Monitoring OpenFlow Virtual Networks via Coordinated Switch-Based Traffic Mirroring

Sogand Sadrhaghighi, Mahdi Dolati, Majid Ghaderi, and Ahmad Khonsari,

***Abstract*—As network virtualization becomes ubiquitous, legacy hardware-based traffic monitoring systems are no longer viable for dynamic traffic inspection at arbitrary locations in virtual networks. In this paper, we present the design and evaluation of Open Virtual Tap (OVT), a software-defined solution to replace hardware taps for traffic monitoring in OpenFlow virtual networks by utilizing mirroring capabilities of OpenFlow switches. The key idea behind OVT is the joint configuration of all switches in the substrate physical network in order to efficiently mirror flows from all virtual networks. We show that such a design avoids inefficiencies that result from existing software-based traffic mirroring solutions in which each virtual network configures its own switches independently of other virtual networks. We evaluate OVT using model-driven simulations as well as Mininet experiments with realistic applications for intrusion detection and video telephony analysis. Specifically, in our experiments, we observe that OVT can achieve up to 20% improvement in flow coverage compared to existing traffic mirroring approaches.**

***Index Terms*—Network monitoring, Traffic mirroring, Virtual networks, Software-defined networks.**

I. Introduction

Motivation. Network virtualization has emerged as a core technology in modern networks, specially in multi-tenant datacenter networks. In its general form, network virtualization enables multiple virtual networks (VNs) to coexist on the same substrate physical network through abstraction and isolation mechanisms. While network virtualization brings about substantial benefits such as greater operational efficiency and improved network security, it also creates a new set of challenges for managing VNs. Network management functions that rely on direct access to physical network devices (*e.g.*, switches) do not work properly when deployed in VNs. In particular, to detect network events ranging from performance impairments to security breaches, network operators require continuous, real-time network traffic monitoring at the granularity of packets [1], [2]. Traditionally, monitoring network traffic at packet-level is accomplished using hardware tap devices [3]. However, such hardware middleboxes are less useful in VNs. Once hardware taps are installed, which require access to physical network devices, they cannot be dynamically deployed to another location. Without the ability to capture and analyze traffic at arbitrary points in the network, many management tasks such as fault diagnosis, root cause analysis and malware detection can not be effectively executed. The necessity of packet-level traffic monitoring in VNs motivates adopting software-based solutions. Software-based taps can be dynamically deployed to different locations in a network, making them particularly well-suited for environments where VNs are created and deleted on the fly.

A few recent works [4]–[6] design software-defined packet-level traffic monitoring systems. These works utilize either *port mirroring* feature available on most commodity switches [7] or *flow mirroring* feature available on programmable OpenFlow switches [8]. In port mirroring, a switch duplicates the passing traffic to a designated mirroring port. While simple and widely supported on even legacy switches, port mirroring leads to inefficient network traffic monitoring as the traffic mirroring is at the port-level granularity. In flow mirroring, on the other hand, the traffic mirroring is at the flow-level granularity. While flow mirroring provides a more efficient traffic monitoring solution, it is only supported on programmable switches, *e.g.*, OpenFlow switches. Alternative works on softwarized network monitoring based on Open vSwitch [9] can be found in [10]–[12]. All the aforementioned works, however, are designed with a *single* network in mind. Although they are software-based solutions and can be deployed in VNs, they do not work efficiently as they do not explicitly account for having *multiple* VNs running on the same substrate network.

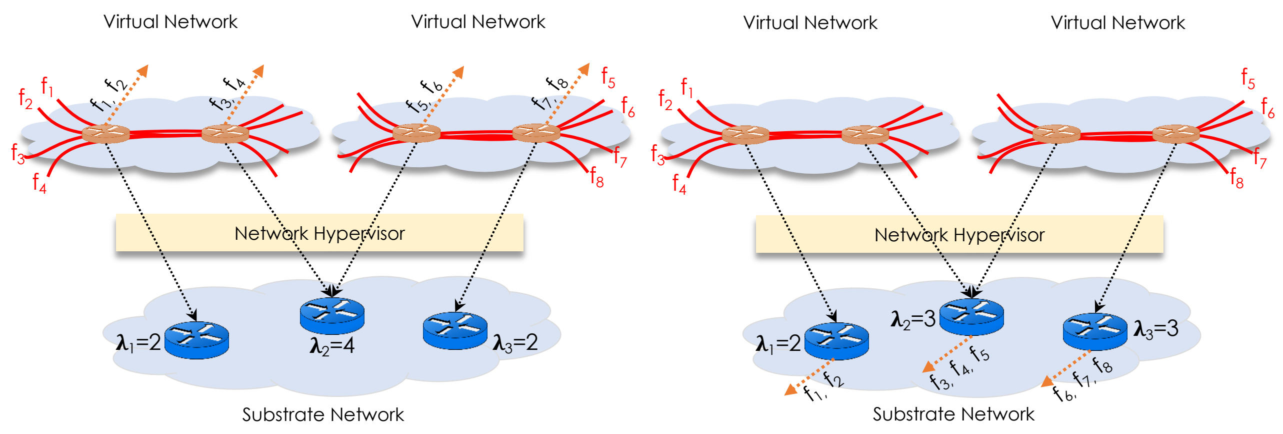
As a motivating example, consider the network environment depicted in Fig. 1, which depicts two VNs running on the same substrate network. Each VN has two virtual switches and four flows that pass through both switches. Flows have the same normalized traffic rate of 1. The substrate network has three physical switches such that two virtual switches, one from each VN, are mapped to a single physical switch, a common scenario in network virtualization. In OpenFlow networks, switch virtualization can be achieved by slicing the available flow table space on the physical switch among the virtual switch instances using an OpenFlow network hypervisor such as OpenVirtex [13] or FlowVisor [14]. Assume that a software-defined traffic mirroring system, such as SoftTap [6], is deployed in each VN. To maximize the number of mirrored flows and balance the mirroring load among switches, SoftTap mirrors two flows on each virtual switch in each VN. While individual mirroring configuration appears optimal from the perspective of each VN, it leads to a highly unbalanced mirroring load on physical switches in the substrate network.

Sogand Sadrhaghighi and Majid Ghaderi were supported by Natural Sciences and Engineering Research Council of Canada and Alberta Innovates.

Sogand Sadrhaghighi and Majid Ghaderi are with the Department of Computer Science, University of Calgary, Canada. (Email: sogand.sadrhaghighi@ucalgary.ca; mghaderi@ucalgary.ca)

Mahdi Dolati is with the Department of Electrical and Computer Engineering, University of Tehran, Iran. E-mail: mahdidolati@ut.ac.ir

Ahmad Khonsari is with the Department of Electrical and Computer Engineering, University of Tehran, and Institute for Research in Fundamental Sciences (IPM), Iran. E-mail: a_khonsari@ut.ac.ir



(a) Independent flow mirroring.

(b) Coordinated flow mirroring.

Fig. 1: Two VNs sharing a physical switch. By jointly configuring flow mirroring of all switches in the network, the maximum mirroring load on switches can be reduced by 25% compared to configuring switches in each VN independently.

Specifically, the maximum mirroring load on physical switches in this example will be equal to 4. Clearly, this mirroring configuration is sub-optimal. If mirroring decisions were made jointly for all VNs by a central controller, a more efficient mirroring configuration would be to mirror three flows on the shared switch and distribute the remaining flows among the other two switches. In this case, the maximum mirror load on physical switches would be 3, a 25% improvement compared to the first scenario. A lower mirroring load means that for the same mirroring capacity, more flows can be mirrored in the network (see Section VI). In general, if each VN has N virtual switches and F flows (where typically $F \gg N$), and only one switch is shared between the two VNs, then the maximum mirroring load when each VN mirrors its own flows independently will be equal to $\lceil 2F/N \rceil$. On the other hand, if mirroring decisions are made jointly for both VNs, the maximum mirroring load on physical switches will be equal to $\lceil 2F/(2N-1) \rceil$. This results in a maximum load ratio of $N/(2N-1)$, which converges to 50% for large N.

From this example, it is clear that to have an efficient switch-based traffic mirroring system, the mirroring configurations of all physical switches must be adjusted jointly across all VNs. Such adjustment requires global network information as well as the ability to dynamically change mirroring configurations of physical switches as VNs and their traffic change.

Our Approach. Our approach is based on global optimization of switch mirroring configurations with respect to traffic and routing layouts in all VNs sharing a substrate network. To this end, we present the design and evaluation of Open Virtual Tap (OVT), a software-defined tap for traffic mirroring in OpenFlow VNs. By employing physical switches for traffic mirroring, OVT is able to provide pervasive visibility covering as many flows as possible in the network, while scaling out as the network expands. OVT does not require any switch modifications and can be implemented on top of any OpenFlow-based network virtualization technology.

The key idea behing OVT is to transparently proxy flow mirroring requests from VNs to a substrate-level *orchestrator* that has global visibility of physical switches via the OpenFlow network hypervisor. Specifically, OVT has a local *agent* that is deployed as an OpenFlow application in each VN. From the perspective of VNs, the local agent receives flow mirroring queries within the VN and proxies them to a central orchestrator that runs on the network hypervisor. The orchestrator, then, pools all flow mirroring queries together and invokes the OVT *optimizer* to compute globally optimal mirroring configurations using either port or flow mirroring strategy. The OVT optimizer is a crucial component of OVT, which has to be scalable, efficient and near-realtime. Thus, a major portion of this work is dedicated to designing efficient algorithms for the OVT optimizer. Depending on the mirroring strategy utilized in the network, *i.e.*, port or flow mirroring, the optimizer has to consider a different set of constraints when computing mirroring configurations. In particular, with flow mirroring, a portion of the flow table space on each switch has to be reserved for OVT rules. Recall that on OpenFlow switches, forwarding rules are implemented using TCAM (Ternary Content-Addressable Memory). As such, it is critical to keep the reserved space to a minimum as TCAM capacity on OpenFlow switches is quite limited [15], [16]. Additionally, TCAM is an expensive and power hungry resource.

Contributions. We address the problem of packet-level traffic monitoring in OpenFlow VNs using switch-based mirroring. Previous software-based solutions for packet-level monitoring do not explicitly account for having *multiple* VNs running on the same substrate network, possibly resulting in increased monitoring overhead. We propose OVT, a monitoring system in OpenFlow VNs, that transparently proxy flow mirroring requests from VNs to a substrate-level orchestrator. The orchestrator maintains global substrate-level switch visibility and computes network-wide switch mirroring configurations such that the *aggregate* switch mirroring load generated by all VNs is minimized. Consequently, reducing the maximum switch mirroring load, improves flow visibility, *i.e.*, the percentage of flows that are fully mirrored. To the best of our knowledge the idea of coordinated flow mirroring across multiple VNs to improve flow visibility is novel.

The main contributions of this paper are summarized below:

- We present the design and evaluation of OVT, a scalable software-defined tap for flow mirroring in OpenFlow VNs based on traffic mirroring capabilities of OpenFlow switches.
- We formulate the problem of determining the global mirroring switch configurations under port and flow mirroring strategies, while accounting for switch mirroring constraints such as limited mirroring and TCAM capacities.
- We design polynomial-time approximation algorithms with provable approximations ratios for computing globally optimal switch configurations, which is an NP-hard problem in its exact form.
- We present simulation results to study the performance and scalability of OVT in large networks with realistic traffic patterns and compare that with existing approaches. We also present Mininet experiments to show the utility of OVT when used with practical applications in a realistic network environment.

Organization. Section II reviews the related work on traffic monitoring. In Section III we discuss the design of OVT. Sections IV and Section V are dedicated to algorithm design for OVT Optimizer under port and flow mirroring strategies, respectively. Performance evaluation results are presented in Section VI, while Section VII concludes the paper.

II. Related Works

In this section, we review existing works on network traffic monitoring in SDNs. For each monitoring method, we first study the virtualization-unaware approaches. We, then, have a closer look at the software-based monitoring systems that consider the existence of VNs in their design.

Statistics-based Methods. Various studies utilize the built-in OpenFlow statistics collection mechanisms for statistics collection in physical SDNs. Examples of such works are presented in [17] and [18], both of which minimize per-switch TCAM usage when installing OpenFlow statistics collection rules. Both approaches incur long delays and overhead since they require the SDN controller to periodically poll the switches. With respect to the above problems, the work [19] minimizes the latency of statistics collection, and the work [20] minimizes the collection overhead. However, none of the indicated approaches address specific challenges of statistics collection in the presence of multiple VNs, *e.g.*, non-isolated statistics between VNs.

Some recent works propose statistics-based monitoring frameworks for VNs. Examples of such frameworks are presented in [21]–[23]. In particular, Lattice [21] proposes a statistical-based monitoring system for resource and service monitoring by deploying multiple management systems on top of the physical network infrastructure. The work [22] applies Lattice for monitoring datacenter slices and investigates Lattice's applicability for dynamic monitoring of sliced end-to-end infrastructures. Similarly, the work [23] utilizes the mapping between virtual and physical networks to collect per-VN statistics. Specifically, the proposed system minimizes the statistics collection delay of each VN, by periodically feeding the VN controllers with relevant statistics. The works in this category are limited to flow statistics collection. There are, however, applications that require the actual packets' metadata or payload information, *e.g.*, deep packet inspection.

Sampling Methods. Traffic sampling is the traditional approach for low-overhead fine-grained packet-level information collection. OpenFlow specification, however, does not support traffic sampling. Consequently, the works [24]–[26] study adding standard sampling modules, similar to Netflow [27] and sFlow [28], to OpenFlow switches. Alternatively, the works [29] and [30] propose sketch-based sampling frameworks for software switches. All of the above mentioned works use probabilistic sampling and are biased towards long flows. To lessen the bias and account for short flows, the works [31] and [32] propose time-based and flow-based sampling in SDNs, respectively. Although these solutions are deployable in softwarized switches such as Open vSwitch, their sub-optimal sampling configuration in the presence of multiple VNs cause severe sampling load-imbalance across physical switches.

Virtualization-aware traffic sampling has been discussed in [33] and [34]. Particularly, a sFlow-based monitoring system for multi-tenant SDNs is implemented in [33]. The authors, however, do not take into account the traffic layout of each VN, which makes their approach susceptible to load imbalance. To address this issue, authors in [34] employ the knowledge of traffic layout by estimating the traffic matrix from sampled link measurement data.

Packet-level Methods. Packet-level monitoring provides a thorough picture of the network that is not achievable through statistics-based and traffic sampling approaches. Planck [4] is a solution that employs the port mirroring technique to extract global network information, from physical switches, on a millisecond time scale. Since Planck does not address the optimization problem of selecting the mirrored ports, it suffers from low flow coverage and high mirroring redundancy. Alternatively, the per-flow mirroring technique is used to verify the routing compliance and detect faults in datacenters in [35] and [36], respectively. Both of these works are compatible with the capabilities of existing commodity switches. Further work on softwarized packet-level monitoring has been proposed in [11], [12] and [6]. In particular, the authors in [11] use the extended Berkeley packet filter (eBPF) to implement a software-based monitoring system. Their solution is limited to systems with communicating co-located VMs, since it relies on the host OS to duplicate the packets. At the same time, the work [12] proposes a software-based tap to duplicate the traffic between OpenStack-based VMs, using the group table feature. This work restricts flow duplication to the edge switches and lacks the required flexibility to balance the mirroring load, leading to performance degradation and monitoring loss. Authors in [6] design a system named SoftTap to address the problem of balancing the mirroring load across switches in SDNs. The work addresses calculating mirroring configurations using port and flow mirroring strategies. None of these works, however, consider the effect of having multiple VNs on the same substrate network and can make inefficient localized mirroring decisions that reduce the flow coverage and increase the load imbalance. In contrast, OVT is based on global optimization of mirroring configurations with respect to traffic layout in all VNs. It is worth mentioning that, there exist high-performance commercial products for processing captured network traffic at high rate. For example, Arista provides a monitoring system, called DANZ Monitoring Fabric (DMF) [37], to process high-rate traffic collected through physical taps and mirroring ports. DMF, however, does not address the problem of specifying the mirroring configurations for target flows, which results in inefficient traffic collection and lower flow visibility.

Orthogonal to our work, the work [38] provides a packet anonymizer, which allows obfuscating sensitive information and can be used to address privacy concerns of packet-level traffic monitoring schemes such as OVT.

Compiled Queries in the Programmable Data Plane. With the emergence of programmable switches, some works have sought to offload all or partial monitoring tasks to be executed on the switches. Examples of such works are [39]–[42]. In particular, UnivMon [39] and Marple [40] specify a set of fixed operations that can be executed on a programmable switch. However, the current programmable switches limited computational and memory resources cannot meet the requirements posed by general applications, requiring more complex operations. Sonata [41] reduces the memory requirements of the queries by introducing a method to refine each query and ensure that available resources focus only on traffic that satisfies the query. However, as shown in [43], still, not all queries can fit within the limited resources of the switches. To address this problem, *Flow [42] exports telemetry data from packet traces and offloads the rest of the processing to the application software. This approach, however, does not address the question of efficient network-wide collection of the required flow information.

In-band Network Telemetry (INT). In-Band Network Telemetry (INT) is a framework for collecting telemetry items and switches internal state information from the data plane. In INT, the switches read the instructions and add the required data to the packet headers as the packets traverse the network. Existing INT implementations have high overheads due to per-packet operation. To address this problem, the authors in [44] employ a probabilistic packet processing scheme. Similarly, the work [45] reduces the overhead by keeping the packet length constant. The main limitation of these works is that they rely on specialized data planes that are not yet widely available in production environments. While OVT can be implemented using such switches to allow finer-grained monitoring, *e.g.*, mirroring just the packet headers, it is not bound to these switches.

Host-based Monitoring. An alternative approach to the above-mentioned network monitoring works is to offload network monitoring to end-hosts. In these works, users define monitoring events, which are then installed as triggers on each end-host. Incoming packets may trigger a notification to the controller when an event is detected. Following that, the controller polls all end-hosts to aggregate all traffic for detecting network-wide events. Examples of such works are presented in [46]–[48]. In particular, PathDump [46] presents a monitoring framework for a limited number of applications using statistics at the flow granularity, while Trumpet [47] defines a limited number of events at packet granularity. Confluo [48] improves Trumpet by introducing a new data structure that allows for a broader range of telemetry applications. These works, however, require sending information to a central controller for network-wide event detection. This aggregation may be challenging due to limited network resources at each end-host server. For example, the number of network interface cards in hosts is usually limited and using one of them only for the monitoring purpose is not ideal. On the other hand, switches have more available ports, and on many occasions, some of these ports are un-used [49]. These un-used ports can then be configured for mirroring in OVT.

III. System Design

The high-level design of OVT is depicted in Fig. 2. As shown in the figure, OVT has four components, namely Agent, Collector, Optimizer and Orchestrator. Each VN has a local agent, which implements the functionality of a virtual tap for the VN. Analyzer applications that require traffic capture send their queries to their local agent. Agents, in turn, communicate with the orchestrator, which collects all mirroring queries. The orchestrator then communicates with the optimizer, which computes a global mirroring configuration. Following that the orchestrator installs appropriate forwarding rules on OpenFlow switches in the substrate network using the network hypervisor southbound interface. All mirrored traffic is then stored on the collector before retrieved by analyzer applications. A more detailed description of each component of the design is provided below.

Network Virtualization. In our design, network virtualization is implemented on top of an OpenFlow-based hypervisor such as OpenVirtex [13]. Specifically, each VN is controlled by its own OpenFlow controller and has its own address space and topology. From the VN's perspective, each OpenFlow controller is directly in charge of configuring its data plane elements. However, in reality, all such communications are intercepted by the network hypervisor, which is in charge of mapping VNs (*i.e.*, virtual switches and virtual address spaces) to the physical substrate network.

Control Flow. The execution sequence of OVT is as follows. Within each VN, an analyzer application sends traffic monitoring queries, *i.e.*, set of flows to be mirrored, to the OVT agent. The agent then forwards the received monitoring queries to the OVT orchestrator module. The orchestrator communicates with the hypervisor to find the mapping between VN flows to the substrate network flows as well as the current usage of mirroring resources on physical switches, *e.g.*, the total port rates and TCAM usage. Following that, the orchestrator passes the collected information, including the substrate network topology, set of flows to be mirrored and physical switches resource usage, to the OVT optimizer module. Subsequently, the optimizer computes an optimal global mirroring configuration for all physical switches and sends it back to the orchestrator for deployment. Finally, according to the computed mirroring

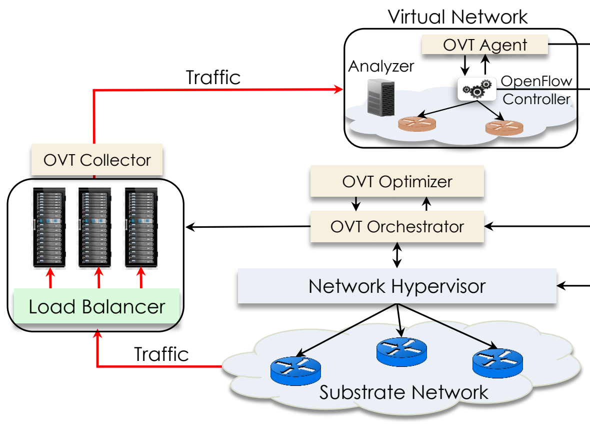


Fig. 2: High-level architecture of Open Virtual Tap (OVT). The red arrows show the data plane, while black arrows show the control plane.

configuration, the orchestrator installs forwarding rules on physical switches. This is achieved through the use of the southbound OpenFlow interface of the hypervisor.

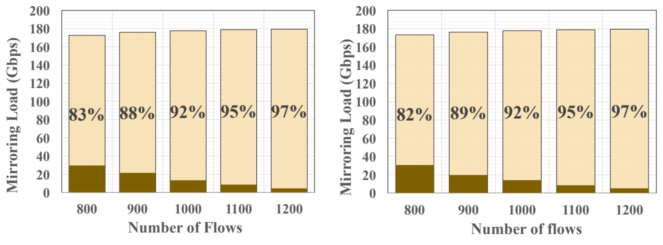
OVT Agent. The agent is implemented as an OpenFlow application on each VN's controller and receives mirroring queries from traffic analyzer applications. A mirroring query specifies either a particular set of flows to be mirrored or a wildcard request to mirror all flows in the VN. For example, the former could be a set of flows of interest, specified by a light-weight statistics-based IDS application [50], running on the analyzer server. Alternatively, the first few packets of an arriving flow can be forwarded to a light-weight analyzer application to decide whether a flow is of interest for further analysis [51]. In the latter case, the agent communicates with the local OpenFlow controller to determine all active flows in the VN. It is important to note that multiple traffic analyzer applications in the network may send mirroring queries to the agent. The agent forwards all received queries to the orchestrator. The orchestrator assigns a unique ID to each query and returns the set of assigned query IDs to the agent, which, in turn, returns the query IDs to corresponding analyzer applications. As explained later, the query IDs are used by analyzer applications to retrieve mirrored data from OVT collector. While in the current OVT design, it suffices to perform query aggregation at the orchestrator level, the OVT agent can also aggregate mirroring queries received from different analyzer applications by eliminating repetitive flows. To this extent, the agent would keep a mapping table, mapping the aggregated query to the corresponding analyzer applications. Using the mapping table, the agent would forward the obtained (from the orchestrator) query ID to all corresponding analyzer applications. We note that analyzer applications that rely on specific events to trigger traffic monitoring are well-suited for OVT. Furthermore, our design can be extended to prioritize specific analyzer applications (*e.g.*, by attaching a suitable utility function to each application [52]). In such cases, the agent assigns priorities to queries, indicating the monitoring value of the query (see Section V-B).

OVT Collector. The OVT collector consists of a load balancer and multiple storage servers. The number and capacity of storage servers depend on the size of the network and the type of filtering/pre-processing required from the collector. The load balancer distributes packet traces across the servers based on the available capacity and the performance of the storage servers [53]–[55]. Additionally, the load balancer ensures that the traced packets of the same flow are sent to the same server (*e.g.*, using the hash value of the packet headers). Alternatively, there are commercial systems such as Arista's DANZ Monitoring Fabric (DMF) [37] for analyzing and processing the mirrored traffic, which can be integrated into OVT.

When configuring mirroring on switches, the orchestrator installs flow rules that specify the collector (*i.e.*, the address of the load balancer in front of the storage servers) as the destination of the mirrored traffic. The orchestrator also sends the list of mirroring queries, query IDs and virtual-to-physical flow mappings to the collector. When an analyzer application attempts to retrieve mirrored traffic from the collector, it specifies a query ID, which the collector will use to identify the relevant traffic data to forward to the analyzer as well as any address translation that is required to convert from physical to virtual flows (*e.g.*, NAT functionality). The storage servers in the collector can be connected to switches by direct links or over the same substrate network, albeit the latter comes at the cost of increased load on the network.

OVT Orchestrator. The orchestrator receives mirroring queries from the agents. It performs query aggregation by storing all flows specified by all monitoring queries in the *set* $\mathcal{F}$, and thus, omitting the duplicated flows. The orchestrator then communicates with the hypervisor to obtain the mapping between VNs and the substrate network. For instance, a flow in a VN may be specified by its source and destination IP addresses in the VN's address space. To determine which physical switches are on the path of the virtual flow, the orchestrator needs to map the virtual source and destination addresses to their corresponding physical addresses. In addition to computing the mappings, the orchestrator also obtains usage information about the mirroring resources on physical switches, *e.g.*, the total port rate and the TCAM usage. Depending on how virtual switches are mapped to physical switches, the orchestrator attempts to aggregate mirroring requests from multiple agents. This aggregation results in fewer flow entries on physical switches since physical network resources can be mapped to one or more VNs. Since the destination of all mirrored traffic is the collector, which has precise knowledge of queries, aggregated queries do not violate VN segregation.

OVT Optimizer. The optimizer receives information about the set of flows to be mirrored on the substrate network switches. Considering the mirroring resource constraints on switches, it then solves a constrained optimization problem to compute a global mirroring configuration. This configuration, depending on physical switches mirroring capabilities, can be computed at port or flow granularity. To show the critical role of the optimizer in the performance of OVT, we have simulated a datacenter network and implemented an optimizer algorithm based on port mirroring as in Planck [4], where



(a) Web server traffic. (b) Cache server traffic.

Fig. 3: Simulation on a 12-pod FatTree topology showing that redundant flow monitoring results in substantial overhead. For each number of flows, plots show the total rate of mirrored traffic as well as the percentage of the mirrored traffic that is redundant.

every switch mirrors all its ports to its mirroring port. The datacenter topology is a FatTree consisting of 12 pods. Two real-world traffic workloads, namely web and cache server [56] are used to simulate traffic flows in the datacenter. We change the number of flows in the network and compute the amount of mirroring overhead (*i.e.*, unnecessarily duplicated traffic) under each traffic workload. The results are presented in Fig. 3. As can be seen from the figure, the amount of overhead is substantial, reaching up to 97% of the total mirrored traffic volume.

From this example, it is clear that in a useful switch-based traffic mirroring system, the optimizer must compute a global mirroring configuration accounting for the network's routing layout and traffic distribution. The global mirroring configuration results in minimizing the mirroring overhead and consequently maximizing the flow coverage. Designing efficient algorithms for the optimizer is the main focus of the next two sections. We define two different optimization problems, namely port-based traffic mirroring (PortTM) and flow-based traffic mirroring (FlowTM), for the port and flow mirroring strategies, respectively. It is crucial to keep these problems as simple as possible, as they have to be repeatedly solved at runtime by the optimizer. Given that both problems are NP-hard, we design approximation algorithms to solve each problem efficiently. The formal definition of PortTM and FlowTM, and the designed approximation algorithms are discussed in detail in Sections IV and V, respectively.

Notation. The following notational conventions are used in the rest of the paper: Sets are typeset in calligraphic font, *e.g.*, set $\mathcal{Z}$. The cardinality of set $\mathcal{Z}$ is denoted by Z (*i.e.*, typeset in regular font). Given variable Φ_z for $z \in \mathcal{Z}$, we define $\overline{\Phi}_z = \max_{z \in \mathcal{Z}} \Phi_z$.

IV. Port-Based Traffic Mirroring

In this section, we formally define the port-based traffic mirroring (PortTM) problem, also studied in [6], and show that it is NP-hard. We then focus on designing a polynomial-time approximation algorithm to solve the problem.

Definitions. To characterize the inputs of the PortTM optimization program, we use $\mathcal{F}$ and $\mathcal{W}$ to represent the set of flows that have to be mirrored and the set of physical network switches, respectively. We define $\mathcal{P}_w$ to be the set of ports of switch $w \in \mathcal{W}$. For simpler reference, we define

TABLE I: Summary of notations used in the PortTM problem.

Input	Definition
$\mathcal{F}$	Set of all flows
$\mathcal{W}$	Set of all switches
$\mathcal{P}$	Set of all ports in the network
$\mathcal{P}_w$	Set of ports of switch w
$\mathcal{P}_f$	Set of ports that flow f meets
r_p	Traffic rate of port p
Variable	**Definition**
x_p	Decision variable of mirroring port p

$\mathcal{P} = \cup_{w \in \mathcal{W}} \mathcal{P}_w$ to be the set of all switch ports in the network. We say that f meets p if flow f traverses switch $w \in \mathcal{W}$ via port $p \in \mathcal{P}_w$. We define $\mathcal{P}_f$ to be the set of ports that flow $f \in \mathcal{F}$ meets on its path. The path of a flow can be obtained using different path detection methods, *e.g.*, [57]. Based on the obtained paths, we can then specify the set of flows $f \in \mathcal{F}$ that meet port $p \in \mathcal{P}$ and denote them by $\mathcal{F}_p$. The traffic rate of an outgoing port $p \in \mathcal{P}$ is represented by r_p. We can estimate the traffic rate of the outgoing ports by deploying a load estimation mechanism such as the one presented in [58]. Table I summarizes the notations used in the PortTM problem.

A. Problem Formulation

To formulate PortTM, we define binary decision variable x_p to show whether port $p \in \mathcal{P}$ is mirrored or not. Specifically, port $p \in \mathcal{P}$ is mirrored if $x_p = 1$, otherwise $x_p = 0$. Also, we define continuous decision variable λ as PortTM's objective, representing the maximum mirroring load of any switch[1] in the network. The value of λ is computed based on the values of x_p. Problem 1 presents the Integer Linear Programming (ILP) formulation of PortTM. Constraint (1b) ensures that flow $f \in \mathcal{F}$ is mirrored on at least one port it meets on its path, *i.e.*, $\mathcal{P}_f$. Constraint (1c) shows λ's calculation, *i.e.*, the maximum mirroring load on network switches. By introducing λ, we have essentially linearized the objective $\min \max_{w \in \mathcal{W}} \lambda_w$, where λ_w denotes the mirroring load on switch w. Although PortTM resembles the well-known set cover problem, the definition of its objective makes it considerably different. Nevertheless, the following theorem demonstrates that set cover problem can be reduced to PortTM, proving the NP-hardness of PortTM.

Theorem 1. *PortTM problem is NP-hard.*

Proof. Please refer to Appendix A for the proof. □

Furthermore, despite the formulation similarity between PortTM and min-congestion flow routing (MCFR) problem, they are fundamentally different. Consider ports and switches to be equivalent to paths and links in MCFR, respectively. Each flow has to select a path to minimize congestion on links. However, unlike MCFR, in PortTM the mirroring load of each flow depends on the mirroring configuration of other flows. In particular, if two flows use the same mirroring port, the congestion of the corresponding switch is identical to that when only one of them selects that port.

[1]Physical or software switches.

Problem 1: Port-Based Traffic Mirroring (PortTM)

$$
\begin{aligned}
\textbf{PortTM:}\quad & \min \lambda & & \text{(1a)}\\
\text{s.t.}\quad & \textstyle\sum_{p\in\mathcal{P}_f} x_p \ge 1, & \forall f\in\mathcal{F} \quad & \text{(1b)}\\
& \textstyle\sum_{p\in\mathcal{P}_w} x_p r_p \le \lambda, & \forall w\in\mathcal{W} \quad & \text{(1c)}\\
& x_p \in \{0,1\}. & \forall p\in\mathcal{P} \quad & \text{(1d)}
\end{aligned}
$$

B. Algorithm Design

To efficiently solve the PortTM problem, we apply the randomized rounding technique. To this end, we ignore the integrality constraints of decision variables x_p and allow them to take fractional values in the interval $[0,1]$. This procedure, known as *relaxation*, converts the PortTM problem to a *Linear Program* (LP) that can be efficiently solved in polynomial-time using techniques such as the interior point method. However, the obtained fractional values of decision variables can not be used directly, as partial port mirroring is not possible with the port mirroring strategy. Consequently, a port should either be fully mirrored or not mirrored at all.

The standard approach is to *round* the obtained fractional values to integer values. An important factor is to preserve the feasibility of constraints and ensure that the objective does not substantially deteriorate. We can then use the results of the rounding as the mirroring configurations. To produce a high-quality solution, the rounding procedure should exploit information in the values of relaxed variables. As such, we interpret the fractional values of decision variables as *probabilities* to design a randomized rounding procedure. Moreover, since the randomized solution does not guarantee mirroring of all flows, we repeat the rounding procedure multiple times. We will use a parameter K_ϵ that determines the number of rounding iterations needed to ensure that all flows are mirrored with the probability of at least $1-\epsilon$ for any $\epsilon > 0$ (see equation (3)).

Algorithm. Our proposed algorithm based on randomized LP rounding, named ϵ-LPR, is outlined in Algorithm 1. In ϵ-LPR, we use $\widetilde{x}_p$ and $\hat{x}_p$ to show the fractional and the rounded values of the decision variable x_p, respectively. ϵ-LPR obtains the fractional values of the decision variables by solving the relaxed version of the problem in line 1. The algorithm, then, repeatedly executes the rounding procedure for K_ϵ times. The number of mirrored flows, the obtained objective value, and the switch configurations from the i-th rounding iteration are stored in $\hat{\theta}_i$, $\hat{\lambda}_i$, and set $\{\hat{x}_p\}_i$, respectively (see lines 11 and 12). In each iteration, the rounding procedure starts by initializing all $\hat{x}_p$'s to zero (see line 3). Then, for each port $p \in \mathcal{P}$ a random number is drawn from the uniform distribution over the interval $[0,1]$, by calling the uniform random number generator `rand()`. The generated random number is compared with $\widetilde{x}_p$. If the random number is less than or equal to $\widetilde{x}_p$, the value of $\hat{x}_p$ is set to one, otherwise it is set to zero. In lines 7-10, the number of mirrored flows is computed. In line 13, the index of the configuration with the highest number of mirrored flows is determined and stored in set $\mathcal{B}$. Among the configurations specified by $\mathcal{B}$, the one with the lowest mirroring load is determined in line 14 and its associated configuration is returned in line 15 as the final solution.

Alg. 1: Randomized LP Rounding (ϵ-LPR)

```
procedure ε-LPR (F, W, K_ε)
1    {$\widetilde{x}_p$} ← SolveLP($\mathcal{F}, \mathcal{W}$)        /* External LP solver */
2    for $i \leftarrow 1$ to $K_\epsilon$ do
3        $\{\hat{x}_p\} \leftarrow \{0\}$
4        foreach $p \in \mathcal{P}$ do
5            if rand() $\le \widetilde{x}_p$ then
6                $\hat{x}_p \leftarrow 1$
7        $\hat{\theta}_i \leftarrow 0$  /* Number of mirrored flows in iteration i*/
8        for $f \in \mathcal{F}$ do
9            if $\sum_{p\in\mathcal{P}_f} x_p \ge 1$ then
10               $\hat{\theta}_i \leftarrow \hat{\theta}_i + 1$
11       $\hat{\lambda}_i \leftarrow \max_{w\in\mathcal{W}} \sum_{p\in\mathcal{P}_w} \hat{x}_p r_p$
12       $\{\hat{x}_p\}_i \leftarrow \{\hat{x}_p\}$
13   $\mathcal{B} \leftarrow \mathrm{argmax}_{1\le i\le K_\epsilon} \hat{\theta}_i$
14   $i^* \leftarrow \mathrm{argmin}_{i\in\mathcal{B}} \hat{\lambda}_i$
15   return $\{\hat{x}_p\}_{i^*}$
```

Coverage Analysis. We show that to cover all flows with probability of at least $1-\epsilon$, it is sufficient to set K_ϵ to $\lceil \frac{\ln\epsilon}{\ln(1-(1-\frac{1}{e})^F)} \rceil = O(2^{-F}\log(1/\epsilon))$. First, we compute the coverage probability after one round of randomized rounding, *i.e.*, calling ϵ-LPR with $K_\epsilon = 1$. In this case, we show that ϵ-LPR mirrors all flows with probability $(1-\frac{1}{e})^F$. Let E_p denote the event that flow f is mirrored by port p, *i.e.*, port p is mirrored and $p \in \mathcal{P}_f$. The probability that flow f is mirrored by any port on its path is given by $\mathbb{P}\left\{\cup_{p\in\mathcal{P}_f} E_p\right\}$, which is expressed as,

$$
\begin{aligned}
\mathbb{P}\left\{\cup_{p\in\mathcal{P}_f} E_p\right\} &= 1 - \textstyle\prod_{p\in\mathcal{P}_f}(1-\widetilde{x}_p)\\
&\ge 1 - (1-\frac{1}{P_f})^{P_f} \ge 1 - \frac{1}{e}\,.
\end{aligned} \tag{2}
$$

The first inequality follows from the fact that the product of a set of variables with fixed sum is maximized when all of them are equal, and noting that $\sum_{p\in\mathcal{P}_f} \widetilde{x}_p = 1$. The second inequality follows from the inequality $(1-\frac{1}{x})^x \le \frac{1}{e}$, which is valid for $x \ge 1$. Consequently, the probability of mirroring all flows in one round of the algorithm is at least equal to $(1-\frac{1}{e})^F$. Next, consider invoking randomized rounding K_ϵ times. In this case, the mirroring probability of ϵ-LPR is at least equal to $1-(1-(1-\frac{1}{e})^F)^{K_\epsilon}$. Thus, to ensure that the mirroring probability is greater than $1-\epsilon$, it suffices to have,

$$
K_\epsilon = \left\lceil \frac{\ln\epsilon}{\ln(1-(1-\frac{1}{e})^F)} \right\rceil = O(2^{-F}\log(1/\epsilon))\,. \tag{3}
$$

Theorem 2. *ϵ-LPR runs in polynomial time.*

Proof. The complexity of solving a linear program with N variables using the interior point method is $O(N^{3.5})$ [59]. The two nested loops in the algorithm take $O(K_\epsilon P)$ additional time. □

Theorem 3. *ϵ-LPR attains the approximation ratio $\gamma_\delta = \frac{3\log(W/\delta)}{\log\log(W/\delta)}$ with probability $(1-\delta)$.*

Proof. To analyze the approximation ratio, we first present the following form of the Chernoff bound in lemma 1 that we use in the proof.

Lemma 1. (Chernoff bound [60]) *Let $X_1, ..., X_k$ be independent random variables over the interval $[0,1]$ such that*

TABLE II: Summary of notations used in the FlowTM problem.

Input	Definition
$\mathcal{F}$	Set of all flows
$\mathcal{W}$	Set of all switches
$\mathcal{F}_w$	Set of flows that pass through switch w
$\mathcal{W}_f$	Set of all switches on the path of flow f
r_f	Traffic rate of flow f
M_w	Mirroring capacity (bandwidth) of switch w
T_w	Number of available flow entries in switch w for mirroring
$\mathcal{I}_w$	The set of mirrored flows on switch w
Variable	**Definition**
$x_{f,w}$	Decision of mirroring flow f on switch w

$\mathbb{E}\left[\sum_{1\le i\le k} X_i\right] \le 1$. *For any* $N > 2$ *and* $b \ge \frac{3\log N}{\log\log N}$, *the following inequality holds,*

$$\mathbb{P}\left\{\sum_{1\le i\le k} X_i \ge b\right\} < 1/N\,. \tag{4}$$

We need to show $\mathbb{P}\{\hat{\lambda} \ge \gamma_\delta \lambda\} \le \delta$. To this end, define random variable Ω_p for each port p as,

$$\Omega_p = \begin{cases} \frac{r_p}{\widetilde{\lambda}}, & \text{if port } p \text{ is mirrored} \\ 0, & \text{otherwise}\,. \end{cases}$$

The expectation of the sum of random variables Ω_p over all ports of a given switch w is given by,

$$\begin{aligned}\mathbb{E}\left[\sum_{p\in w}\Omega_p\right] &= \sum_{p\in w}\mathbb{E}\left[\Omega_p\right] = \sum_{p\in w}\frac{r_p}{\widetilde{\lambda}}\widetilde{x}_p \\ &= \frac{1}{\widetilde{\lambda}}\sum_{p\in w} r_p\widetilde{x}_p \le 1\,.\end{aligned} \tag{5}$$

Using lemma 1, and given that $\Omega_p \in [0,1]$, we set N to be $W/\delta > 2$. As such, we obtain that,

$$\mathbb{P}\left\{\sum_{p\in w}\Omega_p \ge \gamma_\delta\right\} < \delta/W\,. \tag{6}$$

Recall that $\hat{x}_p$ denotes the integral solution obtained by rounding $\widetilde{x}_p$. Therefore, we have,

$$\sum_{p\in w}\Omega_p = \sum_{p\in w}\frac{r_p}{\widetilde{\lambda}}\hat{x}_p\,. \tag{7}$$

By substituting (7) in (6), the following relation is obtained,

$$\mathbb{P}\left\{\sum_{p\in w} r_p\hat{x}_p \ge \widetilde{\lambda}\gamma_\delta\right\} < \delta/W\,. \tag{8}$$

Since $\widetilde{\lambda} \le \lambda$, it follows that

$$\mathbb{P}\left\{\sum_{p\in w} r_p\hat{x}_p \ge \lambda\gamma_\delta\right\} \le \mathbb{P}\left\{\sum_{p\in w} r_p\hat{x}_p \ge \widetilde{\lambda}\gamma_\delta\right\} < \delta/W\,. \tag{9}$$

Based on the definition of the PortTM problem, we have the relation $\hat{\lambda} = \max_{w\in\mathcal{W}}\sum_{p\in w} r_p\hat{x}_p$, which yields,

$$\begin{aligned}\mathbb{P}\left\{\hat{\lambda} \ge \gamma_\delta\lambda\right\} &= \mathbb{P}\left\{\max_{w\in\mathcal{W}}\sum_{p\in w} r_p\hat{x}_p \ge \gamma_\delta\lambda\right\} \\ &\le \sum_{w\in\mathcal{W}}\mathbb{P}\left\{\sum_{p\in w} r_p\hat{x}_p \ge \gamma_\delta\lambda\right\} \le \delta,\end{aligned} \tag{10}$$

where, the inequality follows from the *union bound.* □

V. Flow-Based Traffic Mirroring

In this section, we formally define the flow-based traffic mirroring (FlowTM) problem and show that the problem is NP-hard. An instance of this problem without port and TCAM capacity constraints is studied in [6]. To efficiently solve the problem, we focus on designing a polynomial-time approximation algorithm. We emphasize that, compared to PortTM, FlowTM is a fundamentally different problem. For example, PortTM is not concerned with the available TCAM capacity, *i.e.*, number of available TCAM entries for installing flow rules, in switches while for FlowTM this is an essential constraint.

Problem 2: Flow-Based Traffic Mirroring (FlowTM)

$$\begin{aligned} \textbf{FlowTM}: \quad & \max \sum_w \sum_f x_{f,w} & & \text{(11a)} \\ \text{s.t.} \quad & \textstyle\sum_{f\in\mathcal{F}_w} x_{f,w} r_f \le M_w, & \forall w \in \mathcal{W} \quad & \text{(11b)} \\ & \textstyle\sum_{f\in\mathcal{F}_w} x_{f,w} \le T_w, & \forall w \in \mathcal{W} \quad & \text{(11c)} \\ & \textstyle\sum_{w\in\mathcal{W}_f} x_{f,w} \le 1, & \forall f \in \mathcal{F} \quad & \text{(11d)} \\ & x_{f,w} \in \{0,1\} & \forall f \in \mathcal{F}, w \in \mathcal{W} \quad & \text{(11e)} \end{aligned}$$

Definitions. The set of flows to be mirrored is denoted by $\mathcal{F}$. A flow is defined using any combination of packet header fields consistent with OpenFlow rules. The set of switches that are used for mirroring is denoted by $\mathcal{W}$. Each switch $w \in \mathcal{W}$ has a mirroring port capacity M_w and an available TCAM capacity T_w. We define $\mathcal{W}_f$ to be the set of switches on the path of flow f. Similarly, we define $\mathcal{F}_w$ to be the set of flows that pass through switch w. Additionally, the traffic rate of flow f is denoted by r_f. The value of r_f can be estimated using various methods from the past flow observations. For example, the works [61]–[65] employ statistical models, such as autoregressive moving average, to estimate the traffic rates. Other works, *e.g.*, [66]–[68], employ machine learning techniques to estimate flow rates using flow-level features, for example, byte and packet counters, protocol, IP addresses, port numbers, and flow durations. The orchestrator can periodically poll the OpenFlow switches, *e.g.*, through OpenFlow `OFPFlowStatsRequest` [8], to extract flow-level features. To account for estimation errors, one approach is to allocate a headroom in each mirroring port to absorb the prediction errors. Table II summarizes the notations used in the FlowTM problem.

A. Problem Definition

Let $x_{f,w}$ denote a binary decision variable that indicates whether flow f is mirrored on switch w or not. The objective is to maximize the number of mirrored flows in the network. Problem 2 presents the formulation of FlowTM as an ILP. Constraint (11b) ensures that, on each switch, the mirroring load is less than the mirroring port capacity. Knowing that for each mirror, a flow rule needs to be installed on the switch, constraint (11c) ensures that the number of mirroring rules does not exceed the TCAM capacity on the switch. Constraint (11d) ensures that each flow is not mirrored on more than one switch along its path.

Theorem 4. *FlowTM problem is NP-hard.*

Proof. Please refer to Appendix B for the proof. □

B. Algorithm Design

This section presents the design of an efficient approximation algorithm for solving the FlowTM problem. We also present an extension of this algorithm to account for flow priorities in a mirroring query. The extended algorithm provides a mechanism to prioritize mirroring the flows with a high priority

Alg. 2: Max Mirrored Flows (MMF)

```
procedure MMF ($\mathcal{F}$, $\mathcal{W}$)
    $\mathcal{S} \leftarrow$ sort-desc($\mathcal{F}, \{r_f | f \in \mathcal{F}\}$)
    $\mathcal{I}_w \leftarrow \emptyset$                /* Set of mirrored flows on $w$ */
    $\lambda_w \leftarrow 0$        /* Total rate of all mirrored flows on $w$ */
    $\{\hat{x}_{f,w}\} \leftarrow \{0\}$
    foreach $w \in \mathcal{W}$ do
        if $\mathcal{S} \neq \emptyset$ then
            for $f \in \mathcal{S} \cap \mathcal{F}_w$ do
                if $r_f + \lambda_w \leq M_w$ and $I_w < T_w$ then
                    $\mathcal{I}_w \leftarrow \mathcal{I}_w \cup \{f\}$
                    $x_{f,w} \leftarrow 1$
                    $\lambda_w \leftarrow \lambda_w + r_f$
                    $\mathcal{S} \leftarrow \mathcal{S} - \mathcal{I}_w$
        else
            break
    return $\{x_{f,w}\}$
```

coefficient. The objective of the base FlowTM is to maximize the number of mirrored flows. As such, flows that have lower rates are more likely to be selected for mirroring. This may lead to an undesired behaviour where high-rate flows are not mirrored by OVT. The inclusion of flow priorities into the problem alleviates this situation.

Algorithm. The proposed algorithm, named max mirrored flows (MMF), is presented in Algorithm 2. The algorithm starts by sorting the flows inversely proportional to their transmission rates in descending order. In line 1, the sorted flows are stored in set $\mathcal{S}$. Then, the set of mirrored flows in the switches is determined iteratively. We define $\mathcal{I}_w$ and λ_w to be the set of mirrored flows and the mirroring load on switch w, respectively (see lines 2-3). For each switch, the set of mirrored flows is examined in the order of the flow's appearance in $\mathcal{S}$. Flow $f \in \mathcal{F}_w$ is selected to be mirrored on switch $w \in \mathcal{W}$, if the capacity of the mirroring port is respected and there is available TCAM capacity on the switch (see line 8). If a flow satisfies these two criteria, it is removed from the set $\mathcal{S}$ and is added to the set $\mathcal{I}_w$. Also, the current mirroring load of the switch and the corresponding decision variable $x_{f,w}$ are updated accordingly (see lines 9-12).

Theorem 5. *MMF runs in $\mathcal{O}(W\overline{F}_w + F\ln(F))$.*

Proof. The algorithm starts by sorting the flows, which takes $\mathcal{O}(F \ln F)$. MMF then iterates through all switches, and for each switch creates the set of not-yet-mirrored flows from the set of flows that traverse a switch. The set of flows that traverses each switch can be found in $\mathcal{O}(1)$. In the worst case, for any switch, all flows traversing the switch can be mirrored on it. As such, the inner loop (line 7) is executed for at most $\overline{F}_w$ times. All updates (line 9-line 12) can be run in $\mathcal{O}(1)$. Therefore, the cost of selecting the mirrored flows for all switches is $\mathcal{O}(W\overline{F}_w + F\ln(F))$. ☐

Theorem 6. *MMF attains the approximation ratio of* 2.

Proof. Let $\mathcal{I}_w$ be the set of flows that are mirrored on switch w using the MMF algorithm. Let $\hat{\mathcal{I}}^\star_w$ denote the set of flows that are mirrored on switch w by the optimal algorithm, but are not mirrored by the MMF algorithm at all (*i.e.*, $\hat{\mathcal{I}}^\star_w \cap \bigcup_{w\in\mathcal{W}} \mathcal{I}_w = \emptyset$). Recall that $\hat{I}^\star_w$ and I_w show the set cardinalities of $\hat{\mathcal{I}}^\star_w$ and $\mathcal{I}_w$, respectively. Note that the flows in $\hat{\mathcal{I}}^\star_w$ were available to MMF at the time of specifying the mirrored flows on switch w but were not selected. Lemma 2 shows that MMF is optimal for the single-switch sub-problem. As such, we can conclude that $I_w \geq \hat{I}^\star_w$. Consequently, it is possible to obtain the following lower-bound for the total number of mirrored flows in the network using the MMF algorithm:

$$\sum_{w\in\mathcal{W}} I_w \geq \sum_{w\in\mathcal{W}} \hat{I}^\star_w. \tag{12}$$

Let OPT be the optimal objective value of the FlowTM problem obtained by the optimal algorithm. We have: $\sum_{w\in\mathcal{W}} \hat{I}^\star_w \leq$ OPT. Now, observe that if $\sum_{w\in\mathcal{W}} \hat{I}^\star_w \geq \text{OPT}/2$, then MMF attains the approximation ratio of 2. On the other hand, if $\sum_{w\in\mathcal{W}} \hat{I}^\star_w < \text{OPT}/2$, by the definition of $\hat{\mathcal{I}}^\star_w$'s, MMF has left out less than half of the flows that are mirrored by the optimal algorithm and thus it attains the approximation ratio of 2. The combination of these two possibilities proves the approximation ratio of 2. ☐

Lemma 2. *MMF is optimal for the single-switch sub-problem.*

Proof. Let $f_1, \ldots, f_m$ with $r_{f_1} \leq \cdots \leq r_{f_m}$ represent the flows mirrored by the MMF algorithm on a single switch. Similarly, let $f'_1, \ldots, f'_n$ with $r_{f'_1} \leq \cdots \leq r_{f'_n}$ represent the flows mirrored by the optimal algorithm. If $m = n$ (*i.e.*, the MMF and the optimal algorihm mirror the same number of flows) the theorem is proved. We show that $m < n$ is impossible. For the sake of contradiction assume that $m < n$. Since MMF selects flows in the order of increasing rates, we have:

$$\sum_{i=1}^{m} r_{f_i} \leq \sum_{i=1}^{m} r_{f'_i}. \tag{13}$$

Consider flow f'_{m+1} which is not selected for mirroring by the MMF algorithm. Based on the constraints in the problem (2), we know that MMF stops mirroring flows either because of the available number of flow rule entries or the mirroring port capacity violation. If the procedure is terminated because of the available number of flow rule entries, the algorithm has already found the optimal solution (*i.e.*, recall that all flows regardless of their rate require a single flow rule). Therefore, we can conclude that the reason for not selecting f'_{m+1} is the mirroring port capacity constraint, *i.e.*,:

$$r_{f'_{m+1}} + \sum_{i=1}^{m} r_{f_i} > M_w. \tag{14}$$

Consequently,

$$\sum_{i=1}^{n} r_{f'_i} = \sum_{i=m+1}^{n} r_{f'_i} + \sum_{i=1}^{m} r_{f'_i} \tag{15}$$

$$\geq r_{f'_{m+1}} + \sum_{i=1}^{m} r_{f_i} > M_w, \tag{16}$$

which contradicts the fact that $f'_1, \ldots, f'_n$ are the feasible optimal solution of the problem. ☐

Extension of FlowTM with flow priorities. FlowTM assumes all flows have the same priority in a mirroring query. However, there may be scenarios where flows in a mirroring query have different priorities. To this extent, OVT can be extended to prioritize flows to ensure that the limited mirroring capacity of switches is better utilized. Flow priorities can be assigned at different levels and based on various factors: 1) The orchestrator prioritizes the flows based on the source and destination VN subscription model. 2) The agent prioritizes the flows based on the significance of the analyzer application. For example, the set of flows specified by a security analyzer may be more critical than that of a performance analyzer. 3) Each analyzer can prioritize different flows in its requested monitoring queries based on the impact each flow would have on the application's performance and accuracy. For example, a security analyzer prioritizes flows that target critical systems in the network, *e.g.*, accounting databases. Consequently, it is possible to obtain a task-dependent priority assignment scheme to maximize the utility of analyzer applications. To account for flow priorities, when deciding on the flow mirroring configurations, the FlowTM problem, used in the optimizer module, should also be extended. To this end, for each flow f, we add a coefficient $\alpha_f \in [0, 1]$, representing the flow's priority in a mirroring query. In this case, since the objective is not maximizing the number of mirrored flows anymore, using all TCAM capacity, *i.e.*, available TCAM entries, does not imply the solution's optimality. Additionally, we cannot simply mirror the flows by sorting them in descending order of $\frac{\alpha_f}{r_f}$ (*i.e.*, priority by rate ratio). To illustrate this, consider the following example of a single-switch sub-problem. Assume a switch with 3 Mbps mirroring port capacity, 2 available TCAM entries and two traversing flows, *i.e.*, f_1 ($r_{f_1} = 1$ Mbps and $\alpha_{f_1} = 2$) and f_2 ($r_{f_2} = 3$ Mbps and $\alpha_{f_2} = 3$). When choosing the flows based on the decreasing order of the ratio of priority to rate, f_1 is chosen to be mirrored on the switch. However, the optimal solution is to mirror f_2. The example shows that by simply using the priority to rate ratio, we do not efficiently utilize the mirroring capacity of the switch.

To account for different flow priorities while efficiently utilizing the mirroring capacity, we modify the MMF algorithm as follows. Similar to the base MMF, the mirrored flow set on the switches is iteratively determined. In each iteration, we consider each switch as a 2-dimensional knapsack, *i.e.*, a knapsack with 2 constraints, one representing the mirroring port capacity and the other representing the TCAM capacity. We then use the approach presented in [69] to select the set of mirrored flows on the switch, such that the value of the mirrored flows is maximized and the constraints (11b) and (11c) are satisfied. Following this, the current mirroring load of the switch and the corresponding decision variable are updated accordingly. Additionally, the set of mirrored flows on the switch is removed from the set of not-yet-mirrored flows. This procedure continues until all flows are mirrored or all switches are assigned a set of mirrored flows based on their mirroring capacity.

Theorem 7. *Extended MMF attains the approx. ratio of* 6.

Proof. The approach presented in [69] for the single-switch sub-problem attains a 3-approximation ratio. This results in $\sum_{f \in \mathcal{I}_w} \alpha_f \geq \sum_{f \in \hat{\mathcal{I}}^\star_w} \alpha_f / 3$. The equation (12) then is modified as follows:

$$\sum_{w \in \mathcal{W}} \sum_{f \in \mathcal{I}_w} \alpha_f \geq \frac{\sum_{w \in \mathcal{W}} \sum_{f \in \hat{\mathcal{I}}^\star_w} \alpha_f}{3}. \tag{17}$$

Then the theorem is established following the arguments presented in the proof of Theorem 6. □

Discussion. Although it is possible to solve the base FlowTM problem using the approach presented in [69], the MMF algorithm attains a better approximation ratio (*i.e.*, 2 compared to 6) and has a lower computational complexity as it does not rely on solving an LP. These are important considerations that substantially affect the performance of OVT Optimizer.

The current design of OVT assumes that a flow can be mirrored on any switch along its path. However, there could be scenarios where a flow is required to be mirrored in a specific zone in the network. For example, a flow is only required to be mirrored after the firewall. In such scenarios, as opposed to the original PortTM and FlowTM problems, only a subset of ports/switches along the flow's path, belonging to the flow's monitoring zone, can be used for mirroring. To this extent, constraints (1b) and (11d) can be modified to account for the monitoring zone of flow f in the PortTM and FlowTM problems, respectively. Note that both ϵ-LPR and MMF algorithms and their analyses apply to this case.

VI. Evaluations

In this section we evaluate the performance of OVT using simulations and Mininet experiments. In particular, we simulate our system and analyze it through the lens of system performance and micro benchmarks. Furthermore, we use Mininet to study the applicability of our system on real-worl applications through application performance.

Methodology. To study the performance of OVT, we present three sets of evaluation results:

1) *System Performance:* Using simulations, we present the benefits of VN awareness in OVT compared to independent VN traffic mirroring approaches in subsection VI-A.
2) *Micro Benchmarks:* We perform an extensive number of simulation-based micro benchmarks to assess the performance and scalability of the algorithms used in OVT optimizer, *i.e.*, ϵ-LPR and MMF. Particularly, in subsection VI-B, we present the comparison results with alternative mirroring algorithms such as those proposed in [6]. The comparisons are done in terms of performance metrics such as mirroring flow coverage, switch load and utilization on varying network sizes.
3) *Application Performance:* We demonstrate the real-world applicability of OVT by implementing it in the Mininet environment in subsection VI-C. We show the advantages of OVT when used in conjunction with two practical applications, namely an intrusion detection system (IDS) and a video telephony analyzer (VTA). We investigate the effects of utilizing port and flow mirroring algorithms on the performance of these applications.

A. System Performance

Setup. System performance analysis is conducted using simulations, implemented in Python 3.6. As our substrate physical network, we use the USA topology (24 switches and 43 links) from the topology Zoo dataset [70]. We set the mirroring port capacity of all switches to 1 Gbps. Even though 10 Gbps network links are common, a lower value is selected to reduce the experiments' run time. Additionally, the TCAM capacity of each switch is set to 136 rules [71]. While newer OpenFlow switches have higher TCAM capacity, the chosen value is sufficient for the purpose of this study. Furthermore, we consider running two and three VNs on top of the substrate network. VNs share all physical switches, *i.e.*, each VN has 24 virtual switches simulating the USA topology. On each VN, we choose uniformly at random the source and destination of each flow. The flow rates are generated based on the results presented in [72]. In particular, 10% of the generated flows transmit at less than 4 Mbps. 60% of the flows transmit at a rate in $[4, 9]$ Mbps and the remaining 30% transmit at higher than 9 Mbps. The size of the mirroring queries is the same for all VNs. Finally, all computations are carried out on a computer with an Intel® CoreTM i5-7360U processor at 2.3 GHz and 8 GB of RAM. The reported results are averaged over 10 runs.

Baseline. In these experiments, OVT is compared with a baseline mirroring system, in which each VN computes its mirroring configuration independently from other VNs. We refer to the baseline mirroring system as IND. In the experiments, both OVT and IND use the same mirroring algorithms, namely ϵ-LPR ($K_\epsilon = 10$) and MMF. Thus, any differences between their performances are solely due to joint optimization of mirroring configurations in OVT.

Performance Metric. These experiments are focused on *Flow Coverage* metric defined as the percentage of flows that are fully mirrored by at least one switch in the network. A higher flow coverage means that more flows are mirrored and as a result the mirroring capacity is better utilized.

Results. Fig. 4 and Fig. 5 compare the flow coverage of OVT and IND, when using port and flow mirroring algorithms, respectively. From Fig. 4 we can observe that OVT has higher flow coverage compared to IND. Specifically, as the query size grows, the flow coverage of OVT decreases with a lower slope compared to that of IND. For example, for the query size of 5000 flows, OVT achieves 10% higher coverage compared to IND with 2 VNs. Interestingly, IND with 2 and 3 VNs achieve a similar flow coverage for all query sizes. The reason is that when VNs decide on the port configurations independently, most ports are mirrored to cover the flows. This results in similar behaviour of IND with 2 and 3 VNs. From Fig. 5 we can observe the following points. First, OVT achieves a higher flow coverage compared to IND. Specifically, using flow mirroring, for a query size of 3000 flows OVT achieves more than 20% higher flow coverage compared to IND. Note that the lower difference between OVT's and the IND's flow coverage, when using port mirroring as opposed to flow mirroring algorithms, is due to the latter's finer granularity and control scheme. Second, we can see that flow coverage for IND with 2 and 3 VNs diverges for larger query sizes. This behaviour is expected since, as the number of VNs sharing the same physical switches increases, independent mirroring of the flows results in more inefficient use of mirroring resources on switches.

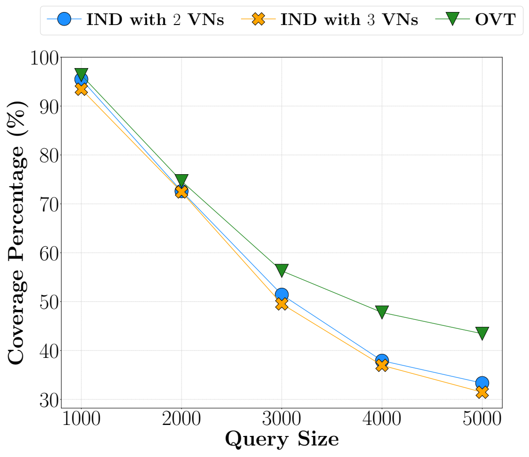


Fig. 4: Performance of OVT and IND when using port mirroring.

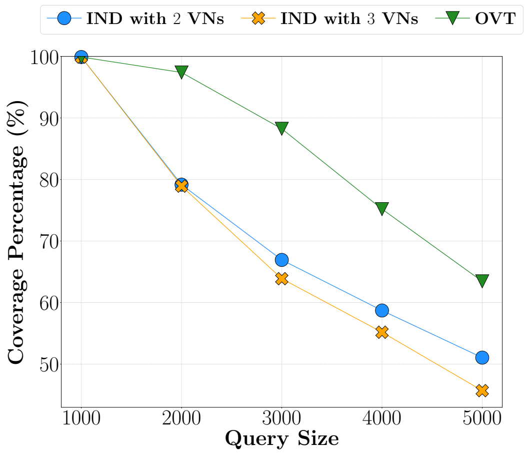


Fig. 5: Performance of OVT and IND when using flow mirroring.

B. Micro Benchmarks

Setup. Micro benchmarks are conducted using simulations, implemented in Python 3.6. Similar to the system performance experiments, in the ϵ-LPR algorithm, we set $K_\epsilon = 10$. All computations are carried out on a computer with an Intel® CoreTM i5-7360U processor at 2.3 GHz and 8 GB of RAM. The reported results are averaged over 10 runs.

In this set of experiments we use the following network topologies:

1) *Small-scale ISP (USA)* topology chosen from the topology Zoo dataset [70] with 24 switches, 24 hosts and 43 links. Note that for the rest of the paper, we refer to this topology as the USA topology.
2) *Large-scale ISP* topology, presented in [73] and accessible through [74], with 315 switches, 315 hosts and 1944 links. Note that for the rest of the paper, we refer to this topology as ISP topology.
3) 12-pod *FatTree* topology with 36 core switches, 72 aggregate switches, 72 edge switches and 72 servers. Note that a FatTree topology of this size is larger or equal to the ones commonly used in the literature [75], [76].

The setup of bandwidth and TCAM capacity is similar to the system performance experiments, presented in subsection VI-A.

Implemented Algorithms. To compare ϵ-LPR and MMF with existing mirroring algorithms, we have also implemented the following algorithms:

- **OPT**: The optimal solution of ILPs obtained by the Gurobi optimizer [77]. Note, we can calculate the optimal solution only for small instances of the problems, *e.g.*, small networks with a few switches.
- **PMA**: An alternative algorithm to ϵ-LPR proposed in [6]. PMA determines port mirroring configuration of switches based on their contributions to the objective, *i.e.*, the maximum switch load.
- **FMA**: An alternative algorithm to MMF proposed in [6]. FMA determines the flow mirroring configuration of switches without considering the mirroring port capacity or available TCAM capacity.
- **Planck**: An alternative algorithm for solving the PortTM problem, presented in [4]. Planck mirrors all ports on all switches to a mirroring port.
- **Stroboscope**: An alternative algorithm to MMF presented in [35]. Stroboscope mirrors a flow on a minimum number of switches along its path to confirm the flow's path compliance.

Traffic Generation. To generate a flow, we at random choose a source and a destination and, between them, establish constant-rate traffic. The traffic rate depends on the network topology under the experiment. Specifically, we classify the USA and ISP topologies as ISP environments and use the results of [72] to generate flow rates. In particular, 10% of the generated flows transmit at less than 4 Mbps, 60% of the flows transmit at a rate in $[4, 9]$ Mbps and the remaining 30% transmit at higher than 9 Mbps. We classify the FatTree topology as a datacenter environment and employ two widely used datacenter workloads to generate flow rates. The first workload is based on web server traffic [56], [78], [79] and the second workload is based on a cache server [56], [78].

Performance Metrics. We compute and report the following performance metrics:

- *Flow Coverage*: The percentage of flows that are fully mirrored by at least one switch in the network. A higher flow coverage means that more flows are fully mirrored in the network and as a results the mirroring capacity of the network is utilized more efficiently.
- *Maximum Switch Load*: The mirroring load on a switch with the highest mirroring load in the network. The lower the maximum switch load, the more balanced the mirroring load distribution in the network.
- *Mirroring Cost*: The total mirroring load in the network divided by the number of fully mirrored flows. A higher mirroring cost means that more flows are being mirrored per fully covered flow. As such, a higher mirroring cost suggests either lower flow coverage or higher redundant mirroring.
- *Mirroring Utilization:* The mirroring load of the unique mirrored flows divided by the total mirroring capacity of the network. Higher mirroring utilization means that more unique flows are being mirrored in the network.
- *Runtime*: The amount of time the algorithm takes to compute the mirroring configuration of all switches in the network.

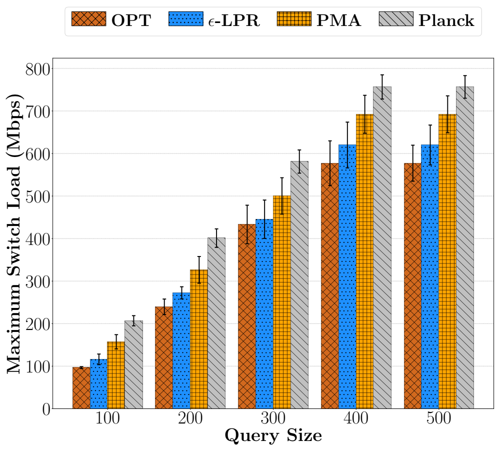


Fig. 6: Comparison of the port mirroring algorithms with the optimal solution on the USA topology.

The performance metrics are reported for different *query sizes*. The query size refers to the number of flows that are submitted by all OVT Agents to the orchestrator for mirroring.

Comparison with Optimal. We compare the performance of port and flow mirroring algorithms with optimal solutions, in Fig. 6 and Fig. 7, respectively. The optimal solutions are obtained by solving the corresponding ILPs. Specifically, Fig. 6 demonstrates the comparison between different port mirroring algorithms and the optimal solution, in terms of the maximum switch load, on the USA topology. Note that the lower the maximum switch load is, the more evenly the mirroring load is distributed among the switches, and, as such more flows can be mirrored in the network. We can see that the maximum switch load achieved under ϵ-LPR is, on average, 9% higher than the optimal's. This close performance between the ϵ-LPR and the optimal algorithms suggests that the real-world approximation ratio of ϵ-LPR is substantially better than the one derived in our analysis. We can also see that ϵ-LPR achieves 15% and 27% lower mirroring load on the most loaded switch compared to PMA and Planck, respectively.

Different from the PortTM problem, the objective of the FlowTM problem is defined as maximizing flow coverage in the network. As such, Fig. 7 shows the performance of different flow mirroring algorithms on the USA topology. The following observations are in order. First, MMF on average has 2.6% less flow coverage than the optimal, which is better than the calculated theoretical ratio (*i.e.*, approximation ratio of 2). Second, even for small query sizes, MMF has a better flow coverage compared to both FMA and Stroboscope. Particularly, for the query size of 3000 flows, the maximum flow coverage difference between the ϵ-LPR and the FMA algorithms is 11.4% (*i.e.*, 340 flows). Finally, Stroboscope has the worst performance, missing more than half of the flows for query sizes over 3000 flows.

Coverage Analysis. Fig. 8(a) demonstrates the differences between port and flow mirroring algorithms on the USA topology. Specifically, Fig. 8(a) illustrates the flow coverage differences, while Fig. 8(b) shows the mirroring cost differences, between different port and flow mirroring algorithms. From Fig. 8(a) we can observe the following points. First,

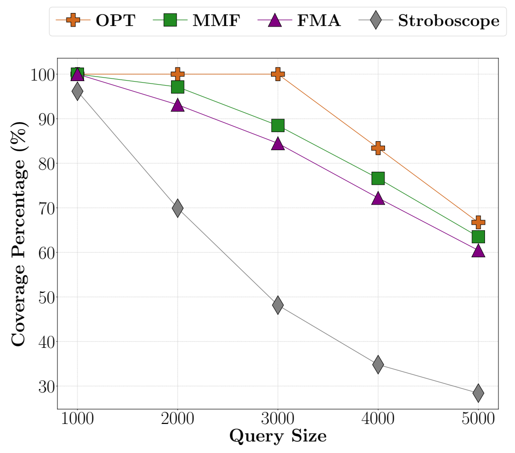
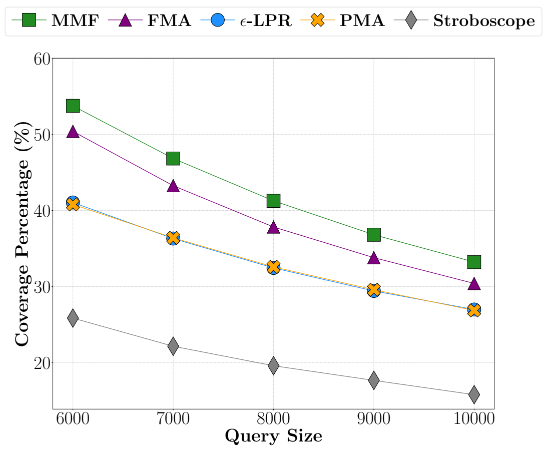
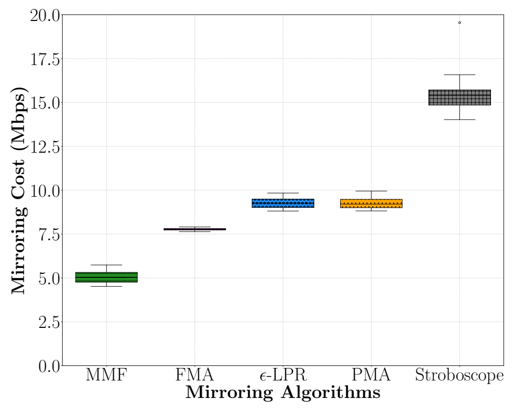


Fig. 7: Comparison of the flow mirroring algorithms with the optimal solution on the USA topology.



(a) Flow coverage analysis.



(b) Mirroring cost analysis.

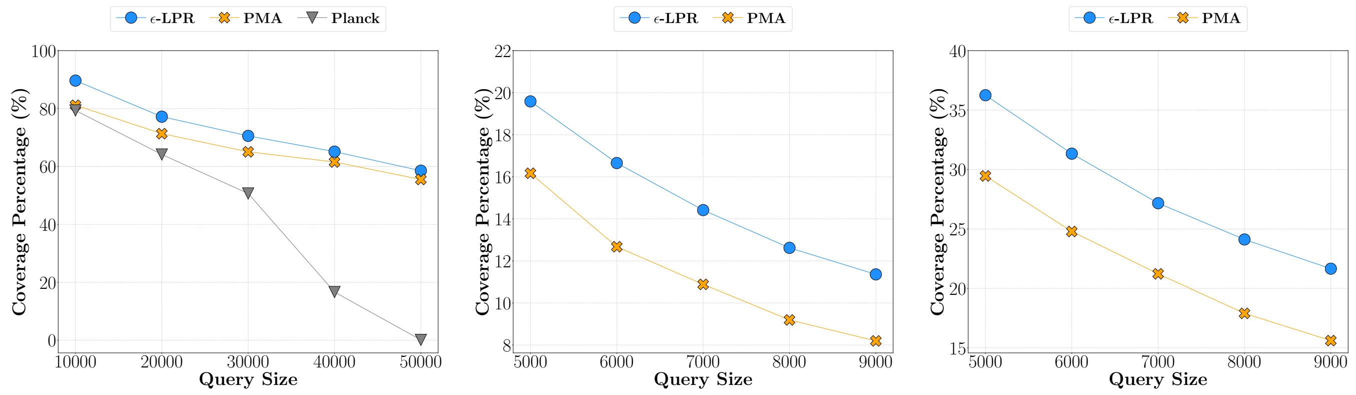
Fig. 8: Coverage analysis of different mirroring algorithms on the USA topology.

for all query sizes, MMF by at least 7.7% outperforms all other mirroring algorithms. When comparing MMF with FMA, MMF achieves a higher flow coverage since it calculates the mirroring configurations accounting for mirroring port and TCAM capacities. Additionally, on average, MMF obtains 20% higher flow coverage compared to ϵ-LPR and PMA. Such results are expected since ϵ-LPR and PMA mirror flows at the port granularity, resulting in mirroring redundancy. Second, ϵ-LPR and PMA present a similar trend. The reason is that both algorithms, for query sizes of 6000 flows, mirror most of the ports on most of the switches in the network. Additionally, the mirroring capacity of the switches is limited, and since the network is saturated, both algorithms will have similar mirroring losses and show similar behaviour. Finally, although Stroboscope is a flow mirroring algorithm, it has worse coverage compared to port mirroring algorithms, *i.e.*, ϵ-LPR and PMA. This low coverage is due to Stroboscope mirroring the same flow on multiple switches on its path.

Fig. 8(b) shows the mirroring cost distribution of different algorithms on the USA topology for query sizes of 6000 to 10000 flows. The following observations are in order. First, we can see that MMF, compared to PMA, achieves a lower mirroring cost. The reason is that, in addition to higher flow coverage, MMF also tends to mirror smaller flows. Second, FMA has a tighter mirroring cost distribution compared to MMF. This suggests that for varying query sizes, FMA compared to MMF has more consistent mirroring configurations. Similar to the flow coverage analysis, we observe a similar trend between ϵ-LPR and PMA. Finally, Stroboscope achieves the worst mirroring cost while having the lowest coverage.

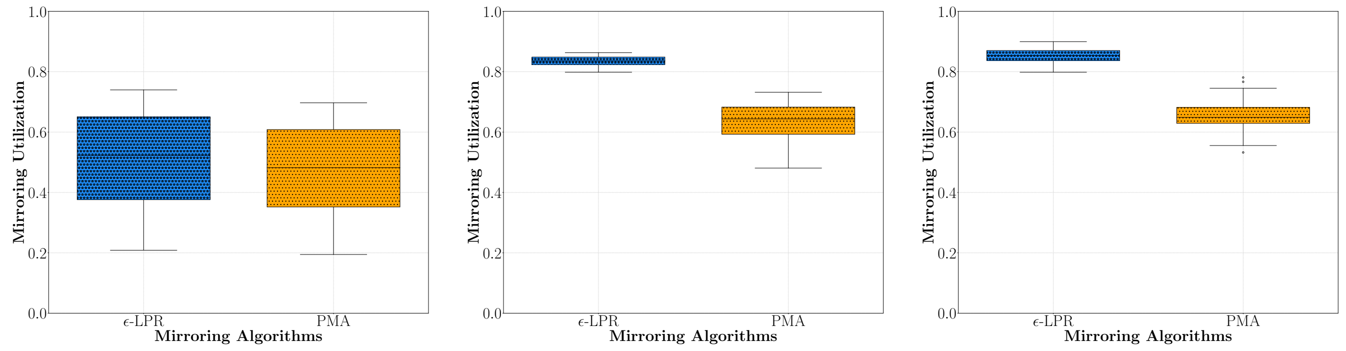
ϵ-LPR and PMA behave similarly on the small-scale USA topology. However, this does not necessarily hold for other topologies. To further investigate, we have compared the performance of port mirroring algorithms on larger topologies. Fig. 9(a) shows the comparison on a large-scale ISP topology, while Fig. 9(b) and Fig. 9(c) show the comparison on the FatTree topology under web and cache workloads, respectively. Note that Planck is omitted from Fig. 9(b) and Fig. 9(c), since it has 0% flow coverage (*i.e.*, it achieves only partial flow coverage). We use larger query sizes for the ISP topology. The reason is that the ISP topology has more switches compared to the FatTree topology, and as such, smaller query sizes can be fully mirrored by all algorithms. From the figures, we can observe that, for all scenarios, ϵ-LPR outperforms PMA. The difference between the two, however, depends on both the topology and the traffic type. In particular, on average, ϵ-LPR achieves 5.3% higher flow coverage compared to PMA on the ISP topology, while having a 3.5% and 6.3% higher flow coverage on the FatTree topology under web and cache application traffic, respectively. As expected, regardless of the topology, increasing the query size results in the flow coverage decrease. The reason is that for large query sizes, most switches are already utilizing their full mirroring capacity. As such, an increase in the query sizes results in mirroring loss.

Mirroring Utilization Analysis. Fig. 10 shows the mirroring utilization comparison between ϵ-LPR and PMA. Specifically, Fig. 10(a) shows the mirroring utilization, for query sizes of 10K to 50K flows, on the ISP topology. Fig. 10(b) and Fig. 10(c), on the other hand, show the mirroring utilization on the FatTree topology for query sizes of 5000 to 9000 flows under web and cache server traffic, respectively. Note that, compared to the ISP topology, the query sizes on the



(a) ISP topology. (b) FatTree topology (web server traffic). (c) FatTree topology (cache server traffic).

Fig. 9: Coverage analysis of port mirroring algorithms on large-scale topologies.



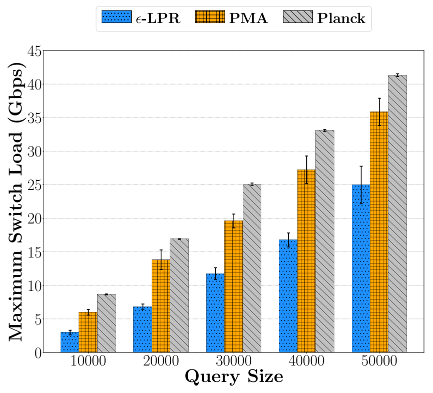
(a) ISP topology. (b) Fattree topology (web server traffic). (c) Fattree topology (cache server traffic).

Fig. 10: Mirroring utilization analysis of port mirroring algorithms on large-scale topologies.

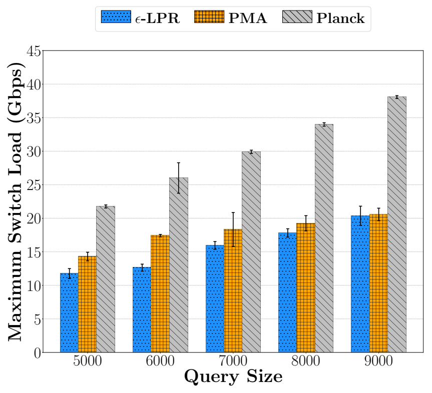
FatTree topology are smaller. The reason is that FatTree topology has fewer switches compared to the ISP topology and as such, larger query sizes result in very low coverage. We can observe that both algorithms have similar mirroring utilization on the ISP topology. However, on the FatTree topology, ϵ-LPR better utilizes the mirroring capacity of the network. This suggests that ϵ-LPR, compared to PMA, mirrors more unique flows or equivalently mirrors less redundant flows in the network. Specifically, ϵ-LPR achieves a minimum of 0.8 mirroring utilization under both loads. Additionally, the mirroring utilization of PMA varies in a wider range of values, resulting in a lower minimum utilization as compared to ϵ-LPR. For example, PMA's minimum mirroring utilization is 0.5 for the web server traffic, while ϵ-LPR has a minimum mirroring utilization of 0.8.

Maximum Switch Load Analysis. We minimize mirroring losses in ϵ-LPR by balancing the mirroring load among different switches. As such, we compare the maximum switch load of different port mirroring algorithms. Fig. 11 shows this comparison between ϵ-LPR, PMA and Planck on the ISP and FatTree topologies. We can observe that in all scenarios, ϵ-LPR outperforms PMA and Planck. Additionally, ϵ-LPR's behaviour shows less dependency on the topology compared to PMA. One can observe this behaviour by comparing the maximum switch load of ϵ-LPR and PMA with Planck's across different topologies. In particular, on the ISP topology, the difference between ϵ-LPR and Planck, in terms of the maximum switch load, is on average 53%. On the FatTree topology (web server traffic) this difference is 48% (*i.e.*, only a 5% difference with the ISP topology results). On the other hand, the maximum switch load difference between PMA and Planck is 20% and 39% on the ISP and FatTree topologies, respectively.

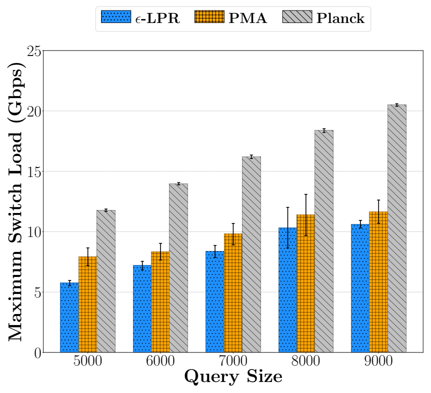
Runtime Analysis. Table III shows the runtime comparison of different mirroring algorithms on the USA topology. We can observe that port mirroring algorithms are faster than flow mirroring algorithms. This is expected since the number of decision variables in the PortTM problem is lower than the number of variables in the FlowTM problem. In particular MMF is an order of magnitude slower than PMA. However, observe that MMF is on average 2 orders of magnitude quicker than FMA, providing a runtime improvement. The reason is that MMF does not rely on solving the LP, speeding up the algorithm. The runtime complexity of both ϵ-LPR and MMF grow with the query size (see Theorem 2 and Theorem 5). In fact, the results presented in Table III further confirm this dependency. Please note that increasing the network's size does not necessarily increase the runtimes of the algorithms, as their runtimes depend on the set of flows to be mirrored and their corresponding path, *i.e.*, the number of ports/switches the flows traverse. For example, the work [80] reports that the average path length for four different datacenter network topologies (*i.e.*, Multi-tiered, FatTree, BCube, Flattened Butterfly), consisting of thousands



(a) ISP topology.



(b) Fattree topology (web server traffic).



(c) Fattree topology (cache server traffic).

Fig. 11: Maximum switch load analysis of port mirroring algorithms on large-scale topologies.

TABLE III: Runtime (in sec) on the USA topology.

Mirroring Algorithm		Query Size				
		6000	7000	8000	9000	10000
Flow Mirroring	MMF	3.4	3.4	3.5	3.5	4.3
	FMA	321.9	321.9	341.8	411.7	411.7
Port Mirroring	ϵ-LPR	0.54	0.54	0.55	0.59	0.61
	PMA	0.33	0.32	0.32	0.33	0.36

of servers, is less than 6.

C. Application Performance

To study OVT performance in realistic network settings, we built an IDS and a VTA application on top of OVT and studied their performance in Mininet. The IDS uses OVT to monitor network traffic for detecting the "Zorro" malware [41]. To this end, IDS performs deep packet inspection (DPI) to analyze the packet payloads and classifies them as contaminated by Zorro malware (*i.e.*, containing "Zorro" string) or benign. VTA monitors the Zoom traffic [81] to infer the quality of experience and adjust the streaming parameters based on that.

Mininet Setup. We use a 2-pod *FatTree* topology, consisting of 8 hosts and 10 OpenFlow 1.3 switches. To decrease the runtime of the experiments, we configure the capacity of all mirroring ports to be 10 Mbps. Additionally, we set the size of the forwarding tables on switches, using the default size of Open vSwitch forwarding tables (*i.e.*, 100 rules). We connect all switches to a host to collect mirrored flows and configure ONOS [82] as our SDN controller.

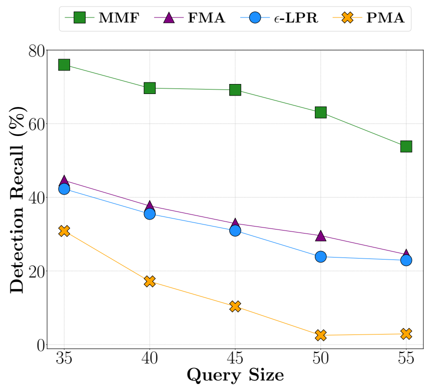
ONOS uses OpenFlow protocol to communicate with OpenFlow switches and maintains a state graph of the switches using the OpenFlow Discovery Protocol (OFDP). Furthermore, it exposes a northbound API to the OpenFlow applications, in this case, OVT Agent. We use ONOS proactive forwarding to install routing flow rules. The reason is that, as packets are transmitted, ONOS reactive forwarding module installs rules on demand, which may result in varying routes for the same flow in between different runs [83].

To compute mirroring configurations, we have implemented ϵ-LPR and MMF algorithms in Python 3.6. Moreover, for the sake of comparison, we have also implemented PMA and FMA algorithms presented in [6]. Once mirroring configurations are computed using one of the above algorithms, we configure each switch, accordingly, by installing OpenFlow rules. Note, at the high level, an OpenFlow rule is composed of match and action sets. The former identifies a set of flows, and the latter defines the actions to be applied to the set members, in our case, output ports of the flows. In our experiments, we increase query sizes from 35 to 55 with a step of 5. Finally, the reported results are the average of 5 runs.

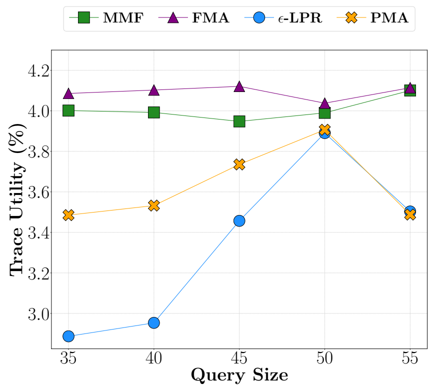
IDS Setup. In this experiment, we generate Telnet traffic, on port 23, with traces of the "Zorro" attack. We generate the traffic with flow rates from a datacenter web server traffic distribution, available in [56]. However, the reported distribution is based on measurements in a datacenter with link capacities of 10 Gbps. For the sake of Mininet experiments runtime, we scale down the rates by a factor of 1000. We use 1500 byte packet sizes since it is the maximum transmission unit (MTU) for the Ethernet. We contaminate $[4-5]\%$ of the packets with the "Zorro" string to represent malicious behaviour. This percentage is slightly higher than what is assumed in the literature (*i.e.*, 3% of the internet traffic [84]), to increase the probability of observing losses of the Zorro packets. We report the following application performance metrics:

- *Detection Recall*: Number of mirrored Zorro packets divided by the total number of transmitted Zorro packets. A higher detection recalls means that more contaminated packets are detected by the IDS application.
- *Trace Utility*: Number of mirrored Zorro packets divided by the total number of mirrored packets. A higher trace utility means that the mirroring capacity of switches are better utilized and more of the packets of interest are collected.

IDS Results. Fig. 12(a) presents the detection recall of the IDS application when using different mirroring algorithms. The following observations are in order. First, increasing the query sizes results in lower detection recall. This is due to increased mirroring loss when introducing more flows into the network. Second, we can observe that flow mirroring algorithms have higher detection recall compared to port mirroring algorithms. This behaviour is expected since port mirroring algorithms mirror more redundant packets, thus, losing the packets of interest, *i.e.*, Zorro packets. Third, MMF has the highest detection recall compared to other algorithms, including FMA, also a flow mirroring algorithm. Specifically, MMF compared to FMA, achieves an average of 32% higher

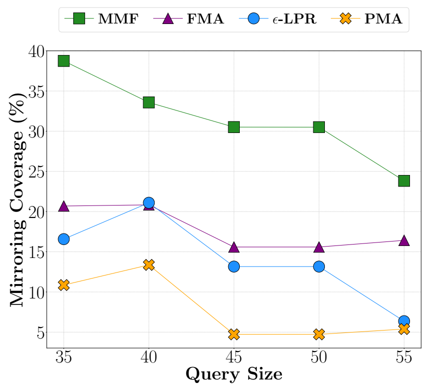


(a) Detection recall.



(b) Trace utility.

Fig. 12: IDS application.



(a) Mirroring coverage.

Fig. 13: VTA application.

detection recall. Interestingly, we can see that FMA behaves similar to ϵ-LPR. The reason is that both algorithms depend on solving the LP. Although these two algorithms solve the problem at different granularities, they behave similarly when the set of flows given to FMA includes most flows traversing a port. This can be observed on topologies with limited number of paths and distinct source-destination pairs. Finally, we can see that ϵ-LPR compared to PMA, shows an 18% detection recall improvement.

Fig. 12(b) shows the trace utility of different mirroring algorithms. Note that since Zorro packets are only included in $[4-5]\%$ of the packets, trace utility is limited to at most 5%. We can see that both flow mirroring algorithms, *i.e.*, MMF and FMA, achieve higher and more stable utility compared to port mirroring algorithms. This behaviour is expected since both MMF and FMA do not mirror redundant packets, resulting in a more stable relation between the total number of mirrored packets and the number of lost Zorro packets, *i.e.*, the number of Zorro packets that are not mirrored. We can also observe that FMA achieves a slightly higher utility than MMF, which is explainable through FMA's lower detection recall and mirroring coverage. In particular, since the number of Zorro packets is substantially lower than that of benign packets, losing one contaminated flow has a higher impact on the total number of mirrored packets compared to the number of mirrored Zorro packets. Additionally, for query sizes below 50 the trace utility has an increasing slope for both port mirroring algorithms. The reason is that only a tiny portion of the total traffic contains the string "Zorro". Therefore, as the query sizes grow, the mirroring losses increase with most of the losses occurring for non-Zorro packets. This increase in trace utility is observable to a certain point (*e.g.*, query size of 50 flows), after which, as can be seen at query size of 55 flows, more losses occur for Zorro packets.

VTA Setup. To represent the video telephony traffic in Mininet, we generate flows based on traffic characteristics of Zoom application. Specifically, we generate UDP traffic on port 8801 and use Zoom bandwidth requirements available at [81] to decide on flow rates. In particular, we assume that flows belong to either a one-to-one or a group video call of 720p or 1080p HD video quality. Using the specified parameters, a flow rate for a Zoom call is between $[1.2-1.8]$ Mbps or $[2-3]$ Mbps, respectively. To decide on the packet sizes, we capture and analyze packets of a 1 minute Zoom call using Wireshark [85]. We find that Zoom calls on average generate UDP packets of 1000 bytes. We report the following application performance metric:

- *Mirroring coverage*: The percentage of the uniquely mirrored packets divided by the total number of transmitted packets. A higher mirroring coverage means that the mirroring resources of the switches are better utilized.

VTA Results. Fig. 13 shows the measurement results of the VTA application. We can observe a decreasing trend in the mirroring coverage. The reason is that as more flows are introduced into the network most mirroring ports become saturated, increasing the mirroring losses. We can also observe that MMF compared to FMA achieves an average of 43% higher mirroring coverage. This higher coverage is due to MMF accounting for mirroring port capacity of switches. Interestingly, we can see that FMA and ϵ-LPR follow a similar trend. As explained earlier, this is due to both algorithms relying on solving the LPs. Although they solve the problem at different granularities, for large query sizes, FMA needs to cover most flows on a port which makes it behave similar to ϵ-LPR. Specifically, when the network is saturated FMA will end up covering most of the flows traversing a port, similar to the ϵ-LPR.

VII. Conclusion

In this paper, we presented OVT, a flow mirroring system for virtual networks that utilizes mirroring capabilities of commodity OpenFlow switches to create a software-defined network tap. We showed that by carefully computing switch mirroring configurations, OVT is able to provide flow coverage functionality similar to hardware taps. To achieve this, we formulated two optimization problems for computing switch mirroring configurations and designed fast and efficient algorithms to solve each problem. In addition to the theoretical results, we also conducted experiments using model-driven simulations as well as realistic Mininet emulations in a variety of networks and application scenarios. The results are compared with existing network traffic mirroring approaches. Extending OVT to mirror flows on multiple switches along their path can facilitate the debugging of network functions that modify/drop packets. Additionally, incorporating programmable switches in OVT results in a more flexible design. In particular, with the advent of programmable switches, OVT can reduce the mirroring load by partially mirroring the traffic destined to analyzer applications that do not require the whole packet, *e.g.*, DDoS detection. The mirroring load can be further reduced using probabilistic methods, a worthwhile extension on switches with constrained resources. Finally, in this work, we used a randomized strategy to round the fractional solutions of the relaxed integer program. Investigating other frameworks, *e.g.*, *pipage rounding*, for solving extensions of our formulations is a promising avenue for designing monitoring solutions with improved worst-case performance.

References

[1] C. Avin, M. Ghobadi, C. Griner, and S. Schmid, "On the complexity of traffic traces and implications," *ACM Meas. Anal. Comput.*, vol. 4, no. 1, 2020.

[2] O. Michel, J. Sonchack, E. Keller, and J. M. Smith, "Packet-level analytics in software without compromises," in *Proc. USENIX HotCloud*, Jul. 2018.

[3] Gigamon Inc., "Understanding network tap devices," accessed Jun. 29, 2021. [Online]. Available: https://www.gigamon.com/products/access-traffic/network-taps.html

[4] J. Rasley, B. Stephens, C. Dixon, E. Rozner *et al.*, "Planck: Millisecond-scale monitoring and control for commodity networks," in *Proc. ACM SIGCOMM*, Aug. 2014.

[5] P. J. Moyer, "SDN-based mirroring of traffic flows for in-band network analytics," Patent 20 170 048 312A1, Feb., 2017.

[6] S. Sadrhaghighi, M. Dolati, M. Ghaderi, and A. Khonsari, "SoftTap: A software-defined tap via switch-based traffic mirroring," in *Proc. IEEE NetSoft*, Jul. 2021.

[7] Juniper Networks, "Understanding port mirroring and analyzers," accessed Jun. 29, 2021. [Online]. Available: https://www.juniper.net/documentation/en_US/junos/topics/concept/port-mirroring-qfx-series-understanding.html

[8] Open Networking Foundation, "OpenFlow switch specification v1.5.1," accessed Jan. 20, 2022. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[9] "Open vSwitch," accessed Jun. 29, 2021. [Online]. Available: https://www.openvswitch.org/

[10] S. Jeong, D. Lee, J. Li, and J. W.-K. Hong, "OpenFlow-based virtual tap using open vSwitch and DPDK," in *Proc. IEEE/IFIP NOMS*, Jul. 2018.

[11] J. Hong, S. Jeong, J.-H. Yoo, and J. W.-K. Hong, "Design and implementation of eBPF-based virtual tap for inter-VM traffic monitoring," in *Proc. IEEE CNSM*, Nov. 2018.

[12] S. Jeong, J. H. You, and J. W. K. Hong, "Design and implementation of virtual tap for SDN-based OpenStack networking," in *Proc. IFIP/IEEE IM*, Apr. 2019.

[13] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe *et al.*, "OpenVirteX: Make your virtual SDNs programmable," in *Proc. ACM SIGCOMM HotSDN*, Aug. 2014.

[14] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller *et al.*, "Can the production network be the testbed?" in *Proc. USENIX OSDI*, Nov. 2010.

[15] J.-P. Sheu, W.-T. Lin, and G.-Y. Chang, "Efficient TCAM rules distribution algorithms in software-defined networking," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 2, 2018.

[16] S. Zhao, D. Li, K. Han, and Z. Zhu, "Proactive and hitless vSDN reconfiguration to balance substrate TCAM utilization: From algorithm design to system prototype," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 2, 2019.

[17] N. Grover, N. Agarwal, and K. Kataoka, "LiteFlow: Lightweight and distributed flow monitoring platform for SDN," in *Proc. IEEE NetSoft*, Jun. 2015.

[18] H. Xu, S. Chen, Q. Ma, and L. Huang, "Lightweight flow distribution for collaborative traffic measurement in software-defined networks," in *Proc. IEEE INFOCOM*, Jul. 2019.

[19] K. B. Nougnanke, M. Bruyère, and Y. Labit, "Low-overhead near-real-time flow statistics collection in SDN," in *Proc. IEEE NetSoft*, Jul. 2020.

[20] G. Tangari, D. Tuncer, M. Charalambides, Y. Qi *et al.*, "Self-adaptive decentralized monitoring in software-defined networks," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 4, 2018.

[21] S. Clayman, A. Galis, and L. Mamatas, "Monitoring virtual networks with Lattice," in *Proc. IEEE/IFIP NOMS Workshops*, Apr. 2010.

[22] F. Tusa, S. Clayman, and A. Galis, "Dynamic monitoring of data center slices," in *Proc. IEEE NetSoft*, Jun. 2019.

[23] G. Yang, H. Jin, M. Kang, G. J. Moon *et al.*, "Network monitoring for SDN virtual networks," in *Proc. IEEE INFOCOM*, Jul. 2020.

[24] S. Shirali-Shahreza and Y. Ganjali, "FleXam: Flexible sampling extension for monitoring and security applications in OpenFlow," in *Proc. ACM SIGCOMM HotSDN*, Aug. 2013.

[25] J. Suárez-Varela and P. Barlet-Ros, "Towards a NetFlow implementation for OpenFlow software-defined networks," in *Proc. IEEE ITC*, Sep. 2017.

[26] ——, "SBAR: SDN flow-based monitoring and application recognition," in *Proc. of the Symposium on SDN Research*, Mar. 2018.

[27] B. Claise, "Cisco systems NetFlow services export v9," accessed Jun. 29, 2021. [Online]. Available: https://www.ietf.org/rfc/rfc3954.txt

[28] sFlow, "Making the network visible," accessed Jun. 29, 2021. [Online]. Available: https://sflow.org/

[29] C. Kattadige, K. N. Choi, A. Wijesinghe, A. Nama *et al.*, "Seta++: Real-time scalable encrypted traffic analytics in multi-Gbps networks," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, 2021.

[30] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner *et al.*, "Nitrosketch: Robust and general sketch-based monitoring in software switches," in *Proc. ACM SIGCOMM*, Aug. 2019.

[31] X. Wang, X. Li, S. Pack, Z. Han *et al.*, "STCS: Spatial-temporal collaborative sampling in flow-aware software-defined networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, 2020.

[32] R. Cohen and E. Moroshko, "Sampling-on-demand in SDN," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, 2018.

[33] S. U. Rehman, W.-C. Song, and M. Kang, "Network-wide traffic visibility in OF@TEIN SDN testbed using sFlow," in *Proc. IEEE APNOMS*, Sep. 2014.

[34] H. Baek, C. Jin, G. Jiang, C. Lumezanu *et al.*, "Polygravity: Traffic usage accountability via coarse-grained measurements in multi-tenant data centers," in *Proc. ACM SoCC*, Sep. 2017.

[35] O. Tilmans, T. Bühler, I. Poese, S. Vissicchio *et al.*, "Stroboscope: Declarative network monitoring on a budget," in *Proc. USENIX NSDI*, Apr. 2018.

[36] Y. Zhu, N. Kang, J. Cao, A. Greenberg *et al.*, "Packet-level telemetry in large datacenter networks," in *Proc. ACM SIGCOMM*, Aug. 2015.

[37] Arista Networks, "Pervasive network monitoring with DANZ monitoring fabric," accessed Nov. 29, 2021. [Online]. Available: https://www.arista.com/assets/data/pdf/Whitepapers/DMF_Technical_Solution_Guide.pdf

[38] T. Favale, M. Trevisan, I. Drago, and M. Mellia, "α-mon: Traffic anonymizer for passive monitoring," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, 2021.

[39] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar *et al.*, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *Proc. ACM SIGCOMM*, Aug. 2016.

[40] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal *et al.*, "Language-directed hardware design for network performance monitoring," in *Proc. ACM SIGCOMM*, Aug. 2017.

[41] A. Gupta, R. Harrison, M. Canini, N. Feamster *et al.*, "Sonata: Query-driven streaming network telemetry," in *Proc. ACM SIGCOMM*, Aug. 2018.

[42] J. Sonchack, O. Michel, A. J. Aviv, E. Keller *et al.*, "Scaling hardware accelerated network monitoring to concurrent and dynamic queries with * Flow," in *Proc. USENIX ATC*, Jul. 2018.

[43] O. Michel, J. Sonchack, G. Cusack, M. Nazari *et al.*, "Software packet-level network analytics at cloud scale," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 1, 2021.

[44] S. Tang, D. Li, B. Niu, J. Peng *et al.*, "Sel-INT: A runtime-programmable selective in-band network telemetry system," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 2, 2019.

[45] Q. Zheng, S. Tang, B. Chen, and Z. Zhu, "Highly-efficient and adaptive network monitoring: When INT meets segment routing," *IEEE Trans. on Netw. and Serv. Manag.*, no. 3, 2021.

[46] P. Tammana, R. Agarwal, and M. Lee, "Simplifying datacenter network debugging with PathDump," in *Proc. USENIX OSDI*, Nov. 2016.

[47] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Trumpet: Timely and precise triggers in data centers," in *Proc. ACM SIGCOMM*, Aug. 2016.

[48] A. Khandelwal, R. Agarwal, and I. Stoica, "Confluo: Distributed monitoring and diagnosis stack for high-speed networks," in *Proc. USENIX NSDI*, Feb. 2019.

[49] K. Chen, A. Singla, A. Singh, K. Ramachandran *et al.*, "OSA: An optical switching architecture for data center networks with unprecedented flexibility," *IEEE/ACM Trans. Netw.*, vol. 22, no. 2, 2013.

[50] A. S. da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN," in *Proc. IEEE/IFIP NOMS*, Apr. 2016.

[51] A. Bianco, P. Giaccone, S. Kelki, N. M. Campos *et al.*, "On-the-fly traffic classification and control with a stateful SDN approach," in *Proc. IEEE ICC*, May 2017.

[52] S. Shakkottai and R. Srikant, "Network optimization and control," *Foundations and Trends in Networking*, vol. 2, no. 3, 2008.

[53] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long *et al.*, "Ceph: A scalable, high-performance distributed file system," in *Proc. USENIX OSDI*, Nov. 2006.

[54] W. Xie, J. Zhou, M. Reyes, J. Noble *et al.*, "Two-mode data distribution scheme for heterogeneous storage in data centers," in *Proc. IEEE Big Data*, Oct. 2015.

[55] K.-i. Ishikawa, "ASURA: Scalable and uniform data distribution algorithm for storage clusters," *arXiv preprint arXiv:1309.7720*, 2013.

[56] A. Roy, H. Zeng, J. Bagga, G. Porter *et al.*, "Inside the social network's (datacenter) network," in *Proc. ACM SIGCOMM*, Aug. 2015.

[57] J. Wang *et al.*, "FlowTracer: An effective flow trajectory detection solution based on probabilistic packet tagging in SDN-enabled networks," *IEEE Trans. on Netw. and Serv. Manag.*, vol. 16, no. 4, 2019.

[58] C. Hardegen *et al.*, "Predicting network flow characteristics using deep learning and real-world network traffic," *IEEE Trans. on Netw. and Serv. Manag.*, vol. 17, no. 4, 2020.

[59] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proc. ACM STOC*, Dec. 1984.

[60] R. Kleinberg, "Notes on approximation algorithms," accessed Jun. 29, 2021. [Online]. Available: www.cs.cornell.edu/courses/cs6820/2013fa/handouts/approx_algs.pdf

[61] Y. Wang, D. Jiang, L. Huo, and Y. Zhao, "A new traffic prediction algorithm to software-defined networking," *Springer Mobile Networks and Applications*, vol. 26, no. 2, 2021.

[62] R. F. Fouladi, O. Ermiş, and E. Anarim, "A DDoS attack detection and defense scheme using time-series analysis for SDN," *Elsevier Journal of Information Security and Applications*, vol. 54, no. 1, 2020.

[63] J. Suh, T. T. Kwon, C. Dixon, W. Felter *et al.*, "Opensample: A low-latency, sampling-based measurement platform for commodity SDN," in *Proc. International Conference on Distributed Computing Systems*, Jun. 2014.

[64] B. L. Dalmazo, J. P. Vilela, and M. Curado, "Predicting traffic in the cloud: A statistical approach," in *Proc. IEEE International Conference on Cloud and Green Computing*, Oct. 2013.

[65] M. Kodialam, T. Lakshman, and S. Mohanty, "Runs based traffic estimator (RATE): A simple, memory efficient scheme for per-flow rate estimation," in *Proc. IEEE INFOCOM*. IEEE, Mar. 2004.

[66] A. Lazaris and V. K. Prasanna, "An LSTM framework for modeling network traffic," in *Proc. IFIP/IEEE IM*, May 2019.

[67] Y. Li, H. Liu, W. Yang, D. Hu *et al.*, "Inter-data-center network traffic prediction with elephant flows," in *Proc. IEEE/IFIP NOMS*, Apr. 2016.

[68] D. Andreoletti, S. Troia, F. Musumeci, S. Giordano *et al.*, "Network traffic prediction based on diffusion convolutional recurrent neural networks," in *Proc. IEEE INFOCOM WKSHPS*, Apr. 2019.

[69] Y. Zhang *et al.*, "Approximation for knapsack problems with multiple constraints," *Springer JCST*, vol. 14, no. 4, 1999.

[70] The university of Adelaide, "The internet topology Zoo," accessed Jun. 29, 2021. [Online]. Available: http://www.topology-zoo.org/index.html

[71] A. X. Liu, C. R. Meiners, and E. Torng, "TCAM Razor: A systematic approach towards minimizing packet classifiers in tcams," *IEEE/ACM Trans. Netw.*, vol. 18, no. 2, 2009.

[72] J. L. García-Dorado, A. Finamore, M. Mellia, M. Meo *et al.*, "Characterization of ISP traffic: Trends, user habits, and access technology impact," *IEEE Trans. Netw. Service Manag.*, vol. 9, no. 2, 2012.

[73] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp and P. Francois, "A declarative and expressive approach to control forwarding paths in carrier-grade networks," in *Proc. ACM SIGCOMM*, Aug. 2015.

[74] "Declarative and Expressive Forwarding Optimizer (DEFO)," accessed Jun. 29, 2021. [Online]. Available: https://sites.uclouvain.be/defo/

[75] Y. Li, R. Miao, H. H. Liu, Y. Zhuang *et al.*, "HPCC: High precision congestion control," in *Proc. ACM SIGCOMM*, Aug. 2019.

[76] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi *et al.*, "PINT: Probabilistic in-band network telemetry," in *Proc. ACM SIGCOMM*, Jul. 2020.

[77] Gurobi Optimization LLC., "Gurobi Solver," accessed Jun. 29, 2021. [Online]. Available: https://www.gurobi.com

[78] S. Hu, W. Bai, G. Zeng, Z. Wang *et al.*, "Aeolus: A building block for proactive transport in datacenters," in *Proc. ACM SIGCOMM*, Jul. 2020.

[79] M. Alizadeh, S. Yang, M. Sharif, S. Katti *et al.*, "pFabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, 2013.

[80] F. Yao *et al.*, "A comparative analysis of data center network architectures," in *Proc. IEEE ICC*, Jun. 2014.

[81] Z. H. Center, "Zoom system requirements for Windows, macOS, and Linux," accessed Jun. 29, 2021. [Online]. Available: https://support.zoom.us/hc/en-us/articles/201362023-System-requirements-for-Windows-macOS-and-Linux

[82] ONOS Community, "Open source network operating system (ONOS)," accessed Jun. 29, 2021. [Online]. Available: https://onosproject.org/

[83] G. M. Suranegara, I. N. Ichsan, and E. Setyowati, "Reactive forwarding and proactive forwarding performance comparison on SDN-based network," *Jurnal Online Informatika*, vol. 5, no. 1, 2020.

[84] S. Yoon, T. Ha, S. Kim, and H. Lim, "Scalable traffic sampling using centrality measure on software-defined networks," *IEEE Communications Magazine*, vol. 55, no. 7, 2017.

[85] Wireshark org., "What are network taps," accessed Jun. 29, 2021. [Online]. Available: https://www.wireshark.org

[86] R. M. Karp, *Reducibility among Combinatorial Problems*. Springer, 1972.

APPENDIX A
PROOF OF THEOREM 1

Proof. We describe a polynomial time reduction from an instance of the *Set Cover* problem [86] to an instance of PortTM with a *single* switch. In the Set Cover problem, we are given a ground set $\mathcal{U} = \{u_1, \ldots, u_n\}$ and a collection of subsets $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_K\}$ such that their union is $\mathcal{U}$. Selecting subset $\mathcal{S}_k$ incurs cost c_k. A cover is a subset $\mathcal{C} \subseteq \mathcal{S}$ of sets $\mathcal{S}_k$ whose union is $\mathcal{U}$. The objective of the Set Cover problem is to find a cover whose cost is minimum. To reduce Set Cover to PortTM, map each element $u_i \in \mathcal{U}$ to a flow and each subset $\mathcal{S}_k$ to a port p_k in the PortTM problem, where all ports are connected to a single switch. Set $r_{p_k} = c_k$. Then, λ is the cost of the minimum set cover that covers all elements u_i. □

APPENDIX B
PROOF OF THEOREM 4

Proof. The problem is NP-hard via reduction from *Partition* problem [86]. In Partition problem, we have a set of numbers $\mathcal{A} = \{a_1, \ldots, a_n\}$ such that $\sum_{a_i \in \mathcal{A}} a_i = 2C$. The goal is to decide if there is a subset $\mathcal{B} \subset \mathcal{A}$ such that $\sum_{a_i \in \mathcal{B}} a_i = C$. For each number a_i, we assume there is a flow f_i with traffic rate $r_i = a_i$ in FlowTM. All flows traverse two switches that their mirroring and TCAM capacities are equal to C and n, respectively. If there is a partition $\mathcal{B}$ that sums to C, then all flows can be mirrored in FlowTM (*i.e.*, the optimal value of FlowTM is equal to n). □