current operation mode. A CME periodically tracks the packet arrival rate (EWMA with $\alpha = 0.75$ over a window of 100 samples) and compares with a threshold to switch modes.

General to Lite Transition When the packet arrival rate exceeds the rate supported by the General mode, the CME triggers a *switch over* to Lite mode. The CME marks all the hash table rows as 'dirty'. The first PME that finds a bucket within a 'dirty' row performs the cleanup, which involves reordering the flow entries and then marking the row as 'clean'. The PME gains exclusive access to the row (i.e., allow no concurrent flow updates) and reorders the flow records honoring the Lite mode's logical boundary. We do this lazily, as

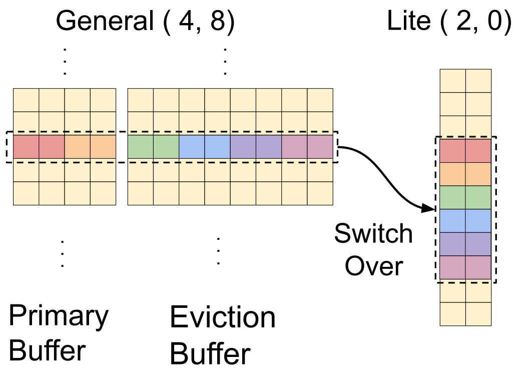


Figure 3.12: General to lite mode

packets arrive, as it is slower for a single CME to reorder all flow records, taking at most 14μs. The 80 PME on the sNIC process packets in parallel for other FlowCache rows even if one PME has exclusive access to a specific row. However, some packets end up waiting for the clean up process to relinquish the exclusive access to a row. We observed this wait time to be less than 5μs, before it can make its own state updates.

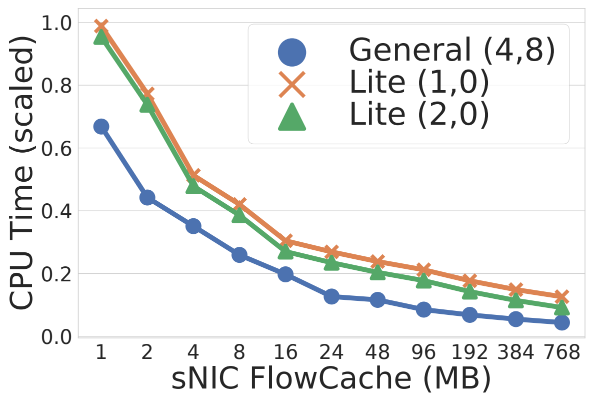


Figure 3.13: Host snapshotting overhead

Lite to General Transition When the packet arrival rate drops below a threshold, the CME initiates the transition to General mode. Flow entries do not have to be reordered and the eviction process can run as usual. For example, a General mode's row consists of 12 buckets, which is sub-divided to support six rows of two buckets each in the Lite mode. Therefore, the packet is forwarded to the same '12 buckets' in physical memory. Since all 12 buckets will be probed in the General Mode and since the candidate buckets in the Lite mode are a subset of the General mode, correctness is ensured.

3.2.3 Sub-components for Host Support

The host can provide a large amount of memory (> 200GB) and storage ($\sim$2TB) compared to the switch and sNIC. We leverage the host as a global pool to collect and store all flow-related information over multiple snapshots for detailed forensics. We leverage the sNIC to aggregate flow records and export them to the host periodically (every 5 seconds). Since the sNIC may export a particular flow's entry several times due to eviction or being aged out, the host is responsible to correctly aggregate each flow's information. We DMA flow records by borrowing the implementation from [269]. The host caches these flows with a $2^{30} \times 1$ hash table. The entries from host cache are periodically (per measurement interval) flushed to a Redis [99] datastore for flow-logging. The host CPU also has the capability to process packets through SR-IOV ports (DPDK [190]). Some IDS/IPS components cannot be offloaded to the sNIC as they require complex operations (See §3.1). We dedicate distinct SR-IOV ports for each supported function on the host. Host NFs include: 1) Zeek[116] for IDS/IPS scripts, 2) Timing Wheel[315] to buffer and release packets, and 3) NFs using the host's larger memory pool.

3.3 Implementation

Low Cost to Add Monitoring Features. The required state for baseline flow logging is $768MB$ to cache 25 million flow records, including timestamps and packet counters. Given the sNIC has 8 GB memory and supports bulk operations, additional attributes can be added to the flow entry to track flow-state, enabling support of additional monitoring features. However, the packet processing overhead must be minimized to perform lossless tracking. The baseline FlowCache, without any additional monitoring features, which supports flow logging consumes most of the sNIC cycles (80.32% of the total cycles). Flow logs exported to the host (always enabled) can be analyzed offline for heavy hitter detection, heavy changes, cardinality estimation, flow size estimation, and Slowloris. Our eviction policy ensures FlowCache processes packets at near-line-rate. Table 3.1 shows that the cycles consumed by other monitoring features is very small compared to FlowCache, and therefore do not reduce packet processing throughput. This is because of the parallel processing across a large number of PMEs on the sNIC, including hardware support of atomic operations.

Symmetric Hash Function: Detectors require session based flow-state tracking. We need to ensure IP packets in the reverse direction map to the same bucket as that of the forward direction. Hence, we use a symmetric hash function[326].

Table 3.2: sNIC Comparison

Attribute / sNIC	**Bluefield MBF1L516A -ESNAT**	**LiquidIO OCTEON TX2 DPU**	**Netronome Agilio LX**
Processor	2.5 GHz [84],[11]	2.2 GHz [68]	1.2 GHz [76]
Parallelism	16 cores [84]	36 cores [68]	96 cores [76]
L1 Size	32 KB [84]	32 KB [241]	64 KB [67]
L1 Access Time	5.0 ns [241]	8.3 ns [241]	13 ns [113]
L2 Size	1 MB [84]	24 MB [68]	256 KB [67]
L2 Access Time	25.6 ns [241]	55.8 ns [241]	51 ns [113]
DRAM Size	16 GB [241]	16 GB [241]	8 GB [79]
DRAM Access Time	132.0 ns [241]	115.0 ns [241]	137 ns [113]
Atomic Primitives	Yes [241]	Yes [241]	Yes [113]
Programmability	GNU [84]	GCC [68]	Micro C/P4 [67]

3.3.1 Generality of sNIC Implementation

In this section, we study the generality of our implementation on the Netronome sNIC, the potential for adoption with other, such as BlueField and LiquidIO sNICs. The details of the 3 sNICs are listed in Table 3.2. Using this, we ran a discrete event simulation with a CAIDA trace[21] containing almost 2 billion packets, where all packets are reduced to 64B to create a worst case stress test. All 3 sNICs support programmability and atomic primitives with a cache structure and multiple packet processing cores. We model the number of cycles consumed in the FlowCache for the BlueField and LiquidIO sNICs by performing the measurements on our Netronome sNIC (e.g., no. cycles for hits / misses). We then estimate the packet processing latency and the number of packets processed per second across all compute units based on the different processor speeds and memory access latencies (Table 3.2 specification). Using our trace-driven simulation, we derive the packet throughput for BlueField and LiquidIO sNICs to be 40.7 and 42.2 Mpps respectively, compared to the throughput for Netronome sNICs, which was 43 Mpps. The reason for their slightly lower throughput compared to Netronome is because of the fewer number of processing cores. SmartWatch is a monitoring platform. But, thanks to its P4 and SR-IOV capability, it can support common data plane functionalities such as switching, tunneling, and QoS, which are typically supported by OVS [167] in today's DCs. All three sNICs support OVS offload and SR-IOV [84, 76, 70, 68]. SmartWatch and OVS can act as the monitoring and connection tracking modules of the pipeline, respectively.

3.4 Evaluation

Testbed: We evaluate the effectiveness of SmartWatch on our local testbed consisting of Linux servers (kernel ver. 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. We use three packet generators, each running Moongen[176] to replay PCAP traces at the high rate (43 Mpps using 64B packets) for our stress tests. Attack and background traces are timestamp shifted and then merged using editcap [33] and mergecap [72], respectively. To truncate packets to 64B, we use tcprewrite[110].

First, we compare SmartWatch against monitoring systems deployed on the host, such as Zeek. Second, we show how SmartWatch can reduce the memory (SRAM) pressure on a P4Switch for cooperative monitoring for detecting covert timing channels [329] and website fingerprinting [142]. Third, we show how FlowCache helps improve long (5 sec) and short (¡200μs) timescale traffic analysis. Lastly, we compare cooperative monitoring technique to Sonata[194] in terms of accuracy. In our experiments, all detectors that are based on flow logging are processed offline on the host, while other attacks, such as SSH Brute Forcing, are detected online.

Evaluation Traces: We use four different traces: 1) CAIDA Traces [21] (years 2015 to 2019) containing 1 to 1.9 billion packets., 2) For stress testing, we created traces with 64B packets with CAIDA traces, 3) For targeted attacks, we used official test traces from Zeek IDS [116], and 4) Univ. of Wisconsin DC measurement traces [145].

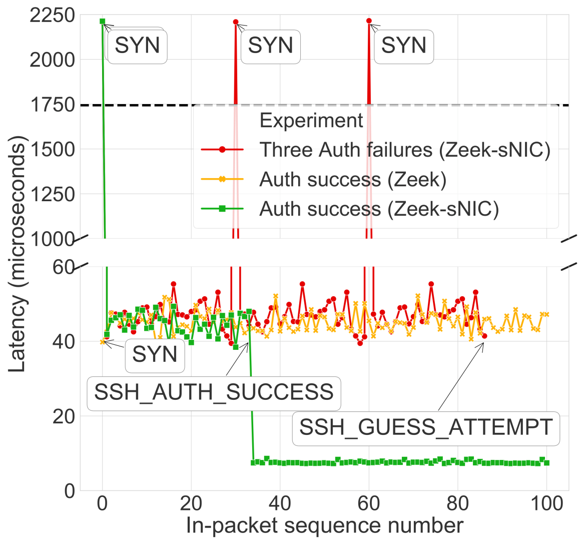


Figure 3.14: SSH Brute-forcing

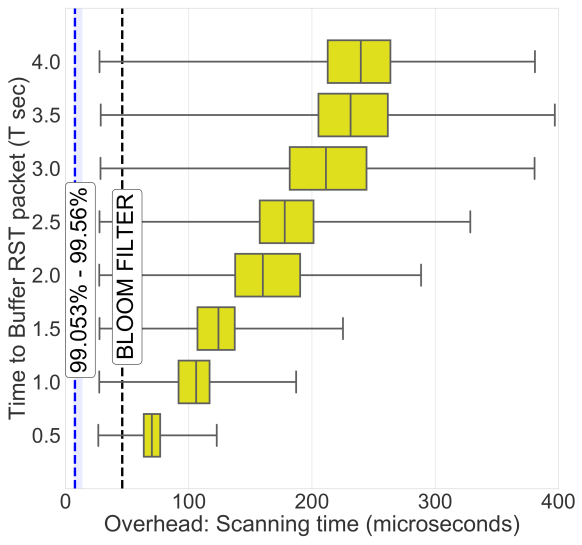


Figure 3.15: Forged TCP RST

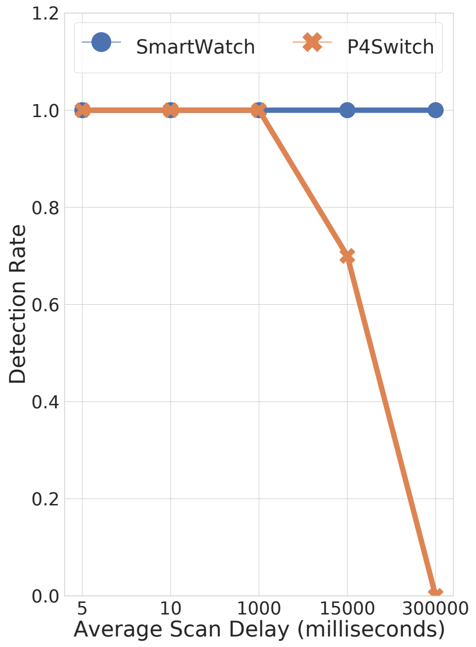


Figure 3.16: Port Scan

SSH brute-forcing

Attack: Many nodes with distinct source IPs use different username/password combinations on SSH login servers [211]. For this attack we leverage the host running Zeek[116] and partition processing tasks between the sNIC and host. The host is only involved in the authentication phase, and if successful packets avoids PCIe transactions to the host.

Detection: Track the number of failed SSH login attempts, ψ, by a remote node in a given time interval. Raise an alert when ψ exceeds a threshold[120].

P4Switch Role: For the SmartWatch framework, the P4Switch measures if the number of SSH connections attempts have exceeded a threshold. Flow subsets with significant number of authentication attempts observed in P4Switch are forwarded to the sNIC. The P4Switch cannot conduct fine-grained analysis by itself as it cannot determine the number of failed SSH connection attempts ψ for a specific remote node, since they are encrypted. It is not possible to heuristically determine connection attempt outcomes in the P4Switch as it requires per-packet state transitions (see §2.2).

SmartWatch Role: As a new SSH connection arrives, we pin the flow entry within the sNIC until the outcome of the connection attempt is determined. FlowCache ensures that subsequent packets of pinned entries are forwarded to a host NF running Zeek. Zeek heuristically guesses the login attempt outcome by tracking connection state transitions and the amount of data communicated [50]. If the host NF detects authentication success, Flow-

Cache unpins the flow entry (evicted as needed) and does not send subsequent packets to the host NF by updating the match action table. If ψ exceeds a threshold, the P4 table blacklists the source IP. Here, the gain is experienced by all packets beyond the SSH authentication phase. The sNIC tracks approved vs. non-approved flows, causing only 1.24% packets in the SSH trace to go to the host (Table 3.1). The remaining packets of the trace avoid costly transfers to the host.

Evaluation: Figure 3.14 shows the SSH packet latency for three scenarios: 1) successful authentication with SmartWatch 2) successful authentication with baseline Zeek, and 3) multiple authentication failure attempts with SmartWatch. Here, 3 failure attempts in 30 minutes generates an *SSH_GUESS _ATTEMPT* event. We utilize traces available in the Zeek package for this experiment. Once a SSH connection is approved by Zeek (*SSH_AUTH _SUCCESS*), packets are no longer processed in the Zeek NF, reducing the average packet processing latency by 77% compared to baseline Zeek. In Fig. 3.14, SYN packets result in unavoidable latency spikes up to $\sim$2250μs because of the need to remove connection state.

Similar Attacks: Expiring SSL certificates[122], FTP brute-forcing[117], and Kerberos ticket traffic[119]. Similar to SSH connections, Zeek can scrutinize the validity of the connection in the host. Following Zeek's approval, their packets are entirely processed in the sNIC.

TCP Forged Resets

Attack: An adversary disrupts TCP connections by sending a forged TCP RST packet to either end of a connection. We leverage the host running a timing wheel where potentially forged RST packets can be buffered until the RST packet is classified as malicious or benign. In this example, the sNIC is responsible for steering the smallest possible traffic subset to the host, thus reducing the high latency PCIe transactions.

Detection: This can be detected using 1) RST packets with outdated SEQ numbers; 2) Multiple RST packets with increasing SEQ number; 3) race conditions between the RST packet and in-flight end-host data packets. Here, we focus on the race condition between the RST packet and data packet as it requires the most flow-state tracking and is difficult for attackers to avoid. Race conditions are unlikely if the RST packet has been generated by an end-host. Hence, it is recommended that a monitor maintain state for a time interval T=2 seconds from the arrival of the RST packets to determine whether the RST is genuine or forged [320]. On the host, we implement a timing wheel [315] to buffer RST packets.

P4Switch Role: In the SmartWatch framework, the P4Switch measures if the number of RST packets exceeded a threshold. Flow subsets with a large number of RST packets seen in the P4Switch are forwarded to the sNIC. The P4Switch by itself cannot conduct this fine-grained analysis (see §2.2). The precise victim in the flow subset requires tracking the arrival of RST packets and inspecting the sequence of subsequent packets [320].

SmartWatch Role: FlowCache steers TCP RST packets to the host via a dedicated SR-IOV port and pins the flow entry in sNIC FlowCache. The RST packet is released by the timing wheel after $T = 2$ seconds if no race conditions are identified (i.e., not forged). On arrival of a genuine data packet right after a forged RST packet, sNIC FlowCache notifies the timing wheel (i.e., state-tracking) after which the forged RST packet is discarded from the timing wheel, without reaching the destination. sNIC FlowCache records are unpinned when the buffered RST packet is released to its destination, or when a forged RST is actually detected. The gain is experienced by all packets arriving prior to the monitor seeing a RST packet in a connection. In our experiments, only 0.95% packets of the CAIDA data center trace go to the host and experience the additional processing delay of the monitoring NF and PCIe transaction (Table 3.1).

Evaluation: Only unique RST packets should be inserted into the timing wheel while duplicate RST or data-after-RST must be immediately notified to operator (i.e., an attack). Ensuring uniqueness requires the timing wheel to be scanned while buffering the RST packet, potentially degrading packet processing latency. This processing can be bypassed for some packets using a Bloom Filter, accelerating the RST buffering operation. Fig. 3.15 shows the packet processing latency and percentage of packets experiencing it for different values of T for the 2018 CAIDA trace [21]. As T increases, so does the scanning time, as more buffered RST packets have to be checked. The blue vertical line is the mean round-trip latency for packets processed solely in sNIC FlowCache. This accounts for 99.053% packets of the trace. As for RST packets, uniqueness identified using the Bloom Filter incurs an avg. 411 ns extra processing time and accounts for 69.7% of RST packets. Remaining RST packets incur extra latency due to the scan operation on the timing wheel, which is necessary to identify the previous (unexpired) RST packet.

Similar Attacks: Similarly, we can detect TCP Incomplete Flows. Instead of looking for race conditions, we check if a SYN packet wasn't followed by DATA packet for some time [291]. SYN packets aren't blocked, as in forged TCP RST.

Port Scan Attacks

Attack: Port scan is a common method for discovering exploitable channels (i.e., open ports) on network servers[215]. SmartWatch partitions the monitoring between the host and the sNIC. The sNIC inspects and reports to the host the outcome of TCP three-way handshake (i.e., incomplete vs. established). The host tracks this over longer time scales, to classify if the remote node is a scanner or benign. But, no packets are forwarded to the host.

Detection: [215] describes a detection scheme using the number of failed connection attempts (i.e., failed three-way handshake) as an indicator to identify scanners. It determines the outcome of the ith connection attempt from remote server r as an indicator variable ϕ_i^r. [215] then runs a hypothesis test determining whether the remote node r is an attacker or not.

P4Switch Role: For the SmartWatch framework, the P4Switch measures the number of connection attempts. Flow subsets with significant number of connection attempts observed in the P4Switch will be forwarded to the sNIC. The P4Switch itself cannot conduct fine grained analysis as it cannot track the connection outcomes which requires flow-state tracking over long time scales, requiring significant amounts of P4Switch memory (see §2.2).

SmartWatch Role: FlowCache computes ϕ_i^r by tracking flow state on a per-packet basis. It waits a short period of time to see the responses for the SYN packet: a SYN ACK (successful); a RST (incorrect service); or no response (incorrect destination/port) from the destination. The flow record is pinned until ϕ_i^r is determined (e.g., 1 if it completes the three-way establishment handshake, 0 otherwise). The indicator variable ϕ_i^r for the flow is stored in the flow record and gets exported to the host. The host then classifies the remote node r as an attacker/benign using hypothesis testing[215].

Evaluation: We use NMAP [82] to generate scanning traffic with different scanning intervals. We merged this attack traffic with the Univ. of Wisconsin datacenter measurement trace [145]. Thus, the attack traffic is hiding in a much larger data stream. Larger scanning intervals become more difficult to detect (i.e., paranoid scanner). Figure 3.16 shows the detection rate in relation to different scanning delays comparing SmartWatch and (standalone) P4Switch. As SmartWatch carries out memory intensive operations, it can track protocol state transitions, allowing for it to compute the indicator variable ϕ_i^r and detect scans with long scanning intervals.

Similar Attacks: Detecting DNS amplification by computing the amplification factor $\frac{sizeof(response)}{sizeof(request)}$ instead of ϕ_i^r [216].

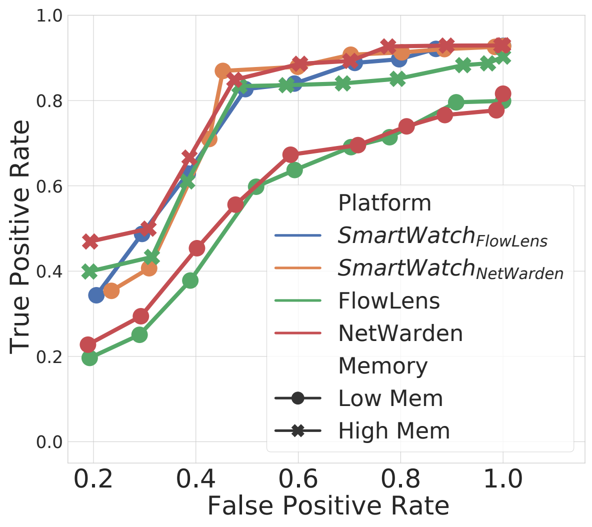


Figure 3.17: Covert Timing

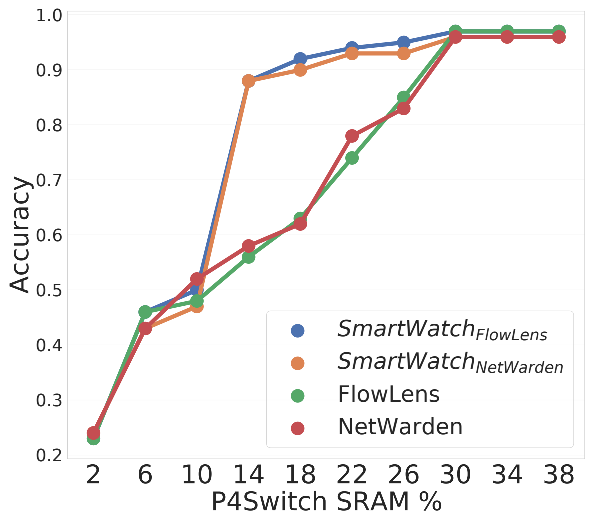


Figure 3.18: Website Fingerprinting

3.4.1 Reducing P4Switch memory pressure

We study two monitoring usecases in this section and detail how SmartWatch assists a P4Switch (NetWarden[329] or FlowLens[142]) consume less SRAM and control plane cores, aiding common forwarding operations in the DC.

Covert Timing Channel

Evasion Covert timing channels can exfiltrate secret data by modulating the inter-packet delays (IPDs) of network traffic, e.g., by using large (small) IPDs to encode ones (zeros)[329]. SmartWatch helps achieve the same True Positive and False Positive Rate (TPR and FPR) with 8 times less P4Switch SRAM occupancy to collect the IPD distribution. Further, no control-plane or co-located server resources are used.

Detection: Since the modulated traffic trace would have different IPD distributions (bi-modal) from those of usual traffic (normal), timing channel detectors look for statistical deviations (KS-Test[123]) between a given IPD distribution and a known-good distribution as obtained from training data[329]. We use a CAIDA[21] workload, where 90% flows are benign and the other 10% are modulated by the attacker to leak data. The modulation ranges from 1µs to 100µs.

P4Switch Role: Since iterative-refinement does not support IPD collection, we compare against two implementations of the P4Switch. FlowLens maintains a flow lookup table, assigning a flow offset to each flow ID. The flow offset locates the flow's set of bins to store the IPD distribution. NetWarden is similar, but instead of using k bins for each connection, it uses k CountMin Sketches to collect the IPD for all connections. The NetWarden dataplane consists of pre-checks that executes range checks on the IPD distribution. On the other hand, FlowLen's control plane reads the batch of collected data from the switch when a timer expires. We have extended these two P4Switch data structures

($SmartWatch_{FlowLens}$, $SmartWatch_{NetWarden}$) to forward packets to SmartWatch's sNIC subsytem when a pre-check[329] range query is satisfied.

SmartWatch Role: On the sNIC, we program the flow IDs that were determined suspicious on the switch (e.g., pre-check). For all the programmed flows, we maintain fine-grained bins (e.g., bin size = 1μs), intended to detect modulation between 1-100μs. Since the number of flows directed to the sNIC is small, this is feasible. On the sNIC's CME, when a timer expires we carry out the complete statistical test (KS-Test) within the sNIC and classify the channel as benign or if it is being modulated by an attacker to leak information. Flows programmed on the sNIC are pinned on SmartWatch's FlowCache to prevent evictions. The benefit of this sNIC-based co-design is less SRAM resource consumption on the P4Switch along with the complete elimination of the need to use CPU cores in the switch's control-plane (or co-located host) to run the statistical test.

Evaluation: P4Switches have a limited amount of SRAM (order of 100MB [257]) that is required for tables and registers [142]. FlowLens and Sonata occupy less than 40% and 20% (e.g., 8 of 32 Mb per stage) SRAM, respectively[142, 194]. As SmartWatch concurrently leverages both their data structures for the P4Switch deployment, it would leave only 40% SRAM total to support common forwarding behaviors, like access control, rate limiting or encapsulation. We show SmartWatch can have the P4Switch operate with substantially more, 75% SRAM available for general operations (instead of only 40%) while achieving similar True and False Positive Rate (TPR/FPR). We consider a high and low memory implementation of NetWarden and FlowLens. For high memory FlowLens implementation, we set the quantization level (QL - influencing bin size and number of bins) to 0, causing each flow to take up 3000 bytes. In contrast, we set the QL to 3 for the low memory implementation (376 bytes per flow). For NetWarden, the low-memory implementation uses a CountMin Sketch with 8 times less memory (0.5 MB as opposed to 4 MB) by altering the Sketch's dimensions. The sNIC fine-grained bins alongside the CME running the KS-Test ensures the complete statistics calculation is carried out for packets forwarded to the sNIC-host subsystem, attaining similar TPR and FPR, despite substantially lower SRAM occupancy (Fig. 3.17). In SmartWatch, when a timing covert channel is detected, we simply copy over the packet contents to the sNIC memory and create a new packet after a random delay. However, given the limited sNIC memory, when the sNIC 's buffers exhaust, we then do this on a host NF.

Website Fingerprinting

Evasion: Allows users to hide the destination address behind a proxy and the content of website visits from external observers using encryption [142]. SmartWatch helps achieve the same accuracy with 14% P4Switch SRAM occupancy compared to 30% needed for FlowLens and NetWarden.

Detection: Identify which sites are access by collecting the flows' packet length distributions (PLD) and feeding them to a Naive Bayes classifier[142]. We use widely used traces containing web page accesses over OpenSSH [346, 273].

P4Switch/SmartWatch Role: The P4Switch and SmartWatch play the same role as with the covert timing channel detection case, except we now collect PLD instead of IPD, and the sNIC CME runs a Naive Bayes classifier instead of KS-Test.

Evaluation: The CME classifies destination IPs as a "hidden destination address" or not. We calculate the accuracy using a Multinomial Naive-Bayes classifier, which leverages the PLD of the incoming and outgoing data of a connection as features [142]. Fig. 3.18 shows the website fingerprinting accuracy with respect to the P4Switch SRAM occupancy. With SmartWatch, we can bring down this occupancy to 14% from 30% and still achieve good accuracy (¿ 90%). SmartWatch sees a steep drop in accuracy around 10% SRAM occupancy because the range checks cannot identify what traffic needs to be sent to the sNIC-host subsystem (Fig. 3.18).

3.4.2 Traffic Analysis

In this section, we compare SmartWatch's FlowCache to common Sketch designs for traffic measurement over long-timescales (5 sec) and examine its ability to perform fine-grained traffic measurement ($< 200\mu s$) compared to a P4Switch.

Volumetric Analysis

For packet rates below the capacity of SmartWatch, we support completely lossless flow-logging (distinct from flow state tracking). When simultaneously running all monitor-

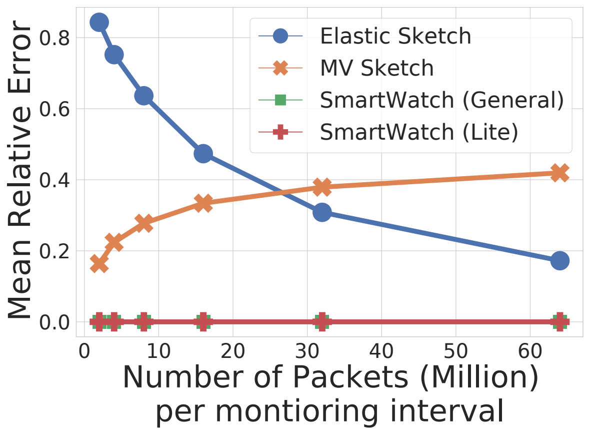


Figure 3.19: Heavy Hitter Detection

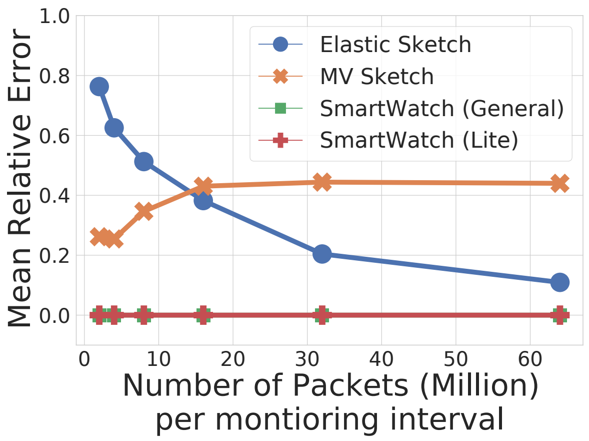


Figure 3.20: Heavy Change Detection

ing functions shown in Table 3.1, the packet throughput is not impacted by additional features since they consume minute fractions of the cycles compared to FlowCache (Table 3.1). The baseline FlowCache has flow-logging always active, so it can be used for heavy hitter, heavy change detection, etc. As each PME operates at 1.2 GHz[76], the small number of cycles used for additional monitoring features has only a small impact on packet processing latency.

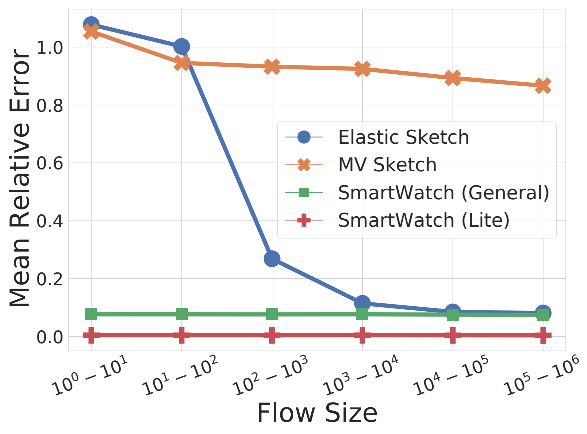


Figure 3.21: Flow Size Distribution

Throughput: Fig. 3.23 compares SmartWatch to Sketch based mechanisms. We implemented Elastic Sketch [331], Nitro Sketch [244], and MVSketch [312]. We compare SmartWatch running FlowCache in both the General and Lite modes: In this experiment, all the monitoring functions listed in Table 3.1 are simultaneously running in SmartWatch. We use Fig. 3.23 to guide us in only using 3 of the total 80 MEs for background processing (e.g., tracking the packet rate for switching between Lite and General mode). The remaining are PMEs (x-axis). The General Mode supports loss-less monitoring for packet rates below 30 Mpps. For higher rates, up to the max. of 43 Mpps, SmartWatch performs loss-less monitoring using the Lite mode. The control-loop (i.e., specific query adaptation with P4Switch) ensures that the traffic sent to SmartWatch is limited to what it can process. The only platform that yields higher throughput than SmartWatch is Nitro Sketch [244], but that is because it performs packet sampling to reduce the average memory operations per packet. CountMIN Sketch throughput is low due to multiple hash calculations per packet [244].

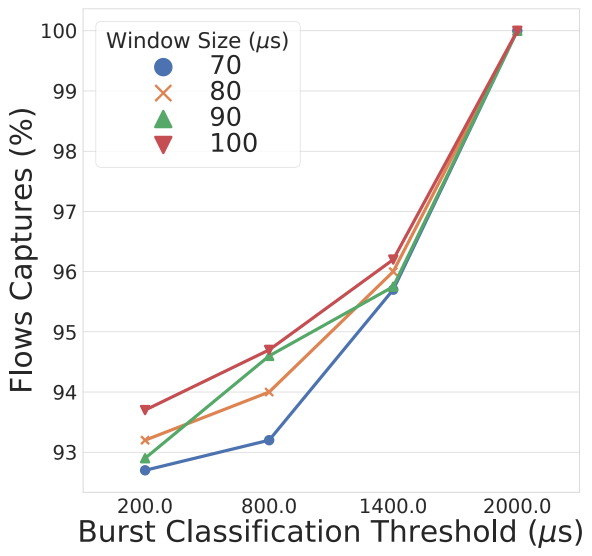


Figure 3.22: Microburst Error

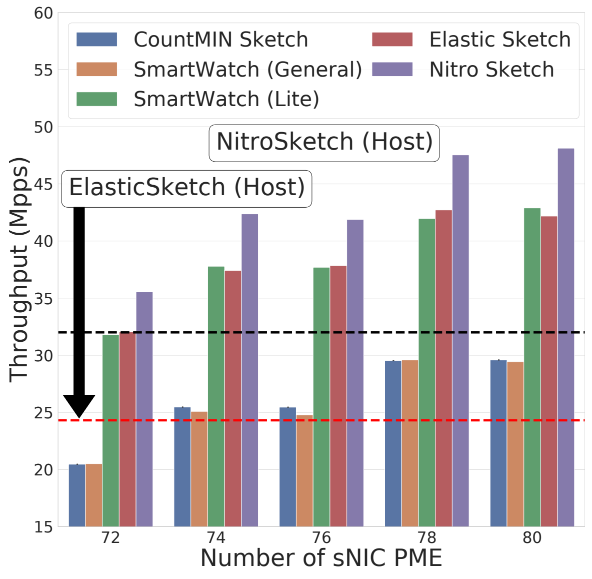


Figure 3.23: Throughput

Accuracy for Volumetric Analysis: For all experiments we use the CAIDA traces[21] from years 2015 to 2019, and reduce the packet size to 64 Bytes, to be replayed at 43 Mpps. First, we conduct heavy hitter detection. We use a predefined threshold for a heavy hitter (0.001% of total packets received in the monitoring interval) and vary the monitoring interval from 2 to 64 million packets. Second, we conduct heavy change detection. The predefined threshold for heavy change is 0.05% of the total changes across two consecutive intervals. Third, we compared the platforms based on the collected flow size distribution. For heavy hitter and heavy change detection, as the monitoring interval increases so does the error in Sketch based methods due to more hash collisions. SmartWatch's lossless monitoring approach instead evicts flow records to the host, preventing any accuracy drop. For heavy hitter and heavy change detection, both modes of SmartWatch have overlapping lines, with zero mean relative error. For flow size distributions, Elastic and MV Sketch prioritize the retention of heavy flows, causing the small flows to be inaccurate. SmartWatch, on the other hand, tracks all flows in a lossless manner and has lower error rate. However, for flow size distributions, the Lite mode has a higher accuracy compared to General mode as the latter does not sustain the high packet arrival rate in this experiment.

Micro-bursts

Anomaly: Micro-bursts are congestion events that (typically) last $< 200\mu s$, 40% of inter-burst gaps are $< 100\mu$s [339]. In this task the sNIC is responsible for reporting the culprit flows in a microburst without any approximation.

Detection: ConQuest [157] and BurstRadar [214] propose detecting micro-bursts when queuing delays go above an operator-specified threshold and then report the responsible flows.

P4Switch Role: In SmartWatch, the P4Switch identifies the link experiencing the microbursts and forwards the flow subsets that suffered the microburst event to the sNIC. Fine-grained analysis at sNIC accurately identifies the source of micro-burst (unlike the overestimation in [157]).

SmartWatch Role: FlowCache works with a linear array, L, of size 96MB storing the unique IP 5-tuple entries, to accurately report details of flows causing microbursts to the host, along with the packet count. We use a doubly-linked entry (i.e., reference from FlowCache entry to L entry and vice versa) to ensure connection uniqueness in L and to quickly locate flow entries in FlowCache from L. SmartWatch monitors the queuing delay on the sNIC as a trigger to activate micro-burst analysis. The PMEs calculate the difference between the current timestamp and the MAC ingress timestamp to compute per-packet queuing delay. When this delay exceeds the threshold, PMEs flag it, and generate an identifier for this micro-burst event. Subsequent packets update FlowCache and then L. Once the micro-burst ends, the CME is responsible for scanning L. The small size of L allows for rapid scanning for computing metrics of interest (within 200ms). A micro-burst ends when the queuing delay drops below a threshold. Following this, the contributing flows are identified by scanning the log, and all FlowCache records are allowed to be evicted to the host.

Evaluation: We use the Wisconsin trace [145], replayed at 10x the original rate, to detect and identify contributing flows of the micro-burst patterns as in [156]. We test each burst event by quantifying the flows present in the ground truth vs. the fraction in L, reflecting how SmartWatch identifies (and reacts to) bursts. Then, there is no error introduced in SmartWatch when reporting the flow responsible for the queue build-up. But, there may be false micro-bursts identified due to a conservative setting of the threshold (of queuing delay). The number of bursts estimated in SmartWatch vs. ground truth was higher by 1.32% to 8.23%, for queuing delay thresholds ranging from 2000μs to 200μs. Fig. 3.22 shows that as we reduce the queuing threshold to classify a micro-burst, we miss a fraction of the flows that were a part of the burst in the ground truth. A burst classification threshold of 200μs, captured 92.7% of the flows in the ground truth. But, a burst classification above 1700μs identifies all 100% of flows.

Similar Attacks: Worm detection where we lookup the hash of the combined payload and destination IP and check whether the worm signature match (i.e., stored in L)[302].

3.4.3 Effectiveness of Co-op Monitoring

Detection Rate

SmartWatch and Sonata allow processing traffic across multiple links incident on the switch, achieving an aggregate terabit scale monitoring. The standalone host will have the highest detection rate because of the highest degree of flexibility and most memory.

However, it is the least scalable option, as shown in Figure 3.4. Of all the stealthy attacks detected by the host, we show the fraction of such attacks that are detected by SmartWatch and Sonata in Table 3.3. The drop in detection rate for Sonata is because of the lack of fine-grained processing. In SmartWatch, the higher detection rate is due to fine-grained processing for flow-subsets. The slight reduction in detection rate for SmartWatch relative to standalone host is due to the attacks expiring within the P4Switch before those packets are forwarded to the sNIC.

Table 3.3: Detection rate relative to host

Attack	**Sonata**	**SmartWatch**
Slowloris	0.44	0.94
Zeek SSH Bruteforcing	0.24	0.79
Zeek Expiring SSL certificate	0.68	0.68
Zeek FTP Bruteforcing	0.25	0.81
Zeek Kerberos Ticket Monitoring	0.73	0.78
In-Sequence Forged TCP RST	0.11	0.80
TCP Incomplete Flows	0.84	0.93
Stealthy Port Scan	0.4	0.90
DNS Amplification	0.38	0.88
EarlyBird Detection Worms	0.59	0.70

P4Switch State

Figure 3.2 and 3.3 show the P4Switch State vs. the traffic volume directed from P4Switch to SmartWatch with CAIDA traces from different years [21]. Here, we study SSH Brute Forcing and Port Scan attacks . When the P4Switch switch redirects traffic to SmartWatch, SmartWatch identifies SSH authentication attempts that succeed and also the source IPs that are not IP scanners. These are benign flows and their packets no longer have to be forwarded to SmartWatch from the P4Switch. However, when installing rules in the P4Switch that whitelists benign flows, SmartWatch needs to be wary of the amount of state used in the P4Switch . To solve this problem, we borrow the 'hoverboard' intuition from Andromeda [167] . FlowCache in SmartWatch detects heavy benign flows and installs them in the P4Switch switch. Selecting heavy flows that are benign (e.g., top-k) as opposed to mice flows helps reduce the amount of redirected traffic to SmartWatch, with relatively few rules installed in the P4Switch. In Figs. 3.2 and 3.3 we see that there exists a knee, beyond which whitelisting flows does not reduce P4Switch state further.

3.4.4 Conclusion

In SmartWatch we introduce a monitoring pipeline suitable for data center anomaly detection. The subsequent chapters shall build upon SmartWatch for 1) carrying container placement based on measured container-pair affinities, 2) offloading the cellular data plane for carrying out mobility prediction, and 3) collecting traffic matrices for cellular-wide anomaly detection.

Chapter 4

pMACH: Graph based Container Placement

4.1 Introduction

Striking the right balance between conflicting scheduling requirements such as over-provisioning to satisfy an application's service level agreements (SLA) vs. tightly packing servers to save power in a data center (DC) can be challenging. Tightly packing containers is necessary to achieve high server utilization and power saving [316, 198, 240, 283] by turning off idle servers. In general, DCs operate at $\sim 20\%$ server utilization [253, 220, 247] and 10% network utilization[292, 198] in order to meet application SLAs. However, this results in high overall DC power consumption as more servers remain powered on.

While there exists some prior work to minimize both power and task completion time [345], they are not incremental, leading to a significant number of container migrations.

They ignore the cost of container migrations when adapting to workload changes or when the workload is consolidated to a smaller number of servers to reduce power consumption. Container migration (e.g., CRIU[65]) also results in downtime [37], and frequent migrations can adversely impact task completion times and are likely to result in SLA violations [314]. Thus, it is desirable to have a DC scheduler that simultaneously reduces power, task completion time, and container migrations and is also scalable to DC scales. The challenges are several - the need to operate servers efficiently [322], support fluctuating workloads [292], account for application container affinity [170], and account for migration overheads [37].

Today's DCs typically employ some form of heuristic-driven bin packing such as RC-Informed[165], Borg[317], pMapper[318] and others [63, 278, 12]. These solutions do not consider container affinity, potentially resulting in hosted cloud applications having higher latency [345] due to large inter-container communications. State-of-the-art task placement frameworks such as Borg [317] and RC-Informed [165] pack containers in highly utilized servers. Borg aims to reduce stranded resources while RC-Informed over-subscribes CPU resources at 125% [165], as a way of minimizing the number of servers deployed. To minimize power consumption, pMapper[318] determines the target utilization for each server based on the power model for the server. It then places VMs on servers using a bin-packing algorithm, trying to meet the target utilization on each server. E-PVM [135] places containers on the server with the lowest utilization, so as to leave large headroom for load spikes and achieve low task completion time.

Goldilocks [345] is another approach for scheduling latency sensitive tasks in a DC. It balances task completion time and energy, benefiting from placing frequently communicating containers together. However, it uses a centralized, periodic graph partitioning and scheduling policy using Metis [219], which does not scale to large DCs consisting of tens of thousands of servers. The change in container graph going from one epoch to the next may be incremental, but re-partitioning the entire graph, as in [345], results in a lot of container migrations. Vertices can be moved from one partition to another due to repartitioning. As vertex migrations correspond to container migrations, they are expensive and must be minimized. Furthermore, their work does not consider the overhead associated with transmitting the traffic matrix.

A DC cluster of several thousand servers, switches and links is typically broken up into smaller identical units. These units are called pods, comprising of several hundred servers along with the top-of-the-rack and aggregation switches. The DC network provides high-performance connectivity between all pods in the DC. We propose pMACH a Two-Tier distributed scheduling framework to adaptively 'right size' the DC by first considering a pod-level partitioning of containers, and then repartitioning the container sub-graph within a pod. pMACH schedules groups of containers (pMACH is generic, and may be used for scheduling VMs as well) of a partition on a server. It minimizes container migrations by adopting an incremental partitioning technique. pMACH's main focus is on achieving scalability using a Two-Tier partitioning algorithm, and executing the algorithm in an entirely distributed manner, unlike the centralized approach that has been the state-of-the-art. pMACH's strengths are:

- **Scalability:** pMACH can schedule a large number of containers over a cluster of ten thousands of servers in a relatively short time.

- **Multi-objective optimization:** pMACH balances between power consumption, task completion time, and task migrations.

- **Efficient:** pMACH only requires a small amount of processing resources (few cores on a select server in each pod of the DC), and uses network offload to relieve CPU cores of scheduler related activity.

- **Practical:** rather than assuming the container communication graph, pMACH collects the needed information in real-time on a Smart network interface card (sNIC).

pMACH significantly reduces task completion time as containers that frequently communicate with each other are placed together in the DC topology. Power saving is achieved by having a minimal number of servers, so that unused servers can be turned off. Container migrations are reduced by accounting for dirty vertices (vertices that are moved from their original group to another group in the graph), thereby minimizing downtime. We consider three mechanisms to perform hierarchical partitioning of the container graph, namely, ParMetis Base partitioning, ParMetis Adaptive partitioning[297], and Tabu Search. Both ParMetis offerings (e.g. Base and Adaptive) are highly parallelized. The difference between them is that Adaptive partitioning reduces container migrations and is faster to deal with workload variation. Tabu Search is a widely used meta-heuristic for graph partitioning as shown in [239, 290, 209] and allows us to provide a multi-objective cost formulation, accounting for container migration costs. Tabu Search however has poor scaling properties for larger graphs. Hence, we propose a hierarchical Two-Tier partitioning architecture that combines the advantages of both ParMetis Adaptive partitioning and Tabu search.

To obtain the container graph, we use a sNIC to collect the communication graph and provide it to the appropriate ParMetis graph partitioning worker nodes. This helps us save crucial CPU cycles. We use an efficient data stream summarization [330] to derive the edge weights with reasonable accuracy to allow frequently communicating container pairs to be placed together, to minimize task completion time.

Both testbed measurements and large-scale trace-driven simulations show that pMACH saves 13.44% more power compared to other scheduling systems. It speeds task completion, reducing the 95th percentile by a factor of 1.76-2.11 compared to existing container scheduling schemes. Compared to the static graph-based approach[345], our incremental partitioning technique reduces the migrations per epoch by 82%. Our major contributions include:

- A distributed scheduling system to scalably schedule containers across tens of thousands of servers.
- A Two-Tier scheduler composed of ParMetis Adaptive partitioning and Tabu Search to help reduce the partitioning time and container migrations.
- An efficient telemetry data structure on the sNIC to obtain the container graph in real-time.
- We implemented pMACH in a DC testbed (Cloudlab[175]) using 16 servers. We also implemented a large-scale flow-level simulation to demonstrate the scalability of pMACH.

4.2 Motivating Experiments

In this section, we carry out several experiments to measure the impact of container affinity, energy consumption and migrations on the performance of a DC. We also test how all these factors can be improved through graph partitioning.

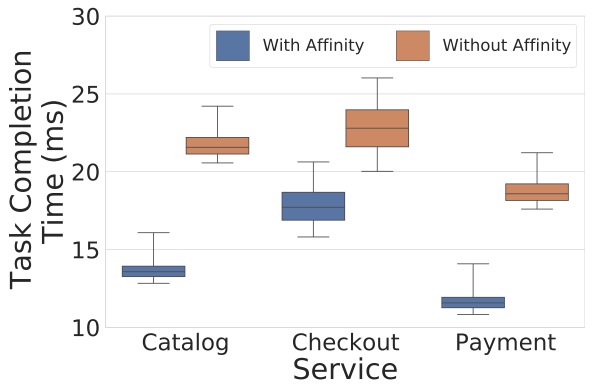


Figure 4.1: Affinity vs Response Time. The wicks represent 5th and 95th%tile.

Container Affinity: Previous work [345] shows that it can achieve 2.6 times better task completion time compared to alternatives such as E-PVM, RC-Informed, and p-Mapper by grouping frequently communicating containers together. To understand this in our context, we utilize a 10-tier Kubernetes microservice application provided in [39] on a testbed with four servers connected by an intermediate switch. First we let the Kubernetes scheduler decide the container placement by itself and in the second scenario we place the high affinity container pairs together (e.g. CheckoutService with PaymentService) by setting the nodeName [60] configuration. In Figure 4.1 we show that by exploiting affinities across three services (Catalog, Checkout, and Payment) the 95th percentile response time sees a speedup of 1.5, 1.2, and 1.5 times, respectively. This is a different workload compared to [345], resulting in a lower speedup. Clearly, container affinity must be considered while placing the containers, unlike other bin packing approaches such as E-PVM and RC-Informed.

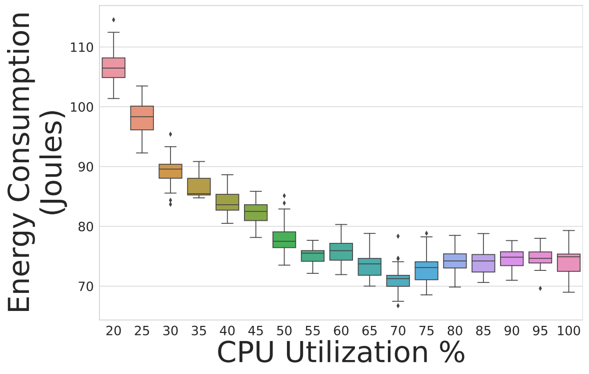


Figure 4.2: Peak Energy Efficiency at 65%-75% CPU utilization

Energy Consumption: Figure 4.2 shows a boxplot of the energy consumption in Joules, measured using RAPL [221] with respect to the load on the CPU. In this experiment, we use a 16 core Intel(R) Xeon(R) CPU D-1548 2.00GHz x86_64 architecture CloudLab [175] instance. We toggle the CPU utilization on all CPU cores of the instance (x-axis in Figure 4.2) and study the total energy consumed by the CPU package (y-axis in Figure 4.2) to complete a buffer I/O workload[64]. We observe that the energy consumed is lowest around 65% to 75% CPU utilization, displaying a 'U' curve for energy consumption. Similar observations have been made in the past [323, 345], generally referring to it as Peak Energy Efficiency, which is defined as the point achieving the maximum number of operations completed per watt. Such a strategy saves more total server power, while leaving a larger headroom to deal with instantaneous load fluctuations. The non-linear relationship between CPU load and power curve may be attributed to the cubic reduction in processing power with a linear reduction in performance for DVFS and dynamic overclocking, such as with Intel's TurboBoost [55, 202, 325, 323].

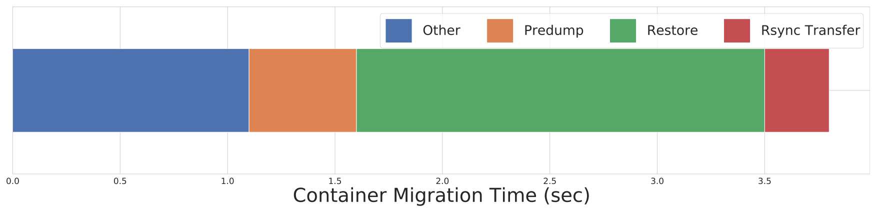


Figure 4.3: Migration Time Breakdown

Migrations: Consolidation can contribute to considerable power savings by turning off both servers and network switches and links. Maintaining affinity among containers is important to reduce communication overhead and reduce task completion time. The byproduct of consolidation and enforcing affinity are migrations. Containers will have to be moved at scheduling epochs, resulting in container migration overheads (both additional processing and communication) and undesirable downtime. Figure 4.3 shows that using CRIU [65] a Memcached container instance from CloudSuite[23] takes upwards of 3.5 seconds to migrate. Furthermore, the image predump, image transfer using rsync, and image restore require stopping application execution resulting in application downtime. Overall, it is important to minimize migrations, which was not considered earlier ([345]). The typical sources of migrations delays are 1) checkpoint/restore 2) writes to remote storage, and 3) scheduling delays [109].

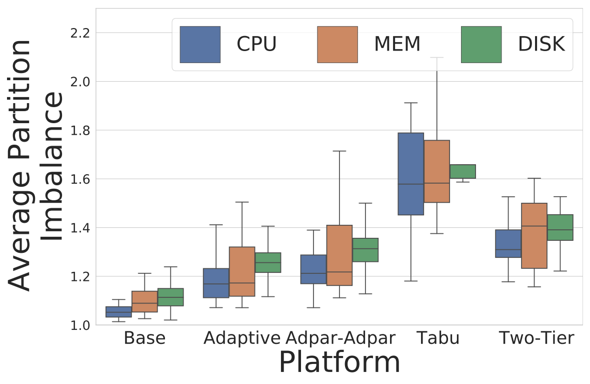


Figure 4.4: Partition Balance

Since we transform the container placement problem to a graph partitioning algorithm, we must consider how effective the partitioning is. ParMetis distributes vertices among cores (e.g. MPI workers) to parallelize the algorithm while balancing the load on the MPI workers. The technique can operate in two modes, namely, COUPLED or UNCOUPLED. These two approaches vary in partitioning time. In the COUPLED approach all vertices that belong to the same original partition are placed within the same CPU core before the next partitioning phase starts. The advantage of the COUPLED approach is that partitioning time is much lower because of increased local computation and reduced communication between the cores [95]. One constraint while running in the COUPLED mode is that the number of MPI workers must equal the original and target number of partitions. As we explore the partitioning time in this section by varying the number of partitions (e.g. 1024), we cannot physically have those many MPI workers, forcing us to operate in the UNCOUPLED mode. Later in the paper, we explain how we can scalably respect this constraint imposed by the COUPLED approach and leverage its speedup.

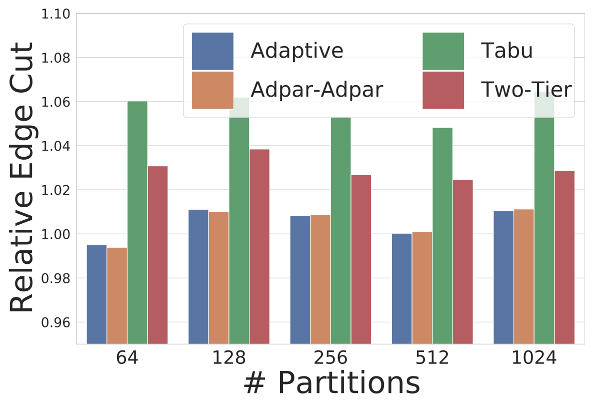


Figure 4.5: Relative Edge Cut

Graph partitioning imposes significant processing requirements with large graphs. Even though it is possible to have a 256-node MPI cluster [171] that can quickly partition such graphs, it would be impractical and expensive to have such a dedicated cluster in a DC. We perform the **graph partitioning in a distributed manner** in the DC by carefully designating CPU cores in selected servers in each pod of the DC. Taking advantage of the high-bandwidth links in the DC, and intelligently splitting it into a hierarchical solution, we are able to partition the large graph rapidly.

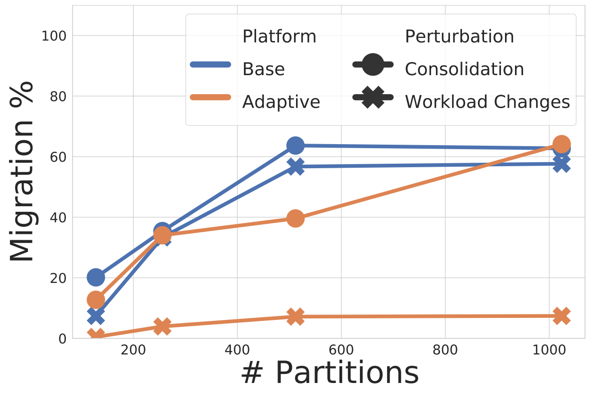


Figure 4.6: ParMetis Migrations

To manage the scale of the problem, we envisage partitioning the graph first at a coarser granularity (Tier-1). Then, we partition each subgraph (Tier-2). Adpar-Adpar refers to the partitioning result, where the first tier of partitioning employs the ParMetis Adaptive partitioning, generating X partitions. The second tier partitioning then internally partitions the graph, using ParMetis Adaptive partitioning, into $\frac{Target\ \#Partitions}{X}$ partitions. We also define Two-Tier, where the first tier carried out by ParMetis Adaptive partitioning and the second tier is carried out using Tabu Search. We carry out several experiments to compare these partitioning techniques.Explained in the next para. The partitioning was done using a single CPU core.

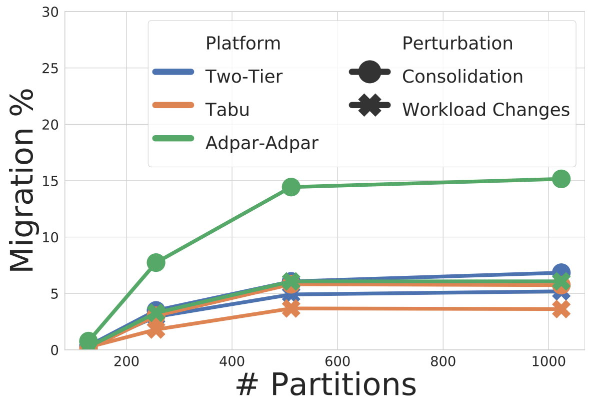


Figure 4.7: Migrations (non-ParMetis

Container Graph: We utilize a trace derived from the CDF of DC traffic, using a NS3-based DC simulation [27]. We obtain the communication matrix time series for different workloads. This yields both the container-pair connectivity and the edge-weights that represent the amount of communication per epoch. The container graph vertex weights are the percentage utilization, measured when running CloudSuite[23] container instances on

CloudLab[175] servers. Using the docker stat API[31], we measure the CPU, memory and disk utilizations. We use three different workloads (Memcached, Hadoop , and web-search using Apache Solr). The simulation generates the connectivity graph and the communication (i.e., edge weights), and the testbed measurements provide the utilization (i.e., vertex weights). We combine the two based on the application type (e.g. Memcached, Hadoop, Websearch). If an edge connects a Memcached client to a server, then the vertex weight is that of the Memcached client and server. This graph has 4 million vertices. We consider three metrics for partition goodness and relate them to the container placement.

Imbalance: Vertices represent the container resource requirements in a multi-dimensional form (e.g. CPU, MEM, DISK). Our goal is that each partition (i.e., server resource) should see close to the same, balanced, load. Therefore partition imbalance (e.g., deviation from mean partition weight) must be minimized. Figure 4.4 shows the imbalance, for every dimension, computed as the average absolute difference between each partition's weight and the mean partition weight. In this experiment we set the number of partitions to 1024. Figure 4.4 shows that ParMetis Base partitioning has the lowest partition imbalance. Adaptive partitioning is more imbalanced as it tries to minimize migrations (e.g. it prioritizes vertex moves that place the vertices back to their original partition). The hierarchical approach, Adpar-Adpar, has a similar imbalance as Adaptive partitioning as it relies on the same mechanism. Tabu Search results in higher imbalance as it penalizes vertex migrations more. Finally, Two-Tier (i.e., Adaptive at the coarse level and then Tabu Search) depicts slightly higher imbalance compared to adaptive partitioning as the second tier Tabu Search heavily penalizes container migration within the sub-graph.

Edge Cut: Edge weights represent the communication between the containers, and the edge cut denotes the communication intensity between partitions. As partitions represent the placement of containers on a physical server in our case, the more we reduce the edge cut, the more we take advantage of container pair affinity with frequently communicating containers being placed closer together. Figure 4.5 shows the edge cut relative to base partitioning, $\frac{e}{e_{BP}}$, where e denotes the edge cut provided by the partitioning algorithm and e_{BP} denotes the edge cut afforded by Base partitioning. Adaptive partitioning and Adpar-Adpar depict very similar edge cut and sometimes even lower than Base partitioning. As Tabu Search heavily penalizes vertex migrations that are required to minimize edge cut, standalone Tabu search has a higher edge cut. Finally, Two-Tier improves over Tabu search by leveraging the lower edge cut across coarse-grained partitions generated by Tier-1.

Migrations: The graph is repartitioned periodically to take into account the change in workload that may result in different assignment. We express migrations as a percentage of the total number of containers running during a given time interval. Figure 4.6 and 4.7 shows the percentage of containers migrated as a consequence of two different types of perturbations: namely, **workload changes** caused by partitioning every 10 minutes considering the graph snapshots at that time; and **consolidation** where the number of target partitions are reduced by one to save energy. For hierarchical partitioning schemes, as the Tier-2 sub-graph partitions will correspond to servers, consolidation only reduces the target number of partitions in the Tier-2 step. The target number of partitions for the Tier-1 partitioning remains unchanged as the coarsened partitions correspond to pods as described below. Figure 4.6 shows that Base partitioning is ill-suited, similar to Metis used in [345], for DC container placement, since the number of migrations is very high. This is because Base partitioning does not take the previous partitioning result into account and only tries to aggressively minimize edge cut and imbalance. However, Adaptive partitioning, shows very few migrations for workload changes as it takes the previous partitioning solution into account. But, it fails to yield the same low number of migrations when there is consolidation as compared to the original partitioning (e.g., previous epoch's partitioning result). It uses migrations to mitigate either the poor balance or edge cut when one partition is taken away.

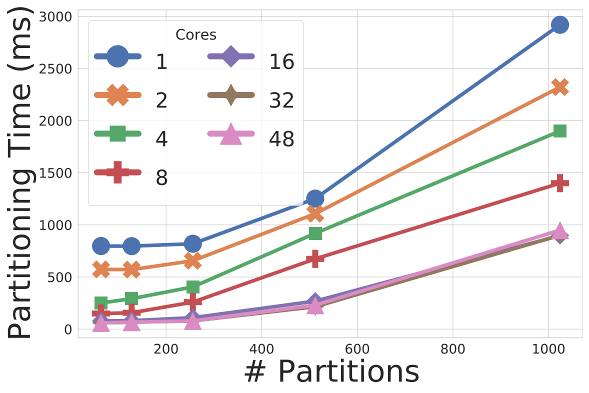


Figure 4.8: ParMetis Base

Figure 4.7 shows that none of the approaches using ParMetis (e.g., Base or Adaptive) performs as well as Tabu Search when considering consolidation in terms of vertex migrations. Tabu Search's custom cost function allows us to provide a higher penalty for migration. With hierarchical approaches, the first tier is not impacted by consolidation because the target number of partitions is fixed, but the the number of partitions for the second tier may reduce. Adpar-Adpar suffers because the second tier Adaptive partitioning has too many migrations. The Two-Tier scheduler drastically outperforms other approaches, except Tabu-Search, when considering consolidation. This is because the target number of partitions for Tier-1 is unchanged and Tier-2 heavily penalizes migrations by assigning a higher weight to migrations in its formulation, while also including edge cut and imbalance.

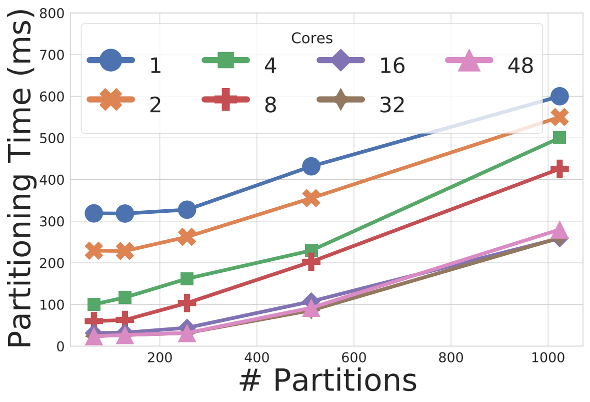


Figure 4.9: ParMetis Adaptive

4.3 pMACH: Distributed Container Scheduling

We need to rapidly partition container graphs with minimal communication overhead. End-hosts are responsible for transmitting sub-graphs to designated partitioning workers **(pWorker)** in the pod, avoiding communication across pods (e.g., which is unavoidable in the centralized approach). We do this in the background, with the partitioning task in epoch t operating over data gathered in epoch $t-1$. Next, the entire container graph is partitioned using Adaptive Partitioning. It's highly parallel and focuses on container-pairs that communicate across neighbouring pods. Adaptive partitioning also minimizes edge cut in the output partitioning. Lastly, to factor the cost of migrations, since Adaptive Partitioning is poor at consolidation, we run Tabu Search on the same pWorker, which is slower but operates on a smaller graph. Altogether, the distributed architecture for graph partitioning is a key innovation that makes wide-scale, real-time scheduling that is adaptive to workload changes, practical.

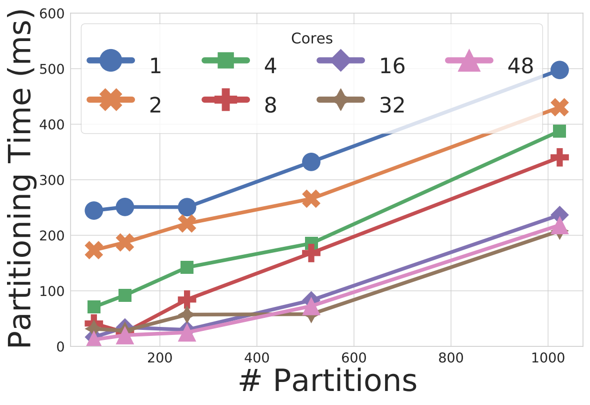


Figure 4.10: Adpar-Adpar

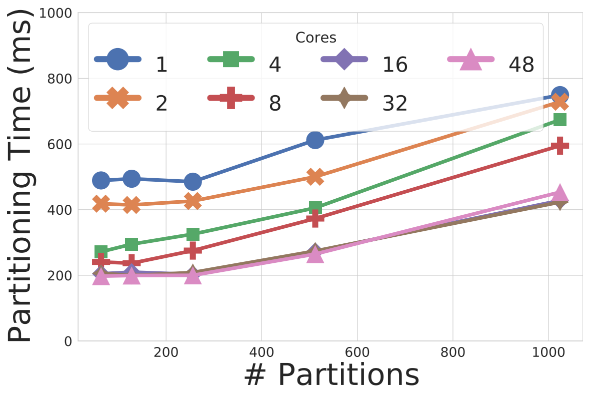


Figure 4.11: Two-Tier

A scheduler in a DC is typically responsible for the placement of tasks among a large number, typically of the order of 10,000 servers (e.g., as in Google's DCs [317]). Furthermore, as in a fat-tree DC network (DCN) architecture (e.g., [271]), we consider a k-ary fat tree network with $\frac{k^3}{4}$ end hosts. In our context that would roughly translate to 11,664 servers distributed among 36 pods handled by one scheduler. To help manage this scale, factoring in the complexity of the graph partitioning algorithm, pMACH uses a Two-Tier hierarchical scheduler, as shown in Figure 4.12. pMACH develops a two-level graph partitioning algorithm, namely the ParMetis Adaptive partitioning as the first tier and Tabu Search as the second tier. Tier-1 is responsible for partitioning the container graph over pods using the large-scale, scheduler-wide communication graph as input. Tier-2 is responsible for intra-pod scheduling using a smaller pod-wide communication graph as input. This design is inspired by DC such as [177, 317] that use pods as a logical and physical clustering of DC resources, creating a modular solution that can adapt to different-size DCs.

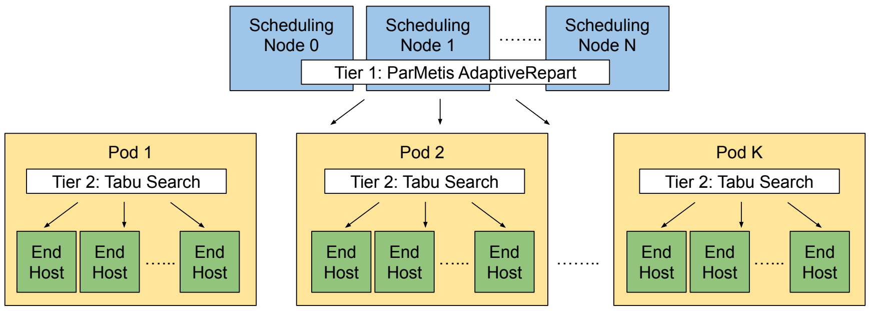


Figure 4.12: Two-Tier Hierarchical Scheduler

Adaptive partitioning of a graph with 4 million vertices, with a target of 36 partitions, takes 101ms compared to 5619ms with Tabu Search. This speedup holds even as the number of partitions increase. As the Tier-2 input container sub-graph is small, partitioning takes at most 71ms. Therefore, intra-pod scheduling is managed by Tabu Search for scheduling the containers over fewer, $\frac{k^2}{4}$ (324) servers. Since pMACH employs an epoch based scheduler, we place new containers according to the Best-Fit algorithm[16] with a 70% cap on utilization (i.e., to operate at Peak Energy Efficiency).

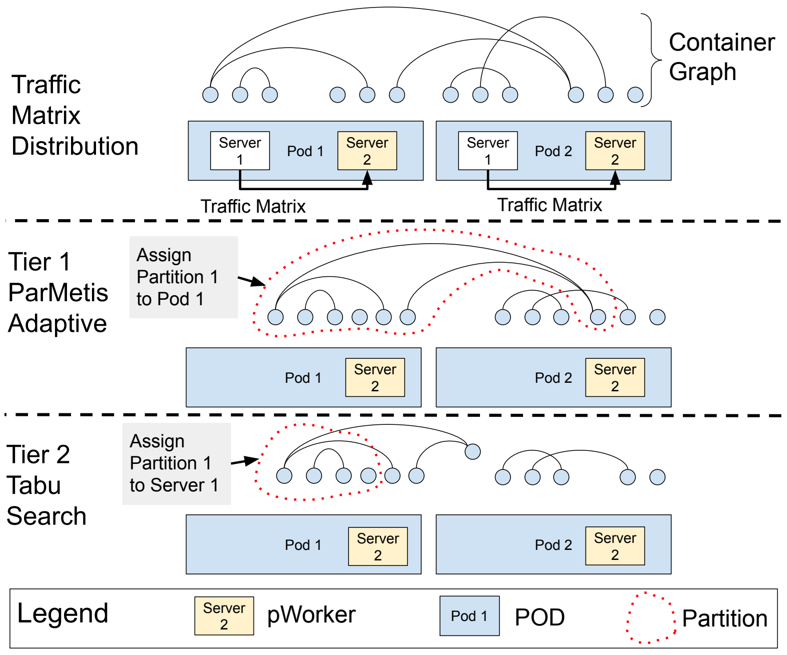


Figure 4.13: Distributed Scheduling

In Figure 4.13, we show the three stages in our distributed container scheduling technique to schedule containers on to servers. The end hosts transmit the communication graph to the designated pWorker within the pod for the purposes of graph partitioning. Next, the Tier-1 ParMetis Adaptive partitioning program is invoked. We designate one pWorker (one core on one select server) per pod to partition the graph. If a scheduler's domain consists of k pods, then Adaptive partitioning will take the container graph as input and generate k partitions. In the next stage, the same pWorker in each pod concurrently run independent instances of the Tier-2 Tabu Search optimization problem where the input graph consists only the containers running in this pod plus the graph changes made by Tier-1 scheduler (i.e., inter-pod migration). Once the serialized Tabu Search procedure for the pod terminates, we migrate containers corresponding to dirty vertices from the original partition to the new partition, where the partition ids correspond to server ids. Some containers may have to be moved to another pod as per the Tier-1 partitioning output.

$$\min \sum_{1 \leq i < j \leq n} |E^c_{i,j}| \tag{4.1}$$

$$W^1_d \approx W^2_d .. \approx W^n_d, \ where \ d \in D \tag{4.2}$$

$$\forall P_i, \sum_{j \in P_i} A^c_j \leq B^c_i, \ where \ 1 \leq i \leq n \tag{4.3}$$

The Tier-1 scheduler runs ParMetis adaptive partitioning using the objective function to minimize edge cut (Eq. 4.1). It seeks to ensure the partition weights are almost balanced (Eq. 4.2). Eq. 4.3 guarantees that the partition resource demands do not exceed the pod capacities. Here, n represents the number of partitions, $E^c_{i,j}$ is the sum of edge weights between partition i and j, W^i_d represents the d^{th} dimension weight of partition i, where $d \in D$. The dimensions D include CPU, memory, and disk. B^c_i represents the capacity of pod i. P_i is the container group assigned to pod i and A_j represents the resource demands of container j. This cost formulation does not explicitly factor the cost of migrations, but it tries to minimize migrations by leveraging the previous partitioning result along with the assumption that graph changes are small.

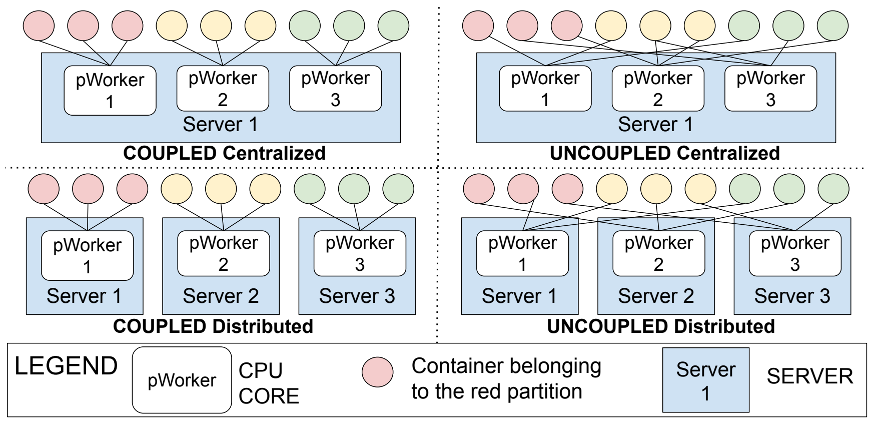


Figure 4.14: Vertex distribution over pWorker

Our MPI program can run on a centralized server, where the computation is distributed over CPU cores or in a distributed manner. According to Figure 4.14, pMACH can distribute vertices (e.g., containers) over pWorkers in four different ways (e.g., centralized/distributed, each being COUPLED/UNCOUPLED). We measure pMACH's partition time overhead for different implementations of "Two-Tier" as shown in Fig. 4.14. Partitioning time is the dominant time for scheduling, followed by the migration related down-times, whose impact on the task completion time is studied in section 4.4.1. In Figure 4.15, we see the partitioning time breakdown for Two-Tier, composed of the latency to run ParMetis Adaptive repartitioning (Adaptive) and Tabu Search on a container graph that contains 6 million vertices. We experiment with a distributed approach (16 servers each with 1 core) and a centralized server based approach (1 server with 16 cores). For each, we explore the COUPLED and UNCOUPLED approach.

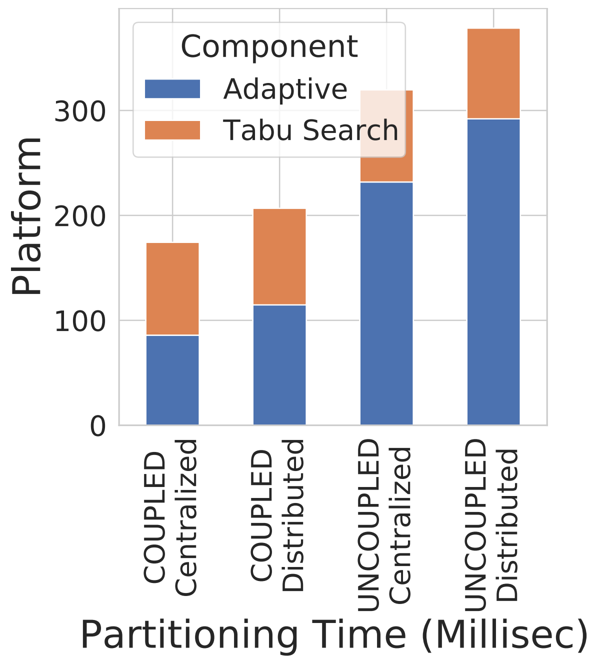


Figure 4.15: pMACH Partitioning Time

The COUPLED approach is on average 1.8 times faster than the UNCOUPLED approach. By switching from the UNCOUPLED to COUPLED Tier-1 partitioning scheme, we see a significant speedup because Tier-1 is responsible for processing a large-scale container graph, dominating the total partitioning time. We choose the COUPLED over UNCOUPLED approach for its lower partitioning time. In a centralized approach, the data collected across end-hosts in all pods must be transmitted over the network to a single server (or a group of servers). Typically, communication spanning pods is more expensive since more links are traversed. The distributed approach, reserves one pWorker per pod, resulting primarily in intra-pod communication (from end hosts in the pod to the pod's designated worker). Therefore, the COUPLED distributed approach reduces the communication overhead across pods. Hence, we opted for the COUPLED distributed approach over the COUPLED centralized approach. The COUPLED-centralized scheduler is 1.18 times faster compared to the COUPLED-distributed approach. This is because workers communicate over shared memory in the COUPLED-centralized scheduler as opposed to network links. We tolerate this partitioning time penalty to minimize cross-pod communication during the traffic matrix distribution phase. *In summary, we select the **COUPLED distributed** scheduling mechanism as it is scalable and faster.* The complexity analysis for Tier 1 and 2 can be found [218] and [279], respectively.

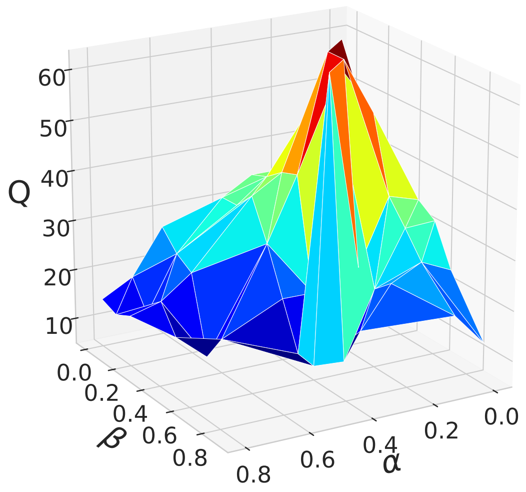


Figure 4.16: Coeff selection

We partition the intra-pod container sub-graph at the Tier-2 scheduler once the Tier-1 scheduler has computed and transmits the changes to the container graph to each of the pods. The intra-pod container graph is the graph involving the containers (i.e., the vertices) within that pod. The Tier-2 scheduler runs a sequential Tabu Search algorithm using an objective function as shown in Eq. 4.4.

$$\begin{aligned}
\min \quad & \alpha \times EC + \beta \times IB + \gamma \times DV \\
EC \quad & = \sum_{1 \le i < j \le n} |E^c_{i,j}|/E \\
IB \quad & = \frac{1}{n|D|} \sum_{d \in D} \sum_{1 \le i \le n} |W^i_d - \overline{W_d} \ \ |/\overline{W_d} \\
DV \quad & = \sum_{v \in V} I_{P_{t-1}(v) \neq P_t(v)}/|V|
\end{aligned} \tag{4.4}$$

$$\forall Q_i, \sum_{j \in Q_i} A^c_j \le S^c_i, \ where \ 1 \le i \le n \tag{4.5}$$

The objective function consists of three parameters (e.g α, β, and γ) that act as weights to the edge cut (EC), imbalance (IB), and dirty vertices (DV). The indicator variable $I_{P_{t-1}(v) \neq P_t(v)}$ equals 1 when the vertex is assigned to a different partition as compared to the original partition, otherwise 0. To ensure edge cut, imbalance, and dirty vertices are dimensionless, we normalize the multi-objective function using the following: E represents the total edge weight and $|V|$ represents the total number of vertices. Equation 4.5 ensures that the resource demands do not exceed server capacity. S^c_i represents the capacity of server i. Q_i is the container group assigned to server i. By setting a large value for γ relative to α and β, we penalize vertex migrations more. Therefore, even in the event of consolidation, container migrations are low.

We employ Tabu Search for iterative refinement of our graph partitioning solution, following [209]. Each solution (e.g., candidate) provides a cut, which assigns containers to servers. We start with an original partition. At each iteration, we compute the neighborhood solutions (e.g., current solution + a candidate vertex migration). From this, we filter away Tabu moves and then select the vertex move that satisfies Eq. 4.4. Next, the current solution is updated and the selected move is stored in a Tabu list for a predefined tenure (e.g., next t iterations). The Tabu list ensures that a local minimum is not returned by discouraging the search from coming back to previously-visited solutions. Under certain circumstances a move that is in the Tabu list can be selected, which is referred to as the aspiration criterion. In our technique, a Tabu move will be selected if it yields a solution that is better than the best solution so far. The Tabu Search program stops if a fixed Max Iterations value is reached or if there was no improvement in b iterations. Next we describe how we select the coefficient for Tabu Search. First, we compute different partitioning outputs of the same graph by varying α and β (e.g., $\alpha+\beta+\gamma=1$). We then summarize the partition quality Q as $\frac{1}{N(EC)}+\frac{1}{N(IB)}+\frac{1}{N(DV)}$ (see Eq. 4.4) where N scales the individual quantity to the range $[0,1]$ across all partitioning outputs of the same graph. Fig 4.16 shows how Q changes with α and β. This plot summarizes multiple partitioning outcomes over several input graphs, using the median Q. We observe (α, β, γ) equal to (0.234, 0.512, 0.254) maximizes the median value of Q and use it for subsequent experiments.

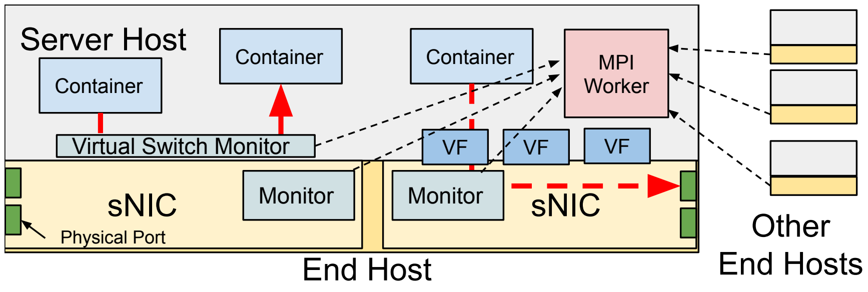


Figure 4.17: Pod Architecture: Monitoring and Graph Partitioning

Unlike previous approaches for scheduling and power management in DCs where the communication graph is assumed to be known (or ignored), pMACH explicitly accounts for it and utilizes a sNIC to collect the container-level communication graph. The intra-host container communication graph is collected by the software switch running on the host[245] while the inter-host container communication graph is collected by the sNIC, which are a often used in today's DC networks [185]. We assume every node contains a sNIC to offload the traffic matrix collection, reducing the overhead on the host while collecting the information at link speed [242, 185].

Figure 4.17 shows the pod-architecture. All servers in the pod are responsible for generating the communication graph. The container resource utilization weights along with intra-host and inter-host container communication weights are transmitted to the designated server within that pod for graph-partitioning. The vertex weights are derived using the docker stat API[31]. The edge weights are measured in the sNIC and software switch. As communication weights help characterize affinity, estimating them helps us trade-off between space (memory and bandwidth) vs. partitioning quality. Reducing bandwidth consumption helps reduce interference for latency sensitive user traffic. Reducing the partitioning quality degrades application task completion time. It is possible to seek a balance between space and application performance. We describe four data plane algorithms below that we evaluated to collect the communication graph edge weights. Based on the analysis in section 4.4.2, we choose Elastic Sketch.

Confluo: Uses a data structure called Atomic MultiLog that supports highly-concurrent read-write operations [222]. Since all the edge weights are accurately recorded, Confluo occupies a lot of space, but has no error estimating edge weights.

CountMIN Sketch: A compact space data structure for summarizing data streams. It uses hash functions to map container-pair communication events to frequencies (e.g., edge weight) at the expense of overcounting due to collisions [163].

Elastic Sketch: It consists of two parts: a “heavy” part recording high-affinity container pair communication weights and a “light” part recording low-affinity container pair communication weights. The heavy part is a hash table while the light part is Count-MIN Sketch [330].

Nitro Sketch: It combines a Count Sketch[154] with a sampling strategy to reduce the number of communication edge weight update operations. It is the fastest, but results in higher error estimating communication weights [245].

The Tier-1 ParMetis scheduler equates pods to partitions while the Tier-2 Tabu Search scheduler equates servers to partitions. This is possible because both ParMetis and Tabu Search support heterogeneous partitions. In ParMetis Adaptive partitioning, the user can supplement the *tpwgts* argument to regulate the fraction of vertex weight that should be distributed to each partition (e.g. pod) for each dimension[95]. Likewise in Tabu Search, vertex moves that violate server capacity can be deemed as illegal moves (e.g., non-neighboring moves). Therefore, both ParMetis and Tabu Search can capture heterogeneous pod and server capacities. Different CPU speeds (e.g., GHz) can also be captured by adjusting vertex weights.

4.4 Evaluation

We evaluate pMACH on a testbed implementation and compare it with a number of alternatives published in the literature, viz., E-PVM[135], MPP[318], Goldilocks[345], Best-Fit[16] (e.g. Borg stand-in) and RC-Informed[165]. We also do limited measurements with a sNIC. Finally, we carry out large scale simulations to predict the performance of DCs.

Testbed: We run the Cloudsuite benchmark, which has 432 containers on a testbed containing 16 servers (each with 20 cores) on Cloudlab[175]. The container graph is obtained from running the CloudSuite[23] workload components Memcached, Hadoop MapReduce, and Apache Solr (an equal number of instances of each). The vertex weights, depicting server resource consumption is measured using the docker stat API[31]. CloudSuite benchmarks comprise multiple containers with communication between each other, and the graph's edges characterizes this connectivity. We consider the provisioning of new services where containers are created and killed in the trace at different points in time based on real production datacenter traces [27]. We used the IPTraf monitoring tool [57] to measure the communication rate between pairs of containers, which is used as the edge weight. IPTraf monitors the virtual port for each container. Similar to [198, 345], servers with no active containers are turned off to reduce power consumption. The epoch length is 10 mins. We empirically determined that scheduling decisions with epoch lengths < 10 mins, made it prone to transient changes, unsuitable for partitioning.

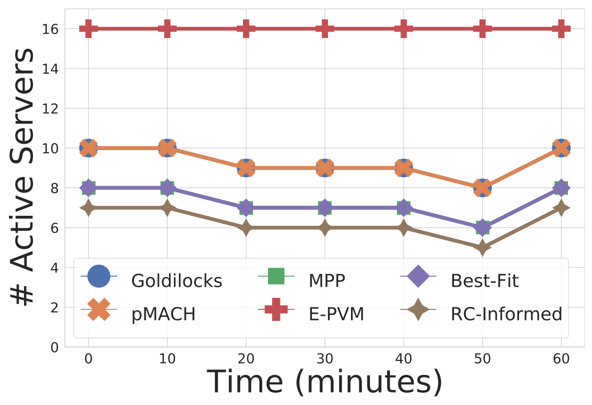


Figure 4.18: Active Servers

4.4.1 Testbed Results

Partitioning Quality: In Figure 4.18 we plot the number of active serves over time for different scheduling techniques. It is observed that E-PVM occupies the largest number of servers, as it places containers in the least utilized server. The bucket based RC-Informed technique yields the lowest number of active servers due to CPU over subscription. Best-Fit packs containers at 95% utilization, yielding fewer servers compared to Goldilocks and pMACH that pack at 70% utilization to operate at peak energy efficiency. MPP tries to minimize power consumption by greedily increasing the target utilization on the server. Its curve overlaps with that of Best-Fit in Fig. 4.18. In Figure 4.19 we see that the power consumption with Goldilocks and pMACH is the lowest, a reduction of 13.44% compared to RC-Informed. RC-Informed consumes less power than E-PVM, Best-Fit, and MPP because it occupies fewer servers. Goldilocks and pMACH consume less power by running at utilizations that result in peak energy efficiency in terms of tasks completed for the energy consumed. Packing at 70% utilization also provides more head room to sustain CPU utilization spikes. In these experiments we have three different container workloads running (e.g. Memcached, Hadoop, and Web-Search). In Figure 4.20 we show the 95th percentile task completion time for the Twitter Memcached Workload by varying the RPS between 44K to 440K across the entire testbed, which effectively varies the resource utilization for the containers. The task completion time is measured at the Memcached-client as it issues get and set requests to the Memcached-servers. As expected, Goldilocks and pMACH show a substantial improvement, with the 95th percentile task completion time speedup of 2.011 as it takes account of the container pair affinity.

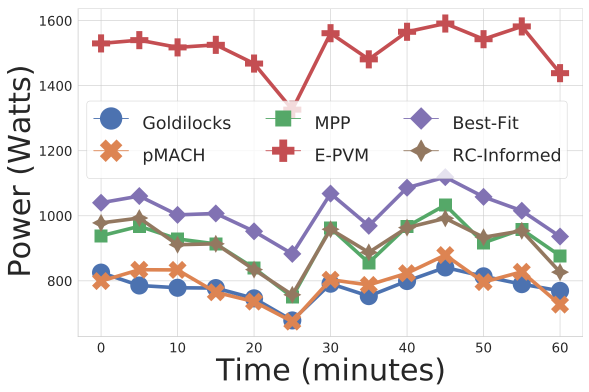


Figure 4.19: Power Consumption

Goldilocks did not do incremental graph partitioning. At every epoch, Goldilocks re-partitioned from scratch and assumed the containers migrate to their new location, while ignoring the overhead of migrations. Here, we incorporate the impact of container migrations of pMACH in the application-level metric of task completion time. In this experiment, containers are migrated as per the scheduler's decision, using CRIU. Much of the scheduling activity is performed concurrently with task execution. But, the downtime due to migrations directly impacts the task completion time. The average downtime is 2.39s. However, the parallelism in performing 44 migrations with all 16 servers results in 6.06s downtime. While this downtime could be reduced by live migration optimizations, e.g., [327], it is crucial to reduce the amount of migrations, as we strive with pMACH. pMACH still achieves close to Goldilocks' completion time by minimizing dirty vertices. Figure 4.21 shows container migration events as a percentage of total number of containers. Unlike Goldilocks, MPP and pMACH, the other scheduling mechanisms distribute containers only when they arrive, and have no migrations. Goldilocks has average 51.8% migrations per epoch. But, pMACH only has 8.83% migrations, benefiting from its incremental partitioning approach. MPP has the least migrations as it migrates containers only when the server's utilization deviates from the target utilization. This results in poorer task completion time and higher power consumption than pMACH.

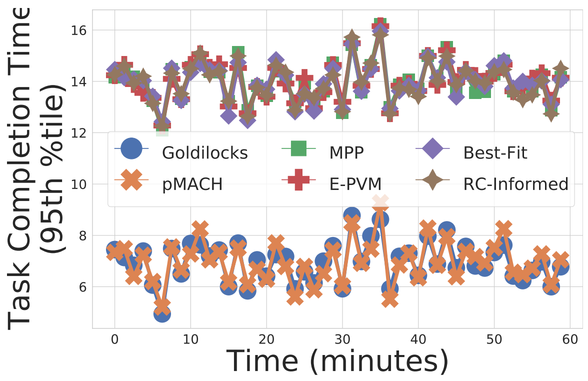


Figure 4.20: Task Compl. Time (Memcached)

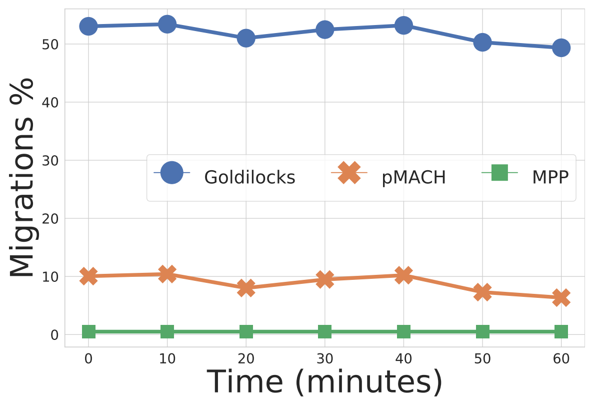


Figure 4.21: Container Migrations

Energy Consumption: We now verify if indeed running at 70% utilization is efficient for overall DC energy consumption. Figure 4.22 shows the energy consumption, measured using RAPL[221], for 16 servers against different levels of packing capacities using pMACH's scheduler. Consistent with previous work on Peak Energy Efficiency [322, 323] we observe that the total power consumption is the lowest when the utilization per server is capped at 70%.

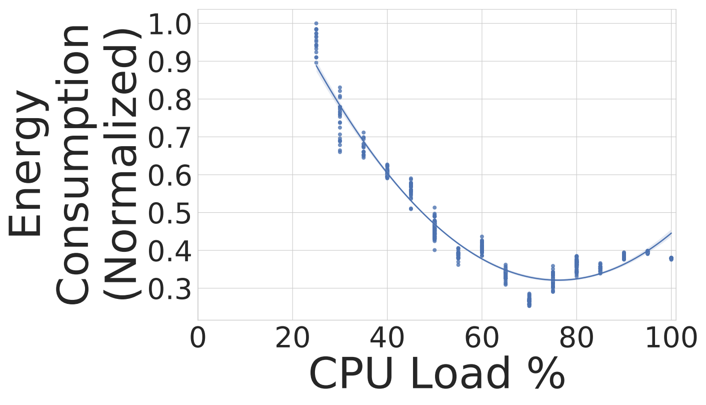


Figure 4.22: CPU Load vs. Normalized Energy

4.4.2 Distributed Data Collection

We study different ways to collect the communication graph on the sNIC using the trace to derive the partitioning quality in section 4.4.1. This section emphasizes the reduction in network bandwidth for transmitting the graph and the resulting tradeoff in application performance. We replay the packet trace over a sNIC, transfer the data collected from the sNIC to the host, and partition the communication graph using the data collected from the sNIC as well as the intra-host container communication obtained by the cluster metrics collection framework. All the approaches other than Confluo [222] use some sort of probabilistic approximation, allowing them to have lower packet processing and data transmission overhead, but are more prone to estimation error. In all the data structures the tuple key is the source and destination container IP. We also use a local testbed consisting of server with 20 Intel Xeon 2.20GHz CPU cores and 256GB memory running Linux (4.4.0-142). It has Netronome Agilio LX 2×40 GbE sNICs which have 8GB DDR3 memory and 96 flow processing cores.

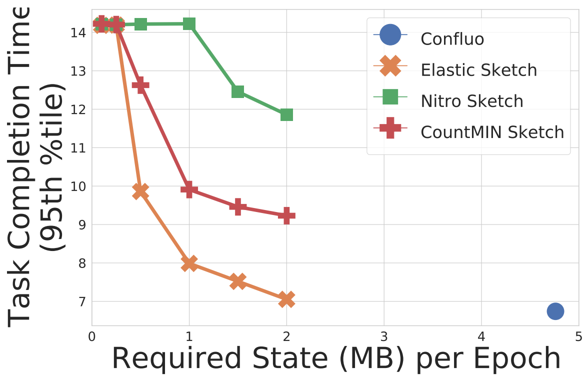


Figure 4.23: Sensitivity to amount of state

Figure 4.23 shows the task completion time vs the overall memory usage at all the servers in a 16 server implementation. We run the pMACH graph partitioning technique to determine the container placement, similar to the testbed experiments of section 4.4.1. We compute the 95th percentile of task completion time for the Memcached workload. Memory usage reflects the amount of data that must be transferred over the wire. Confluo consumes the most amount of memory compared to all other platforms as it must track all edges, but also yields the lowest task completion time. Nitro Sketch and CountMIN Sketch result in high task completion times because of the edge weight overestimation that results in many of the low-affinity containers pairs being scheduled together. These low-affinity containers compete with container-pairs that actually have high affinity and thus result in overall poor placement. We observe Elastic Sketch has a negligible increase in task completion time compared to Confluo but with a substantial memory usage reduction of 2.38 times. This is because Elastic Sketch prioritizes the retention of heavy flows, in turn preserving container-container affinity.

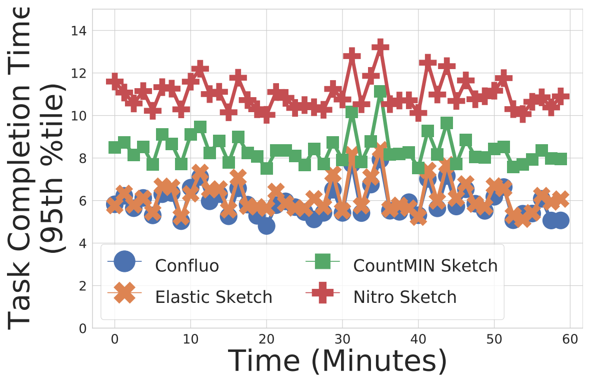


Figure 4.24: Time Series

Figure 4.24 shows the result of the experiment in the form of a time series, where the memory state of the data summarization approach is transferred every epoch. Confluo transfers 4.76MB of data per epoch, while the other dataplane algorithms transfer only about 2MB data per epoch. Clearly, Elastic Sketch, despite using 57.98% less bandwidth, has low application task completion time, matching Confluo.

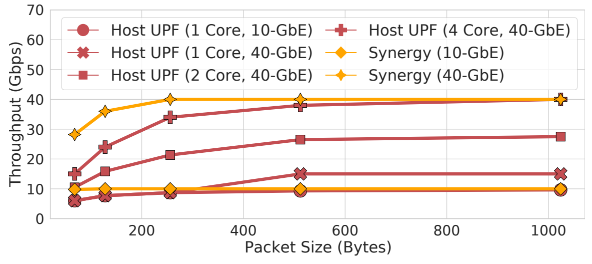


Figure 4.25: Throughput

4.4.3 Large Scale Simulation Results

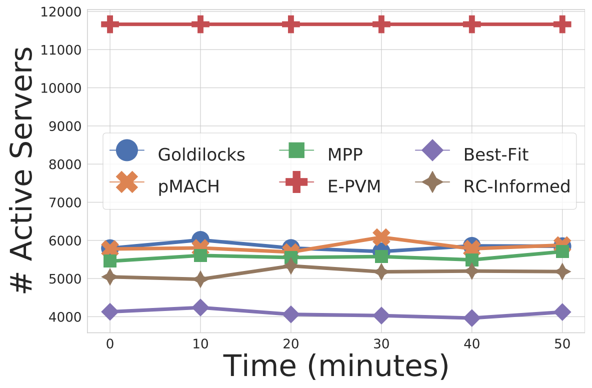


Figure 4.26: Active Servers

We also performed a flow-level, large scale simulation with a 36-ary fat tree topology, with $11,664$ servers and a total of $104,958$ containers (e.g., targeting 20-30% utilization for baseline E-PVM[253, 220, 247]). We utilize a trace from a NS3-based DC simulation [27]. We obtain the communication matrix time series for different workloads (i.e., Memcached, Hadoop, and Microsoft Web Search). This represents the container graph edge weights and connectivity. We then merge the graph edges with container graph vertex weights per application (e.g., Memcached). We ran actual instances of containers from the Cloudsuite [23] benchmark on CloudLab[175] and measured resource demands to get the vertex weights.

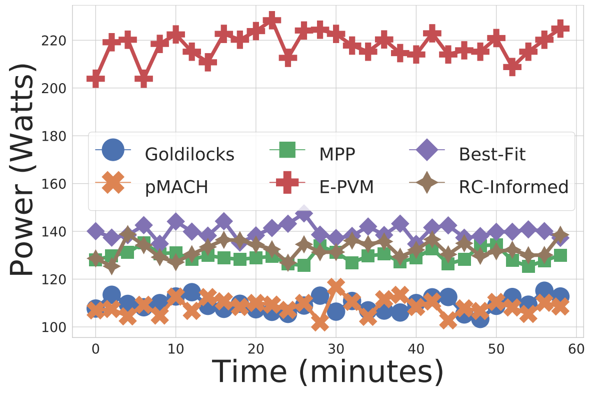


Figure 4.27: Power Consumption

In Figure 4.26, we see E-PVM always chooses the least utilized server, but all 11664 servers are active. RC-Informed requires the least number of servers because it permits oversubscription of server resources. Best-Fit requires a slightly higher number of servers, because it sets target of 95% utilization. Similarly, MPP does not oversubscribe server resources, but greedily selects the servers to provision, based on the power model. Goldilocks and pMACH use more of servers as they operate at a lower target of 70% utilization. But,

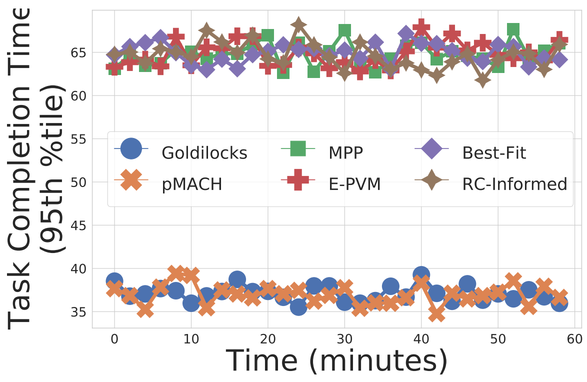


Figure 4.28: Task Completion Time

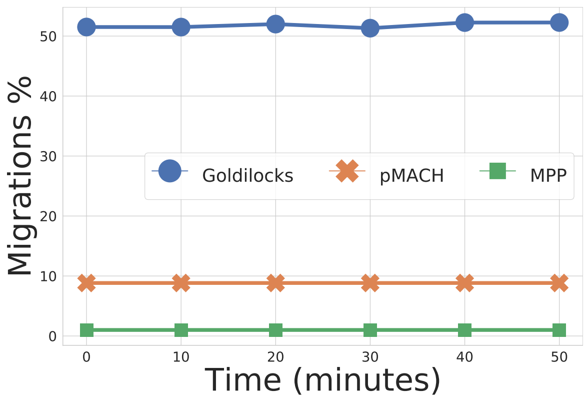


Figure 4.29: Container Migrations

as we see in Figure 4.27 for the power consumption, using the power model we measured on CloudLab[175], Goldilocks and pMACH consume the least energy as they operate at peak energy efficiency. In Figure 4.28 we compute the 95th percentile task completion time for Apache Solr search engine in the testbed and vary the search request rate. We use the processing time distribution for search queries based on a benchmark measurements made in CloudLab[175]. We use packet latency obtained from measuring the container pair communication latency in our testbed using the Arista 7050SX-72Q switch. We ignore the queuing delays based on typically low link utilizations in DCs ($< 25\%$) [146]. By accounting for container-pair affinity, the 95th percentile task completion speeds up by 1.76x. Lastly, in Figure 4.29 we see that the migrations reduce from 51.70% with Goldilocks to 8.7% with pMACH. MPP's migrations remain below 1.3%, but consumes 19% more power and 76% longer task completion time. This is because MPP does not operate at peak energy efficiency and ignores container affinity.

4.4.4 Conclusion

In pMACH we capture the container pair affinities and use that to better container placement, resulting it lower energy consumption and lower task completion time. pMACH shall also be the basis for our query processing engines introduced in our third contribution, namely 5GDMon, that provides a cellular-wide monitoring infrastructure.

Chapter 5

Synergy: Faster Handovers using Mobility Prediction

5.1 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, 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[92]. 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 [36] 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, contribute to constraining the performance[274]. Since the 5GC supports a large number of UEs connected to multiple base stations, facilitating a wide range of critical applications and services[259], achieving high performance for the 5GC is key. Utilizing network acceleration to implement 5GC NFs can substantially improve throughput.

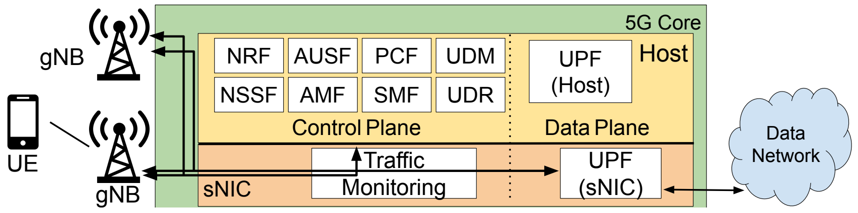


Figure 5.1: Synergy 5GC Architecture

Another avenue for network acceleration is using programmable switches. While programmable switches (P4Switch) for the 5GC data plane packet processing show promise[249], they have two drawbacks. First, P4Switches do not have large buffers or the ability to hold packets as required by the 5GC's UPF [249]. To overcome this feature gap, [249] 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 [258]) limits its ability to support flow state tracking, even though it can forward traffic at very high rates [274]. 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. 5.1. 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[274]. 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 [94] 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 [108].

Beyond implementing the core functions for a UPF on the sNIC to be compliant with the 3GPP specification [2, 3], 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 handover (as in Sec. 9.2.3.2.2 in [6]), 'Smart buffering' of packets within the UPF has been proposed as a way of reducing the latency in L^{2}5GC [210] and CleanG [260]. 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[210] and Free5GC[36].

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× when buffering within the sNIC instead of the buffering within the host. Compared to other sNIC-based flow state management approaches such as DeepMatch[208] and SmartWatch[274] 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 . 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[274] (§5.3.1). Synergy increases its capacity by 44×, to support up to 12 million flows (§5.2) 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[81] 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[341]. 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.

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. 5.1) 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 [2, 3]. The control plane must setup a unique tunnel endpoint identifier (TEID) to tunnel dataplane traffic between the gNB and UPF using GTP[249]. 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[249].

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)[125]. 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[249].

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[124, 249].

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 [260, 249].

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[210]. 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 [49]. These require completing handovers as quickly as possible. Along with the many control message exchanges [25], 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.

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[36], with the dataplane (UPF) implemented using the DPDK[190] libraries.

UPF Programmability: The UPF must dynamically update FARs and PDRs along with their priorities (see §5.1). 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[329]. 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 [249] 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.

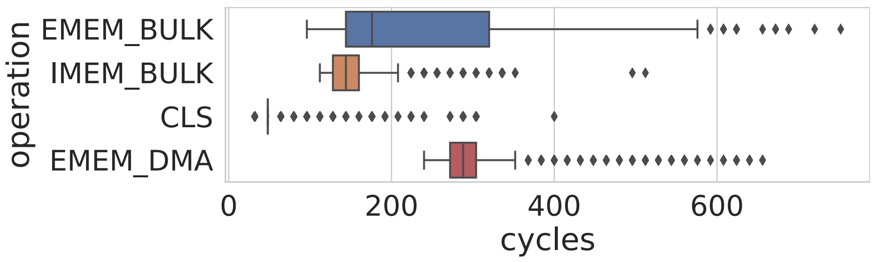


Figure 5.2: Memory Latency sNIC

5.2 Design

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 [208]. Switching contexts takes 2 cycles (non-preemptive threads scheduling). The sNIC can be programmed in both Micro-C, which is an extended subset of C-89, and P4 [274]. 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 [208].

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[208]. Table 5.1 shows the memory hierarchy on the 40 GbE sNIC [208], in terms of capacity and usage. Fig. 5.2 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 5.1: 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)

We leverage the buffering mechanism we develop on the UPF to improve the performance of handover and paging. The sNIC is more efficient for packet processing compared to host-based systems. However, others have taken the approach of just buffering

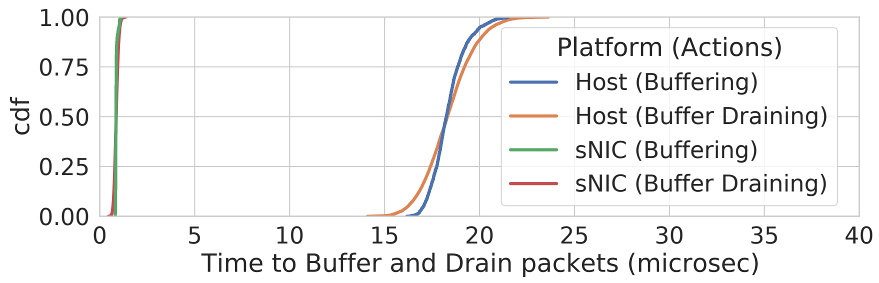


Figure 5.3: Latency to Buffer/Drain Packets

the packets on the host while carrying out the rest of the features on a P4Switch [249]. 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* [101]. 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. 5.3 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).

On the sNIC, we load a P4 match action table for PDR matching. However, the capacity of this table is only 64K entries[196], forcing us to resolve P4 table misses on the host. Other platforms such as T4P4S[319] and NetFPGA[75] that are based on other vendors have similar capacity constraints for individual P4 tables[196], 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.

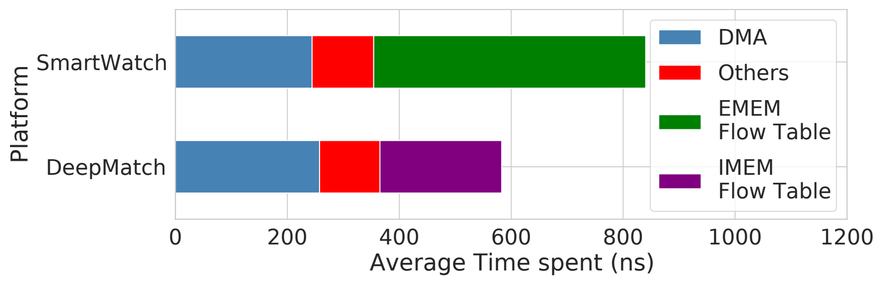


Figure 5.4: Time spent to buffer packets in sNIC

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[274] and DeepMatch[208], 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 (§5.2) because of its large DRAM capacity.

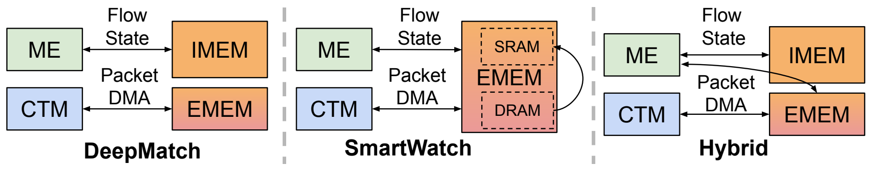


Figure 5.5: Hybrid Memory Architecture

Fig. 5.4 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. 5.2. 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. 5.5 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. 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 [274].

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 [208]. 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. As shown in Fig. 5.6, 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. 5.7

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.

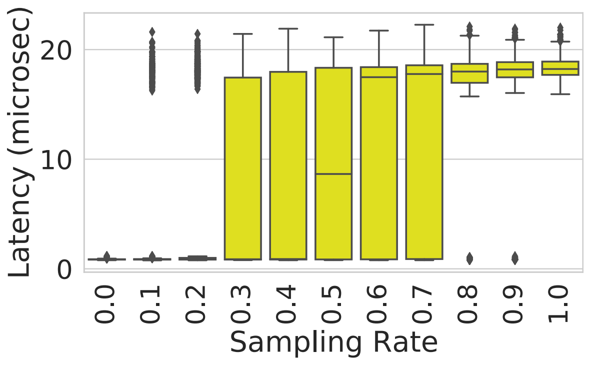


Figure 5.6: Latency vs. Sampling Rate

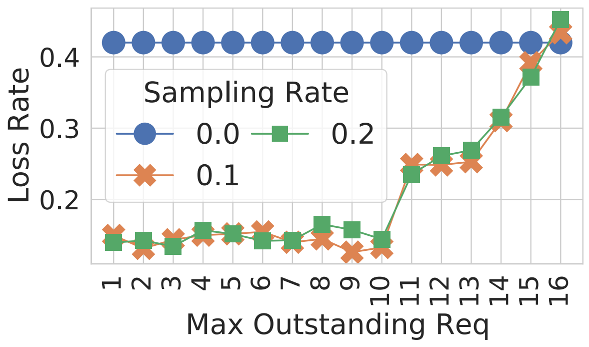


Figure 5.7: Loss vs. Max outstanding DMA

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[81]. 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 [43]), which is triggered by the source gNB based on measurement reports[71].

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[127]. 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[127]. 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 [255].

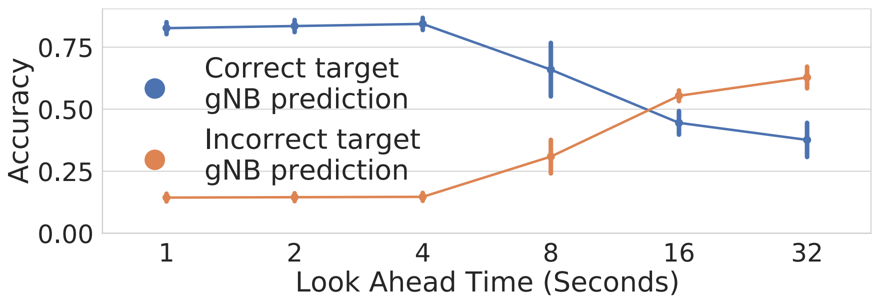
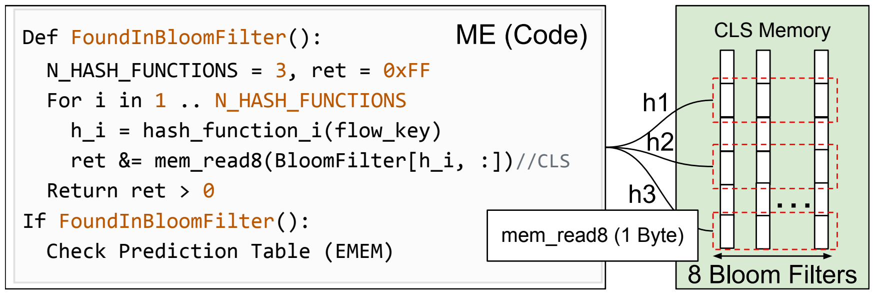


Figure 5.8: Prediction Accuracy

We use a SUMO-based [104] vehicular mobility dataset [114] for subsequent experiments. For our experiments, we use the mobility predictor introduced in [232] since it is decentralized and shown to be highly accurate [232, 255]. 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[232].

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[126]. Fig. 5.8 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.



```
Def FoundInBloomFilter():
  N_HASH_FUNCTIONS = 3, ret = 0xFF
  For i in 1 .. N_HASH_FUNCTIONS
    h_i = hash_function_i(flow_key)
    ret &= mem_read8(BloomFilter[h_i, :])//CLS
  Return ret > 0
If FoundInBloomFilter():
  Check Prediction Table (EMEM)
```

Figure 5.9: Bloom Filter Architecture

We take the N2Handover codebase[42] and divide it into two phases corresponding to prediction and handover. The prediction phase is primarily concerned with prepopulating 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[193] 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 5.2), 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_read*8 (e.g., 1 Byte) can access all parallel Bloom Filters at the same hash index. Fig. 5.9 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[150].

On replaying the SUMO trace[114] 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.

We now describe the Synergy architectural design for the 5G UPF within the sNIC (shown in Fig 5.10) 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 §5.2 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.

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 §5.3.3. 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 §5.2 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 §5.3.3) 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.

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 [208]. 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.

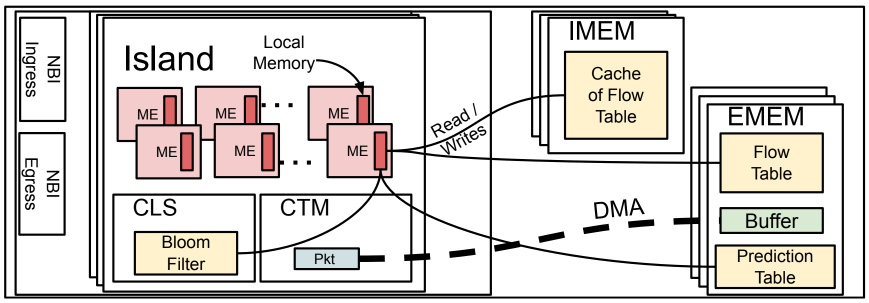


Figure 5.10: Synergy Architecture

Fig. 5.11 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* [77]’ provided by the Netronome sNIC to drop traffic exceeding the limits.

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[275] 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[210].

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 [207]. 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

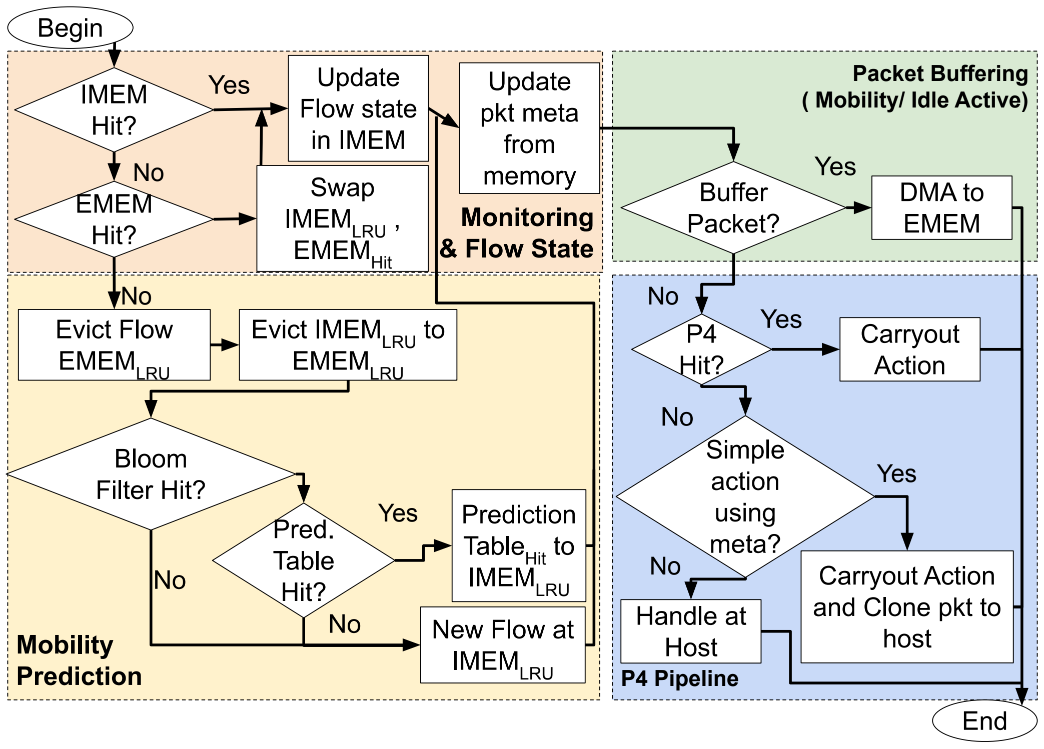

Begin
IMEM Hit?
Yes
Update Flow state in IMEM
Update pkt meta from memory
No
EMEM Hit?
Swap IMEM$_{LRU}$, EMEM$_{Hit}$
Monitoring & Flow State
Packet Buffering (Mobility/ Idle Active)
Buffer Packet?
Yes
DMA to EMEM
No
Evict Flow EMEM$_{LRU}$
Evict IMEM$_{LRU}$ to EMEM$_{LRU}$
P4 Hit?
Yes
Carryout Action
No
Bloom Filter Hit?
Simple action using meta?
Yes
Pred. Table Hit?
Yes
Prediction Table$_{Hit}$ to IMEM$_{LRU}$
Carryout Action and Clone pkt to host
No
No
No
New Flow at IMEM$_{LRU}$
Handle at Host
Mobility Prediction
P4 Pipeline
End

Figure 5.11: Synergy Flow Chart

uses the file prefix for the huge page it access to, which is provided to it by the primary DPDK process [32]. Location reports are encrypted to maintain user privacy. The crypto module of our target sNIC device[26] decrypts these location reports. The required keys are provided to the sNIC by the NF that is co-located on the same host.

Due to Synergy's buffering and monitoring capabilities, the throughput achieved with Synergy is not as high as what is achievable with P4Switch [249]. 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.

Here we study the generality of our UPF implementation on the Netronome Agilio LX sNIC [8], 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 [274]. Given the similarity of the architectures and capabilities, Synergy should in principle be portable to any of the other sNICs.

5.3 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[285] and a SUMO-based vehicular mobility dataset[114]. 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[232].

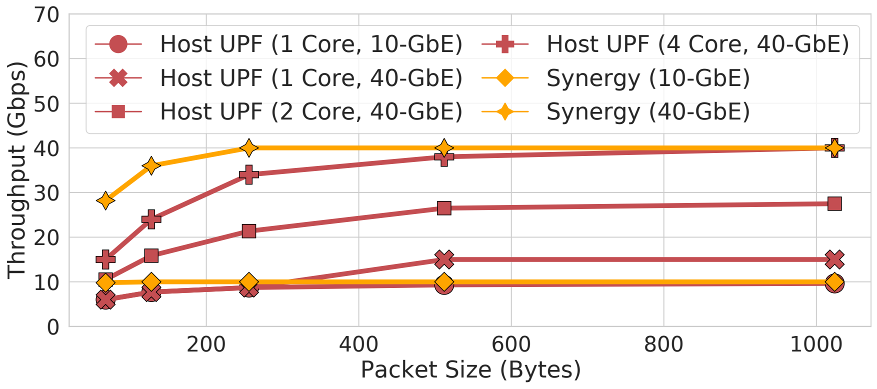


Figure 5.12: Tunneling Throughput

5.3.1 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 5.2). Recall “SRAM only” and “DRAM w/aux. cache” are derived from the SmartWatch and DeepMatch designs, respectively. For the host UPF, we use Free5GC[36], with UPF implemented on top of DPDK[190]. 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. 5.13, 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$\times$ 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. 5.14. 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.

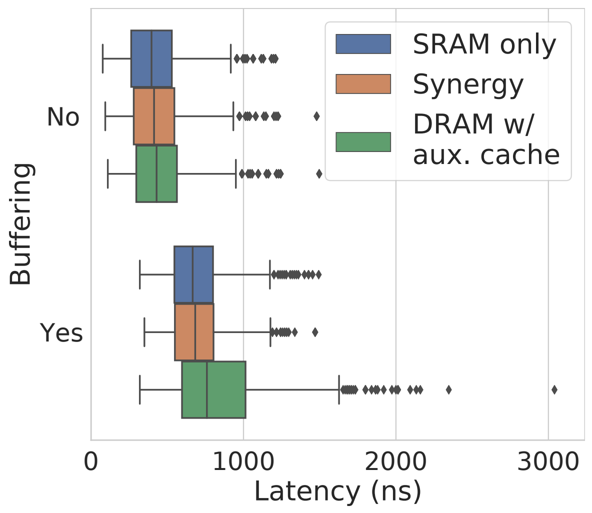


Figure 5.13: Latency Distribution

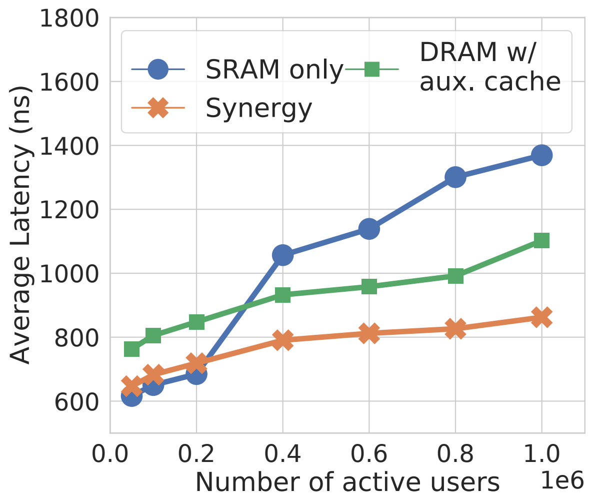


Figure 5.14: Latency wrt users

5.3.2 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

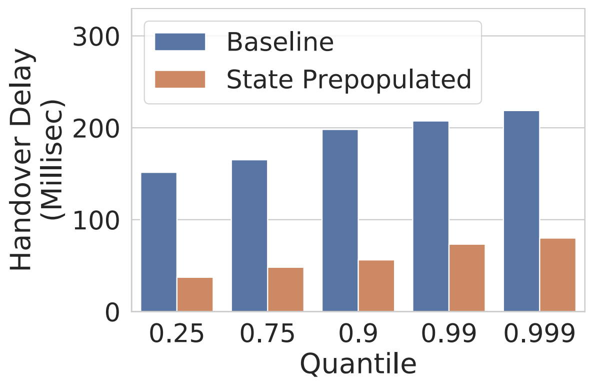


Figure 5.15: Speedup via state prepopulation

DPDK [36] 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[114]. Fig. 5.16 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. 5.16), 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. 5.17 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

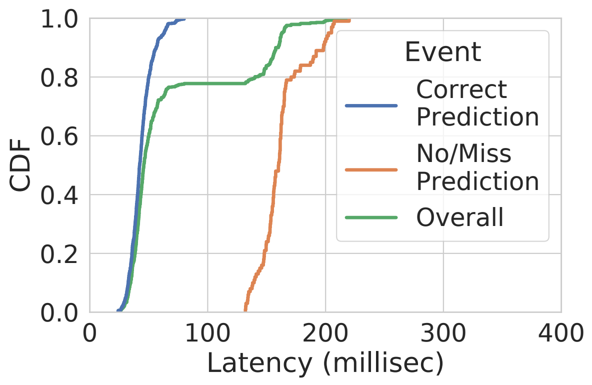


Figure 5.16: Synergy latency distribution

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. 5.2). 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. 5.2). 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.

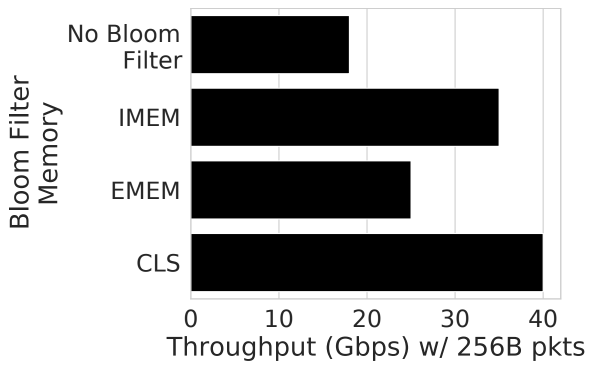


Figure 5.17: Throughput Analysis

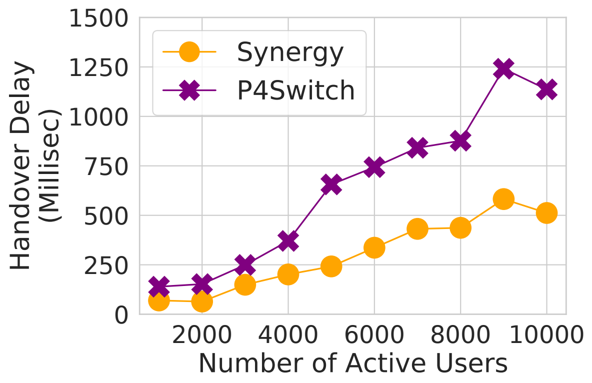


Figure 5.18: sNIC/P4Switch P4-based UPF

5.3.3 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 [329], using the same values for P4 table and rule updates as with the sNIC UPF. Fig. 5.18 shows the handover delay with respect to the number of active users in a SUMO-based vehicular mobility trace [114]. 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.

5.3.4 Conclusion

In Synergy we offload a critical data plane component to the SmartNIC to capture mobility patterns and use it towards mobility prediction. 5GDMon is a distributed implementation of Synergy that detects anomalies in the constrained cellular environment.

Chapter 6

5GDMon: Monitoring Cellular Networks

6.1 Introduction

Cellular Networks have become predominantly IP-based data communication infrastructures. As such, cellular networks are also increasingly vulnerable to attacks, just as any other data communication network. Cellular networks have limited resources, especially radio resources, that must be managed carefully to provide the best quality of experience for as many active users as possible. Resource management and ensuring the cellular network's infrastructure and users are protected against attacks requires monitoring network traffic as in traditional IP networks.

Limited radio resources require careful resource allocation and scheduling to meet the quality of experience expected by users. The Open Radio Access Network (O-RAN) architecture is seeking to evolve the cellular RANs to virtualized RANs with software-based components connected via open interfaces, with the use of ‘intelligent controllers’ (such as the non-real-time and near-real-time RAN intelligent controllers) to help make informed decisions based on data collected from the operation and traffic demand observed on segments of the network[129, 281]. The O-RAN software framework splits the RAN processing into several sub-components, with a Central Unit (CU), a Distributed Unit (DU), and a Radio Unit (RU)[201] together performing the processing that a traditional monolithic 5G base station (gNB) would perform (see Fig. 6.1). The RAN intelligent controllers (RICs) are tasked with streaming telemetry from the RAN so that they can provide intelligence to a Service Management Orchestrator (SMO) to deploy control actions and policies for resource allocation and management of the traffic by the CU, DU, and RU of the O-RAN environment. Network control functions manage the RAN by utilizing applications (called xApps) along with the RICs. A number of O-RAN RU and DU units may be managed by an SMO and RIC complex. A number of O-RAN complexes may be backhauled to a 5G cellular core network which is the main interface to the rest of the data network (including the Internet). Thus, by its nature, the overall O-RAN-based cellular infrastructure is widely distributed, with a number of vantage points for monitoring traffic and exercising control for varying subsets of the traffic carried by the overall cellular network. The traffic observed at the cellular core is the aggregation of all the traffic at the different O-RAN subnetworks. Traffic monitoring, closely coupled to the cellular network architecture can help in resource management, identifying anomalies, and combating attacks. Given the network’s distributed nature, monitoring needs to be performed at multiple vantage points (e.g., close to each of gNB and the cellular core (see Fig. 6.2).

Cellular networks are prone to typical attacks observed on the internet, such as DDOS, Port Scans, and botnet attacks. Beyond this, unlike data centers and the Internet in general, resources are constrained, and therefore, likely warrant greater protection. In this work, we explore a distributed monitoring approach leveraging the software-based O-RAN and cellular core network infrastructure to thwart distributed attacks. Distributed attacks are difficult to detect if monitored solely at the core network since a considerable amount of traffic from many O-RAN subnetworks would be aggregated at the core, and the anomalous traffic could easily fly under the radar of traffic monitor at the core. Additionally, just monitoring traffic at each edge independently may also not identify distributed attacks as the volume of traffic at each distinct O-RAN subnetwork may not be high (although still capable of impacting the performance of legitimate traffic given the limited radio network bandwidth). But the aggregate load from these distributed attacks may be significant. An efficient traffic monitoring approach through summarization from the different O-RAN subnetworks can help to identify threat information accurately (see Fig. 6.2).

In this paper, we argue that cellular attack detection requires: aggregation, refinement, and filtering. This is because of the limited radio resources and the heterogeneity of cellular cells. The purpose of traffic aggregation is to group monitored data that have a shared characteristic[231], thus minimizing the memory overhead to maintain statistics for the group. Refinement lets us zoom into traffic subsets, thereby adaptively allocating more memory resources only to those traffic subsets that would return higher detection accuracy[263, 195]. Lastly, filtering ensures that we only transmit to a monitor only those traffic subsets that help to evaluate a query result[197].

We borrow inspiration from Jaal[138], a distributed monitoring framework, that utilizes a clustering mechanism to aggregate and transmit traffic summaries to extract the required information from traffic spread through the network. We leverage [197] for traffic filtering, which identifies traffic of interest on a network-wide basis by measuring traffic at various monitors and comparing it to a dynamically configured threshold. Traffic below the threshold is not transmitted to the global traffic analyzer to reduce data transmission overheads. We refer to this work as CMY because it uses the Cormode–Muthukrishnan–Yi upper bound [164] as the basis for setting the thresholds. Lastly, we are inspired by the traffic-refinement strategy from another monitoring framework, Dream[263]. It hypothesizes that beyond a certain detection accuracy, provisioning more memory resources to a single monitoring task will not increase the detection accuracy. Therefore, it adaptively allocates memory to monitoring tasks. We shall refer to this property as refinement. In other words, zooming in by allocating more memory to attain more fine-grained, or less aggregated, data for a traffic-subset. The desideratum of this work is as follows:

Monitoring in a heterogeneous cellular network: The cellular network includes a range of gNB sizes, such as Macro Cells, Micro Cells, etc. The traffic handled by different gNBs can also greatly vary, meaning that any monitor must be able to analyze the traffic adaptively. Jaal's traffic summaries consume considerable memory. However, accuracy drops significantly if we configure Jaal's parameters to reduce memory consumption. This is because of a lack of traffic refinement. As Dream[263], we must allocate more memory resources to those selected traffic subsets that improve overall detection accuracy. Jaal does not adapt to the heterogeneous traffic intensities as seen in the different cell sizes.

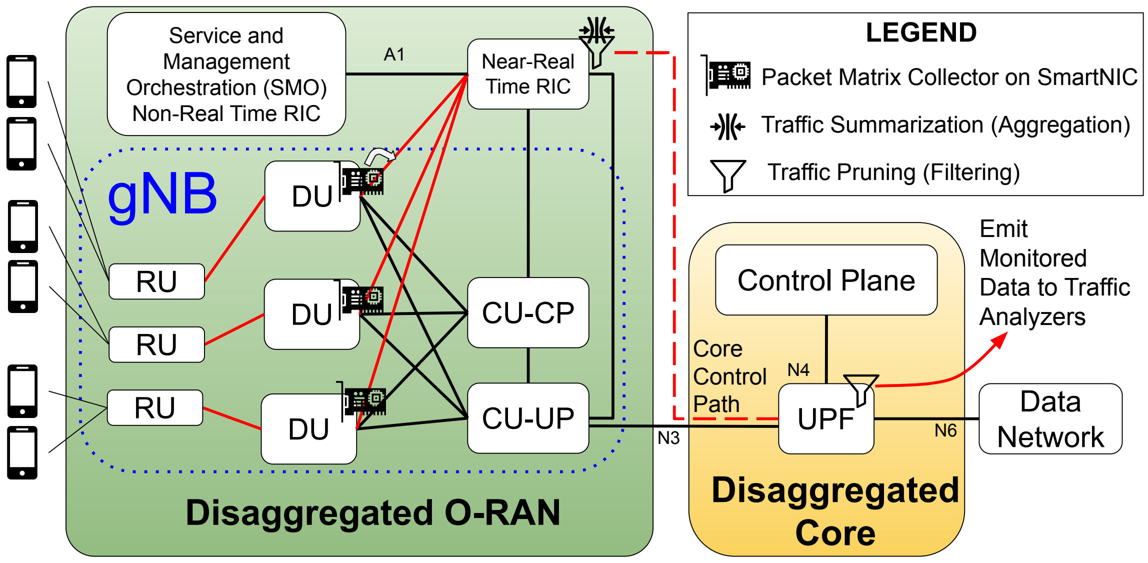

Service and Management Orchestration (SMO) Non-Real Time RIC
A1
Near-Real Time RIC
gNB
DU
RU
CU-CP
CU-UP
Disaggregated O-RAN
LEGEND
Packet Matrix Collector on SmartNIC
Traffic Summarization (Aggregation)
Traffic Pruning (Filtering)
Control Plane
Core Control Path
N4
UPF
N3
N6
Data Network
Emit Monitored Data to Traffic Analyzers
Disaggregated Core

Figure 6.1: Monitoring an O-RAN based Cellular Net

Detection at Periphery: Cellular resources are scarce, and therefore we want to be able to detect attacks as rapidly as possible at the periphery. In Dream, if one of the monitored prefixes is "interesting" from the perspective of a specific task, it divides that prefix to monitor into traffic subsets and uses more memory to monitor it. However, the refinement proposed in Dream is slow in the cellular context, mainly because it only considers IP prefixes and ignores other fields, such as flags, necessary to isolate benign vs. malicious traffic. For example, during a SYN Flood attack[195], using the SYN flag will better help discriminate benign and malicious traffic subsets rather than simply using IP prefixes.

Low Communication Overhead: Cellular monitoring overhead must be low while maintaining reasonable accuracy. CMY mainly configures thresholds and does not aggregate traffic based on subsets having similar traits. Therefore, the number of messages sent using CMY can be very high (400k messages for just 20 sites per epoch)[197]. To overcome this problem, we first used sampling, as is done in NitroSketch[245]. However, this only increased the convergence time with perceptible accuracy degradation. Elastic Sketch and Defeat suffer from the inherent problem of sketches, which involves trading off memory vs. accuracy and causes overestimation due to hash collisions. Furthermore, despite being invertible for five-tuple (e.g., can recover flows from sketch data structure itself [312]), dedicated sketches will have to be deployed as several detectors require data beyond just the five-tuple (e.g., SYN Flag). This results in higher memory requirement.

Low Processing Overhead: Network acceleration is pivotal as we will have to identify threats from high-speed networks. This can be achieved using hardware acceleration; in particular, we shall use SmartNICs to combat this problem. Using SmartNICs helps us minimize the loss-rate (e.g., unaccounted packets). Once the data is monitored, it will be summarized in software, before being emitted to the query processor.

The key differentiator between CMY[164] and 5GDMon is that CMY does not leverage the traffic aggregation (like Dream [263] or Jaal [138]), leading to high communication overhead. Furthermore, the query computation is performed at a central node (e.g., root), which also results in a high load at the root. The key differentiator between Dream [263] and 5GDMon is that Dream focuses on tracking anomalies by monitoring IP prefixes while 5GDMon, considers a more feature-rich packet header vector to detect anomalies. Since Dream utilizes limited switch TCAM memory, it restricts the number of active monitoring tasks. 5GDMon leverages memory and compute heterogeneity. A SmartNIC (with limited memory and compute capability, like switches in Dream), allows processing high volumes of traffic using hardware support. The host in 5GDMon (with more memory and CPU cores) then summarizes traffic streams. Contributions:

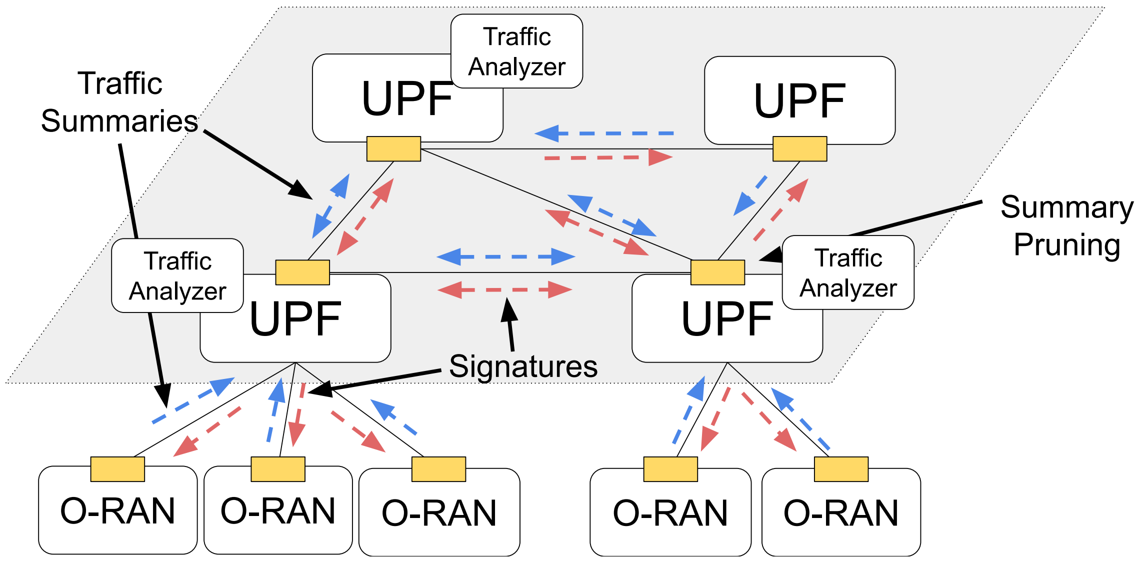


Figure 6.2: Global Cellular Monitoring Components

- A distributed monitor deployed on ORAN and UPF devices.
- Design a novel data summarization technique to aggregate packet logs collected from multiple vantage points. Our approach supports zooming into arbitrary traffic subsets by using hierarchical clustering that yields dendrograms which are then split appropriately to yield the desired level of summarization.
- Design a data collection mechanism on the SmartNIC, which runs on the distributed unit, to leverage acceleration, combating the high traffic volume.
- We combine this approach with a state-of-the-art pruning technique, but operate over the summarized logs instead of raw packet logs to reduce data transmission overheads.
- We evaluate our platform against simulation-generated traces as well as real-world traces.

6.2 5GC Preliminaries

The User Plane Function (UPF) is the first access point for packets flowing through the 5G Core of a cellular network. User Equipment (UE), or mobile devices, connect to the IP network through the UPF with their radio access provided through base stations (gNB)[276]. A recent standard, the Open RAN (O-RAN), enables the Radio Access Network (RAN) to be vendor-independent and software-based, disaggregating the various data and control components involved in forwarding data and managing the radio resources. O-RAN allows for innovative uses of machine learning (in 5GDMon, we use an unsupervised machine learning technique, as shown in §6.3.2) to optimize and regulate the RAN radio resources. O-RAN enablers include the following:

Disaggregation: As depicted in Fig. 6.1, the disaggregation of the RAN separates gNBs into their functional components, extending the functional disaggregation and software-based functionality concept put forth by 3GPP for Next Generation Node Bases[5]. The gNB is divided into a Central Unit (CU), a Distributed Unit (DU), and a Radio Unit (RU)[281]. This logical division enables the deployment of different functions in different parts of the network and on diverse hardware platforms[280]. For instance, the DU, which helps with lower levels of the protocol stack, including the physical layer, can offload a packet monitoring thread for a more efficient collection of traffic logs in 5GDMon as shown in §6.3.6.

RAN Intelligent Controller (RIC): The second enabler are the RICs, which bring in programmable components that can execute optimization processes with closed-loop control to manage the RAN. This encompasses two logical controllers that can serve as monitoring points of the RAN. The RICs analyze this data and utilize AI and ML algorithms to decide and implement control policies and actions on the RAN to provide data-driven, closed-loop control that can automatically optimize the network by RAN slicing, load balancing, handovers, scheduling policies, etc.,[147, 280]. In 5GDMon, we implement traffic summarization (§6.3.2) and pruning (§6.3.4) in the Near Real Time RIC.

Open Interfaces: The O-RAN Alliance has established technical specifications that outline open interfaces connecting various components of the O-RAN architecture. The inter-RAN interfaces from the 3GPP specifications[5] facilitate the gNB disaggregated architecture. These O-RAN interfaces expose the data analytics and telemetry to the RICs, enabling control and resource management and other deployment optimizations[280]. In 5GDMon, we use these open interfaces to communicate which traffic subsets need to be investigated more via signatures (see §6.3.5). This allows us to push more monitoring resources to the traffic subsets that provide the greatest return in accuracy, as envisioned in Dream [263].

5G monitoring is both bandwidth-consuming and latency-sensitive. The bandwidth overhead is unavoidable as the traffic monitoring is distributed, with multiple monitoring entities communicating with one another. Latency sensitivity stems from the requirement to achieve low threat detection time. The main objective of this work is to ensure monitoring is supported within the O-RAN architecture and its open interfaces. Multi-dimensional data, such as network logs, are acquired and processed via clustering algorithms to summarize the traffic. To study the scale of cellular-wide monitoring, we use the gNB coordinates in the city of Cologne[114]. In Fig 6.3, each dot represents a gNB, and its position within the plot indicates its geographical coordinates in kilometers. With

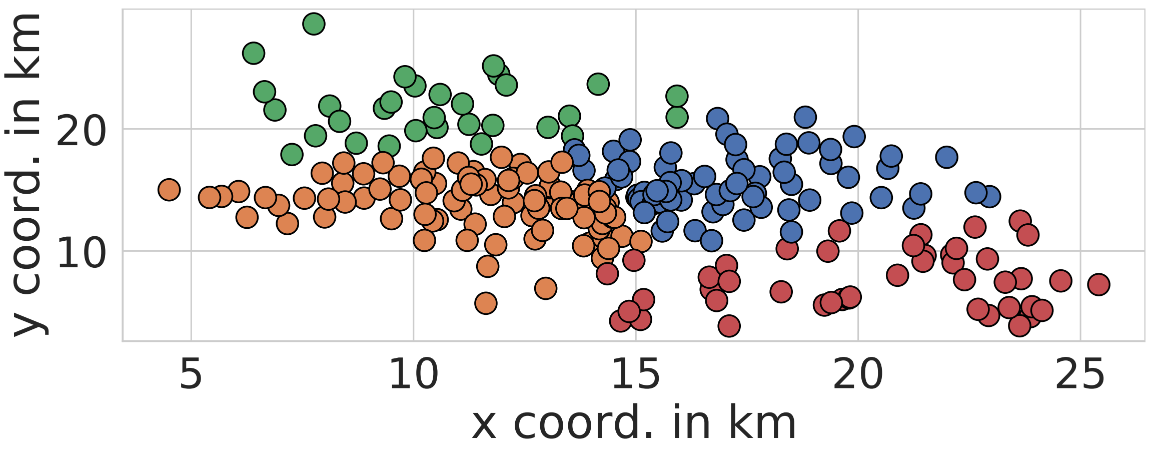


Figure 6.3: Geo-distribution of gNB.

vehicle mobility, the associated UE would have attached to the gNB that currently provides the best radio conditions (received signal strength (RSSI)). Here, the colors represent which UPF the gNB communicates with, which is the gateway to the cellular core (5GC). Since this data is not publicly available, we use K-Means to determine a static allocation of gNB to UPFs, as done in [187]. In this example, we estimate four UPFs serve the 247 gNBs.

Even though O-RAN provides disaggregation, analytical capabilities through RICs, and open interfaces in a virtualized environment, how to combine these operational principles such that it allows for the secure operation of the 5G infrastructure remains unspecified. 5GDMon addresses key aspects of this problem. As described in [83], three interconnected aspects must be addressed, namely multi-channel utilization, network monitoring, and automation of security processes. Network monitoring provides operators with key information by observing and analyzing traffic activity within the 5G network. 5GDMon presents a monitoring algorithm that scales well to support multiple gNBs by leveraging unsupervised machine-learning techniques and network acceleration. Secondly, security workflows must use automation to improve responsiveness to threats. In 5GDMon, we use automatic signature generation and iterative traffic refinement [195] to analyze the traffic at the required level of granularity. This also means that analysis will adapt to its deployment, whether it is macro or micro cells[66]. Lastly, Multi-channel utilization involves using several channels with varying performance and security characteristics to ensure a secure environment that honors the operator's intended resource allocations. In 5GDMon, we throttle the traffic that is inferred to be malicious (see §6.4 on AVIS resource provisioning).

6.3 5GDMon: Design

Cellular radio resources are limited, and it is important that we prevent adversaries from unreasonably consuming resources and thus impacting the QoE of other users. The goal of our work is to design a network telemetry system to thwart distributed attacks that can be effective even for large-scale cellular networks. Monitored data captured at software-based O-RANs (i.e., traffic summaries) and transmitted to traffic analyzers (i.e., UPFs resident at the 5G Core) must be informative and concise. We prune traffic summaries and load-balance the traffic sent from a large number of ORAN-based monitors to the traffic analyzers to efficiently utilize network bandwidth and ensure timely, accurate threat detection. Traffic queries are also distributed across traffic analyzers to ensure the compute-load (to process the queries) is balanced while keeping the data-transmission overhead low (see §6.3.3). Fig. 6.4 depicts a flowchart for 5GDMon monitoring. Packets are first collected to form a packet matrix (dimensions: packet batch size x packet features). Next, the packets matrix is converted to a packet dendrogram that links all the packets based on similarity, thus forming a tree (see Fig. 6.5). A dendrogram is used to compactly represent hierarchical

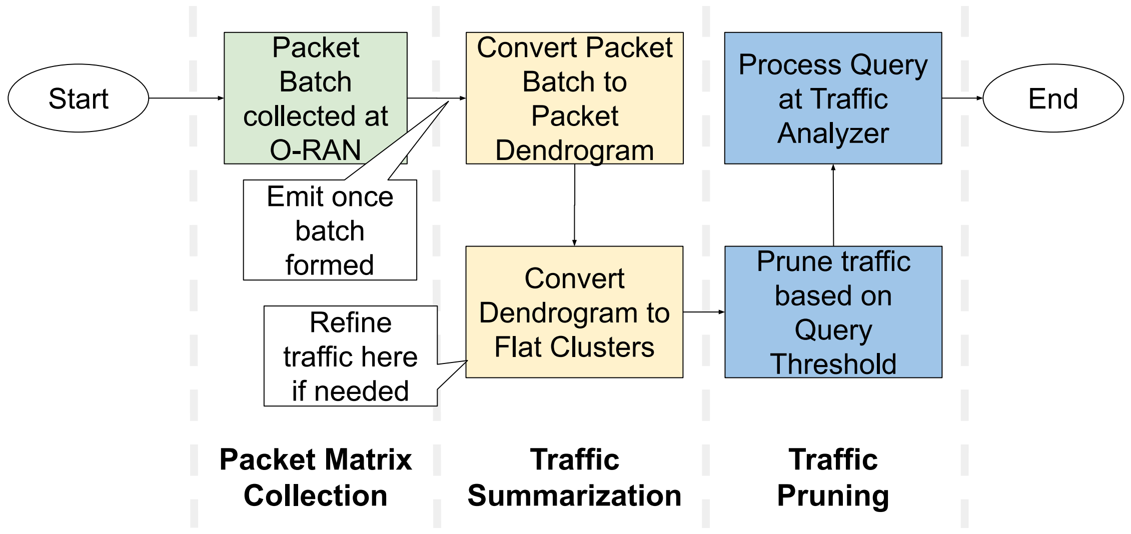

Start
Packet Batch collected at O-RAN
Emit once batch formed
Convert Packet Batch to Packet Dendrogram
Convert Dendrogram to Flat Clusters
Refine traffic here if needed
Process Query at Traffic Analyzer
Prune traffic based on Query Threshold
End
Packet Matrix Collection
Traffic Summarization
Traffic Pruning

Figure 6.4: 5GDMon distributed monitoring pipeline

clustering in the form of a tree diagram, displaying the relationships between groups of similar data (packets in our case)[29]. In 5GDMon, the leaves represent packets. Each link connects a pair of clusters at a time (packets at the bottom level) based on the similarity of the clusters (e.g., using the Ward-method[115]). Depending on the desired accuracy vs. space trade-off, we cut the dendrogram to form flat clusters (explained in §6.3.2). At this point, each packet is assigned a cluster label. We refer to these matrices as traffic summaries. While traffic summarization aggregates traffic, the entire traffic matrix may not be necessary to compute a specific query result. Therefore, we can employ pruning, as explained in §2.4. Depending on the queries deployed, the traffic is pruned to ensure that we only transmit the summaries required by the query. Depending on where the query is processed, the traffic is finally sent to the appropriate traffic analyzer, as explained in §6.3.3.

6.3.1 O-RAN and Core Pipeline Overview

The DU in Fig. 6.1 hosts the Radio Link Control Protocol[128], the medium access control (MAC), and the physical layer(PHY) [131]. In 5GDMon, the DU collects traffic data, while the Near-RT RIC is responsible for traffic summarization. In general, the DU is also responsible for managing the radio resource allocation for the UEs [87]. On detection of threats, in our design, the DU will be notified to control/restrict the resource allocation of the offending UEs even if they are distributed and are located across many gNBs.

Downlink and uplink data packets are sent to and from the UE through the DU. The DU implements the functional blocks of the L2 layer of a 5G protocol stack. There are generally 8 threads in DU, one of which is the Lower MAC Handler[88]. The Lower MAC has a packet handler that we use to monitor and collect packet matrices. We offload the thread responsible for matrix collection to a SmartNIC, which we discuss in §6.3.6, to maximize the fidelity of the data collected by leveraging network acceleration. Traffic summarization and pruning tasks are deployed in the Near-RT RIC. Fig. 6.1 shows how the monitored data propagates across the disaggregated O-RAN and core. Traffic is monitored at the DU and transmitted to the Near-RT RIC for summarization and pruning. The pruned traffic summaries are then transmitted to the UPF via the core control path. Fig. 6.2 shows how the monitored data propagates through the cellular core network comprising multiple UPFs. A subset of UPFs run traffic analyzers. The traffic summaries will be further pruned at the UPF before they are sent to the appropriate traffic analyzer for query computation.

6.3.2 Traffic Summarization

Traffic summaries aim to reduce communication overhead by aggregating packets while still maintaining high detection accuracy across the network. Traffic summaries outlined, for example, in Jaal[138] using K-Means++[58], aggregate traffic statistics such that it can be transmitted efficiently for query inference. Using K-Means++ ensures that packets with similar attributes are clustered together. In 5GDMon we carry out hierarchical clustering[7] for traffic summarization that is more adaptive. We can cut the dendrogram (e.g., computed for hierarchical clustering) at varying depths to achieve the desired level of summarization. This allows us to analyze the entire traffic stream at a coarse granularity and then zoom into the traffic subsets at a more refined level. Let P denote the packet matrix, where the columns represent the packet header features, and rows depict individual packets. Since each packet header attribute can consist of variable ranges, such as IP address (e.g., 2^{32} addresses) and TCP flags (e.g., Binary), we normalize header values by the maximum possible value of the respective header field (e.g., $normalized(x) = \frac{x}{max(x)}$) as in [138].

As shown in Fig. 6.2, the packet monitoring task begins in the O-RAN nodes, which are the leaves of the ORAN-UPF monitoring hierarchy. On these leaf nodes (O-RAN), we carry out hierarchical clustering to convert the packet matrix (P) to packet clusters (i.e., traffic summaries). The traffic summaries ($\tilde{P}$) and their membership counts (C) are then transmitted upstream to the UPF after the pruning step (see §2.4). The dimensions of the packet matrix P are the packet batch size $\times$ the number of packet features. Packet batching enables summarization via aggregation, and the network operator configures the batch size.

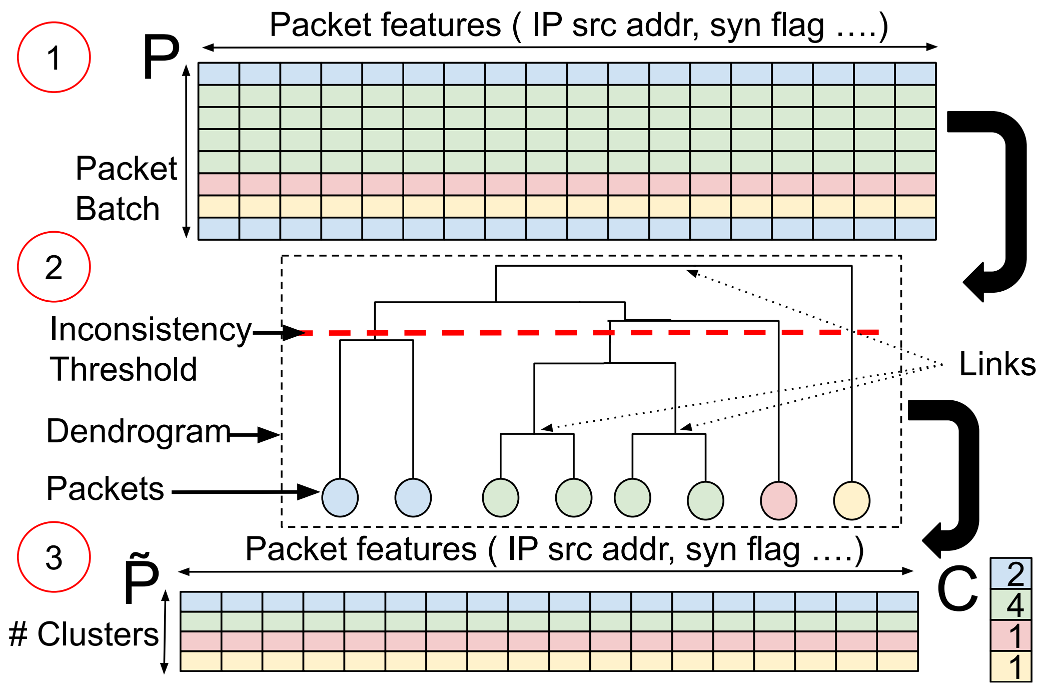

1
P
Packet features (IP src addr, syn flag ….)
Packet Batch
2
Inconsistency Threshold
Dendrogram
Packets
Links
3
P̃
Packet features (IP src addr, syn flag ….)
# Clusters
C
2
4
1
1

Figure 6.5: Clustering Packet Dendrograms

A small packet batch size will lead to frequent clustering operations, slowing the overall processing pipeline. We set the packet batch size to 4000 packets based on experiments with the SmartNIC that we carry out in §6.3.6. The number of clusters is selected dynamically to trade off between excessive summarization and accuracy. As we run hierarchical clustering on P to get traffic summaries $\tilde{P}$, which aggregates similar packets together, the bandwidth overhead of transmitting the packet data upstream reduces. Fig. 6.5 shows an overview of the matrix transformation. The colors tagged to individual packets in P indicate the cluster label assigned to the packet based on hierarchical clustering in the next step. Taking the four packets that have been tagged as the green cluster $\mathcal{G}$ for illustration, we would have $C_{\mathcal{G}}$ equal four, while for all feature $f \in \{src\_ip, dst\_ip...\}$, $\tilde{P}^f_{\mathcal{G}}$ is set to $(\sum_{pkt \in \mathcal{G}} P^f_{pkt})/C_{\mathcal{G}}$.

Hierarchical clustering provides a dendrogram that will have to be converted to flat clusters to form packet clusters (5GDMon traffic summaries). As shown in Fig. 6.5, each non-leaf node (e.g., links) and leaf node represent a potential cluster[277]. Starting from the leaves, which are potential clusters of their own, links connect two clusters at a time to form a tree. The height of the link (h) represents the dissimilarity of the clusters being merged (e.g., low at the leaves and high at the root). Inconsistency is computed for each link by comparing its height (h) with the average height of other links in its subtree ($H = h_0, h_1 ... h_{\#children}$). The larger the inconsistency, the greater the difference between the clusters connected by the link. Inconsistency is measured as $\frac{h - MEAN(H)}{STD\_DEV(H)}$[52, 277]. Next, we determine an inconsistency threshold to determine at what height to cut the dendrogram (see red line in Fig. 6.5). Cutting the tree above the root link, results in one large flat cluster such that the entire packet batch is one cluster (e.g., just one row in the traffic summary). This would be too coarse-grained for accurate traffic analysis. On the other hand, cutting the tree at the lowest level will result in each packet being a cluster of its own. This will result in considerable overhead for transmitting the traffic summaries. The inconsistency increases as we traverse from the leaves to the root, and having an inconsistency threshold dynamically computed ensures we can adapt to different traffic characteristics and volumes. As in [277], we determine the knee of the inconsistency values to be the inconsistency threshold because that is where links transition from low to high inconsistency values, which results in sufficient summarization without merging significantly-dissimilar packet clusters. Fig. 6.9 shows the cumulative distribution of the inconsistency values for all links of the dendrogram. We use a knee detection algorithm[296] to determine at what inconsistency value to cut the graph, as this is the point where we transition from low inconsistency (fine grained-summaries) to high inconsistency (coarse grained-summaries). Thereby, we balance the detection accuracy vs. the size of the traffic summary. In Fig. 6.9, the knee inconsistency value (i.e., blue dot) represents the first inconsistency threshold. Using this, we cut the dendrogram to get flat clusters (traffic summaries) and transmit traffic summary upstream (see Fig. 6.5).

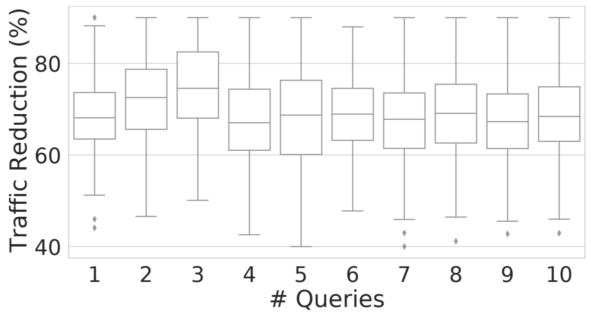


Figure 6.6: Benefit of Pruning

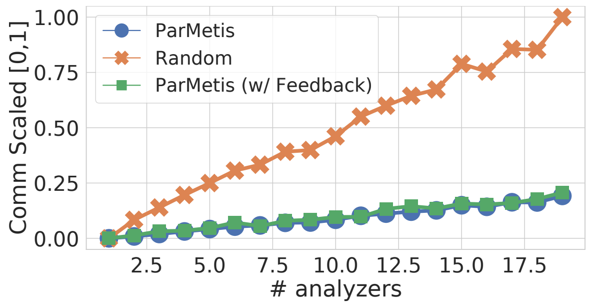


Figure 6.7: Query Partitioning impact on comm

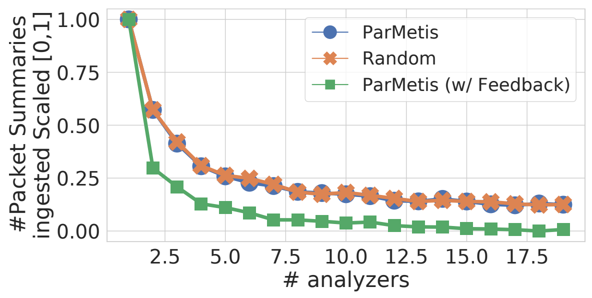


Figure 6.8: Query Partitioning impact on load

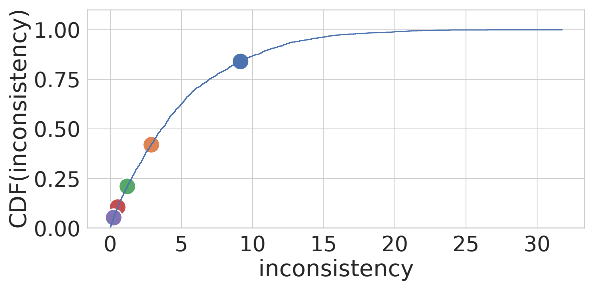


Figure 6.9: Inconsistency Threshold Tuning

6.3.3 Query Partitioning

Any two query pairs may depend on the same packet features. Query-pairs that share packet features should run on the same traffic analyzer (i.e., UPFs) to avoid traffic summaries from being duplicated (i.e., multicasted). On the other hand, query-pairs with disjoint packet features should run on separate traffic analyzers to distribute the query-processing load. To this end, we adopt ParMetis partitioning [297] to loadbalance the query processing tasks while ensuring similar queries (e.g., queries that share packet header fields) are placed closer together, minimizing the need to duplicate summaries. The goal of ParMetis partitioning is to cut a graph in such a way that the number of edges (or edge weights) between partitions are minimized, and each partition contains roughly the same amount of vertices (or vertex weight). In 5GDMon, ParMetis partitions an input graph wherein each vertex corresponds to a query. An edge exists between two queries (vertices) if they share a packet feature. The edge weight equals the number of intersecting fields to ensure the partitioning algorithm considers the magnitude of query-query similarity.

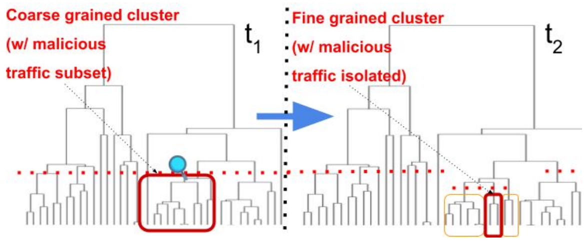


Figure 6.10: Zooming for traffic refinement

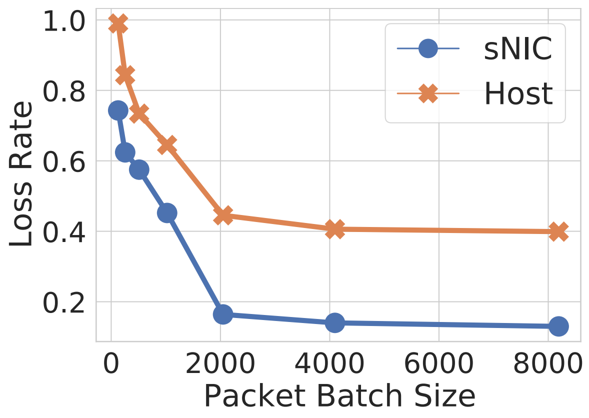


Figure 6.11: sNIC host monitoring

The graph partitioning output places each query on a different partition and then assigns the partition (e.g., a set of similar queries) to traffic analyzers (running on UPFs). The # of partitions (# traffic analyzers) are determined by the network operator. The benefit of carrying out ParMetis partitioning instead of randomly assigning the queries to traffic analyzers is shown in Fig. 6.7 and Fig. 6.8. In this experiment, we run 10 queries and vary the number of traffic analyzers in the x-axis. The y-axis represents the amount of traffic communicated and the average number of traffic summaries ingested across all analyzers, respectively (scaled between 0 and 1). As we increase the number of analyzers, the amount of duplication increases, but the load on each analyzer also decreases (e.g., having 10 analyzers reduces the average load by 88%). The data-point corresponding to one traffic analyzer represents the baseline used in [330, 236, 197, 263]. By deploying ParMetis, as the # traffic analyzers increases, the average load decreases tremendously, and the communication overhead increases only slightly. In this experiment, we use the traffic traces from [285] to represent benign traffic and replay them towards the O-RANs that are geographically organized as per [114]. We insert attack traffic using scapy-based-replayers [195]. It is evident that ParMetis partitioning outperforms all other approaches to balance the load and minimize duplication. We note that load is further balanced by having the vertex weights set to the # traffic summaries processed (e.g., Labelled as 'with Feedback' in Fig 6.8 and Fig. 6.7). Therefore, we use this ParMetis partitioning scheme for subsequent experiments.

Ultimately, each query will be placed on a traffic analyzer. Each UPF becomes aware of the query-to-analyzer mapping so that it can direct the appropriate traffic summaries to the analyzer. On the other hand, the O-RAN will only have to transmit the traffic summary to the UPF responsible for that O-RAN segment (see Fig. 6.2). The UPFs iterate over all traffic analyzers (a subset of UPFs) and determine which traffic summaries are to be emitted to which analyzer based on the query-to-analyzer mapping.

6.3.4 Pruning

Traffic pruning ensures we do not transmit unnecessary traffic summaries to a traffic analyzer. Therefore, it will filter away summaries that do not meet a local threshold as described in §2.4. This tremendously reduces the transmitted traffic as can be seen in Fig. 6.6. The x-axis depicts # queries while the y-axis represents the percentage reduction in the amount of traffic transmitted. Please note, from CMY[164], we borrow the pruning strategy but operate on traffic summaries instead of raw packet logs in order to reduce the data transmission overhead even further.

6.3.5 Signature and Refinement

A signature in 5GDMon is simply a traffic-summary filter transmitted from the analyzers to the O-RAN (where traffic is summarized) for the purposes of refinement (i.e., zooming into the traffic subset). If $\mathcal{P}$ is a traffic summary that exceeds a query threshold (i.e., during coarse-grained analysis), then we send $\mathcal{P}$ to all O-RAN sites. In the subsequent monitoring interval[195], we locate the links of the dendrogram where the previous inconsistency threshold cuts the dendrogram (P_{link}) and identify all links $l \in P_{link}$ where $CosineDist(\mathcal{P}, l) < \theta$. This threshold θ is configured by the network operator (e.g., 0.05 in our experiments). We use cosine distance, as it is bounded between 0 and 1 and is not dominated by any single feature like with euclidean distance[277]. Let us refer to the selected links as $\mathcal{L} = \{l \in P_{link} : \ CosineDist(\mathcal{P}, l) < \theta\}$, which were chosen because they were similar to the signature $\mathcal{P}$. The next task is to determine the new inconsistency threshold on the new subtrees (i.e., dendrograms) so that we can transmit finer-grained summaries than earlier (for the traffic subset). To accomplish this, we only retain the dendrograms under $\mathcal{L}$ and compute the new inconsistency threshold (e.g., orange data point in Fig. 6.9) based on the inconsistency values of the retained dendrograms (see Fig. 6.10). We then cut the new dendrograms and generate the traffic summaries that are transmitted to the UPF after pruning. This step will be repeated multiple times until the malicious traffic subset is identified. Please note that, in parallel, we would also process the original dendrogram (i.e., copy) using the default inconsistency threshold to ensure other queries are never neglected during refinement. Fig. 6.10 shows how the dendrogram is cut when we have to dig deeper into a cluster. The sub-tree that is emphasized by the red boundary at time instance t_1 is zoomed-into at time instance t_2. The rest of the traffic is analyzed at a coarse granularity.

6.3.6 Packet Matrix Collection

Fig. 6.12 shows the data structure maintained in the SmartNIC (sNIC) to compute the traffic logs for queries. We use a simple hash table to track duplicates in a best-effort manner (e.g., references from the hash table to the header log). Here a flow-key is defined as a tuple containing all the header fields that are used for monitoring. Our design strives to minimize duplicates in the header log given the small sNIC memory (e.g., 8GB[274]); however, if there is a miss on the hash table and the flow-key is already present in the header log (unknown when processing the packet), we accept the duplicate flow-key. To ensure failure to update the header log does not lead to packet loss, we decouple the packet processing pipeline from the header log update procedure on the sNIC. When a packet arrives, we copy the packet's features (only those required by queries) from the header and push them into a ring buffer. We set aside four sNIC micro engines (i.e., compute

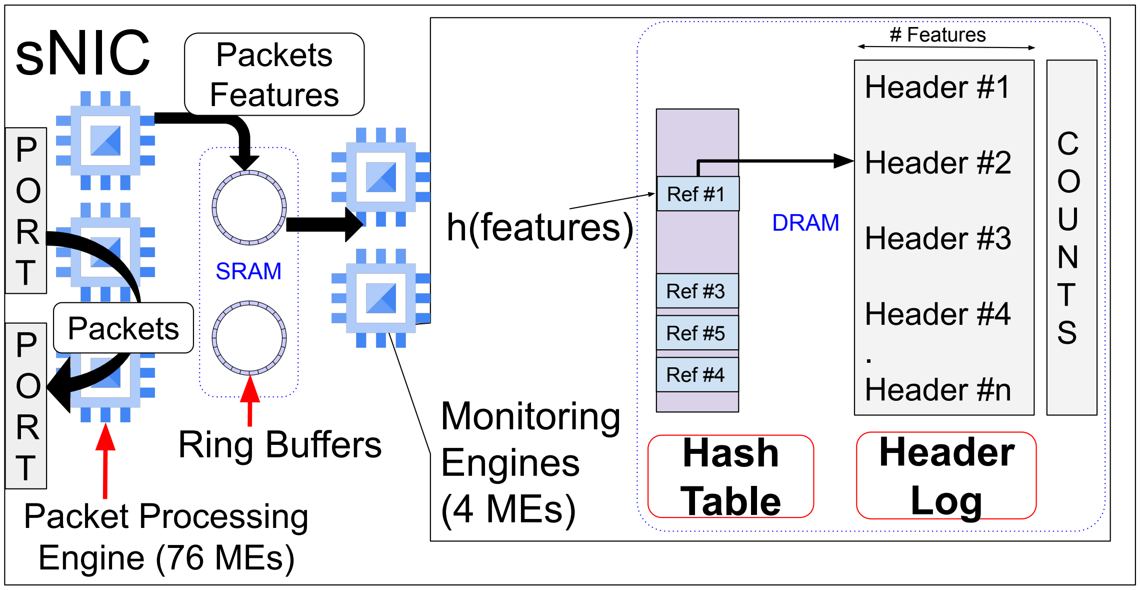


Figure 6.12: sNIC traffic monitoring

engines, called MEs) among the 80 user-programmable MEs. The four MEs are responsible for consuming header features from the ring buffer and updating the header log. The ring buffers are much smaller in size and hosted in SRAM (instead of DRAM), so they are fast to access. Furthermore, since the ring buffer entry is directly appended to the header log (e.g., DRAM), we get to use a bulk operation instead of individually copying part of the packet in smaller chunks on the slow DRAM. Lastly, the header log is DMA'd to the host when the packet batch is full. We do not transmit the hash table as it only helps remove the duplicates stored in the sNIC. Fig. 6.11 shows the loss rate of running this design in the sNIC is 57% lower than the host (which uses Confluo[222]), aggregated across all gNBs when subject to the vehicular-mobility[114] dataset.

6.3.7 Mapping Monitoring to ORAN and Core

The DU collects monitoring data in the O-RAN. 5GDMon offloads the DU's monitoring thread to the sNIC (see §6.3.6) to collect the packet matrix and forward it to the Near-RT RIC. Only the packet handler routine is offloaded to the sNIC, while everything else still runs in software on the host. Implementing which other threads of the DU should be offloaded onto the sNIC is part of our ongoing work. As shown in Fig. 6.1, at the Near-RT RIC, we summarize and prune the traffic and send it upstream to the UPF responsible for the gNB. The pipeline is non-blocking, utilizes huge pages and is pinned to run in as real-time as possible. This is necessary as we want the overhead to be as low as possible. As shown in Fig. 6.2, at the UPF, the data traffic is pruned again and sent upstream towards the traffic analyzer (another UPF), where the query result is computed. Once the traffic analyzer determines that a UE is malicious or if a UE is to be rate limited, the CU and DU are notified to carry out the corresponding action.

6.4 Evaluation

Testbed: The effectiveness of 5GDMon was assessed on a local test setup that comprised Linux servers (kernel version 4.4.0-142). Each server was equipped with 10 CPU cores of Intel Xeon with a clock speed of 2.20GHz, 256GB of memory, and a Netronome Agilio LX 2x40 GbE sNIC with 8GB DDR3 memory and 96 flow processing cores. The stress tests were performed using three packet generators that utilized Moongen[176] to play back PCAP traces at a high rate (43 Mpps) using 64B packets.

Traces: Two traces were employed to depict benign traffic, including a real-world trace[285] obtained from measurements with a 5G network of an Irish mobile operator containing information such as timestamps, coordinates, gNB id, bitrate, and channel quality indicator. This data was collected through UE streaming videos and downloading files while the user was in a vehicle traveling on city streets. The other trace[114] is a dataset of 700k vehicle trips across 247 gNBs, including gNB id, timestamp, and vehicle coordinates. The UE connects to the gNB, offering the best communication conditions determined by path loss[232] at each point in time. The attack traffic was introduced using both synthetic traces generated by Scapy[195] and real-world traces obtained from Zeek[116].

Table 6.1: Packet Header Vector (Features)[93]

Header	**Attributes**
GTP-U	QoS Flow Identifier, Reflective QoS identifier, N-PDU number, Seq number, TEID, Message Type
TCP Header	Source / Destination Port number, Seq number, Ack number, RST/SYN flag
IP Header	Proto, Source / Destination IP addr

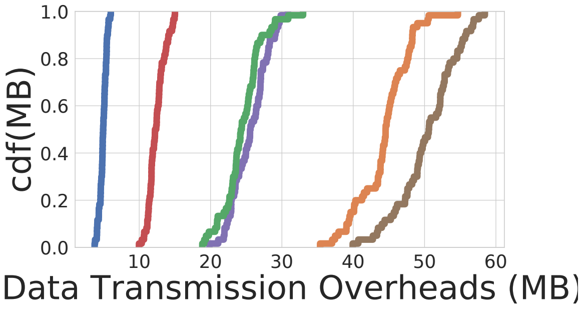


Figure 6.13: Data Transmission Overhead

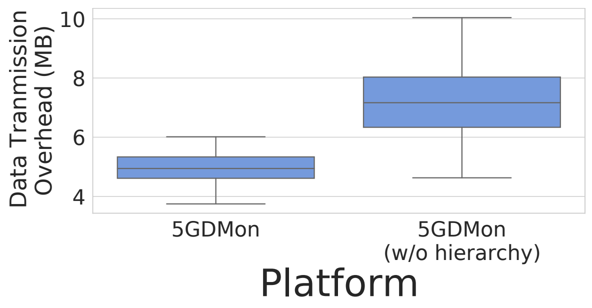


Figure 6.14: Memory Overhead w/o Hierarchy

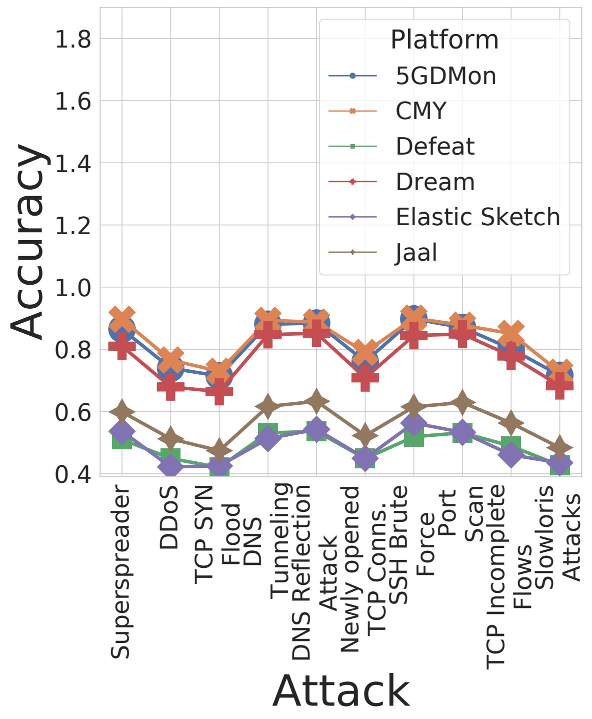


Figure 6.15: Detection Accuracy

Data Transmission Overhead

In Fig. 6.13, we evaluate the communication overhead between the O-RAN and traffic analyzer per one-minute epoch. The reason why Defeat[236] and Elastic Sketch[330] have more overhead is that they are sketch-based mechanisms that always use constant

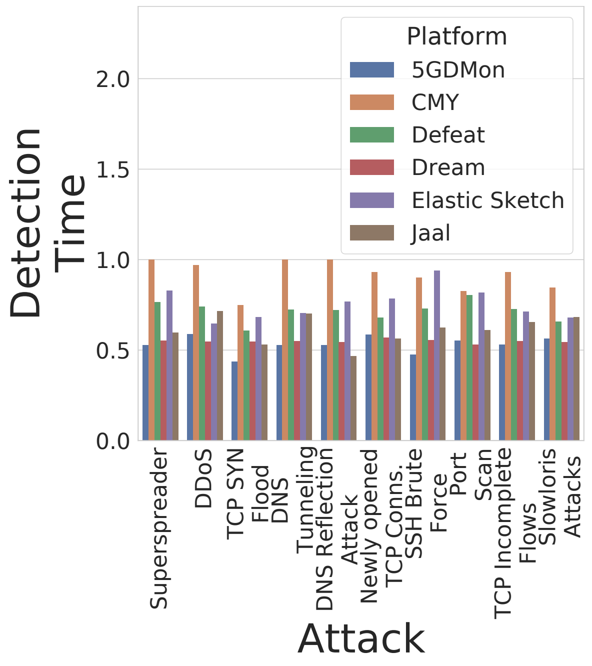


Figure 6.16: Detection Time

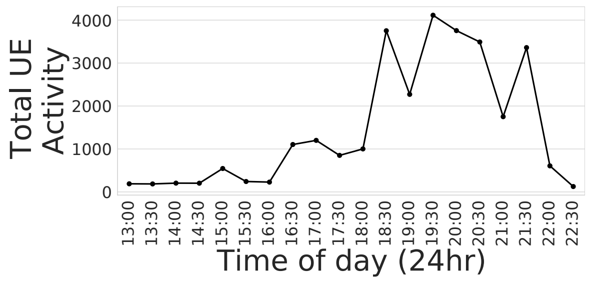


Figure 6.17: Milan Activity

space. Furthermore, due to the richness of the features involved (e.g., including TCP flags), multiple sketches have to be deployed, as we have to construct sketches specific to each and every query. CMY[197] also has high overhead because it lacks aggregation, resulting in a high number of ORAN-UPF messages. As explained in [197], CMY sends 400k messages/epoch when deployed at just 20 sites. Dream [263] is slightly worse than

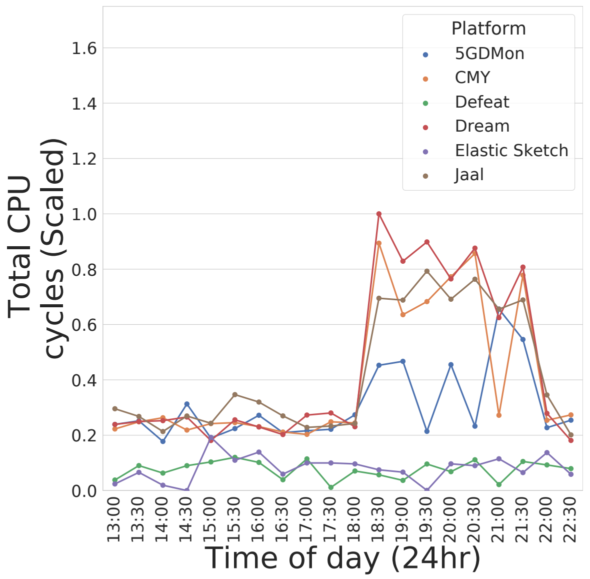


Figure 6.18: CPU runtime

5GDMon in terms of communication overhead because 5GDMon aggressively zooms into malicious traffic subsets. Jaal[138] performs worse because traffic summaries are still large in size ($O(\#summaries \times \#features)$ where $\#summaries = 500$). 5GDMon also uses traffic summaries but utilizes pruning (like CMY, both at the O-RAN and UPF) alongside iterative refinement to tremendously reduce the communication overhead. This works well because a majority of the traffic is likely not attack traffic, and only needs to be processed locally (e.g., Near-RT RIC). That traffic is not transmitted as it does not cross local threshold values (see §6.3.4). We also show the benefit of having a hierarchical monitor (e.g., $ORAN \rightarrow UPF \rightarrow TrafficAnalyzer$) in Fig. 6.14, instead of simply having just the O-RAN and traffic analyzers. Carrying out hierarchical data transmission provides more opportunities to prune the traffic based on local thresholds, allowing us to filter away more traffic.

Accuracy

Secondly, we evaluate the accuracy against several attacks in Fig. 6.15, comparing this work to other platforms. The reason 5GDMon has higher accuracy than Jaal is that 5GDMon can zoom into the traffic subsets instead of just analyzing at a coarse-grained level. Defeat and Elastic Sketch have poorer accuracy as sketch-based approaches are prone to overestimation. CMY achieves similar (and sometimes slightly better) accuracy than 5GDMon because it transmits the exact counts and filters away traffic that cannot theoretically satisfy the query. Lastly, Dream achieves slightly lower accuracy compared to 5GDMon because its aggregation does not consider header attributes other than IP prefixes (e.g., TCP SYN flag).

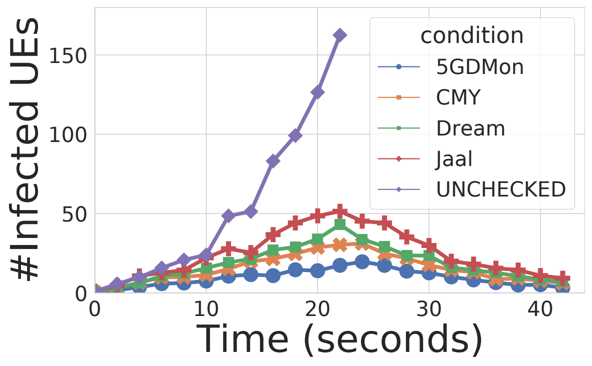


Figure 6.19: Mirai Botnet

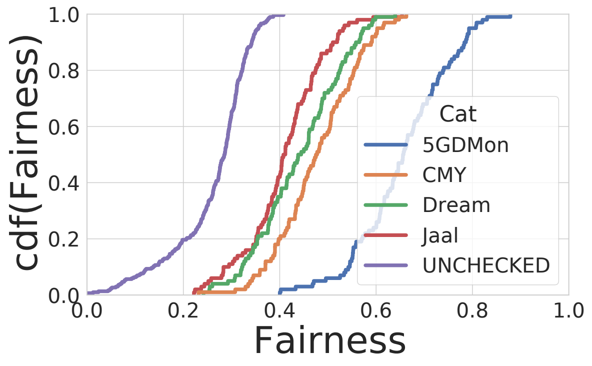


Figure 6.20: Resource Allocation

Detection Time

In Fig. 6.16, we show the detection time scaled such that the maximum is 1. Here we only consider those attacks that have been successfully detected. Jaal, Defeat and Elastic Sketch's static summarization leads to always-coarse traffic summaries. Thus it takes longer to detect the attack as benign traffic is not isolated from malicious traffic. CMY carries out no aggregation and exchanges a lot of messages, causing the detection time to increase. On average 5GDMon and Dream have the lowest detection delay because they zoom into malicious traffic subsets.

Resource Consumption

In this section, we study the monitoring overhead in terms of CPU runtime. For this experiment, we use a cellular mobility dataset from the city of Milan [141] that provides the number of messages, calls, and internet usage, demarcated by squares (i.e., geographical segregation). In this experiment, we focus on the square with a soccer game in the stadium of San Siro (see Fig. 6.17). The activity, which is the sum of sms usage, call usage, and internet usage, surges during that period of time (i.e., 5:30 pm onwards). We use perf to measure the total CPU time (scaled between 0 - 1). As expected, sketch-based approaches such as Defeat and Elastic Sketch have the lowest CPU time (see Fig. 6.18). This is because of the constant time operation involved with both of these sketch monitoring structures. Besides sketch-based approaches, 5GDMon has lower runtime, meaning it is more resource efficient because it operates on less data, as shown in Fig. 6.13.

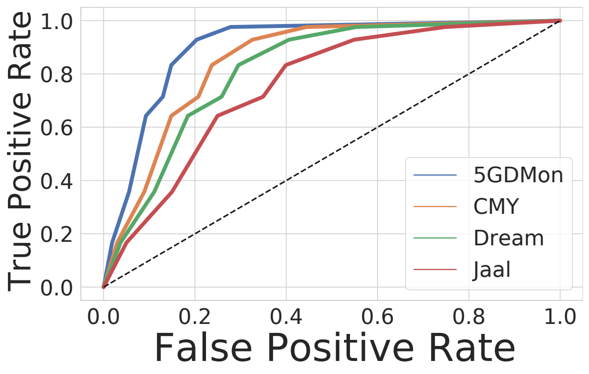


Figure 6.21: Proxy Detection

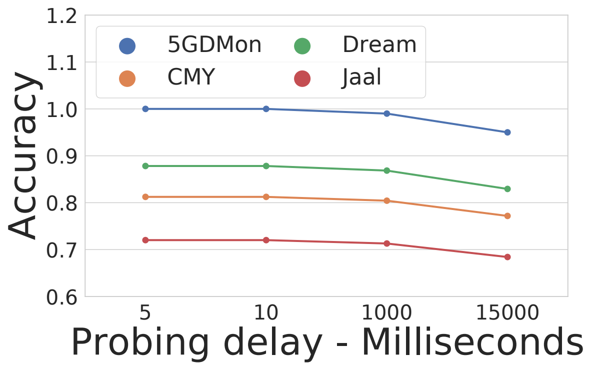


Figure 6.22: TEID bruteforcing

Mirai Botnet

We first consider a Mirai botnet attack[74], where devices are scanned, and once infected they scan the network themselves to infect other devices. This attack is detected by identifying a high variance in the destination IP addresses for target port values 23 and 2323 [138]. We run the experiment in an "unrestricted" setting where there is no detector that removes the infected devices and then compare it against 5GDMon and other related works that depict high accuracy. Since 5GDMon has high accuracy and a quick detection rate, it can detect and remove infected devices quickly as time progresses (see Fig. 6.19). We noticed that the 5GDMon, on average, has 37.99% fewer infected devices than CMY. Fig. 6.23 shows that despite aggressively removing infected devices, the number of legitimate devices removed as a function of time is similar and low (e.g., < 5 UEs incorrectly removed among 1750 UEs) across all monitoring platforms. In all four monitoring platforms, false positives were caused by UEs participating in peer-to-peer (P2P) networks. This may be because of the similarity between botnets and P2P networks (like BitTorrent), which have now become the basis of new botnets like Hajime[200].

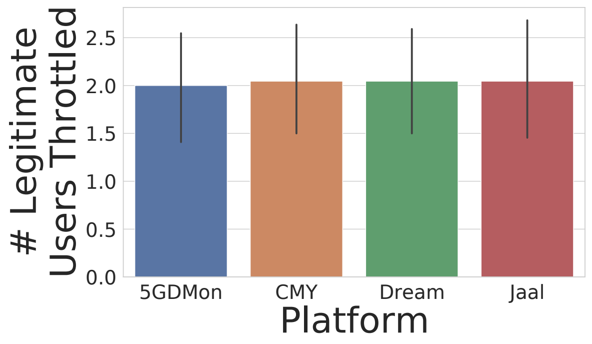


Figure 6.24: Resource Allocation FPR

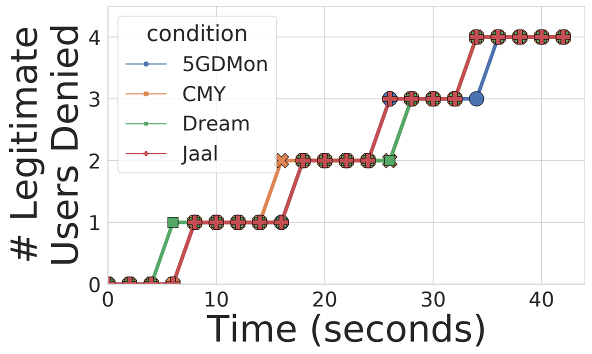


Figure 6.23: Mirai Botnet FPR

Resource Allocation

In Fig. 6.20, we show the fairness for video streaming achieved using the AVIS[155] resource allocation algorithm from an experiment consisting of 242 UEs. We selected AVIS over the state-of-the-art ASAP[336] because ASAP depends on the UE client's video buffer level, which would not be available at the DU. We utilize the alpha-fair[307] metric to evaluate the quality of video delivery that is computed by using the rate-allocation, transmission rate, and the number of users. The anomaly is created by several malicious UEs requesting a high volume of video feeds, causing AVIS to allocate more resources to the malicious UEs that would otherwise have been allocated to legitimate UEs. We then use several detection algorithms to identify and rate-limit such UEs. In Fig. 6.20, we show the alpha-fair metric for all the UEs that are not rate-limited by the resource detector. Because of 5GDMon's ability to quickly and accurately identify heavy hitters, it exhibits 1.35× higher alpha-fairness compared to alternative solutions. Fig. 6.24 shows that 5GDMon and alternative platforms have a low false positive rate, causing on-average two legitimate users to get throttled.

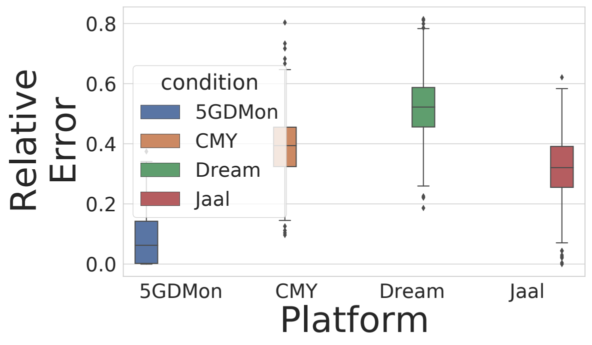


Figure 6.25: Proxy Detection (RE)

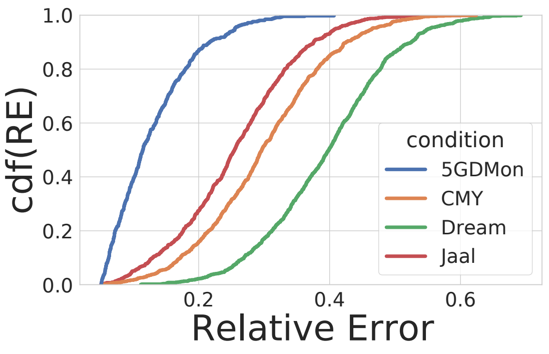


Figure 6.26: TEID bruteforcing (RE)

Proxy Detection

A Proxy app is simply a third-party application that integrates the proxy service into the SDK without the knowledge of the mobile device's (UE's) owner. As noted by [256], any UE can be utilized as a proxy peer by installing a proxy app. Proxy providers grant anonymity to their customers by masking the proxy client's IP address [256]. Proxy providers attain this by directing their client's traffic through various proxy peers (e.g., UEs, whose owners may not realize this use of their device). Since proxy apps abuse on-device resources by relaying traffic, it is imperative to detect the use of a UE as a proxy peer since proxy apps steal cycles that could have been used for legitimate applications on the UE [256]). The UE that is exploited to be a proxy peer typically establishes and maintains one or more long-lasting proxy connections to proxy gateways using TCP. Therefore, it can be detected by observing the gap between TCP SEQ and ACK, both in terms of absolute difference (i.e., $Conn_{SEQ} - Conn_{ACK}$) and ratio (i.e., $\frac{Conn_{SEQ}}{Conn_{ACK}}$) [256]. In our experiment, we deploy this connection gap-based detector (e.g., ratio and absolute difference) with varying thresholds. In Fig. 6.21, we show the ROC curve to detect mobile proxies where we note that the area under the curve of 5GDMon is 5.8% higher than CMY. We also measure the relative error (RE), which is defined as the $|\frac{\gamma-\gamma_q}{\gamma_q}|$ where γ_q is the query threshold and γ is the ground truth (i.e., true count)[158]. In Fig. 6.25, we show that 5GDMon on average achieves 3.92× lower RE than alternatives when varying the query threshold γ_q.

Bruteforce TEID

Similar to SSH Bruteforcing, attackers can identify the TEID using brute forcing, as explained in [111]. The attacker can then tear down the session for which it has learned the TEID, causing a denial of service attack. In this experiment, we scan the TEID space over varying time intervals. If the time delay between consecutive probes is short, it is easier to detect the attack vs. when the probing delay is high. In this experiment we calculate the accuracy and RE of the queries in Fig. 6.22 and Fig. 6.26, respectively. On average, the accuracy improves by 8% to 36% while the RE reduces by 1.96× to 3.04× compared to alternative solutions.

6.4.1 Global Inconsistency Threshold

Locally computing the inconsistency threshold have a few disadvantages. 1) An adversary can generate a traffic stream that causes the inconsistency distribution to be skewed, allowing attacks to fly under the radar by forcing aggressive summarization. 2) The traffic aggregation levels will be sensitive to local traffic characteristics (e.g., relative dissimilarities in a cell cannot be compared against other cells), perturbing the aggregation that occurs in the UPF and Traffic Analyzers. Since we strive to detect network-wide anomalies, it is critical for us to use thresholds that can summarize the traffic at the global scale instead of being based on regional-local characteristics.

To combat this, we maintain inconsistency histograms in the ORAN. The Near-RT RIC is responsible for emitting the inconsistency values to the Non-RT RIC where it is aggregated over five minute epochs. The Non-RT RICs then send the 5-minute inconsistency histogram to a controller where the cdf of the inconsistency values is computed, followed by computing its knee. The knee inconsistency value is then transmitted to all ORANs where it is used to summarize the traffic.

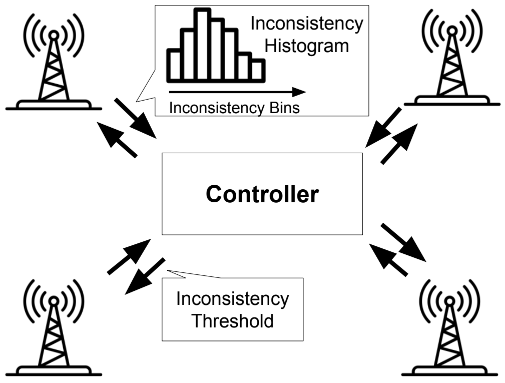


Figure 6.27: Global inconsistency threshold computation

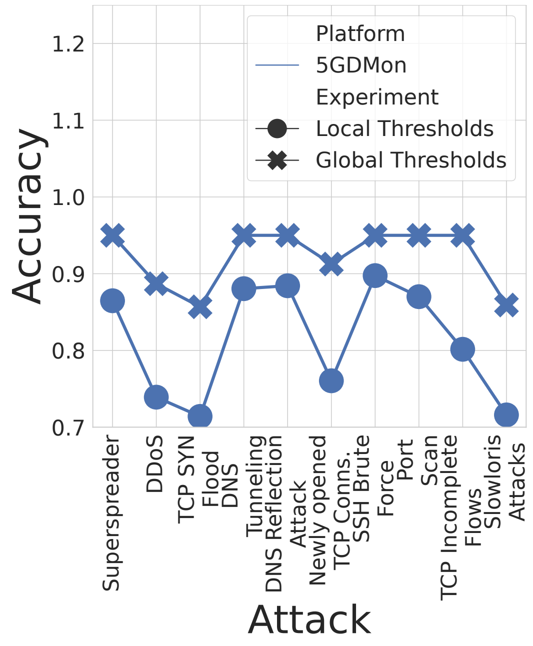


Figure 6.28: Deviation between local and global inconsistency threshold

Chapter 7

Conclusions

Providing a comprehensive monitoring infrastructure that can detect stealthy attacks in the midst of high traffic volumes is a challenge. State-of-the-art monitoring techniques either detect stealthy attacks at very low packet rates or limit their detection capabilities to volumetric attacks for high packet arrival rates. SmartWatch bridges this dichotomy by cooperatively splitting up the monitoring tasks between P4 programmable networking switches, P4-capable SmartNICs and the host. Our proposed control loop helps avoid having to make a trade-off between detection rate vs. packet processing rate. Furthermore, SmartWatch helps reduce the SRAM memory pressure on programmable switches, by reducing the required state on the switches. On the other hand, the SmartNIC helps reduce the packet processing latency even further, by offloading flow-state tracking and flow-logging tasks from the host to the network-centric components of sNIC and P4Switch. In summary, SmartWatch selects the correct monitoring-granularity and monitoring-target to detect both volumetric and stealthy attacks. SmartWatch's network switch and host co-design for cooperative monitoring yields 2.39 times better detection rate compared to just programmable switches, thanks to SmartWatch's fined-grained processing without compromising packet processing throughput. Compared to host-based fine-grained approaches, SmartWatch reduces the packet processing latency by 72.32%

pMACH is a framework that solves the complex provisioning problem in containerized data centers. pMACH includes a novel graph-based locality aware container placement scheme that significantly reduces power consumption, task completion time, and migrations. We show that by operating server resources at Peak Energy Efficiency, we both save power and provide greater headroom for traffic spikes. Our Two-Tier distributed graph partitioning architecture can scale to tens of thousands of servers and compute the partitioning result quickly. By carefully CPU cores in selected servers in each pod of the data center, and taking advantage of the high-bandwidth data center links, we split the graph partitioning into a hierarchical solution. pMACH tracks container-to-container communication and uses data stream summarization techniques to communicate the traffic matrices efficiently to designated servers in each pod for partitioning the graph.

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.

5GDMon is a distributed monitoring solution for 5G cellular networks (and beyond) by leveraging the software-based O-RAN environment. To carefully protect the limited radio resources, 5GDMon focuses on detecting attacks on the 5G infrastructure as well as the end-user mobile devices with high accuracy, while minimizing overhead. 5GDMon uses Agglomerative Clustering to summarize traffic while sup- porting refinement beyond processing and pruning traffic at multiple software-based nodes in the cellular network, both in the RAN and the 5G core. 5GDMon load balances query processing across multiple analyzers and intelligently places queries so as to reduce the communication overhead when queries are related to one another. Overall, this makes the distributed monitor scalable. The monitoring threads in the ORAN are offloaded to a SmartNIC to achieve a low loss rate. Furthermore, the processing pipeline is designed so that the monitoring pipeline does not impede the data forwarding pipeline. Compared to existing approaches Dream and CMY, 5GDMon achieves at least: 2.52× lower data transmission overhead, 1.58× lower botnet penetration, 1.37× better fairness, and 4.78× lower error in detecting proxies.

Bibliography

[1] 2020 ddos attacks.

[2] 3gpp ts23.501 section 4.2: Architecture reference model.

[3] 3gpp ts23.502: System architecture for the 5g system (5gs).

[4] 5g enterprise market.

[5] 5g ng-ran architecture description (3gpp ts 38.401 version 15.2.0 release 15).

[6] 5g nr 3gpp.

[7] Agglomerative clustering.

[8] Agilio lx 2x40gbe smartnic.

[9] Amazon vpc flow logs.

[10] Apache thrift.

[11] Arm cortex-a72 architecture.

[12] Aws ecs bin packing.

[13] Aws region and zone.

[14] Azure encryption.

[15] Bare-metal switches.

[16] Bin packing.

[17] Blockhosts.

[18] Bloom filter tutorial.

[19] Bro cluster architecture.

[20] Bruteforceblocker.

[21] The caida anonymized internet traces. http://www.caida.org/data.

[22] Cisco tetration.

[23] Cloudsuite.

[24] Connected everything: 5g and edge computing solutions.

[25] Control plane messages.

[26] Crypto module nfp.

[27] D-salt.

[28] Default paging cycle.

[29] Dendrogram.

[30] Denyhosts.

[31] Docker stats.

[32] Dpdk multi processor support.

[33] Editcap.

[34] Facebook data center fabric. hhttps://engineering.fb.com/2014/11/14/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/.

[35] Fail2ban.

[36] free5gc.

[37] Gcp live miogration. https://cloud.google.com/compute/docs/instances/live-migration.

[38] Gcp vpc flow logs.

[39] Google cloud platform: Micro services demo.

[40] Gridsearch.

[41] Gtp v1 header.

[42] Handover code free5gc.

[43] Handover required 3gpp.

[44] Hardware-accelerated networks at scale in the cloud.

[45] Hardware acceleration for network services.

[46] Hash collision probabilities.

[47] High performance cloud with hardware acceleration.

[48] Hitachi smart data center.

[49] How do 5g small cells work and where are they located?

[50] How zeek can provide insights despite encrypted communications. https://corelight.blog/2019/05/07/how-zeek-can-provide-insights-despite-encrypted-communications/.

[51] In-band network telemetry - more insight into the network.

[52] Inconsistency threashold.

[53] Int band telemetry.

[54] Intel tofino 2.

[55] Intel turbo boost.

[56] Intel xl710 datasheet.

[57] Iptraf.

[58] K-means.

[59] Kubernetes.

[60] Kubernetes nodename.

[61] Kubernetes pod.

[62] Kubernetes pod limits.

[63] Kubernetes resource bin packing.

[64] Linux manual page: dd.

[65] Live migration using criu. https://criu.org/Docker.

[66] Macropicocell.

[67] Mapping p4 to smartnics.

[68] Marvell liquidio iii.

[69] Marvell liquidio iii.

[70] Marvell user guide.

[71] Measurement reports.

[72] Mergecap.

[73] Microsoft vpc flow logs.

[74] Mirai botnet attack.

[75] Netfpga-sume.

[76] Netronome agilio lx.

[77] Netronome metering.

[78] Netronome microc.

[79] Netronome nfp-6000 flow processor.

[80] The new need for speed in the datacenter network.

[81] Ng application protocol.

[82] Nmap.

[83] Nsf smart-5g: Secure multichannel automated operations through 5g networks).

[84] Nvidia mellanox bluefield.

[85] Nvidia mellanox bluefield.

[86] O-ran alliance.

[87] O-ran architecture overview.

[88] O-ran du overview.

[89] Open ran.

[90] Open5gs.

[91] Opensoc.

[92] Our high-performing core network.

[93] P4 header file for upf).

[94] P4 programming language.

[95] Parmetis manual.

[96] Perfect k-ary tree.

[97] Profiling cpu usage in real time with perf top.

[98] Rc informed presentation netman ai ops.

[99] Redis.

[100] Ric.

[101] Rte mbuf.

[102] Scantron.

[103] Service oriented 5g core networks.

[104] Simulation of urban mobility.

[105] Slowloris.

[106] Snort ips deployment guide.

[107] sshguard.

[108] Synergy opensource code.

[109] Task migration at scale using criu.

[110] Tcprewrite.

[111] Threats to cellular networks (report).

[112] Towards converged smartnic architecture for bare metal & public clouds.

[113] Towards machine learning inference in the data plane.

[114] Vehicular mobility trace.

[115] Ward method.

[116] Zeek.

[117] Zeek ftp bruteforcing.

[118] Zeek http stalling detection.

[119] Zeek kerberos.

[120] Zeek ssh bruteforcing script.

[121] Zeek ssl.

[122] Zeek ssl certificate.

[123] *Kolmogorov–Smirnov Test*, pages 283–287. Springer New York, New York, NY, 2008.

[124] 3GPP. LTE; 5G; Interface between the Control Plane and the User Plane nodes. Technical Specification (TS) 29.244, 3rd Generation Partnership Project (3GPP), 09 2019. Version 15.7.0.

[125] 3GPP. PDR specification. https://www.etsi.org//ts_129244v150500p.pdf, 2021.

[126] Mohamed Abdel-Nasser and Karar Mahmoud. Accurate photovoltaic power forecasting models using deep lstm-rnn. *Neural Comput. Appl.*, 31(7):2727–2740, jul 2019.

[127] Mukhtiar Ahmad, Syed Usman Jafri, Azam Ikram, Wasiq Noor Ahmad Qasmi, Muhammad Ali Nawazish, Zartash Afzal Uzmi, and Zafar Ayyub 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*, SIGCOMM '20, page 648–661, New York, NY, USA, 2020. Association for Computing Machinery.

[128] Sassan Ahmadi. Chapter 7 - the ieee 802.16m medium access control common part sub-layer (part ii). In Sassan Ahmadi, editor, *Mobile WiMAX*, pages 281–319. Academic Press, Oxford, 2011.

[129] Azza H. Ahmed and Ahmed Elmokashfi. Icran: Intelligent control for self-driving ran based on deep reinforcement learning. *IEEE Transactions on Network and Service Management*, 19(3):2751–2766, 2022.

[130] Mustafa Riza Akdeniz, Yuanpeng Liu, Mathew K. Samimi, Shu Sun, Sundeep Rangan, Theodore S. Rappaport, and Elza Erkip. Millimeter wave channel modeling and cellular capacity evaluation. *IEEE Journal on Selected Areas in Communications*, 32(6):1164–1179, 2014.

[131] Alberto Martínez Alba, Jorge Humberto Gómez Velásquez, and Wolfgang Kellerer. An adaptive functional split in 5g networks. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 410–416, 2019.

[132] Adeeb Alhomoud, Rashid Munir, Jules Pagna Disso, Irfan Awan, and A. Al-Dhelaan. Performance evaluation study of intrusion detection systems. *Procedia Computer Science*, 5:173 – 180, 2011. The 2nd International Conference on Ambient Systems, Networks and Technologies (ANT-2011) / The 8th International Conference on Mobile Web Information Systems (MobiWIS 2011).

[133] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. Conga: Distributed congestion-aware load balancing for datacenters. *SIGCOMM Comput. Commun. Rev.*, 44(4):503–514, August 2014.

[134] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, page 63–74, New York, NY, USA, 2010. Association for Computing Machinery.

[135] Y. Amir, B. Awerbuch, A. Barak, R. S. Borgstrom, and A. Keren. An opportunity cost approach for job assignment in a scalable computing cluster. *IEEE Transactions on Parallel and Distributed Systems*, 11(7):760–768, 2000.

[136] Navendu and Jain and Seny Kamara. Attacking data center networks from the inside. Technical Report MSR-TR-2015-52, June 2015.

[137] Fabien André, Stéphane Gouache, Nicolas Le Scouarnec, and Antoine Monsifrot. Don't share, don't lock: Large-scale software connection tracking with krononat. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 453–466, Boston, MA, July 2018. USENIX Association.

[138] Azeem Aqil, Karim Khalil, Ahmed O.F. Atya, Evangelos E. Papalexakis, Srikanth V. Krishnamurthy, Trent Jaeger, K. K. Ramakrishnan, Paul Yu, and Ananthram Swami. Jaal: Towards network intrusion detection at isp scale. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, page 134–146, New York, NY, USA, 2017. Association for Computing Machinery.

[139] Azeem Aqil, Karim Khalil, Ahmed O.F. Atya, Evangelos E. Papalexakis, Srikanth V. Krishnamurthy, Trent Jaeger, K. K. Ramakrishnan, Paul Yu, and Ananthram Swami. Jaal: Towards network intrusion detection at isp scale. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, pages 134–146, New York, NY, USA, 2017. ACM.

[140] Zachary K. Baker and Viktor K. Prasanna. High-throughput linked-pattern matching for intrusion detection systems. In *Proceedings of the 2005 ACM Symposium on Architecture for Networking and Communications Systems*, ANCS '05, page 193–202, New York, NY, USA, 2005. Association for Computing Machinery.

[141] Gianni Barlacchi, Marco De Nadai, Roberto Larcher, Antonio Casella, Cristiana Chitic, Giovanni Torrisi, Fabrizio Antonelli, Alessandro Vespignani, Alex Pentland, and Bruno Lepri. A multi-source dataset of urban life in the city of milan and the province of trentino. *Scientific Data*, 2:150055, 10 2015.

[142] Diogo Barradas, Nuno Santos, Luís Rodrigues, S. Signorello, Fernando M. V. Ramos, and André Madeira. Flowlens: Enabling efficient flow classification for ml-based network security applications. In *NDSS*, 2021.

[143] Adam Bates, Kevin Butler, Andreas Haeberlen, Micah Sherr, and Wenchao Zhou. Let sdn be your eyes: Secure forensics in data center networks. 01 2014.

[144] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. Pint: Probabilistic in-band network telemetry. 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*, SIGCOMM '20, page 662–680, New York, NY, USA, 2020. Association for Computing Machinery.

[145] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 267–280, New York, NY, USA, 2010. ACM.

[146] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, page 267–280, New York, NY, USA, 2010. Association for Computing Machinery.

[147] Leonardo Bonati, Salvatore D'Oro, Michele Polese, Stefano Basagni, and Tommaso Melodia. Intelligence and learning in o-ran for data-driven nextg cellular networks. *IEEE Communications Magazine*, 59(10):21–27, 2021.

[148] Abhik Bose, Diptyaroop Maji, Prateek Agarwal, Nilesh Unhale, Rinku Shah, and Mythili Vutukuru. Leveraging programmable dataplanes for a high performance 5g user plane function. In *5th Asia-Pacific Workshop on Networking (APNet 2021)*, APNet 2021, page 57–64, New York, NY, USA, 2021. Association for Computing Machinery.

[149] Vladimir Braverman, Jonathan Katzman, Charles Seidell, and Gregory Vorsanger. An optimal algorithm for large frequency moments using o(n$^(1 - 2/k))bits. In APPROX - RANDOM$, 2014.

[150] Andrei Broder and Michael Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1(4):485 – 509, 2003.

[151] Nicola Bui and Joerg Widmer. OWL: a reliable online watcher for LTE control channel measurements. *CoRR*, abs/1606.00202, 2016.

[152] Brian Caswell, James C. Foster, Ryan Russell, Jay Beale, and Jeffrey Posluns. *Snort 2.0 Intrusion Detection*. Syngress Publishing, 2003.

[153] Milan Ceška, Vojtech Havlena, Lukáš Holík, Jan Korenek, Ondrej Lengál, Denis Matoušek, Jirí Matoušek, Jakub Semric, and Tomáš Vojnar. Deep packet inspection in fpgas via approximate nondeterministic automata. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 109–117, 2019.

[154] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, page 693–703, Berlin, Heidelberg, 2002. Springer-Verlag.

[155] Jiasi Chen, Rajesh Mahindra, Mohammad Amir Khojastepour, Sampath Rangarajan, and Mung Chiang. A scheduling framework for adaptive video delivery over cellular networks. In *Proceedings of the 19th Annual International Conference on Mobile Computing amp; Networking*, MobiCom '13, page 389–400, New York, NY, USA, 2013. Association for Computing Machinery.

[156] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, Jennifer Rexford, and Ori Rottenstreich. Catching the microburst culprits with snappy. In *Proceedings of the Afternoon Workshop on Self-Driving Networks*, SelfDN 2018, page 22–28, New York, NY, USA, 2018. Association for Computing Machinery.

[157] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, Jennifer Rexford, Ori Rottenstreich, Steven A Monetti, and Tzuu-Yi Wang. Fine-grained queue measurement in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, CoNEXT '19, page 15–29, New York, NY, USA, 2019. Association for Computing Machinery.

[158] Xiaoqi Chen, Shir Landau-Feibish, Mark Braverman, and Jennifer Rexford. Beaucoup: Answering many network traffic queries, one memory update at a time. 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*, SIGCOMM '20, page 226–239, New York, NY, USA, 2020. Association for Computing Machinery.

[159] Xiaoqi Chen, Shir Landau-Feibish, Mark Braverman, and Jennifer Rexford. Beaucoup: Answering many network traffic queries, one memory update at a time. 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*, SIGCOMM '20, page 226–239, New York, NY, USA, 2020. Association for Computing Machinery.

[160] B. Claise. Cisco systems netflow services export version 9. RFC 3954, RFC Editor, October 2004. http://www.rfc-editor.org/rfc/rfc3954.txt.

[161] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[162] Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. In Martín Farach-Colton, editor, *LATIN 2004: Theoretical Informatics*, pages 29–38, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[163] Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, April 2005.

[164] Graham Cormode, S. Muthukrishnan, and Ke Yi. Algorithms for distributed functional monitoring. *ACM Trans. Algorithms*, 7(2), mar 2011.

[165] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the International Symposium on Operating Systems Principles (SOSP)*, October 2017.

[166] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. Gigascope: A stream database for network applications. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, page 647–651, New York, NY, USA, 2003. Association for Computing Machinery.

[167] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, James Alexander Docauer,

Jesse Alpert, Jing Ai, Jon Olson, Kevin DeCabooter, Marc de Kruijf, Nan Hua, Nathan Lewis, Nikhil Kasinadhuni, Riccardo Crepaldi, Srinivas Krishnan, Subbaiah Venkata, Yossi Richter, Uday Naik, and Amin Vahdat. Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 373–387, Renton, WA, April 2018. USENIX Association.

[168] Huynh Tu Dang, Marco Canini, Fernando Pedone, and Robert Soulé. Paxos made switch-y. *SIGCOMM Comput. Commun. Rev.*, 46(2):18–24, May 2016.

[169] Ramraj Dangi, Praveen Lalwani, Gaurav Choudhary, Ilsun You, and Giovanni Pau. Study and investigation on 5g technology: A systematic review. *Sensors*, 22(1), 2022.

[170] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pages 137–150, San Francisco, CA, 2004.

[171] Narayan Desai, Rick Bradshaw, Andrew Lusk, and Ewing Lusk. Mpi cluster system software. In Dieter Kranzlmüller, Péter Kacsuk, and Jack Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 277–286, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[172] Mohammed Dighriri, Gyu Myoung Lee, and Thar Baker. Applying scheduling mechanisms over 5g cellular network packets traffic. In Xin-She Yang, Simon Sherratt, Nilanjan Dey, and Amit Joshi, editors, *Third International Congress on Information and Communication Technology*, pages 119–131, Singapore, 2019. Springer Singapore.

[173] Shi Dong, Khushnood Abbas, and Raj Jain. A survey on distributed denial of service (ddos) attacks in sdn and cloud computing environments. *IEEE Access*, 7:80813–80828, 2019.

[174] Nick Duffield, Carsten Lund, Mikkel Thorup, and Mikkel Thorup. Estimating flow distributions from sampled flow statistics. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 325–336, New York, NY, USA, 2003. ACM.

[175] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.

[176] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. MoonGen: A Scriptable High-Speed Packet Generator. In *Internet Measurement Conference 2015 (IMC'15)*, Tokyo, Japan, October 2015.

[177] Facebook Engineering. Introducing data center fabric, the next-generation Facebook data center network. https://engineering.fb.com/2014/11/14/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/.

[178] Haggai Eran, Lior Zeno, Maroun Tork, Gabi Malka, and Mark Silberstein. NICA: An infrastructure for inline acceleration of network applications. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 345–362, Renton, WA, July 2019. USENIX Association.

[179] Cristian Estan, Stefan Savage, and George Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, page 137–148, New York, NY, USA, 2003. Association for Computing Machinery.

[180] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. volume 32, pages 75–80, 11 2001.

[181] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, pages 323–336, New York, NY, USA, 2002. ACM.

[182] Yixiao Feng, Sourav Panda, Sameer G Kulkarni, K. K. Ramakrishnan, and Nick Duffield. A smartnic-accelerated monitoring platform for in-band network telemetry. In *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN*, pages 1–6, 2020.

[183] C.-N. Fiechter. A parallel tabu search algorithm for large traveling salesman problems. *Discrete Appl. Math.*, 51(3):243–267, July 1994.

[184] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. Azure accelerated networking: Smartnics in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, Renton, WA, 2018. USENIX Association.

[185] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. Azure accelerated networking: Smartnics in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, Renton, WA, April 2018. USENIX Association.

[186] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. Azure accelerated networking: SmartNICs in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, Renton, WA, April 2018. USENIX Association.

[187] Pablo Fondo-Ferreiro, David Candal-Ventureira, Francisco Javier González-Castaño, and Felipe Gil-Castiñeira. Latency reduction in vehicular sensing applications by dynamic 5g user plane function allocation with session continuity. *Sensors*, 21(22), 2021.

[188] Pablo Fondo-Ferreiro, David Candal-Ventureira, Francisco Javier González-Castaño, and Felipe Gil-Castiñeira. Latency reduction in vehicular sensing applications by dynamic 5g user plane function allocation with session continuity. *Sensors*, 21(22), 2021.

[189] Message P Forum. Mpi: A message-passing interface standard. Technical report, USA, 1994.

[190] Linux Foundation. Data plane development kit (DPDK), 2015.

[191] Thomer M. Gil and M. Poletto. Multops: A data-structure for bandwidth attack detection. In *USENIX Security Symposium*, 2001.

[192] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, Dave Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *SIGCOMM*. Association for Computing Machinery, Inc., August 2009. Recognized as one of "the most important research results published in CS in recent years".

[193] Lee L. Gremillion. Designing a bloom filter for differential file access. *Commun. ACM*, 25(9):600–604, sep 1982.

[194] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 357–371, New York, NY, USA, 2018. ACM.

[195] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 357–371, New York, NY, USA, 2018. Association for Computing Machinery.

[196] Hasanin Harkous, Mu He, Michael Jarschel, Rastin Pries, Ehab Mansour, and Wolfgang Kellerer. Performance study of p4 programmable devices: Flow scalability and rule update responsiveness. In *2021 IFIP Networking Conference (IFIP Networking)*, pages 1–6, 2021.

[197] Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford. Network-wide heavy hitter detection with commodity switches. In *Proceedings of the Symposium on SDN Research*, SOSR '18, New York, NY, USA, 2018. Association for Computing Machinery.

[198] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. Elastictree: Saving energy in data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, page 17, USA, 2010. USENIX Association.

[199] Angel Herranz and Juan José Moreno-Navarro. Cache configuration exploration on prototyping platforms. In *Proceedings of the 14th IEEE International Workshop on Rapid System Prototyping (RSP'03)*, RSP '03, pages 164–, Washington, DC, USA, 2003. IEEE Computer Society.

[200] Stephen Herwig, Katura Harvey, George Hughey, Richard Roberts, and Dave Levin. Measurement and analysis of hajime, a peer-to-peer iot botnet. *Network and Distributed Systems Security (NDSS) Symposium.*

[201] Daisuke Hisano, Yu Nakayama, Kazuki Maruta, and Akihiro Maruta. Deployment design of functional split base station in fixed and wireless multihop fronthaul. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2018.

[202] Chung-Hsing Hsu and Stephen W. Poole. Revisiting server energy proportionality. In *2013 42nd International Conference on Parallel Processing*, pages 834–840, 2013.

[203] Qun Huang, Patrick Lee, and Yungang Bao. Sketchlearn: relieving user burdens in approximate measurement with automated statistical inference. pages 576–590, 08 2018.

[204] Qun Huang, Haifeng Sun, Patrick P. C. Lee, Wei Bai, Feng Zhu, and Yungang Bao. Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy. 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*, SIGCOMM '20, page 404–421, New York, NY, USA, 2020. Association for Computing Machinery.

[205] Syed Rafiul Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. Lteinspector: A systematic approach for adversarial testing of 4g lte. In *NDSS*, 2018.

[206] Syed Rafiul Hussain, Mitziu Echeverria, Omar Chowdhury, Ninghui Li, and Elisa Bertino. Privacy attacks to the 4g and 5g cellular paging protocols using side channel information. *Proceedings 2019 Network and Distributed System Security Symposium*, 2019.

[207] Jinho Hwang, K. K. Ramakrishnan, and Timothy Wood. NetVM: High performance and flexible networking using virtualization on commodity platforms. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 445–458, Seattle, WA, April 2014. USENIX Association.

[208] Joel Hypolite, John Sonchack, Shlomo Hershkop, Nathan Dautenhahn, André DeHon, and Jonathan 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*, page 336–350, New York, NY, USA, 2020. Association for Computing Machinery.

[209] Mohammad Jahanian, Jiachen Chen, and K. K. Ramakrishnan. Graph-based namespaces and load sharing for efficient information dissemination in disasters. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–12, 2019.

[210] Vivek Jain, Hao-Tse Chu, Shixiong Qi, Chia-An Lee, Hung-Cheng Chang, Cheng-Ying Hsieh, K.K. Ramakrishnan, and Jyh-Cheng Chen. L25gc: A low latency 5g core network based on high-performance nfv platforms. SIGCOMM '22, 2022.

[211] Mobin Javed and Vern Paxson. Detecting stealthy, distributed ssh brute-forcing. pages 85–96, 11 2013.

[212] Xin Jin and Jiawei Han. *K-Medoids Clustering*, pages 564–565. Springer US, Boston, MA, 2010.

[213] Lavanya Jose, Minlan Yu, and Jennifer Rexford. Online measurement of large traffic aggregates on commodity switches. In *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'11, page 13, USA, 2011. USENIX Association.

[214] Raj Joshi, Ting Qu, Mun Choon Chan, Ben Leong, and Boon Thau Loo. Burstradar: Practical real-time microburst monitoring for datacenter networks. In *Proceedings of the 9th Asia-Pacific Workshop on Systems*, APSys '18, New York, NY, USA, 2018. Association for Computing Machinery.

[215] Jaeyeon Jung, V. Paxson, Arthur Berger, and Hari Balakrishnan. Fast portscan detection using sequential hypothesis testing. pages 211 – 225, 06 2004.

[216] Georgios Kambourakis, Tassos Moschos, Dimitris Geneiatakis, and Stefanos Gritzalis. Detecting dns amplification attacks. In *Proceedings of the Second International Conference on Critical Information Infrastructures Security*, CRITIS'07, page 185–196, Berlin, Heidelberg, 2007. Springer-Verlag.

[217] George Karypis and Vipin Kumar. Multilevel graph partitioning schemes. pages 113–122, 01 1995.

[218] George Karypis and Vipin Kumar. Kumar, v.: A fast and high quality multilevel scheme for partitioning irregular graphs. siam journal on scientific computing 20(1), 359-392. *Siam Journal on Scientific Computing*, 20, 01 1999.

[219] George Karypis and Vipin Kumar. MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0. http://www.cs.umn.edu/ metis, 2009.

[220] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez. Rubik: Fast analytical power management for latency-critical systems. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 598–610, 2015.

[221] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K. Nurminen, and Zhonghong Ou. Rapl in action: Experiences in using rapl for power measurements. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 3(2), March 2018.

[222] Anurag Khandelwal, Rachit Agarwal, and Ion Stoica. Confluo: Distributed monitoring and diagnosis stack for high-speed networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 421–436, Boston, MA, February 2019. USENIX Association.

[223] Changhoon Kim. Programming the network data plane: What, how, and why? APNet Keynote, 2017.

[224] Daehyeok Kim, Zaoxing Liu, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, Vyas Sekar, and Srinivasan Seshan. 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*, SIGCOMM '20, page 90–106, New York, NY, USA, 2020. Association for Computing Machinery.

[225] Tulja Vamshi Kiran Buyakar, Harsh Agarwal, Bheemarjuna Reddy Tamma, and Antony A. Franklin. Prototyping and load balancing the service based architecture of 5g core using nfv. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 228–232, 2019.

[226] Tiina Kovanen, Gil David, and Timo Hämäläinen. Survey: Intrusion detection systems in encrypted traffic. In Olga Galinina, Sergey Balandin, and Yevgeni Koucheryavy, editors, *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, pages 281–293, Cham, 2016. Springer International Publishing.

[227] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: Methods, evaluation, and applications. In *Proceedings of the 2003 ACM SIGCOMM Internet Measurement Conference, IMC 2003*, pages 234–247, 12 2003.

[228] Swarun Kumar, Ezzeldin Hamed, Dina Katabi, and Li Erran Li. Lte radio analytics made easy and accessible. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 211–222, New York, NY, USA, 2014. Association for Computing Machinery.

[229] Swarun Kumar, Ezzeldin Hamed, Dina Katabi, and Li Erran Li. Lte radio analytics made easy and accessible. *SIGCOMM Comput. Commun. Rev.*, 44(4):211–222, aug 2014.

[230] Aleksandar Kuzmanovic and Edward W. Knightly. Low-rate tcp-targeted denial of service attacks: The shrew vs. the mice and elephants. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, page 75–86, New York, NY, USA, 2003. Association for Computing Machinery.

[231] C.A. Kyriakopoulos, G.I. Papadimitriou, P. Nicopolitidis, and E. Varvarigos. Chapter 11 - advanced optical network architecture for the next generation internet access. In Mohammad S. Obaidat and Petros Nicopolitidis, editors, *Smart Cities and Homes*, pages 219–239. Morgan Kaufmann, Boston, 2016.

[232] Ibtissam Labriji, Francesca Meneghello, Davide Cecchinato, Stefania Sesia, Eric Perraud, Emilio Calvanese Strinati, and Michele Rossi. Mobility aware and dynamic migration of mec services for the internet of vehicles. *IEEE Transactions on Network and Service Management*, 18(1):570–584, 2021.

[233] Ashwin Lall, Vyas Sekar, Mitsunori Ogihara, Jun Xu, and Hui Zhang. Data streaming algorithms for estimating entropy of network traffic. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '06/Performance '06, pages 145–156, New York, NY, USA, 2006. ACM.

[234] Yanfang Le, Hyunseok Chang, Sarit Mukherjee, Limin Wang, Aditya Akella, Michael M. Swift, and T. V. Lakshman. Uno: Uniflying host and smart nic offload for flexible packet processing. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pages 506–519, New York, NY, USA, 2017. ACM.

[235] Jihyung Lee, Sungryoul Lee, Junghee Lee, Yung Yi, and KyoungSoo Park. Flosis: A highly scalable network flow capture system for fast retrieval and storage efficiency. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '15, page 445–457, USA, 2015. USENIX Association.

[236] Xin Li, Fang Bian, Mark Crovella, Christophe Diot, Ramesh Govindan, Gianluca Iannaccone, and Anukool Lakhina. Detection and identification of network anomalies using sketch subspaces. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, page 147–152, New York, NY, USA, 2006. Association for Computing Machinery.

[237] Yuanjie Li, Zengwen Yuan, and Chunyi Peng. A control-plane perspective on reducing data access latency in lte networks. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, page 56–69, New York, NY, USA, 2017. Association for Computing Machinery.

[238] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Flowradar: A better netflow for data centers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 311–324, Santa Clara, CA, March 2016. USENIX Association.

[239] A. Lim and Y.-M. Chee. Graph partitioning using tabu search. In *1991., IEEE International Sympoisum on Circuits and Systems*, pages 1164–1167 vol.2, 1991.

[240] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. In *2011 Proceedings IEEE INFOCOM*, pages 1098–1106, 2011.

[241] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. Offloading distributed applications onto smartnics using ipipe. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, pages 318–333, New York, NY, USA, 2019. ACM.

[242] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. Offloading distributed applications onto smartnics using ipipe. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 318–333, New York, NY, USA, 2019. Association for Computing Machinery.

[243] Ming Liu, Simon Peter, Arvind Krishnamurthy, and Phitchaya Mangpo Phothilimthana. E3: Energy-efficient microservices on smartnic-accelerated servers. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 363–378, Renton, WA, July 2019. USENIX Association.

[244] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. Nitrosketch: Robust and general sketch-based monitoring in software switches. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, pages 334–350, New York, NY, USA, 2019. ACM.

[245] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. Nitrosketch: Robust and general sketch-based monitoring in software switches. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 334–350, New York, NY, USA, 2019. Association for Computing Machinery.

[246] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 101–114, New York, NY, USA, 2016. ACM.

[247] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, page 301–312. IEEE Press, 2014.

[248] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. *SIGARCH Comput. Archit. News*, 42(3):301–312, June 2014.

[249] Robert MacDavid, Carmelo Cascone, Pingping Lin, Badhrinath Padmanabhan, Ajay ThakuR, Larry Peterson, Jennifer Rexford, and Oguz Sunay. *A P4-Based 5G User Plane Function*, page 162–168. Association for Computing Machinery, New York, NY, USA, 2021.

[250] Alexander Magnano, Xin Fei, and Azzedine Boukerche. Predictive mobile ip handover for vehicular networks. In *2015 IEEE 40th Conference on Local Computer Networks (LCN)*, pages 338–346, 2015.

[251] Gregor Maier, Robin Sommer, Holger Dreger, Anja Feldmann, Vern Paxson, and Fabian Schneider. Enriching network security analysis with time travel. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, page 183–194, New York, NY, USA, 2008. Association for Computing Machinery.

[252] W. D. Maurer and T. G. Lewis. Hash table methods. *ACM Comput. Surv.*, 7(1):5–19, March 1975.

[253] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap: Eliminating server idle power. *SIGARCH Comput. Archit. News*, 37(1):205–216, March 2009.

[254] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap: Eliminating server idle power. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIV, page 205–216, New York, NY, USA, 2009. Association for Computing Machinery.

[255] Francesca Meneghello, Davide Cecchinato, and Michele Rossi. Mobility prediction via sequential learning for 5g mobile networks. In *2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 1–6, 2020.

[256] Xianghang mi, Siyuan Tang, Zhengyi Li, Xiaojing Liao, Feng Qian, and Xiaofeng Wang. Your phone is my proxy: Detecting and understanding mobile proxy networks. 01 2021.

[257] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan 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*, SIGCOMM '17, page 15–28, New York, NY, USA, 2017. Association for Computing Machinery.

[258] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan 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*, SIGCOMM '17, page 15–28, New York, NY, USA, 2017. Association for Computing Machinery.

[259] Bincheng Wang M. Fareed Arif Syed Rafiul Hussain Omar Chowdhury 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.

[260] Ali Mohammadkhan, K. K. Ramakrishnan, and Vivek A. Jain. Cleang—improving the architecture and protocols for future cellular networks with nfv. *IEEE/ACM Transactions on Networking*, 28(6):2559–2572, 2020.

[261] YoungGyoun Moon, SeungEon Lee, Muhammad Asim Jamshed, and KyoungSoo Park. Acceltcp: Accelerating network applications with stateful TCP offloading. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 77–92, Santa Clara, CA, February 2020. USENIX Association.

[262] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the slammer worm. *IEEE Security and Privacy*, 1(4):33–39, July 2003.

[263] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. Dream: Dynamic resource allocation for software-defined measurement. In *Proceedings of the ACM SIGCOMM Conference*, 2014.

[264] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. Dream: Dynamic resource allocation for software-defined measurement. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 419–430, New York, NY, USA, 2014. Association for Computing Machinery.

[265] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. Scream: Sketch resource allocation for software-defined measurement. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, New York, NY, USA, 2015. Association for Computing Machinery.

[266] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. Trumpet: Timely and precise triggers in data centers. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 129–143, New York, NY, USA, 2016. Association for Computing Machinery.

[267] Fionn Murtagh and Pierre Legendre. Ward's hierarchical agglomerative clustering method: Which algorithms implement ward's criterion? *Journal of Classification*, 31:274–295, 2014.

[268] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 85–98, New York, NY, USA, 2017. Association for Computing Machinery.

[269] R. Neugebauer, G. Antichi, J. F. Zazo, Yury Audzevich, S. López-Buedo, and A. Moore. Understanding pcie performance for end host networking. *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018.

[270] Dan Nicolaescu, Alexander Veidenbaum, and Alexandru Nicolau. Using a way cache to improve performance of set-associative caches. In Jesús Labarta, Kazuki Joe, and Toshinori Sato, editors, *High-Performance Computing*, pages 93–104, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[271] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: A scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, page 39–50, New York, NY, USA, 2009. Association for Computing Machinery.

[272] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: A scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, 39(4):39–50, August 2009.

[273] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '11, page 103–114, New York, NY, USA, 2011. Association for Computing Machinery.

[274] Sourav Panda, Yixiao Feng, Sameer G Kulkarni, K. K. Ramakrishnan, Nick Duffield, and Laxmi 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*, CoNEXT '21, page 60–75, New York, NY, USA, 2021. Association for Computing Machinery.

[275] Sourav Panda, K. K. Ramakrishnan, and Laxmi N. Bhuyan. pmach: Power and migration aware container scheduling. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–12, 2021.

[276] Sourav Panda, K. K. Ramakrishnan, and Laxmi N. Bhuyan. Synergy: A smartnic accelerated 5g dataplane and monitor for mobility prediction. In *2022 IEEE 30th International Conference on Network Protocols (ICNP)*, pages 1–12, 2022.

[277] Weiwu Pang, Sourav Panda, Jehangir Amjad, Christophe Diot, and Ramesh Govindan. CloudCluster: Unearthing the functional structure of a cloud service. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1213–1230, Renton, WA, April 2022. USENIX Association.

[278] Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. Heuristics for vector bin packing. January 2011.

[279] Harun Pirim, Engin Bayraktar, and Burak Eksioglu. *Tabu Search: A Comparative Study.* 09 2008.

[280] Michele Polese, Leonardo Bonati, Salvatore d'oro, Stefano Basagni, and Tommaso Melodia. Understanding o-ran: Architecture, interfaces, algorithms, security, and research challenges, 02 2022.

[281] Michele Polese, Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. Understanding o-ran: Architecture, interfaces, algorithms, security, and research challenges. *IEEE Communications Surveys Tutorials*, pages 1–1, 2023.

[282] Salvatore Pontarelli, Roberto Bifulco, Marco Bonola, Carmelo Cascone, Marco Spaziani, Valerio Bruschi, Davide Sanvito, Giuseppe Siracusano, Antonio Capone, Michio Honda, Felipe Huici, and Giuseppe Siracusano. Flowblaze: Stateful packet processing in hardware. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 531–548, Boston, MA, February 2019. USENIX Association.

[283] George Prekas, Mia Primorac, Adam Belay, Christos Kozyrakis, and Edouard Bugnion. Energy proportionality and workload consolidation for latency-critical applications. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, page 342–355, New York, NY, USA, 2015. Association for Computing Machinery.

[284] Shixiong Qi, Sameer G Kulkarni, and K. K. Ramakrishnan. Understanding container network interface plugins: Design considerations and performance. In *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN*, pages 1–6, 2020.

[285] Darijo Raca, Dylan Leahy, Cormac J. Sreenan, and Jason J. Quinlan. Beyond throughput, the next generation: A 5g dataset with channel and context metrics. In *Proceedings of the 11th ACM Multimedia Systems Conference*, MMSys '20, page 303–308, New York, NY, USA, 2020. Association for Computing Machinery.

[286] Anirudh Ramachandran, Srinivasan Seetharaman, Nick Feamster, and Vijay Vazirani. Fast monitoring of traffic subpopulations. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, IMC '08, pages 257–270, New York, NY, USA, 2008. ACM.

[287] Haakon Ringberg, Augustin Soule, Jennifer Rexford, and Christophe Diot. Sensitivity of pca for traffic anomaly detection. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '07, page 109–120, New York, NY, USA, 2007. Association for Computing Machinery.

[288] Haakon Ringberg, Augustin Soule, Jennifer Rexford, and Christophe Diot. Sensitivity of pca for traffic anomaly detection. *SIGMETRICS Perform. Eval. Rev.*, 35(1):109–120, June 2007.

[289] S. Robertson, E. V. Siegel, M. Miller, and S. J. Stolfo. Surveillance detection in high bandwidth environments. In *Proceedings DARPA Information Survivability Conference and Exposition*, volume 1, pages 130–138 vol.1, 2003.

[290] Erik Rolland, Hasan Pirkul, and Fred Glover. Tabu search for graph partitioning. *Annals of Operations Research*, 63:209–232, 04 1996.

[291] Volker Roth and Randy H. Katz. Listen and whisper: Security mechanisms for BGP. In *First Symposium on Networked Systems Design and Implementation (NSDI 04)*, San Francisco, CA, March 2004. USENIX Association.

[292] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. *SIGCOMM Comput. Commun. Rev.*, 45(4):123–137, August 2015.

[293] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 123–137, New York, NY, USA, 2015. Association for Computing Machinery.

[294] Adam Ruprecht, Danny Jones, Dmitry Shiraev, Greg Harmon, Maya Spivak, Michael Krebs, Miche Baker-Harvey, and Tyler Sanderson. Vm live migration at scale. In *Proceedings of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '18, page 45–56, New York, NY, USA, 2018. Association for Computing Machinery.

[295] Adam Ruprecht, Danny Jones, Dmitry Shiraev, Greg Harmon, Maya Spivak, Michael Krebs, Miche Baker-Harvey, and Tyler Sanderson. Vm live migration at scale. *SIGPLAN Not.*, 53(3):45–56, March 2018.

[296] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pages 166–171, 2011.

[297] Kirk Schloegel, George Karypis, and Vipin Kumar. Multilevel diffusion schemes for repartitioning of adaptive meshes. *JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING*, 47:109–124, 1997.

[298] Altaf Shaik, Ravishankar Borgaonkar, N. Asokan, Valtteri Niemi, and Jean-Pierre Seifert. Practical attacks against privacy and availability in 4g/lte mobile communication systems. In *23rd Annual Network and Distributed System Security Symposium (NDSS 2016)*, United States, 2016. Internet Society. NDSS 2016 ; Network and Distributed System Security Symposium ; Conference date: 21-02-2016 Through 24-02-2016.

[299] Umesh Shankar. Active mapping: Resisting nids evasion without altering traffic. Technical Report UCB/CSD-03-1246, EECS Department, University of California, Berkeley, Dec 2002.

[300] F. Silveira and C. Diot. Urca: Pulling out anomalies by their root causes. In *2010 Proceedings IEEE INFOCOM*, pages 1–9, 2010.

[301] Fernando Silveira, Christophe Diot, Nina Taft, and Ramesh Govindan. Astute: Detecting a different class of traffic anomalies. *SIGCOMM Comput. Commun. Rev.*, 40(4):267–278, August 2010.

[302] Sumeet Singh, C. Estan, G. Varghese, and S. Savage. The earlybird system for real-time detection of unknown worms. 2005.

[303] Shan Sinha, Srikanth Kandula, and Dina Katabi. Harnessing tcp's burstiness with flowlet switching. In *Proc. 3rd ACM Workshop on Hot Topics in Networks (Hotnets-III)*. Citeseer, 2004.

[304] John Sonchack, Adam J. Aviv, Eric Keller, and Jonathan M. Smith. Turboflow: Information rich flow record generation on commodity switches. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pages 11:1–11:16, New York, NY, USA, 2018. ACM.

[305] Haoyu Song, T. Sproull, M. Attig, and J. Lockwood. Snort offloader: a reconfigurable hardware nids filter. In *International Conference on Field Programmable Logic and Applications, 2005.*, pages 493–498, 2005.

[306] Hardik Soni, Myriana Rifai, Praveen Kumar, Ryan Doenges, and Nate Foster. Composing dataplane programs with p4. 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*, SIGCOMM '20, page 329–343, New York, NY, USA, 2020. Association for Computing Machinery.

[307] Ashwin Sridharan, Rakesh K. Sinha, Rittwik Jana, Bo Han, K.K. Ramakrishnan, N.K. Shankaranarayanan, and Ioannis Broustis. Multi-path tcp: Boosting fairness in cellular

networks. In *2014 IEEE 22nd International Conference on Network Protocols*, pages 275–280, 2014.

[308] Mark Sullivan and Andrew Heybey. Tribeca: A system for managing large databases of network traffic. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '98, page 2, USA, 1998. USENIX Association.

[309] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. Simplifying datacenter network debugging with pathdump. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 233–248, Savannah, GA, November 2016. USENIX Association.

[310] Lin Tan and T. Sherwood. A high throughput string matching architecture for intrusion detection and prevention. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 112–122, 2005.

[311] Andrew S Tanenbaum and Herbert Bos. *Modern operating systems*. Pearson, 2015.

[312] Lu Tang, Qun Huang, and Patrick P. C. Lee. Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 2026–2034, 2019.

[313] Yuta Tokusashi, Huynh Tu Dang, Fernando Pedone, Robert Soulé, and Noa Zilberman. The case for in-network computing on demand. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, New York, NY, USA, 2019. Association for Computing Machinery.

[314] B. Vamanan, H. B. Sohail, J. Hasan, and T. N. Vijaykumar. Timetrader: Exploiting latency tail to save datacenter energy for online search. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 585–597, 2015.

[315] G. Varghese and Tony Lauck. Hashed and hierarchical timing wheels: Data structures for the efficient implementation of a timer facility. *ACM SIGOPS Operating Systems Review*, 21:25–38, 11 1987.

[316] Nedeljko Vasić, Prateek Bhurat, Dejan Novaković, Marco Canini, Satyam Shekhar, and Dejan Kostić. Identifying and using energy-critical paths. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, New York, NY, USA, 2011. Association for Computing Machinery.

[317] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[318] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pmapper: Power and migration cost aware application placement in virtualized systems. In Valérie Issarny and Richard Schantz, editors, *Middleware 2008*, pages 243–264, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[319] Péter Vörös, Dániel Horpácsi, Róbert Kitlei, Dániel Leskó, Máté Tejfel, and Sándor Laki. T4p4s: A target-independent compiler for protocol-independent packet processors. In *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–8, 2018.

[320] Nicholas Weaver, Robin Sommer, and Vern Paxson. Detecting forged tcp reset packets. 09 2009.

[321] Cuidi Wei, Ahan Kak, Nakjung Choi, and Timothy Wood. 5gperf: Profiling open source 5g ran components under different architectural deployments. In *Proceedings of the ACM SIGCOMM Workshop on 5G and Beyond Network Measurements, Modeling, and Use Cases*, 5G-MeMU '22, page 43–49, New York, NY, USA, 2022. Association for Computing Machinery.

[322] Daniel Wong. Peak efficiency aware scheduling for highly energy proportional servers. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, page 481–492. IEEE Press, 2016.

[323] Daniel Wong. Peak efficiency aware scheduling for highly energy proportional servers. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, page 481–492. IEEE Press, 2016.

[324] Daniel Wong. Peak efficiency aware scheduling for highly energy proportional servers. *SIGARCH Comput. Archit. News*, 44(3):481–492, June 2016.

[325] Daniel Wong, Julia Chen, and Murali Annavaram. A retrospective look back on the road towards energy proportionality. In *2015 IEEE International Symposium on Workload Characterization*, pages 110–111, 2015.

[326] S. Woo and K. Park. Scalable tcp session monitoring with symmetric receive-side scaling. 2012.

[327] Timothy Wood, K. K. Ramakrishnan, Prashant Shenoy, and Jacobus van der Merwe. Cloudnet: Dynamic pooling of cloud resources by live wan migration of virtual machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '11, page 121–132, New York, NY, USA, 2011. Association for Computing Machinery.

[328] Yaxiong Xie and Kyle Jamieson. Ng-scope: Fine-grained telemetry for nextg cellular networks. *CoRR*, abs/2201.05281, 2022.

[329] Jiarong Xing, Qiao Kang, and Ang Chen. Netwarden: Mitigating network covert channels while preserving performance. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2039–2056. USENIX Association, August 2020.

[330] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements.

In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 561–575, New York, NY, USA, 2018. Association for Computing Machinery.

[331] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 561–575, New York, NY, USA, 2018. ACM.

[332] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 29–42, Lombard, IL, 2013. USENIX.

[333] Lihua Yuan, Chen-Nee Chuah, and Prasant Mohapatra. Progme: Towards programmable network measurement. *IEEE/ACM Trans. Netw.*, 19(1):115–128, February 2011.

[334] Ruan Yuan, Yang Weibing, Chen Mingyu, Zhao Xiaofang, and Fan Jianping. Robust tcp reassembly with a hardware-based solution for backbone traffic. In *2010 IEEE Fifth International Conference on Networking, Architecture, and Storage*, pages 439–447, 2010.

[335] Yifei Yuan, Dong Lin, Ankit Mishra, Sajal Marwaha, Rajeev Alur, and Boon Thau Loo. Quantitative network monitoring with netqre. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 99–112, New York, NY, USA, 2017. Association for Computing Machinery.

[336] Ahmed H. Zahran, Jason J. Quinlan, K. K. Ramakrishnan, and Cormac J. Sreenan. Asap: Adaptive stall-aware pacing for improved dash video experience in cellular networks. *ACM Trans. Multimedia Comput. Commun. Appl.*, 14(3s), jun 2018.

[337] Zeek. Bro scan script. https://github.com/zeek/bro-scripts/blob/master/scan.bro, 2011.

[338] Kaiyuan Zhang, Danyang Zhuo, and Arvind Krishnamurthy. Gallium: Automated software middlebox offloading to 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*, SIGCOMM '20, page 283–295, New York, NY, USA, 2020. Association for Computing Machinery.

[339] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. High-resolution measurement of data center microbursts. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, page 78–85, New York, NY, USA, 2017. Association for Computing Machinery.

[340] T. Zhang, L. Linguaglossa, M. Gallo, P. Giaccone, and D. Rossi. Flowatcher-dpdk: Lightweight line-rate flow-level monitoring in software. *IEEE Transactions on Network and Service Management*, 16(3):1143–1156, 2019.

[341] Junhui Zhao, Shanjin Ni, Lihua Yang, Ziyang Zhang, Yi Gong, and Xiaohu You. Multiband cooperation for 5g hetnets: A promising network paradigm. *IEEE Vehicular Technology Magazine*, 14(4):85–93, 2019.

[342] Zhipeng Zhao, Hugo Sadok, Nirav Atre, James C. Hoe, Vyas Sekar, and Justine Sherry. Achieving 100gbps intrusion prevention on a single server. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1083–1100. USENIX Association, November 2020.

[343] Dong Zhou, M. Kaminsky, Justine Sherry, and S. Ratnasamy. Data structure engineering for high performance software packet processing. 2019.

[344] Kai Zhou, Wei Wan, Xi Chen, Zhijiang Shao, and Lorenz T. Biegler. A parallel method with hybrid algorithms for mixed integer nonlinear programming. In Andrzej Kraslawski and Ilkka Turunen, editors, *23rd European Symposium on Computer Aided Process Engineering*, volume 32 of *Computer Aided Chemical Engineering*, pages 271–276. Elsevier, 2013.

[345] L. Zhou, L. N. Bhuyan, and K. K. Ramakrishnan. Goldilocks: Adaptive resource provisioning in containerized data centers. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 666–677, 2019.

[346] Zhongliu Zhuo, Yang Zhang, Zhi-li Zhang, Xiaosong Zhang, and Jingzhong Zhang. Website fingerprinting attack on anonymity networks based on profile hidden markov model. *IEEE Transactions on Information Forensics and Security*, 13(5):1081–1095, 2018.