SoftTap: A Software-Defined TAP via Switch-Based Traffic Mirroring

Sogand Sadrhaghighi* Mahdi Dolati† Majid Ghaderi* Ahmad Khonsari†
* University of Calgary, Calgary, Canada. Emails: {sogand.sadrhaghighi, mghaderi}@ucalgary.ca
† University of Tehran, Tehran, Iran. Emails: {mahdidolati, a_khonsari}@ut.ac.ir

***Abstract*—With widespread deployment of virtualization technologies in datacenter networks, traditional tools used for network monitoring, such as hardware taps, become unfit. This is due to the inability of hardware solutions for dynamic deployment and virtual network monitoring. This paper presents the design and evaluation of SoftTap, a scalable alternative to hardware taps which is capable of operating over both physical and virtual switches. SoftTap is based on port and flow mirroring capabilities of commodity OpenFlow switches and is not limited to a specific network architecture or topology. A key design challenge in SoftTap is the fast computation of switch mirroring configurations in large-scale deployments. Our design is based on novel polynomial time approximation algorithms that are shown to achieve bounded approximation ratios compared to optimal solutions. We evaluate SoftTap using model-driven simulations as well as realistic Mininet experiments. Specifically, our simulations consider large networks to show the scalability of SoftTap. Mininet experiments, on the other hand, consider its real-world utility by implementing an intrusion detection system (IDS) and a VoIP metering application on top of SoftTap. In our experiments, under SoftTap, IDS achieves up to 25% higher detection recall, while VoIP metering achieves up to 23% less packet loss compared to existing mirroring-based traffic monitoring approaches.**

I. Introduction

Motivation. As datacenters grow in size and proliferate, the importance of having in-depth network-wide visibility significantly increases. Providing such pervasive visibility requires examining network behaviour at the granularity of packets [1], [2]. Traditionally, packet-level traffic traces are collected using hardware tap (**T**est **A**ccess **P**oint) devices, deployed at specific points in the network. Such hardware middleboxes, however, have become less useful as server and network virtualization have become ubiquitous in datacenters. In a virtualized datacenter network, traffic traverses through *virtual* links and switches, making hardware taps ill-suited or even infeasible to deploy (*e.g.*, monitoring traffic between virtual machines that are connected via a virtual switch). Without the ability to capture and analyze packet-level 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. This motivates adopting "virtual" taps, *i.e.*, software-defined traffic monitoring systems, in virtualized network environments. In particular, a software-defined tap does not require a specialized monitoring infrastructure and can be dynamically deployed to different locations in the network, making it well-suited for virtualized environments, where virtual networks are created as overlays on top of a shared physical network substrate.

978-1-6654-0522-5/21/$31.00 ©2021 IEEE

To this end, we design and evaluate a software-defined tap using the built-in *port mirroring* [3] feature available on both commodity physical and software network switches. In port mirroring, a switch duplicates its passing traffic to a designated mirroring port. Port mirroring, however, could suffer from "loss" due to *port over-subscription* (POS) [3], [4]. In POS, traffic from every port on a switch is copied to its mirroring port. If the total rate of traffic at the switch exceeds the capacity of the mirroring port, the overload traffic is dropped. Additionally, while simple and easy to deploy, POS is highly inefficient as it creates a static mirroring configuration independent of the distribution of traffic and the routing layout of the network. Specifically, attempting to mirror all passing flows in POS, *every flow is mirrored at every switch along its route*. This results in most flows being mirrored at multiple switches, which not only increases mirroring traffic but also wastes the mirroring capacity of the network.

As a motivating example, consider the network depicted in Fig. 1 which has three flows and three switches. With POS, only flow f_2 is fully mirrored (on switch S_3), while flows f_1 and f_3 are only partially mirrored, even though the total rate of flows (at $2.5C$) is less than the aggregate mirroring capacity of the network (at $3C$). From this example, we see that to minimize the mirroring loss, the mirroring configuration of switches must be determined with respect to the routing layout and traffic distribution of the network. To this extent, it is required to acquire global network information as well as the ability to dynamically change mirroring configurations of network switches as the network traffic changes. While some switches support dynamic configuration of mirroring ports through vendor-specific APIs (*e.g.*, [5]), most legacy switches do not support such functionality. Motivated by the recent wide-spread availability of programmable switches (*e.g.*, OpenFlow switches), we argue that it is now possible to build a traffic monitoring system based on switch mirroring capability, which not only is low-cost but also provides high fidelity traffic traces on par with traditional hardware taps.

Our Approach. Our approach is based on network-wide optimization of switch mirroring configurations with respect to network traffic and routing layout. Our key idea is that, from traffic monitoring perspective, it does not matter which switch in the network mirrors a flow, as long as each flow is mirrored by at least one switch on its path. This insight leads us to design a system that minimizes mirroring losses by balancing mirroring load among switches. We demonstrate that we maximize *flow coverage*, *i.e.*, the percentage of flows that are mirrored with no loss, by load-balancing among switches. One

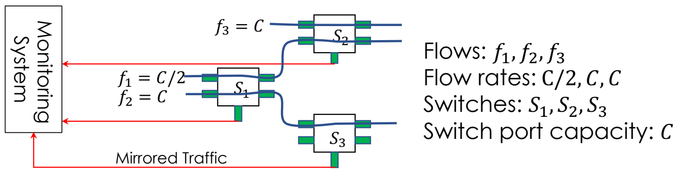


Fig. 1: While POS is unable to fully mirror the flows, SoftTap is able to fully mirror all flows by mirroring flows f_1, f_2, and f_3 on switches S_1, S_3 and S_2, respectively.

of the main challenges we address is how to determine switch configurations in near-realtime, so that the mirroring system can quickly react to network traffic changes. To this end, we present the design and evaluation of SoftTap, a software-defined packet-level tap based on mirroring capabilities of commodity OpenFlow switches. By employing switches for traffic monitoring, our design is able to provide *pervasive* visibility covering every flow in the network, while *scaling out* as the network expands. SoftTap does not require any switch modifications and is agnostic to whether the switches are physical or virtual, as long as they support OpenFlow, and so, is an ideal solution for virtual cloud environments.

At the core of SoftTap is a mirroring orchestrator that is implemented as an OpenFlow application. By solving an optimization model, the orchestrator computes an efficient mirroring configuration, taking into account the routing layout of the network, which is then installed on switches via the OpenFlow protocol. Recall the network example depicted in Fig. 1. While POS was unable to fully mirror the flows, as will be discussed later, SoftTap would be able to fully mirror all flows, while achieving a mirroring rate (at $2.5C$) that is lower than that of POS (at $3C$). To achieve this, SoftTap supports two mirroring strategies, namely *port mirroring* and *flow mirroring*. In port mirroring, as in legacy switches, a subset of ports on each switch is selected for mirroring. In this strategy, for monitoring a single flow, all flows passing through the same port will be mirrored; therefore, it is inevitable that some traffic will be unnecessarily duplicated. Flow mirroring, on the other hand, mirrors traffic optimally by selecting a subset of flows on each switch. While this strategy results in optimal flow coverage, it is expensive to implement as it requires installing additional forwarding rules on each switch (Section II). Our results also indicate that it is slower to compute the optimal mirroring configuration under flow mirroring strategy compared to port mirroring strategy (Section IV).

Contributions. Our main contributions are:

- We present the design and evaluation of SoftTap, a low-cost and scalable switch-based software-defined tap for traffic monitoring in SDNs.
- We formulate the problem of computing the optimal mirror configuration, supported by commodity OpenFlow switches, under port and flow mirroring strategies. We prove that the problem is NP-hard under both strategies and thus, focus on designing approximation algorithms.
- We design polynomial time approximation algorithms for the problem under both mirroring strategies, and analytically bound the approximation ratio of our algorithms.
- We present simulation results to show the performance and scalability of SoftTap in large networks, and compare its performance against two existing mirroring approaches, namely Planck [4] and Stroboscope [6].
- We also present Mininet experiments to show the utility of SoftTap when used with an intrusion detection system (IDS) and a VoIP metering application in a realistic network environment. Our experiments show that, compared to POS-based traffic mirroring approaches, under SoftTap, IDS achieves up to 25% higher detection recall, while VoIP metering achieves up to 23% less packet loss.

Due to space limitation, some proofs, technical details and evaluation results are omitted from this version and can be found in our technical report [7].

II. System Design

In this section, we describe the architecture of SoftTap, and discuss how it interacts with an OpenFlow network. We also present the formulation of port and flow mirroring problems that form the core of the SoftTap orchestrator.

A. *SoftTap Design*

The high-level architecture of SoftTap is depicted in Fig. 2. The main components of the system are described below.

OpenFlow Controller. The OpenFlow controller maintains a state graph of the switches and exposes a northbound API to the OpenFlow applications, in this case, SoftTap and packet analyzer apps. The API provides an abstract view of the network and allows SoftTap to read the state of the switches and dynamically change their mirroring behavior in response to traffic monitoring queries from packet analyzer apps or changes in network traffic (*e.g.*, arrival of a new flow).

Collector Servers. A set of traffic collector servers are connected to network switches. The number of collector servers depends on the size of the network and the type of filtering/pre-processing provided by them. When configuring mirroring on switches, SoftTap communicates with the controller to install forwarding rules that specify a collector server as the destination of the mirrored traffic. Collector servers are selected in a way to achieve load balancing among them.

Analyzer Apps. Analyzer apps, such as IDS, send traffic *monitoring queries* to SoftTap, and analyze resulting collected packet traces. Each analyzer sends monitoring queries that specify a mirroring strategy to be used and a particular set of flows to be monitored, or a wildcard query to monitor all flows. For example, a lightweight controller application using OpenFlow statistics [8] or predefined rules, submits mirroring queries, specifying the set of flows to be mirrored. There could be multiple analyzers that monitor the same traffic. SoftTap collects all such queries and orchestrates a global mirroring configuration to efficiently collect traffic traces from switches. While not considered in our implementation, SoftTap could be extended to consider different priorities among analyzers (*e.g.*, by attaching a suitable utility function to each analyzer [9]).

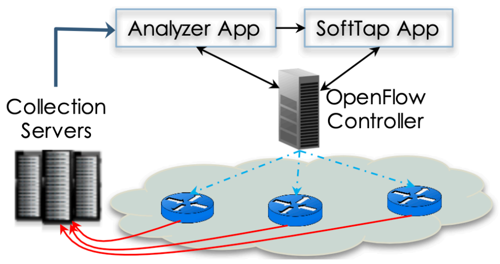


Fig. 2: High-level architecture of SoftTap-based monitoring system.

Our current design provides only a best-effort monitoring service and does not distinguish between different queries.

SoftTap App. The SoftTap app listens for flow monitoring queries from analyzers, invokes the *Mirror Optimizer* module to compute an optimal mirror configuration, and then, communicates with the controller to install appropriate rules on OpenFlow switches using the *Mirror Orchestrator* module. Depending on the underlying mirroring strategy, *i.e.*, port or flow mirroring, the mirror configuration specifies the subset of ports or flows to be mirrored on each switch. Mirror Orchestrator module communicates with the OpenFlow controller to install forwarding rules that implement the mirroring configuration on each switch. At a high-level, an OpenFlow rule is composed of match and action sets, where the former identifies a set of flows and the latter defines the actions to be applied to the members of the set. Adding a new rule depends on the specific OpenFlow controller. In our implementation, in Section IV, we use ONOS controller, which has a REST API that is used to install rules by sending JSON-formatted POST requests to the controller. With port mirroring, only one rule needs to be installed for each mirrored port on a switch. However, with flow mirroring, when flows cannot be aggregated, one rule needs to be installed on the switch for each mirrored flow. From the implementation perspective, mirroring rules do not interfere with existing forwarding rules. In OpenFlow switches the installed rules have to be refreshed periodically. This allows SoftTap to automatically remove completed flows from its set of active flows.

B. Mirror Optimization

The Mirror Optimizer module of SoftTap solves an optimization problem to compute the optimal mirror configuration for a given monitoring query (*i.e.*, set of flows). We define two problems, namely Selective Port Mirroring (SPM) and Selective Flow Mirroring (SFM), corresponding to the port and flow mirroring strategies, respectively. The objective of these problems is to minimize the maximum mirroring load on any switch in order to minimize mirroring losses. To reduce the mirroring configuration computation time, Mirror Optimizer fixes the configuration of existing flows in the optimization problem. It is crucial to keep these problems as simple as possible, as they have to be repeatedly solved at runtime.

Model. The set of flows to be mirrored is denoted by $\mathcal{F}$. The set of switches that can be used for mirroring is denoted by $\mathcal{W}$, where each switch $w \in \mathcal{W}$ has a set of ports $\mathcal{P}_w$. Let $\mathcal{P} = \cup_{w\in\mathcal{W}}\mathcal{P}_w$ denote the set of all switch ports in the network. We say flow f meets port $p \in \mathcal{P}_w$ if p is the *outgoing* port of f on switch w. Let $\mathcal{P}_f$ denote the set of ports that flow $f \in \mathcal{F}$ meets on its path in the network. Similarly, let $\mathcal{F}_p$ denote the set of flows that meet port $p \in \mathcal{P}$. The traffic rate of flow f is denoted by r_f, while the traffic rate at outgoing port p is denoted by r_p.

Selective Port Mirroring (SPM). Define binary decision variable x_p to indicate whether port p is mirrored ($x_p = 1$) or not ($x_p = 0$). Then, SPM can be formulated as an Integer Linear Programming (ILP):

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

Constraint (1b) ensures that every flow is mirrored by at least one of the ports it meets. Constraint (1c) enforces an upper bound on the mirroring load of each switch, which is then minimized across all switches in the objective. Notice that λ is an optimization variable and can be interpreted as the maximum mirroring load of any switch in the network. By introducing λ, we have essentially linearized the objective $\min\max_{w\in\mathcal{W}} \lambda_w$, where λ_w denotes the mirroring load of switch w.

Theorem 1. *Problem SPM is NP-hard.*

Proof. The proof is based on reduction from an instance of the Set Cover problem to an instance of SPM, available in [7]. □

At first glance, SPM appears to be similar to the Set Cover problem. However, we prove that SPM is considerably more general than the Set Cover problem [7]. Additionally, despite the formulation similarity between SPM 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 SPM 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.

Selective Flow Mirroring (SFM). Let $z_{f,w}$ denote the binary decision variable that indicates if flow f is mirrored on switch w or not. Define $\mathcal{W}_f$ and $\mathcal{F}_w$ to be the set of all switches that flow f meets on its path and the set of all flows that pass through switch w, respectively. Then, SFM can be formulated as the following ILP:

$$\begin{aligned} \textbf{SFM:}\quad & \min \lambda && && \text{(2a)}\\ \text{s.t.}\quad & \textstyle\sum_{w\in\mathcal{W}_f} z_{f,w} \geq 1, && \forall f \in \mathcal{F} && \text{(2b)}\\ & \textstyle\sum_{f\in\mathcal{F}_w} z_{f,w} r_f \leq \lambda, && \forall w \in \mathcal{W} && \text{(2c)}\\ & x_{f,w} \in \{0,1\}. && \forall f \in \mathcal{F}, w \in \mathcal{W} && \text{(2d)} \end{aligned}$$

Constraint (2b) ensures that each flow is mirrored by at least one of the switches in its path, while constraint (2c) enforces the upper bound λ on the mirroring load of any switch.

Similarly, SFM also resembles the MCFR problem. Here, we can consider switches to be equivalent to paths. However, we can see that there is no notion of links in SFM. Consequently, using existing solutions of MCFR to solve SFM can lead to sub-optimal solutions as they do not consider the special structure of the problem. In Section III, we will show that SFM is more related to the Partition and Machine Scheduling problems, albeit with rather subtle differences.

Theorem 2. *Problem SFM is NP-hard.*

Proof. The proof is based on reducing the Partition problem to SFM and can be found in [7]. □

III. Algorithm Design

It is crucial for the Mirror Optimizer module of SoftTap to quickly compute a mirror configuration, as long delays effectively lead to increased mirroring losses. Given that SPM and SFM are NP-hard, we focus on designing polynomial time approximation algorithms for these problem.

Notational Convention. Symbols denoting 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 $\overline{\Phi}_z = \max_{z\in\mathcal{Z}} \Phi_z$.

A. Port Mirroring Algorithm (PMA)

SPM is more general than the Set Cover problem [10], and therefore, none of the existing solutions for the Set Cover problem can be directly applied to solve SPM. A standard technique to solve such ILPs is to relax the integrality constraints, solve the resulting LP and then round the fractional solution. However, we design a combinatorial algorithm that avoids solving any LP in the first place, thus performing substantially faster than any rounding-based solution.

Algorithm. Our proposed algorithm is presented in Alg. 1. At the core of the algorithm is a while loop. In each iteration of the loop, the *most cost efficient* not-yet-mirrored port is selected to be mirrored. The most cost efficient port is defined as a port whose selection increases the objective λ by the least amount per each newly mirrored flow (see line 3). After selecting the best port, called p^*, its corresponding decision variable x_{p^*} is set to one. Following that, all the flows that pass through the port p^* are removed from the set $\mathcal{U}$, which initially contains all the flows. The while loop continues until $\mathcal{U}$ becomes empty. In line 7, the set of not-yet-mirrored ports $\mathcal{V}$ is updated by deleting p^*. In line 8, the current maximum mirroring rate is calculated to be used in the next iteration of the while loop. In order to analyze the performance of the PMA, in the process of computing the primal solution, we also build the dual solution of the SPM problem, given as,

Dual:

$$\max \sum_{f\in\mathcal{F}} \alpha_f \tag{3a}$$

$$\text{s.t.} \sum_{f\in\mathcal{F}_p} \alpha_f - r_p\beta_{\omega_p} \le 0, \qquad \forall p \in \mathcal{P} \tag{3b}$$

$$\sum_{w\in\mathcal{W}} \beta_w \le 1, \tag{3c}$$

where, α_f and β_w are the dual variables associated with the constraints (1b) and (1c), respectively. Theorem 3 describes how the values for variables α_f and β_w are computed, as they affect the approximation ratio achieved by the algorithm..

Alg. 1: Port Mirroring Algorithm (PMA)

```
procedure PMA ($\mathcal{F}$, $\mathcal{W}$)
1   $\{x_p\} \leftarrow \{0\}, \lambda \leftarrow 0, \mathcal{U} \leftarrow \mathcal{F}, \mathcal{V} \leftarrow \mathcal{P}$
2   while $\mathcal{U} \ne \emptyset$ do
3       foreach $p \in \mathcal{V}$ do: $c_p \leftarrow \frac{[r_p + \sum_{q\in\mathcal{P}_{\omega_p}} x_q r_q - \lambda]^+}{|\mathcal{F}_p \cap \mathcal{U}|}$
4       $p^* \leftarrow \mathrm{argmin}_{p\in\mathcal{V}}\{c_p\}$
5       $x_{p^*} \leftarrow 1$
6       $\mathcal{U} \leftarrow \mathcal{U} - \mathcal{F}_{p^*}$
7       $\mathcal{V} \leftarrow \mathcal{V} - \{p^*\}$
8       $\lambda \leftarrow \max_{w\in\mathcal{W}} \sum_{p\in\mathcal{P}_w} x_p r_p$
9       Compute dual variables as in Theorem 3
```

Theorem 3. *PMA runs in $O\big((\overline{P}_f + \overline{P}_w \log P)\cdot F\big)$ and attains the approximation ratio R, for $1 \le R \le 1 + M_{w^*}(\overline{P}^*_f - 1)$, where, w^* denotes the switch with the maximum mirroring load, $\overline{P}^*_f$ is the maximum path length of any flow mirrored on w^*, and M_{w^*} is the number of flows mirrored on w^*.*

Proof. The proof is divided into two parts, as follows.

Runtime Analysis. Upon selecting a new port, PMA has to re-compute $|\mathcal{F}_p \cap \mathcal{U}|$ for all other ports that share a flow with the selected port. Each flow meets at most $\overline{P}_f$ ports, and thus the number of these operations is bounded by $(\overline{P}_f \cdot F)$. In each iteration, the mirroring load of one switch changes, which means that the costs of at most $\overline{P}_w$ ports change. Using a Min-Heap data structure, updating port costs can be implemented in $O(\log P)$. Thus, the cost of selecting p^* in all iterations of the loop will be bounded by $O(F \cdot \overline{P}_w \cdot \log P)$.

Approximation Analysis. To analyze the PMA algorithm, we employ the dual fitting technique. To this end, we compute a solution for the dual of the SPM problem (given in (3a)) and then characterize the duality gap to specify the approximation ratio. Let w^* denote the switch with the maximum mirroring load. Consider the following value assignment for variable α_f:

$$\alpha_f = \begin{cases} \frac{r_f}{R}, & \text{if } f \text{ is mirrored on } w^* \\ 0, & \text{otherwise.} \end{cases}$$

From the above, the objective of the dual problem will be equal to $\frac{1}{R}$ of the value of the primal objective. Based on the weak duality theorem, the objective value of the dual problem serves as a lower bound for the primal objective value. Let $\lambda^\star$ be the optimum value of the primal problem. We have,

$$\frac{\sum_{p\in\mathcal{P}_{w^*}} x_p r_p}{R} \le \lambda^\star, \tag{4}$$

or, equivalently, $\sum_{p\in\mathcal{P}_{w^*}} x_p r_p \le R\lambda^\star$, which indicates that the approximation ratio of PMA is given by R, as the LHS of the inequality is the primal objective value. Now, consider the two cases given below for the lower and upper bounds of the approximation ratio R.

Lower Bound: In the case that the flows mirrored on w^* do not pass any other switch in the network, β_w can be calculated as follows: $\beta_w = 1$ if $w = w^*$, and $\beta_w = 0$ otherwise. Evidently, constraint (3c) holds since, $\sum_{w\in\mathcal{W}} \beta_w = \beta_{w^*} = 1$.

We can derive the value of R from constraint (3b) as,

$$\sum_{f\in\mathcal{F}_p} \alpha_f \leq r_p\beta_{w^*} \Rightarrow \tfrac{r_p}{R} \leq r_p\beta_{w^*}, \tag{5}$$

which results in $1 \leq R$.

Upper Bound: Consider the general case in which the flows that are mirrored on switch w^* traverse some other switches. Let $\mathcal{M}_{w^*}$ denote the set of these flows. Furthermore, let $\overline{P}^*_f$ denote that maximum path length of any flow in $\mathcal{M}_{w^*}$. Note that, there are at most $1 + M_{w^*}(\overline{P}^*_f - 1)$ switches (denoted by $\mathcal{W}'$) that forward a flow in $\mathcal{M}_{w^*}$. Additionally, the maximum path length of any flow traversing switch w^* is $\overline{P}^*_f$ and all those paths include the common switch w^*. Consequently, β_w can be computed as follows:

$$\beta_w = \begin{cases} \frac{1}{1+M_{w^*}(\overline{P}^*_f - 1)}, & \text{if } w \in \mathcal{W}' \\ 0, & \text{otherwise.} \end{cases}$$

This value assignment satisfies constraint (3c). We can leverage constrain (3b) to determine the value of R, for any port p that mirrors a flow in $\mathcal{M}_{w^*}$:

$$\sum_{f\in\mathcal{F}_p\cap\mathcal{M}_{w^*}} \alpha_f \leq \frac{r_p}{R} \leq r_p\beta_{\omega_p}, \tag{6}$$

or, equivalently,

$$\frac{r_p}{R} \leq \frac{r_p}{1 + M_{w^*}(\overline{P}^*_f - 1)}, \tag{7}$$

which results in $R \leq 1 + M_{w^*}(\overline{P}^*_f - 1)$. $\square$

We emphasize that this is the worst-case approximation ratio of PMA. As will be shown in Section IV, the approximation ratio achieved by PMA in practice is substantially smaller than its theoretical upper bound.

B. *Flow Mirroring Algorithm (FMA)*

At first glance, SFM resembles two well-studied problems: 1) *Multiple Knapsack* problem where each item can be held only by a subset of available knapsacks [11], and 2) *Scheduling Unrelated Parallel Machines* problem (SUPM), where each job can be assigned to only a subset of available machines [12]. Reduction to [11] can give a 2-approximation algorithm for the number of mirrored flows, but it can not be used to cover all flows with a bounded value for λ. Under the assumption that processing time of each job on a machine is bounded by the optimal value of the problem, the technique in [12] can be used to design a 2-approximation algorithm for SFM, while mirroring all flows. However, to ensure that job assignment restrictions are respected (*i.e.*, each flow is mirrored from a switch that it traverses), we have to set the processing time of those jobs that can not be assigned to a machine to ∞, which violates the assumption needed to establish the proof of the approximation ratio presented in [12]. A general variation of SUPM has been considered in [13] (with assignment restrictions), but their approximation ratio is a function of the maximum and minimum job processing times (*i.e.*, traffic rates) which can be arbitrarily larger than 2. Therefore, in this work, we design an algorithm based on the bisection method, which repeatedly employs an existing result from machine scheduling literature, to find a solution

Alg. 2: Flow Mirroring Algorithm (FMA)

```
1  procedure FMA($\mathcal{W}, \mathcal{F}, \varepsilon$)
2  $\lambda_u \leftarrow \max_{w\in\mathcal{W}} \sum_{f\in\mathcal{F}_w} r_f$, $\lambda_\ell \leftarrow \frac{\sum_{f\in\mathcal{F}} r_f}{|\mathcal{W}|}$
3  if $\lambda_\ell < 1$ then
4      foreach $f \in \mathcal{F}$ do: $r_f \leftarrow \frac{r_f}{\lambda_\ell}$
5  $\lambda_m \leftarrow \lambda_u$
6  while $\lambda_u - \lambda_\ell > \frac{\varepsilon}{2}$ do
7      $\mathcal{J}$, $\mathcal{M}$, $\{t_{j,m}\}$ $\leftarrow$ ToSUPM($\mathcal{F}, \mathcal{W}, \lambda_m + \varepsilon$)
8      $\{y_{j,m}\}$ $\leftarrow$ SolveSUPM($\lambda_m, \mathcal{J}, \mathcal{M}, \{t_{j,m}\}$)
9      if $\sum_{m\in\mathcal{M}} y_{j,m} \geq 1 : \forall j \in \mathcal{J}$ then
10         $\lambda_u \leftarrow \lambda_m$
11         $\{z_{f,w}\} \leftarrow \{y_{j,m}\}$     /* best feasible solution so far */
12     else
13         $\lambda_\ell \leftarrow \lambda_m$
14     $\lambda_m \leftarrow \frac{\lambda_\ell + \lambda_u}{2}$
15 return $\{z_{f,w}\}$
```

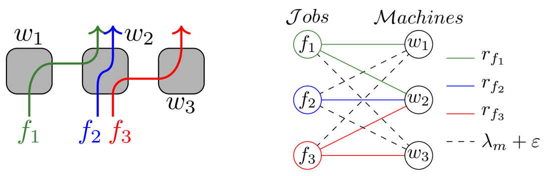


Fig. 3: ToSUPM: Converting an instance of SFM to an instance of SUPM. Each flow f can be considered as a job and each switch w as a machine. If flow f traverses switch w, $t_{f,w}$ is set to the rate of f. Otherwise, $t_{f,w}$ is set to $\lambda_m + \varepsilon$, which is updated in each iteration.

for the SFM problem. We show that our algorithm attains the approximate ratio of $2 + \varepsilon$ for any $\varepsilon > 0$.

Algorithm. The FMA algorithm is described in Alg. 2. It gets an instance of SFM and a threshold ε, which controls the runtime-optimality trade-off, as quantified in Theorem 4. The algorithm starts by computing lower and upper bounds for λ. The upper-bound, λ_u, is computed as the maximum traffic at any switch. The lower-bound, λ_ℓ, is set to the total traffic that should be mirrored divided by the number of switches. On line 3, the algorithm checks if λ_ℓ is less than 1. If so, all flow rates are scaled by the factor $\frac{1}{\lambda_\ell}$. This, ensures that the optimal solution is also greater than 1, which is crucial for proving the approximation ratio of the algorithm. This scaling does not change the solution, because every flow rate is scaled by the same factor. Then, λ_u and λ_ℓ are updated in the while loop by the bisection method until their difference becomes less than or equal to $\frac{\varepsilon}{2}$. In each iteration of the while loop, the method ToSUPM is invoked to convert the current instance of SFM to an instance of SUPM, where $t_{f,w}$ is set to $\lambda_m + \varepsilon$ if flow f does not traverse switch w (see Fig. 3). At this point, SolveSUPM is invoked to check whether or not a machine schedule with makespan λ_m exists (using the technique developed in [14]). If a schedule exists, SolveSUPM constructs a machine schedule with makespan of at most $2\lambda_m$.

Theorem 4. *FMA runs in $O(W^2F^3 \log F \cdot \log(WF/\varepsilon))$ and attains the approximation ratio $2 + \varepsilon$.*

Proof. The proof is divided into two parts, as follows:

Runtime Analysis. The main loop iterates at most $\log \frac{2\Delta}{\varepsilon}$ times, where, we have,

$$\Delta = \max_{w\in\mathcal{W}} \sum_{f\in\mathcal{F}_w} r_f - \frac{1}{W}\sum_{f\in\mathcal{F}} r_f \leq \sum_{w\in\mathcal{W}}\sum_{f\in\mathcal{F}_w} r_f = O(WF) \tag{8}$$

In each iteration of the loop, ToSUPM takes $O(WF)$ time to create the set of processing times $\{t_{j,m}\}$, while SolveSUPM takes $T = O(WF^2 \log F)$ [14] time.

Approximation Analysis. Using the update mechanism in lines 9 to 13, we ensure that the objective remains between λ_u and λ_ℓ. Thus, at the end of the while loop, the distance from the optimal is at most $\frac{\varepsilon}{2}$. Let λ and λ_{FMA} denote the objective achieved by the optimal solution and FMA, respectively. By using SolveSUPM, we know that the objective in the last iteration is at most $2(\lambda + \frac{\varepsilon}{2})$, which yields,

$$\lambda_{\mathsf{FMA}} \leq 2(\lambda + \varepsilon/2) \leq 2\lambda + \varepsilon\lambda = \lambda(2 + \varepsilon), \tag{9}$$

where, the second inequality follows from the fact that $\lambda > 1$, which is guaranteed by the scaling on lines 3 to 4. □

IV. Evaluations

Methodology. We evaluate our system using simulation and Mininet experiments. Simulation experiments are available in [10]. Simulation experiments show the performance and scalability of PMA and FMA ($\varepsilon = 0.02$), through different performance metrics. Mininet experiments are conducted to show real-world performance of SoftTap when used in conjunction with realistic applications, namely IDS and VoIP metering.

Algorithms. In addition to PMA and FMA, we have also implemented the following algorithms for comparison:

- OPT: The optimal solution of ILPs obtained by Gurobi optimizer. Note, we can obtain the optimal solution only for small instances of the problems.
- Planck [4]: A POS-based solution, alternative to PMA, which mirrors all ports of a switch to a mirroring port.
- KPS [6]: An alternative to FMA, presented in Stroboscope [6]. It mirrors a flow on a minimum number of switches along its path to confirm the flow's path compliance.

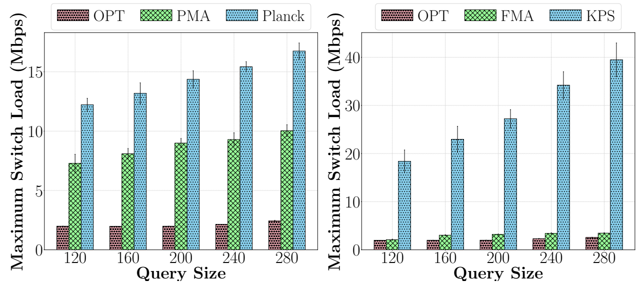
A. Simulation Experiments

Setup. In our simulations, we use two network topologies: 1) 12-pod *FatTree* topology with 180 switches and 72 hosts, 2) *ISP* topology, presented in [15], with 315 switches, 315 hosts and 1944 links. We generate traffic flows between randomly chosen source-destination pairs, with the flow rates randomly selected from a uniform distribution in the interval $[0.6, 2.0]$ Mbps. The reported results are averaged over 10 runs. Simulations are implemented in Python. All computations are carried out on a computer with an Intel® CoreTM i5-7360U processor at 2.3 GHz and 8 GB of RAM.

Metrics. We report the following performance metrics:

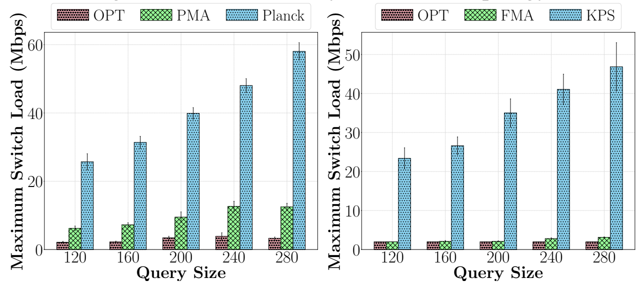
- *Maximum Switch Load*: The maximum mirroring load of any switch in the network.
- *Runtime*: The amount of time it takes to compute the mirroring configuration.
- *Coverage Percentage*: The percentage of flows that are fully mirrored by at least one switch.
- *Coverage Cost*: The network mirroring load divided by the number of flows that are fully mirrored.

These metrics are reported for a varying number of flows in traffic monitoring queries, denoted by *Query Size*.



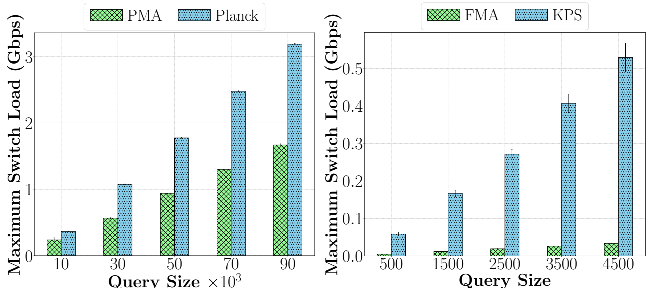
(a) Port mirroring. (b) Flow mirroring.

Fig. 4: Switch load analysis (FatTree topology).



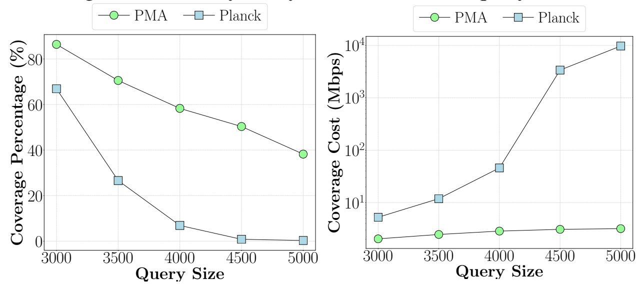
(a) Port mirroring. (b) Flow mirroring.

Fig. 5: Switch load analysis (ISP topology).



(a) Port mirroring. (b) Flow mirroring.

Fig. 6: Scalability analysis with different query sizes.



(a) Coverage Percentage. (b) Coverage Cost (logarithmic).

Fig. 7: Coverage analysis of port mirroring algorithms.

Switch Load Analysis. Fig. 4 and Fig. 5 show the maximum switch load of different algorithms on the FatTree and ISP topologies, respectively. Specifically, from Fig. 4(a) and Fig. 5(a) we can observe two points. First, the approximation ratio of PMA is better than the calculated theoretical ratio. In particular, on the ISP topology, the maximum switch load under PMA is on average 3.1 times the optimal's. Second, regardless of the topology, Planck has the worst performance. Fig. 4(b) and Fig. 5(b) show the maximum switch load of flow mirroring algorithms on the FatTree and ISP topologies, respectively. We see that the approximation ratio of FMA is

TABLE I: Runtime (sec.) on the FatTree topology.

Mirroring Strategy		**Query Size**				
		120	**160**	**200**	**240**	**280**
Port	**PMA**	0.004	0.006	0.005	0.012	0.030
	Optimal	0.036	0.040	0.133	41.29	315.3
Flow	**FMA**	0.234	0.434	0.690	1.045	1.743
	Optimal	0.019	0.040	10.51	120.6	326.1

significantly better than the $(2+\epsilon)$ theoretical ratio. In addition, regardless of the topology, KPS has the worst performance.

Runtime Analysis. Table I shows the comparison of PMA, FMA and the optimal solutions runtime. For both strategies, the optimal algorithms' runtime increases exponentially, making them impractical for any realistically sized network. Observe, when the query size is less than 180, the optimal's runtime is less than FMA's. As the number of flows increases from 180 to 280 (*i.e.*, less than a factor of 2), the optimal's runtime increases to more than 300 sec, 150 times the FMA's.

Scalability Analysis. Note, as we consider large problem instances, we omit the optimal solutions. Additionally, we only present the results on the FatTree topology, as the results on the ISP topology show similar trends. The results are presented in Fig. 6. A few observations are in order. First, PMA outperforms Planck regardless of query size. Specifically, PMA's maximum switch load is on average 45% less than Planck's. Additionally, PMA takes less than 0.95 sec to solve a mirroring request with $90K$ flows. Second, from Fig. 6(b), FMA significantly outperforms the KPS algorithm in terms of the maximum switch load. In particular, the maximum switch load of FMA is on average 92% lower than that of KPS.

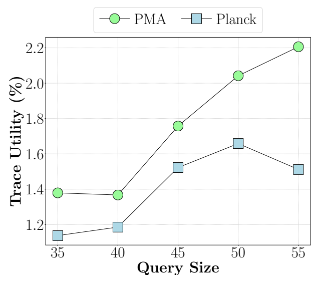
Coverage Analysis. Similar to the scalability analysis, we only present the results for the FatTree topology. For this set of simulations, we assume that switches have a mirroring capacity of 60 Mbps. Fig. 7 shows the comparison between PMA and Planck in terms of coverage percentage and coverage cost. The first observation is that the coverage percentage of Planck is less than the PMA's. Specifically, for more than 4500 flows, Planck has nearly 0% coverage, while PMA has more than 45%. The second observation is that Planck has a higher coverage cost compared to PMA, despite its lower coverage percentage (*e.g.*, Planck has at least three orders of magnitude greater coverage cost than PMA for 4500 flows). This is due to Planck, redundantly, mirroring all ports on all switches.

B. Mininet Experiments

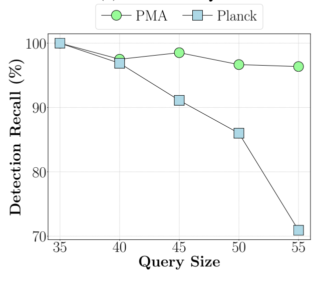
To study SoftTap performance in realistic network settings, we built an IDS and a VoIP metering application on top of SoftTap and studied their performance in Mininet. The IDS uses SoftTap to monitor network traffic for the Zorro malware [16]. The VoIP metering application measures the per host bandwidth consumption of VoIP connections. Note, due to space limitations, we only present PMA's Mininet results.

Applications. We implement the following applications:

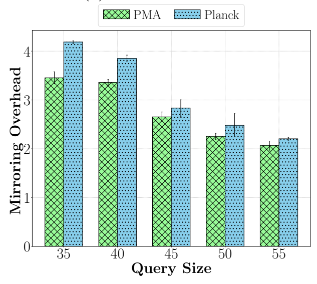
- *IDS Application*: An SDN query-generator module instructs SoftTap to mirror suspicious flows (*i.e.*, Telnet flows on port 23) to a collector node. On the collector node, a DPI-based IDS analyzes the packet payloads and classifies them as contaminated by Zorro malware [16] (*i.e.*, containing "Zorro" string) or benign.
- *VoIP Application*: An SDN query-generator module instructs SoftTap to mirror all VoIP traffic to a collector node, where a VoIP metering application analyzes each host's traffic to determine VoIP usage, and, if necessary, enforces a specific per host bandwidth quota.



(a) Trace utility.

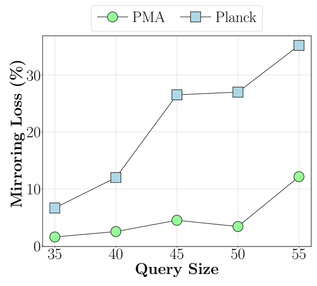


(b) Detection recall.

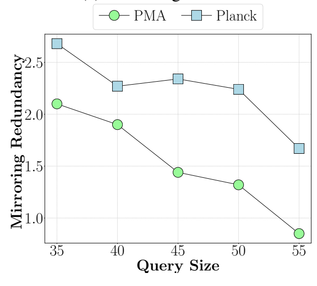


(c) Mirroring overhead.

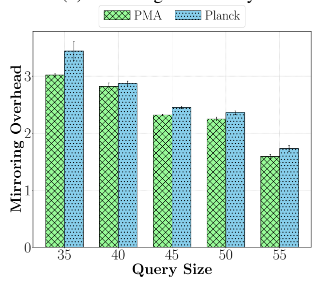
Fig. 8: Mininet measurement results for IDS application.



(a) Mirroring loss %



(b) Mirroring redundancy.



(c) Mirroring overhead.

Fig. 9: Mininet measurement results for VoIP application.

Setup. For both application experiments, we use a 2-pod *FatTree* topology, consisting of 8 hosts and 10 OpenFlow 1.3 switches. We configure ONOS OpenFlow controller as our SDN controller. We generate UDP traffic between randomly selected hosts. The capacity of all mirroring ports is set to be 10 Mbps, chosen to decrease the runtime of the experiments. In Mininet, SoftTap is using PMA as an application on the ONOS controller. In between the runs we increase the query sizes by 5, going from 35 to 55. Finally, the reported results are averaged over 5 runs.

IDS Application Experiments. In this experiment, we generate Telnet traffic, on port 23, containing traces of Zorro attack. The traffic rate is randomly selected between $[25.5, 28.1]$ KB/s. Each flow consists of a random number of packets between 20 and 22, each of size 1278 B. $[4-5]$% of the packets contain "Zorro" string to represent malicious traffic. This is slightly higher than what is assumed in the literature (*i.e.*, 3% [17] for internet traffic), to increase the probability of observing losses of the Zorro packets when comparing the two algorithms. We use the following metrics:

- *Trace Utility*: Number of mirrored Zorro packets divided by the total number of mirrored packets.
- *Detection Recall*: Number of mirrored Zorro packets divided by the total number of transmitted Zorro packets.
- *Mirroring Overhead*: Number of mirrored packets divided by the total number of packets transmitted by mirrored flows. Recall that in port mirroring, some packets are unnecessarily mirrored due to port-level mirroring granularity.

IDS Application Results. Fig. 8 shows the measurement results for the IDS application. Specifically, Fig. 8(a) depicts the trace utility of both PMA and Planck. A few observations are in order. First, for all query sizes, the trace utility of PMA is more than Planck. This suggests that Planck collects more redundant packets per each Zorro packet collected. Second, trace utility has an increasing slope for both algorithms for query sizes below 50. The reason is that only a small 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 mirroring saturation point (*e.g.*, 50 for Planck), after which more losses occur for Zorro packets, as can be seen at 55 flows. This is also observable in Planck's detection recall for a query size of 55 flows.

Fig. 8(b) shows the detection recall of both Planck and PMA. We can observe that, while for small query sizes both algorithms mirror most of the Zorro packets, as the query sizes increase Planck's detection recall drops to nearly 71%, while PMA mirrors 96% of the Zorro packets. This suggests that due to Planck mirroring more redundant packets, the packets that are of interest, *i.e.*, Zorro packets, get lost, because of the capacity limitations on the mirroring ports.

Fig. 8(c) shows the mirroring overhead of both algorithms. As we can observe, the difference between Planck and PMA is larger for smaller query sizes (*i.e.*, in case of 35 queries, Planck is 20% higher than PMA). However, as more flows are introduced to the network this difference diminishes. The reason is that, since for small queries PMA only mirrors small number of ports, the overhead of Planck is substantially higher than PMA. However, as the query sizes grow, PMA introduces more ports to be mirrored and as a result, converges to Planck.

VoIP Application Experiments. To represent the VoIP traffic in Mininet, we generate UDP traffic running on port 5060. The traffic rate is randomly selected between $[38.3, 40.8]$ KB/s. Each flow consists of randomly chosen $30 - 32$ packets, each of size 1278 B. We use the following performance metrics:

- *Mirroring Loss*: The percentage of packets that are not being mirrored on any of the switches.
- *Mirroring Redundancy*: Number of redundant packets divided by the total number of mirrored packets.
- *Mirroring Overhead*: The number of mirrored packets divided by the total number of mirrored flow packets.

VoIP Application Results. Fig. 9 shows the measurement results of the VoIP metering application. In particular, Fig. 9(a) shows the mirroring loss for each algorithm. We can observe that, since for small query sizes most mirroring ports are not saturated, the difference in mirroring loss between the two algorithms is small. However, as the query sizes grow, the difference in mirroring loss starts to grow (*e.g.*, for query size 55, the difference becomes 23%). These results suggest that, similar to the IDS results, PMA achieves better flow coverage by reducing the mirroring redundancy. From Fig. 9(b) we can observe that the mirroring redundancy of both algorithms decreases as the query size increases. This behaviour is expected since for smaller queries, the mirroring port's capacities are not fully saturated, and as such, some of the flows are mirrored on more than one switch. Additionally, we can observe that, for all query sizes, the mirroring redundancy of PMA is smaller than that of Planck. In particular, for a query size of 55, PMA has a redundancy of nearly 0, with 23% less packet loss than Planck. Fig. 9(c) shows the mirroring overhead, which exhibits the same behavior as in the IDS application.

V. Discussion

While SoftTap is able to provide pervasive visibility into an OpenFlow network, the current design has some limitations. First, SoftTap targets applications that require full packet capture and processing, which generally require a single copy of each packet to cope with the processing overhead. Examples of such applications include traffic classification and security functions, *e.g.*, intrusion detection and malware analysis. This means that SoftTap is not targeting applications that require monitoring traffic over multiple switches. Second, current OpenFlow switches do not support partial packet mirroring (*e.g.*, mirroring just the packet header) without modifications [18]. There are, however, emerging programmable switches (*e.g.*, P4 switches [19]) that allow such granular mirroring, although at this time, their cost and availability are limiting their adoption. Third, for some applications, such as network forensics, there are legal requirements on the collected traffic traces (*e.g.*, no loss is tolerated). If SoftTap is used for such applications, the set of flows should be chosen judiciously to avoid any losses. Finally, depending on the routing structure of the network, SFM may require installing additional flow rules in the limited switch flow tables. While this may affect flow table occupancy, most commercial OpenFlow switches (*e.g.*, HP switches) support a list of actions per rule. This feature can be utilized to minimize the SFM flow table overhead by including forwarding and mirroring actions in a single rule.

VI. Related Works

We briefly review existing works on network monitoring that use OpenFlow or are inspired by such.

Flow Statistics Collection. Several works use OpenFlow protocol built-in statistics collection mechanisms to monitor the network [20]–[22]. Specifically, the works [20] and [21] minimize the TCAM usage, while the work [22] focuses on reducing the statistics collection time. Inspired by OpenFlow, the works [23] and [24] use the hash values of flow identifiers to collect per-flow statistics in commodity switches. While these works are limited to flow statistics collection, our work targets applications that require the packets' metadata or payload information.

Traffic Sampling. In traffic sampling, switches are configured to forward 1-in-N packets they receive, to a collector. Currently, OpenFlow specification doesn't support sampling. Therefore, several works focus on designing OpenFlow sampling modules. The work [25] designs such modules to extract flow statistics; while extracting packet payloads using sampling is studied in [26] and [27]. In particular, the work [26] assigns sampling time slots to switches based on historical and active flows and the work [27] allows the controller to determine the sampling rate of each flow.

Traffic Mirroring. Unlike traffic sampling, traffic mirroring provides a full picture of the network. An example of a network monitoring system, based on traffic mirroring, is presented in [4], where the authors demonstrate that port mirroring can be used to extract global network information on a millisecond time scale. Their approach, however, does not make use of SDN capabilities and is solely based on POS with no attempt to optimize flow coverage or mirroring traffic. Examples of per-flow mirroring systems are presented in [6] and [28]. Specifically, in [6] and [28], different from our work, the goal is to mirror a flow on multiple switches in its path, with the former ensuring that the flow follows its expected path and the latter targeting common faults in datacenter networks (*e.g.*, silent packet drops). Akin to flow mirroring, the work [29] proposes a virtual tap to duplicate the traffic between OpenStack based VMs, using the OpenFlow group table feature. The proposed system assumes a pre-computed configuration of the traffic of interest. In contrast, a major contribution of our work is efficiently computing such configurations without VM environment assumptions.

VII. Conclusion

In this paper, we designed and evaluated SoftTap, a traffic monitoring system that utilizes port and flow mirroring capabilities of commodity OpenFlow switches to create a distributed network tap. SoftTap is built upon efficient and fast algorithms for computing mirroring configurations of switches, maximizing the flow coverage. To demonstrate its applicability, we implemented a real-world IDS and a VoIP metering application on top of SoftTap in ONOS controller. Our Mininet experiments show that the detection rate of IDS and the accuracy of metering application were increased by 25% and 23%, respectively. A potential future work is extending SoftTap to programmable switches to increase the flexibility of the system by capturing partial packet payloads or headers.

References

[1] C. Avin, M. Ghobadi, C. Griner, and S. Schmid, "On the complexity of traffic traces and implications," *ACM POMACS*, 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] Juniper Networks, "Understanding port mirroring and analyzers," accessed July 30, 2020. [Online]. Available: https://www.juniper.net/documentation/en_US/junos/topics/concept/port-mirroring-qfx-series-understanding.html

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

[5] Cisco, "Cisco catalyst 6500 series switches," accessed July 30, 2020. [Online]. Available: https://www.cisco.com/c/en/us/support/switches/catalyst-6500-series-switches/series.html

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

[7] S. Sadrhaghighi, M. Dolati, M. Ghaderi, and A. Khonsari, "SoftTap: A software-defined TAP via switch-based traffic mirroring," accessed Dec 9, 2020. [Online]. Available: https://drive.google.com/file/d/1_pv2POLtUdbeR3KB4gYkHVxJRqrtT_rF/view?usp=sharing

[8] 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.

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

[10] Softtap implementation. [Online]. Available: https://github.com/sogand-sadrhaghighi/NetSoft2021

[11] M. Dawande, J. Kalagnanam, P. Keskinocak *et al.*, "Approximation algorithms for the multiple knapsack problem with assignment restrictions," *Journal of Combinatorial Optimization*, vol. 4, no. 2, 2000.

[12] J. K. Lenstra, D. B. Shmoys and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Mathematical Programming*, vol. 46, no. 1, 1990.

[13] D. R. Page, "Approximation algorithms for subclasses of the makespan problem on unrelated parallel machines with restricted processing times," *SOP Transactions on Applied Mathematics*, vol. 2, no. 1, 2015.

[14] D. B. Shmoys and E. Tardos, "Scheduling unrelated machines with costs," in *Proc. ACM SIAM SODA*, Jan. 1993.

[15] DEFO, accessed August 13, 2020. [Online]. Available: https://sites.uclouvain.be/defo/

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

[17] 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.

[18] P. Cheng, F. Ren, R. Shu, and C. Lin., "Catch the whole lot in an action: Rapid precise packet loss notification in data centers," in *Proc. Usenix NSDI*, 2014.

[19] Barefoot Networks, accessed August 13, 2020. [Online]. Available: https://www.barefootnetworks.com/

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

[21] 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.

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

[23] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Proc. USENIX NSDI*, Apr. 2013.

[24] R. B. Basat, G. Einziger, and B. Tayh, "Near optimal network-wide per-flow measurement," in *Proc. IEEE INFOCOM WKSHPS*, Jul. 2020.

[25] Y. Afek, A. Bremler-Barr, S. L. Feibish, and L. Schiff, "Detecting heavy flows in the SDN match and action model," *Computer Networks*, vol. 136, no. 1, 2018.

[26] X. Wang, X. Li, S. Pack *et al.*, "STCS: Spatial-temporal collaborative sampling in flow-aware software defined networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, 2020.

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

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

[29] 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.