



Enhancing Global Network Monitoring with *Magnifier*

Tobias Bühler and Romain Jacob, *ETH Zürich*; Ingmar Poese, *BENOCS*;
Laurent Vanbever, *ETH Zürich*

<https://www.usenix.org/conference/nsdi23/presentation/buhler>

This paper is included in the
Proceedings of the 20th USENIX Symposium on
Networked Systems Design and Implementation.

April 17–19, 2023 • Boston, MA, USA

978-1-939133-33-5

Open access to the Proceedings of the
20th USENIX Symposium on Networked
Systems Design and Implementation
is sponsored by



Enhancing Global Network Monitoring with *Magnifier*

Tobias Bühler*
ETH Zürich

Romain Jacob
ETH Zürich

Ingmar Poesé
BENOCS

Laurent Vanbever
ETH Zürich

Abstract

Monitoring where traffic enters and leaves a network is a routine task for network operators. In order to scale with Tbps of traffic, large Internet Service Providers (ISPs) mainly use traffic sampling for such global monitoring. Sampling either provides a *sparse view* or generates *unreasonable overhead*. While sampling can be tailored and optimized to specific contexts, this coverage–overhead trade-off is unavoidable.

Rather than optimizing sampling, we propose to “magnify” the sampling coverage by complementing it with mirroring. *Magnifier* enhances the global network view using a two-step approach: based on sampling data, it first infers traffic ingress and egress points using a heuristic, then it uses mirroring to validate these inferences efficiently. The key idea behind *Magnifier* is to use *negative* mirroring rules; *i.e.*, monitor where traffic should *not* go. We implement *Magnifier* on commercial routers and demonstrate that it indeed enhances the global network view with negligible traffic overhead. Finally, we observe that monitoring based on our heuristics also allows to detect other events, such as certain failures and DDoS attacks.

1 Introduction

Monitoring transit traffic in Internet Service Provider (ISP) networks is difficult: most operators do not know precisely where traffic enters or leaves their infrastructure. This inability to correlate traffic network-wide makes it hard—if not downright impossible—to detect network-wide problems. As a consequence, operators occasionally learn about routing issues in their own network only via customers calling or opening up tickets.

Operators could use control-plane data to identify where traffic enters and leaves an ISP network; however, that is insufficient. Traffic towards the same destination is often load-balanced between multiple egresses; traffic from the same source prefix often enters via multiple ingresses; importantly, in case of failures or attacks, traffic may not follow the control

plane. Data-plane measurements are thus necessary for accurate flow-level information. Unfortunately, such measurements are hard to scale with the Tbps of traffic crossing ISP networks nowadays.¹ Two common techniques to collect data-plane measurements are packet sampling and traffic mirroring. Both have advantages and disadvantages, making them suboptimal for detecting traffic ingresses and egresses.

Sampling-based approaches such as NetFlow [12] or sFlow [30] provide good coverage at the expense of precision and correctness. Often only a few flows are sampled, and even fewer are sampled at both the ingress and egress. We confirmed this by analyzing a 5-minute slice of NetFlow data (1/1024 sampling rate) extracted from *all* border routers of a Tier-1 ISP in Europe. The slice contains around 40 million flows, where a flow corresponds to packets sharing the same source and destination subnet as well as the same source and destination port. After discarding flows from/to the ISP-owned prefixes, we found that over all sampled transit flows, only 22% are sampled at both their ingress *and* egress, while 41% (resp. 37%) of flows are sampled only at the network ingress (resp. egress). Hence, a traffic matrix such as shown in Fig. 1a locates only 22% of sampled flows; we waste the information from all other sampled flows.

Mirroring-based approaches [33,39] provide high precision and *correctness* at the expense of scalability. Suppose we would mirror all traffic at network border routers. In that case, we could easily enhance packets sharing the same source and destination subnet with their ingresses and egresses, but that would double the network’s traffic. Besides packet mirroring, techniques based on sketches [22] or in-band telemetry [25,26] also excel at gathering precise information but can only do so for a specific share of the traffic.

In this work, we ask ourselves whether we can combine the benefits of sampling and mirroring to mitigate their respective drawbacks. We answer this question positively and present

¹For example, Deutsche Telekom’s IP network has a transit capacity exceeding 30 Tbps and reports up to 10 Tbps of IP traffic on average (3500 PB/month). Source: <https://globalcarrier.telekom.com/business-areas/internet-content/ip-transit>

*The CRediT statement for this work is available in Appendix A.

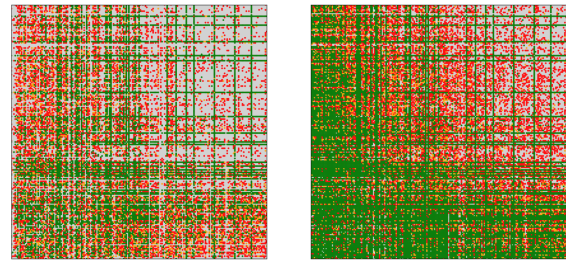
Magnifier, a system that enhances the global network view obtained via sampling using a two-step approach. First, we **infer the ingress and egress of flows** using a heuristic: packets that are “close” in the IP space tend to be routed similarly. This intuition has already been used successfully in other contexts, e.g., to scale heavy-hitter detection using counters [25]. Assuming this holds, we search for the largest IP subnets for which packets appear to enter the network via the same ingress (resp. exit via the same egress) according to the sampling data available. We call these subnets *sentinels*. Fig. 1b shows the sentinel heuristic applied to our Tier-1 NetFlow data, which immediately magnifies the view: not only do we observe more flows for certain ingress–egress pairs (red to green), but we also reveal pairs which were not visible at all in the NetFlow-based matrix (Fig. 1a).

Naturally, this heuristic is not perfect; as sampling is sparse, we may lack important information to correctly identify ingress or egress points, or traffic may simply be rerouted over time. Thus, in a second step, we use **mirroring to validate the inferred ingresses and egresses**. To avoid mirroring a lot of traffic, the key idea is to install mirroring rules where we do *not* expect traffic; i.e., if we infer that subnet s always enters via router R , we install a mirroring rule for s in all ingress routers, except R . In practice, this leads to little mirrored traffic because sentinels are most often correct. Thanks to mirroring, *Magnifier*’s ingress/egress inferences are **guaranteed correct**, which is a key feature of our design and an essential difference from other monitoring tools. Mirrored traffic reveals inference errors or traffic shifts in sub-seconds, which allows *Magnifier* to maintain a correct network view.

The main limitation of *Magnifier* is the number of mirroring rules to install, which, naively, is about one mirroring rule per sentinel on all border routers. For networks forwarding traffic which covers most of the IP space, this vastly exceeds the mirroring capabilities of today’s routers. We thus investigate different strategies to cap the number of rules installed while harnessing most of *Magnifier*’s benefits.

Contributions

- We design *Magnifier*, a network monitoring system that combines sampling with mirroring to enhance the global view on traffic ingresses/egresses (e.g., Fig. 1) while providing correctness guarantees.
- We implement *Magnifier* [3], run it on Cisco Nexus 9300 switches, and demonstrate that *Magnifier* increases the network view coverage with only limited traffic overhead and inference errors using real traffic traces (§ 6).
- We discuss (§ 4.2) and evaluate (§ 6.2.2) different strategies to scale *Magnifier* to large ISP networks by capping the number of mirroring rules required to e.g., the top 1k sentinels while maintaining most of *Magnifier*’s benefits.
- We observe that, even without mirroring, changes in the number of found sentinels create an interesting signal for other monitoring applications, such as failure detection or DDoS protection (§ 6.4).



(a) NetFlow-based matrix. (b) *Magnifier*’s matrix.

Figure 1: By inferring ingress or egress points of sampled flows, *Magnifier* significantly improves the network-wide coverage (Fig. 1b) compared to using sampling only (Fig. 1a). These inferences are guaranteed correct by (the absence of) mirrored packets. Dots represent the number of flows observed from an ingress router (x -axis) to an egress router (y -axis). Grey indicates no flow, red one flow, orange up to 4 flows, and green 5 or more flows. Data source: NetFlow samples from a large Tier-1 ISP.

2 Overview

This section introduces the problem statement (§ 2.1) and *Magnifier*’s main building blocks (§ 2.2). Finally, we illustrate *Magnifier*’s behavior on a simple example (§ 2.3).

2.1 Problem statement

Can we combine the benefits of sampling and mirroring to design an *easy-to-deploy* system that produces *accurate, complete* and *timely* ingress/egress observations in ISP networks, where an “observation” consists of an IP subnet for which we know the correct ingress and egress points?

Ease of deployment The system should be usable in today’s networks with no need for new or specialized hardware.

Accuracy The system should correctly infer subnets’ ingress and egress points.

Completeness The system should generate observations for the largest possible portion of the IP space.

Timeliness The system should update observations in real-time based on newly-collected information; that is, information is processed quicker than it is collected.

2.2 Building blocks

Magnifier extends the coverage of ingress/egress observations using a two-step approach (Fig. 2): based on sampled data, it first *infers* the missing traffic ingress and egress points, then it *validates* these inferences using mirroring.

Inference *Magnifier* cross-correlates the sampled flows to identify IP subnets that are consistently routed via the same ingress or egress routers. For example, suppose we observe *all* sampled flows for a source prefix p enter via ingress router A . *Magnifier* learns that p is an implicit tag for “ingress A ”, which

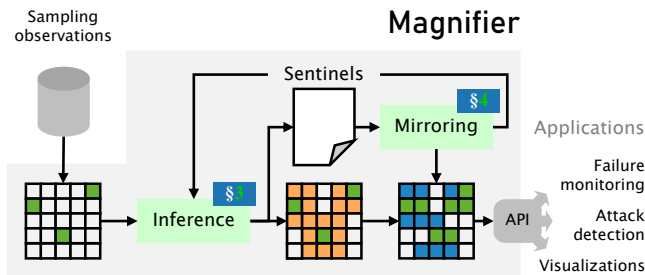


Figure 2: *Magnifier* uses sampled data to infer sentinels that predict IP subnets’ ingress or egress points. *Magnifier* then validates sentinels at runtime using packet mirroring. This way, we can greatly extend the coverage of traffic ingress/egress observations usable by many applications.

enables to map any sampled flow sourced by p as entering via A —even if observed on a different router.

In addition, *Magnifier* leverages the hierarchical nature of the IP space: packets that are “close” in the IP space tend to be routed similarly. Thus, *Magnifier* searches for the largest IP subnets that share the same tags and postulates that all the IPs in these subnets are routed via the same ingresses or egresses. We call these largest subnets *sentinels*. Sentinels significantly extend the coverage of ingress/egress observation (compare Fig. 1). However, these sentinels may be incorrect; sampling may have missed important information, or traffic may simply be re-routed over time. Therefore, *Magnifier* uses mirroring to validate them at runtime.

Validation The key idea behind *Magnifier* is to validate the sentinel inferences using *negative mirroring*; i.e., to deploy mirroring rules where we expect traffic *not* to go. Negative mirroring is efficient because sentinels are often correct in practice; therefore, we mirror only a little traffic. Fundamentally, this guarantees that *Magnifier*’s outputs are correct; all prefixes covered by sentinels either have correctly identified ingress/egress or carry no traffic. Otherwise, traffic is mirrored, which provides additional observations and allows *Magnifier* to maintain and improve its accuracy over time.

Optimization The main limitation of *Magnifier* lies in the number of mirroring rules that can be activated simultaneously on one router. By aggregating subnets together, sentinels effectively limit the number of mirroring rules that must be deployed, but this remains a constraint for large ISP networks. *Magnifier* supports multiple rule deployment strategies to respect a given rule budget per router while optimizing for different properties (e.g., IP space coverage).

2.3 Illustrative example

While mirroring rules generate additional traffic, they are essential to *Magnifier*, illustrated with an example (Fig. 3): p_0 to p_7 are eight /24 prefixes belonging to the same /21; most of the traffic comes from p_0 , with sporadic traffic from other prefixes. Let’s assume that we only sample traffic from p_0 ,

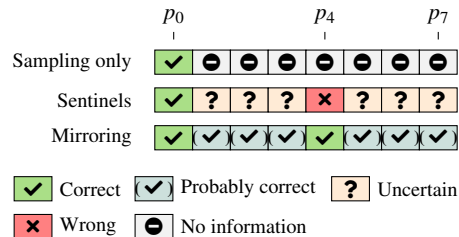


Figure 3: Sampling provides information about the sampled prefixes only. The sentinel inference extends the coverage, but it is uncertain and can make wrong assumptions without any means to detect them. With mirroring, these inferences can be validated, leading to either correct or probable inferences.

which enters at ingress A . One can hypothesize that all p_0 traffic enters via A , but nothing can be said about p_1 to p_7 .

Since no sampled packet contradicts this hypothesis, we infer that all eight /24 enter via A ; the whole /21 is a sentinel for ingress A . This inference is, however, uncertain for seven /24 prefixes without any data. Some traffic from p_4 enters via another ingress, but as long as we do not sample p_4 traffic, we will not detect the wrong inference.

We now use mirroring to validate the sentinel: all routers except A mirror packets for the /21. At first, no packet is mirrored: this indicates either that the sentinel is indeed correct or that there is no traffic *at all* on prefixes that would enter via another ingress. Thus, for the seven prefixes without sampling data, *Magnifier* concludes that the ingress is “probably A ”.

Finally, ingress router B mirrors packets coming from p_4 . *Magnifier* now learns that the /21 sentinel was incorrect. We recompute sentinels, which leads to two /22 sentinels, one for A and one for B . Once the corresponding mirroring rules are installed, *Magnifier* confirms that p_0 and p_4 enter via A and B respectively, and that p_1 to p_3 (p_5 to p_7) probably enter via A (resp. B) as we would otherwise observe mirrored packets.

Conclusion The mirroring rules are essential to validate the sampling-based inferences. Once active, *Magnifier* *guarantees* that the inferences are either correct or that prefixes for which they are wrong do not carry any traffic at all.

3 Ingress & egress identification

In this section, we define the notion of “sentinels” and present an efficient algorithm to find them (§ 3.1). We then discuss sentinel subnet size tradeoffs (§ 3.2) and finally show how *Magnifier* uses these sentinels to match ingress and egress observations in sparsely sampled data (§ 3.3).

3.1 Sentinel search and definition

Definition A sentinel is an *IP subnet* which always enters or leaves the network via *one* network device. Therefore, a sentinel identifies this device whenever a flow from/towards

Algorithm 1 Sentinel search algorithm

```
start ← starting subnet size
end ← ending subnet size
table[IP, device] ← search IPs and network devices
sentinels ← {}
for start ≤ S ≤ end do
  table[IPnew] ← ((IP >> (32 - S)) << (32 - S))
  aggregated ← table.groupby(IPnew)[device]
  result[n] ← numique(aggregated[device])
  sentinels += (result[n] == 1)
  table -= sentinels
end for
return sentinels
```

the IP subnet is observed *somewhere* in the network. As an example, if flows towards 1.2.3.0/24 *only* leave the network via egress router R, we say that 1.2.3.0/24 is an egress sentinel for R. Note that a single sentinel can cover numerous flows.

Types We can distinguish four types of sentinels depending on which IP address we look at (source or destination) and which traffic direction is identified by them (ingress or egress). For example, we can speak about ingress source sentinels. However, unlike specified differently, the remaining sections will only focus on two types of sentinels (i) ingress source sentinels (abbreviated as *ingress sentinels*); and (ii) egress destination sentinels (abbreviated as *egress sentinels*). Currently, *Magnifier* only considers IPv4 sentinels, but *Magnifier* can be applied to the IPv6 address space as well.²

Search algorithm *Magnifier*'s sentinel search algorithm takes flow samples as input. They contain, among others, the identifier of their origin router and the packet source and destination IP addresses. In addition, we define a start and end subnet size over which the algorithm searches for unique subnets to reveal sentinels. Algorithm 1 highlights the main sentinel search steps. The for loop iterates from the start to the end subnet size. In each iteration, we extract the corresponding subnets from the IP addresses of the collected flow samples. All flows belonging to the same subnet are aggregated. If one aggregate only contains samples from the same device, *Magnifier* has found a sentinel, removes the samples from further search iterations, and eventually returns the sentinels.

3.2 Sentinel subnet sizes

Algorithm 1 returns a subnet as a sentinel as soon as it only contains flow samples from one device. However, it is also possible that a smaller subnet would cover all these samples. Fig. 4 shows a simple example. The network forwards traffic from three different /24 subnets. We collect samples from the

²Current IPv6 allocation strategies (e.g., <https://www.ripe.net/publications/docs/ripe-738#5>) are favorable for *Magnifier*. The same AS tends to be allocated large IP blocks that we can use as sentinels.

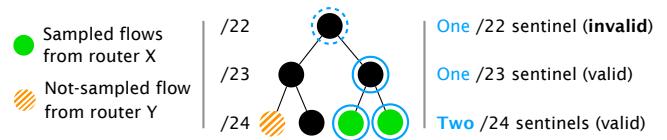


Figure 4: The sentinel amount and coverage depend on the subnet size. The “largest” /22 sentinel is invalid, whereas one /23 sentinel or two /24 sentinels are valid.

two green /24 subnets (ingress router X). Unfortunately, we do *not* sample a packet from the orange /24 subnet (ingress router Y). Algorithm 1 would return the corresponding /22 subnet as an ingress sentinel. After installing corresponding mirroring rules (§ 4.1), *Magnifier* will detect that this sentinel is invalid as it contains flows from two different ingress routers (X and Y). If we would search for smaller subnets, we could either return one valid /23 sentinel or two valid /24 sentinels.

This simple example shows a fundamental tradeoff between the subnet size of found sentinels, the number of sentinels, and their validity. In general, sentinels based on smaller subnets are more likely to be valid but require more mirroring rules to be validated. Experimentally, we find that starting at /16 and ending at /24 yields good performance; starting at bigger sizes does not help as we rarely see such big prefixes in BGP, and it is unlikely that they are unique to a single ingress/egress, while /24 is the smallest globally routed prefix size [35]. As a consequence, the search sizes also influence the number of required validation mirroring rules (§ 4.1) and, therefore, the required router resources.

3.3 Sentinel-based ingress & egress detection

Magnifier uses the found sentinels in two ways. First, for each sentinel type, it tracks the number of sentinels found per device in the network. § 6.4 shows that the number of sentinels is rather stable and changes can reveal unexpected network behavior. The second use case exploits the uniqueness property of sentinels. Let’s assume we have found a (valid) egress sentinel for router X. For each flow towards the sentinel’s subnet—no matter if we observe a corresponding packet on an ingress or another device—we instantaneously know that it will leave the network over X. Similarly, we can identify flow ingresses based on ingress sentinels. *Magnifier* uses this information as input for its ingress/egress observations.

4 Mirroring-based validation

In this section, we explain how *Magnifier* uses traffic mirroring to validate the ingress and egress sentinels produced by the sentinel search algorithm (§ 4.1). To ensure that *Magnifier* can adhere to an operator-given budget of mirroring rules, we introduce two different rule deployment strategies and discuss additional optimization possibilities (§ 4.2).

4.1 Validating found sentinels with mirroring

Magnifier uses *negative* rules to validate the sentinels it finds. Negative rules are placed on devices that are *not* expected to see matching traffic. For instance, to validate that an ingress sentinel belongs to ingress I , *Magnifier* places negative mirroring rules mirroring traffic for the sentinel’s source subnet at all ingress routers except I . The negative mirroring rules will never generate any packets if the sentinel is valid. For invalid sentinels or sentinels which become invalid over time (e.g., a forwarding change), the mirrored packets inform *Magnifier* immediately, and we can update our inferred ingress or egress observations. *Magnifier* then includes the mirrored data in the next sentinel search to find better sentinels.

ACL-based mirroring *Magnifier* relies on existing features such as ERSPAN [9] to mirror traffic from a router. Depending on the router model and capabilities, there are different ways to define the mirrored traffic. We can temporarily mark packets (i.e., [39]) or directly assign a list of subnets to the mirroring session. We use so-called Access Control Lists (ACLs) in all these cases. An ACL is a list of subnets that matches on the forwarded traffic and defines the mirrored traffic. Our “mirroring rules” are entries in an ACL.

Deployment and activation of mirroring rules To deploy its mirroring rules, *Magnifier* interacts with a Python script that runs directly on the router CPU. Via its arguments, *Magnifier* tells the script the mirroring rules to add to the ACL. The script uses e.g., Cisco’s Python API [6] to perform the changes. However, naively adding entries to an ACL that is already connected with an active ERSPAN session can result in unexpected mirroring behavior for at least two reasons: (i) adding new entries takes some time and *Magnifier* cannot predict at which point in time a new mirroring rule is active; (ii) the Ternary Content Addressable Memory (TCAM) region which handles the ACLs/mirroring rules is limited. *Magnifier* handles (i) by pre-deploying inactive mirroring rules and (ii) with techniques explained in § 4.2.

To pre-deploy mirroring rules, *Magnifier* first adds entries to an ACL that is not yet active, i.e., connected with an ERSPAN session. The ACL entries do not yet take space in the TCAM. Once the ACL contains all mirroring rules, another script activates the entire ACL, simultaneously enabling all mirroring rules. In practice, *Magnifier* always iterates between *two* ACLs. One is currently actively mirroring traffic while the other one is populated. Once the second ACL is ready, we switch between them. Due to this deployment strategy, *Magnifier* is not negatively influenced by frequently changing mirroring rules/sentinels (see § D.4) as we always activate a new pre-deployed ACL. Furthermore, this only affects the mirrored traffic; *Magnifier* does not impact the production traffic.

Magnifier can also add a parameter to the scripts which defines how long an ACL should be active. The script will then automatically, i.e., without any external interaction, deactivate the mirroring rules once the defined timeout expires.

4.2 Limiting the amount of mirroring rules

The amount of mirroring rules which a single router can support is limited. Not only is the entire TCAM limited, other features (e.g., traffic engineering) use the same memory space and compete with *Magnifier*’s mirroring rules. For this reason, *Magnifier* supports multiple deployment strategies to adhere to an operator-given budget of mirroring rules. In the following paragraphs, we describe two strategies, but network operators can easily define their own sorting algorithm to control which mirroring rules they deploy first.

Deployment based on sentinel size The first strategy maximizes the sentinel IP space covered by mirroring rules. As each mirroring rule is connected to a sentinel with a specific subnet size, *Magnifier* first orders all sentinels of an ingress or egress based on their subnet size. *Magnifier* then iterates through all network border routers in a round-robin fashion and deploys mirroring rules for the sentinel with the biggest subnet (i.e., the subnet which covers the most IP space). This process ends if either the mirroring rule budget is reached or every mirroring rule is deployed.

Deployment based on sentinel activity The second strategy prioritizes the most active (amount of sampled packets) subnets/sentinels. In other words, we make sure that the inferred ingress or egress points for the most active subnets are validated by mirroring. To this end, *Magnifier* iterates through all border routers in a round-robin fashion and first deploys mirroring rules for the sentinels that are based on the largest number of sampled packets. Random packet sampling—by design—favors large, active flows. Therefore *Magnifier* indirectly deploys mirroring rules for the most active subnets. We evaluate both deployment strategies in § 6.2.2 and § D.3.

Network-specific optimizations *Magnifier* further reduces the amount of mirroring rules using network-specific knowledge. For example, some ISP border routers only connect to customers, and the operator knows exactly which IP addresses belong to them. That limits the possible source addresses entering the ISP over these ingresses (assuming no IP spoofing). On these devices, *Magnifier* does not need to install mirroring rules which belong to IP subnets outside of the customer’s prefixes as we should never receive contradicting traffic.

5 *Magnifier*’s controller

Magnifier’s controller collects and combines the sampled and mirrored packets, finds new sentinels, deploys and activates the corresponding mirroring rules, and uses the newest data to generate accurate and up-to-date ingress/egress observations. This section first explains how the different pieces work together before introducing *Magnifier*’s API. § B contains details about *Magnifier*’s controller placement.

Controller design *Magnifier*’s control flow works in iterations that align to the system component with the longest runtime.

As various tests on real hardware show, this is usually the time it takes to deploy mirroring rules on the routers. Fig. 5 shows the entire process. *Magnifier* uses the collected sampled and mirrored data in iteration $N-2$ (and optionally $N-3$ or older iterations) to compute sentinels and their mirroring rules. Based on the operator given rule budget, *Magnifier* sorts the sentinels according to the deployment strategies in § 4.2. While iteration $N-1$ is running, *Magnifier* pre-deploys the newly computed mirroring rules on the routers. As soon as *Magnifier* deploys the last rule (or once we reach the defined iteration time), it switches to iteration N and activates the pre-deployed mirroring rules after deactivating the old ones. Finally, *Magnifier* uses the collected sampled and mirrored data and the inferred ingress and egress points from the newest sentinels to compute accurate and up-to-date ingress/egress observations.

***Magnifier*'s API** *Magnifier*'s API supports four distinct primitives. First, `enhance_subnet(S)` returns the available ingress and/or egress data related to subnet S . Second `get_interfaces()` returns the relationship between sentinels and their interfaces. *Magnifier* can infer the corresponding interfaces based on sampling data and/or the observed MAC addresses in mirrored packets. Third, `get_matrix()` generates the most up-to-date ingress/egress matrix. Each cell contains the number of observed packets and bytes (reported by sampled packets) for an ingress/egress observation. In addition, a validity bit indicates inferences that are currently validated with mirroring rules. Finally, `get_counts()` outputs the number of found sentinels per device grouped by sentinel type. In § 6.4 we use this API call to detect network problems based on data from a real Tier-1 ISP.

6 Evaluation

This section evaluates *Magnifier* in detail. After introducing the evaluation setup (§ 6.1), we first focus on *Magnifier*'s performance in simulation and on real hardware devices in our lab (§ 6.2). Afterward, we perform a detailed comparison with the Everflow system (§ 6.3) before we highlight that *Magnifier* also works with data from a real ISP (§ 6.4).

6.1 Evaluation setups, datasets, and metrics

Setups We evaluate *Magnifier* in a simulation setup without any resource constraints and a lab setup on real hardware with its corresponding limitations. Our lab setup contains two Cisco Nexus 9300 switches (C93108TC-FX) [10], and a larger Nexus 7009 switch (N7K-C7009) [8]: an older³ but more resourceful model that we use for benchmark experiments.

We illustrate the lab setup in Appendix § D.1. We establish four parallel connections between the two Nexus 9300 switches, each emulating a network ingress. The first switch receives and samples the traffic using sFlow (sampling rate

³Released in 2011 and no longer sold.

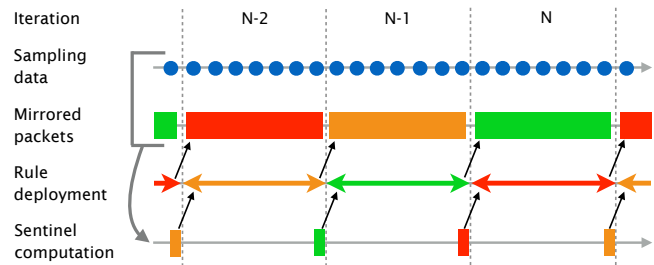


Figure 5: *Magnifier*'s control flow works in iterations based on the mirroring rule deployment time. Rules for sentinels based on $N-2$ are deployed in $N-1$ and active in iteration N .

1/4096⁴). It then forwards to the second switch, which mirrors the traffic according to the configured rules. *Magnifier*'s controller runs on a server and collects sampling and mirroring data. As these switches are limited to 512 mirroring rules, we used a fixed budget of 500 rules per emulated ingress point.⁵ Unless otherwise specified, *Magnifier* prioritizes sentinels according to the activity ordering (§ 4.2).

Our simulation setup is an idealized version of the lab setup. It instantaneously starts mirroring for any prefixes, has unlimited memory space for mirroring rules, and removes rules after their first mirrored packet. The simulator is written in Python and publicly available [3]. Unless specified differently, we always consider *Magnifier*'s iterations to be 60s long. For the N -th sentinel computations, we take sampling and mirroring data from iterations $N-1$ and $N-2$ (see Fig. 5).

We focus on ingress sentinels in the evaluation, *i.e.*, source IP prefixes unique to one ingress. However, the results also apply to egress sentinels. For example, BGP selects the best route for each prefix that is assigned to a single egress. *Magnifier* identifies (part of) these prefixes as egress sentinels (§ 6.4). As a result, one major problem is how to split traffic over different ingress points: *Magnifier*'s performance depends on the assumption that prefixes close in the IP space get routed similarly. We study this dependency using three IP space to ingress mappings: *random* (least favorable for *Magnifier*), *static*, and *permuted* (most favorable). The *random* approach splits the *destination* IP space into n^6 equal slices and assigns one destination IP slice to each ingress point; as a result, source IPs are randomly assigned to one ingress, and this assignment changes frequently. The *static* approach assigns each source /24 prefix *statically* to one random ingress point; however, close IP space is still distributed over different ingresses. Finally, *permuted* splits the *source* IP space into n equal slices and permanently assigns each slice to one of the n ingresses. Then we permute a fixed percentage of /24 source prefixes by moving them to different ingresses. This way, we preserve most of the existing IP structure. A *permuted* 0% assignment results in a perfect mapping for *Magnifier*.

⁴The highest configurable rate on this model; we get the *most* samples.

⁵The four emulated ingresses share the budget. We use TCAM carving [7] to increase the space for our mirroring rules to 2048 (by taking it from other features) to enable the original budget (512) per ingress.

⁶Lab: 1st: 0.0.0.0/2; 2nd: 64.0.0.0/2; 3rd: 128.0.0.0/2; 4th: 192.0.0.0/2

Datasets We use two datasets: one actual packet trace based on CAIDA data and NetFlow samples from a Tier-1 ISP.

The packet trace is based on a 2018 CAIDA trace [4] (1.5 billion packets; one hour long), adjusted to be used in both our setups: (i) We modified the packet MAC addresses to match the lab setup. (ii) We added random payload bytes (removed from CAIDA traces) to match the specified packet sizes. (iii) We moved all destination IPs from 224.0.0.0/4 to a different /4 prefix as this prefix is reserved for IP multicast and led to unexpected packet forwarding on the switches. Replaying the trace at normal speed using `tcpreplay` [16] exhibited anomalies (packet loss and delays). Therefore, we slowed the replay by 10x, resulting in an average of 45k packets per second. Our simulations also use normal and faster speeds to emulate increasing traffic load.

The second dataset contains sampled (rate 1/1024) NetFlow data from all border routers (more than 100) belonging to a large Tier-1 ISP in Europe. The dataset spans over one hour of peak time in the evening of a weekday in 2018. The IP addresses are anonymized by replacing source and destination IPs with the best matching prefix from the full BGP table or the corresponding /24 prefix, whichever is more specific.

Metrics We use the following performance metrics and report the mean over 60 iterations (30 for 2x replay speed).

Coverage Quantifies the amount of traffic for which the ingress point is *correctly* identified. We consider both per-prefix coverage—*i.e.*, the number of /24 covered—and per-packet coverage—*i.e.*, the percentage of covered packets from the input trace.

Mirrored traffic volume Quantifies the overhead in terms of mirrored traffic, as a percentage of the total traffic.

Mirroring rule space Quantifies the number of mirroring rules (ACL entries).

Deployment speed Quantifies how long it takes to either add new mirroring rules or deactivate an installed rule.

6.2 *Magnifier*'s performance

This section details *Magnifier*'s performance. We first show that *Magnifier* greatly enhances the prefix coverage compared to sampling only (up to 80x) and that the ingress points are validated with mirroring rules. This is achieved while mirroring less than 0.3% of traffic. We then analyze methods to limit the number of mirroring rules required. Finally, we confirm that *Magnifier* runs and performs well on real hardware.

6.2.1 Coverage and mirrored traffic volume

We first use our simulation setup to evaluate *Magnifier*'s coverage in different scenarios.

Setup We use our simulation setup and the CAIDA dataset. We vary the trace replay speed (traffic load) and compare the coverage achieved by *Magnifier* by using sampling only. We compute sentinels, install mirroring rules at the start of

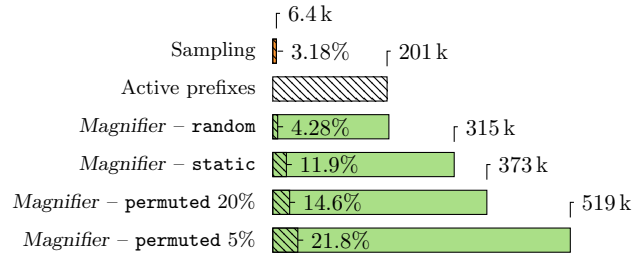


Figure 6: Amount of covered /24 source prefixes by *Magnifier* and sampled data assuming unlimited mirroring resources. 32 border routers, 1/1024 sampling rate, and real replay speed.

each iteration, and compute their coverage values at the end unless mirrored traffic invalidated them. Only these count to the shown coverage values (mean over all iterations).

Per-prefix results Fig. 6 shows the per-prefix coverage with 32 border routers, 1/1024 sampling rate, and real-time replay speed. Sampling covers $\approx 6.4k$ of the active /24 prefixes in the trace, for which we could consider the corresponding ingress point as identified, although without any confirmation that it is valid for all packets belonging to the /24 prefix.

By contrast, we immediately see that *Magnifier* enhances these inferences for all different prefix-to-ingress mappings in at least two ways. First the number of covered /24 prefixes increases to $\approx 200k$ (random), $\approx 315k$ (static), $\approx 370k$ (permuted 20%) and $\approx 520k$ (permuted 5%) respectively. Second, *Magnifier* covers prefixes that are currently active in the CAIDA traces (dashed boxes). The active prefixes increase from $\approx 4\%$ (random) up to $\approx 20\%$ (permuted 5%).

These observations highlight two principles of *Magnifier*: (i) our sentinel heuristic greatly enhances the prefix coverage around sampled data; and (ii) *Magnifier* remains a data-driven system. It has difficulties covering active prefix space that is not sampled using sentinels of reasonable sizes—hence the small % of active prefix coverage (Fig. 6, stripes).

Even more important, for every sentinel validated by mirroring rules, *Magnifier* immediately reports if an ingress inference is no longer valid or enhances new flows (which get active over time) with ingress information. These results are more visible in the per-packet coverage analysis.

Per-packet results Fig. 7 shows the per-packet coverage (left) and mirrored traffic volume (right) with 32 border routers and a 1/1024 sampling rate for varying replay speeds and traffic-to-ingress assignment strategies.

The left plot shows that *Magnifier* achieves an increasing per-packet coverage from $\approx 20\%$ (random) up to $\approx 80\%$ (permuted 5%) which can be surprising given the lower active prefix coverage (Fig. 6). This is explained by the nature of the CAIDA trace, which contains a small number of heavy-hitters and a lot of /24 source prefixes that only carry a few packets. 10% of source /24 IP prefixes account for more than 90% of the packets in 60s trace data (§ D.2). Hence, *Magnifier* often samples and covers these prefixes with sentinels.

These results nicely show the different trade-offs of our assignment strategies. For `random`, the ingress of packets is constantly moving, which makes it difficult to find valid sentinels, while at `permuted 5%`, the assignments are static and *Magnifier* can often find large sentinels which cover a lot of packets. The “real” coverage value is somewhere in between.

To compare, Fig. 7 also contains two sampling-based inferences (without *Magnifier*’s enhancements). The violet line near zero represents a lower bound. We only infer the ingress for the sampled packets. As an upper bound, we consider all the sampled packets in the `permuted 5%` assignment and naively assume that a single sampled packet immediately reveals the ingress point for all other packets belonging to the same source /24 prefix. Note that we can only plot these values because we have the full ground truth data from the CAIDA trace. An operator would *not* know if these inferences are correct. For bigger traffic loads, the upper bound is better than *Magnifier*’s per-packet coverage. This is due to invalidated sentinels: *Magnifier* searches for large sentinels based on sampled packets, likely to come from heavy-hitter flows. Suppose a non-sampled /24 prefix covered by that sentinel is mapped to a different ingress and carries even only one packet. In that case, it triggers a mirroring rule and invalidates the entire sentinel, and *Magnifier* loses all its coverage.

The right plot in Fig. 7 shows a low percentage of mirrored traffic for all assignment strategies (between 0.3% and 0.01%). As expected, a `random` assignment often leads to invalid sentinels and thus more mirrored packets.

Finally, we observe that larger traffic loads yield better performance. With more traffic, *Magnifier* collects more samples per iteration, computes more accurate sentinels, and achieves better coverage and less mirrored traffic. We show additional results in § D.3: Performance decreases with the number of routers in the `random` case—as the previously discussed mapping strategy gets worse—but remains nearly unaffected in the `static` and `permuted` cases (Fig. 15). Moreover, more sampled packets (higher sampling rate) result in better input data and thus performance improvements (Fig. 16).

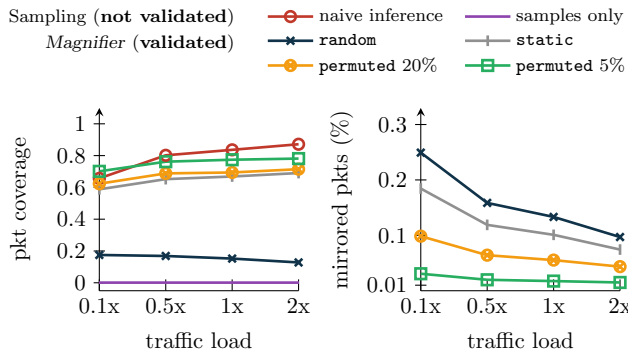


Figure 7: Amount of covered packets and mirrored traffic for different assignment strategies and inferences based on sampled packets only. 32 border routers and 1/1024 sampling rate.

Conclusion *Magnifier* greatly increases the per-prefix coverage compared to sampling (up to 80×) while validating all ingress points with mirroring rules. *Magnifier* achieves this while mirroring less than 0.3% of traffic and translates into a per-packet coverage of up to 80%.

6.2.2 Impact of limited mirroring budget

We now show that *Magnifier* also performs well when limiting the number of mirroring rules installed per router.

Setup We use the same setup as before and compare the coverage achieved by *Magnifier* with different bounds on the number of validated sentinels for two sentinel selection strategies (§ 4.2): `activity` (covering most sampled packets) & `size` (largest subnet sizes). The number of validated sentinels is an upper bound for the number of mirroring rules required per router; in the worst case, all sentinels belong to one router, resulting in one rule per sentinel on *all* the other routers.

Results Fig. 8 compares *Magnifier*’s per-packet coverage achieved with different numbers of validated sentinels: 500, 1k, 5k and unlimited; using the same settings as in Fig. 7. We show results for the `permuted 5%` (left) and `static` (right) assignment strategy, additional plots can be found in § D.3.

More validated sentinels achieve a higher coverage and generate more mirrored traffic. The top `size` sentinels have the highest chance of being invalidated by un-sampled prefixes and generate more traffic than their `activity` counterparts.

The `activity` selection achieves much better per-packet coverage than `size`, which is expected since `activity` prioritizes sentinels covering the most active prefixes. As the trace contains many heavy-hitters (previous discussion), even as few as 500 sentinels are enough to yield good packet coverage. Note that for 0.1× traffic load in the top left plot, the number of sentinels is smaller than 5000, resulting in the same coverage

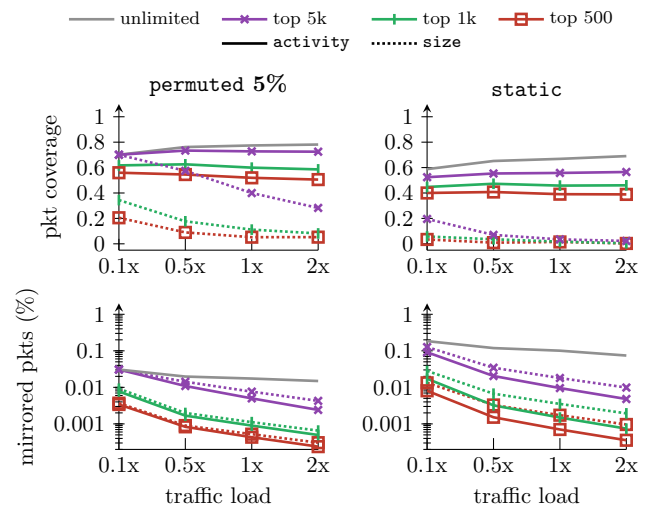


Figure 8: Coverage and mirrored traffic amount for different top sentinels ordered by activity or size.

Technique	Covered /24 prefixes
Sampling	6.4k
[activity] top 500 static	4.1k
[activity] top 500 permuted 5%	29.4k
[size] top 500 static	9k
[size] top 500 permuted 5%	47.3k

Table 1: Covered /24 source prefixes by *Magnifier* and sampling only considering the top 500 sentinels (*activity* and *size* ordering) in the *static* and *permuted 5%* assignments.

values for *activity* and *size*. We also see that the *activity* coverage values remain more or less constant even if the traffic load increases which is not the case for *size*.

The *size* selection favors sentinels centered around sparse samples in a relatively empty prefix space; this results in a low per-packet coverage (Fig. 8), but in a large per-prefix coverage (Table 1). As we can see, *activity* really prioritizes sentinels around a few selected prefixes resulting in fewer covered prefixes than sampling (*static* assignment). However, a *size* ordering can easily exceed the number of covered prefixes by up to seven times, even if we only take the top 500.

Conclusion *Magnifier*'s performance is maintained when limiting the deployed mirroring rules. The top 1k *activity* sentinels are sufficient to achieve up to $\approx 50\%$ packet coverage while mirroring less than 0.05% of traffic (*static* case).

6.2.3 Comparison with the lab setup

We now show that our hardware-based results match the simulation ones, validating *Magnifier*'s performance in practice.

Setup We use our lab setup (Nexus 9300 switches), which has two main differences from the simulation: we only have 500 mirroring rules per router, and there are delays to install and delete rules. We use the *random* assignment strategy and fill the 500 mirroring rules with the top 500 *activity* sentinels. For a fair comparison with the simulation, we consider iteration times of 60s. *Magnifier* needs ≈ 20 s to install all mirroring rules and then activate them. Afterward, we start to delete the rules which mirror packets. We compare this with the corresponding simulation results *i.e.*, 4 border routers, 1/4096 sampling rate, and $0.1 \times$ replay speed.

Results Fig. 9 shows the amount of covered /24 prefixes for sampled data only and the validated sentinels. We first notice that the coverage for sampled packets in our simulation (297) is slightly higher than on the switches (268). This can be explained by the different setups. All four ingress routers run on one Nexus 9300 (§ 6.1), which is not transparent to the sFlow-based sampling unit. Therefore, we get random packet sampling over all the traffic while the simulation performs packet sampling for each ingress device independently. This also shows in the achieved coverage values using the top 500 *activity* sentinels: ≈ 10.4 k prefixes in the simulation, ≈ 8.7 k prefixes on the hardware. We also have to consider that

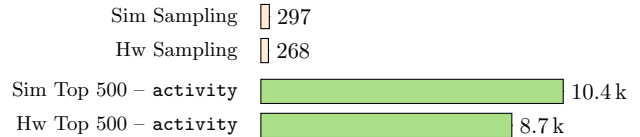


Figure 9: Covered /24 source prefixes by *Magnifier* and sampled data in simulations and on Nexus 9300 switches. *random* assignment, 1/4096 sampling rate, and $0.1 \times$ replay speed.

we need additional time to deploy the mirroring rules on the switch. Thus, a few more sentinels get invalidated compared to the simulations; and no longer count to the coverage values. The packet coverage values (not shown) are also comparable between the simulation (17.0%) and the hardware (16.1%).

Finally, we evaluate the percentage of mirrored traffic. We notice that the deactivation of active mirroring rules works well. In the worst case (active rules mirror for the entire 60s), *Magnifier* would mirror 2.3% of the overall traffic. This value is reduced to 1.4% if we start to deactivate rules. However, we are still above the optimal simulation results (less than 0.1%), where we can deactivate mirroring instantaneously.

Conclusion The hardware results closely follow our simulations regarding achieved coverage. However, *Magnifier* needs more time to install and deactivate mirroring rules, resulting in additional mirrored packets. To reduce the amount of mirrored traffic, operators can use existing hardware features to rate-limit the mirrored traffic on the switch [11].

6.2.4 Micro-benchmarks

We now perform micro-benchmarks on the hardware switches to assess (i) how many mirroring rules each device supports, how long it takes to (ii) deploy them, and (iii) deactivate them.

Results—Mirroring rule space With the default configuration of the Nexus 9300 switch, we can deploy 512 rules and up to 2048 in the current lab setup (TCAM carving [7]). On the Nexus 7009, we can deploy ≈ 32 k rules using one TCAM bank and ≈ 128 k rules if we chain all four TCAM banks together.

Results—Rule deployment time We measure how long it takes to deploy a set of mirroring rules on our two devices. During our tests, we realized that deploying the rules over multiple parallel sessions between *Magnifier* and the switches is beneficial. Four parallel sessions worked well for us. Table 2 shows the mean deployment times over ten measurements each. They include the session setup and round-trip time between *Magnifier* and switch. We see that the deployment time is not strictly linear in the number of rules. We conjecture that caches and buffers allow deploying a small number of rules quickly, but this no longer works for larger number of rules. We also see that the (newer) Nexus 9300 switch needs less time than the Nexus 7009. We can deploy 2000 rules in ≈ 18 s, which matches our observations in § 6.2.3 (500 rules for 4 ingresses on one device). We expect that the rule deployment time will continue to decrease with more powerful/newer devices.

Number of rules	100	500	1000	2000	5000
Nexus 9300	3.5s	5.5s	8.4s	17.8s	112s
Nexus 7009	2.6s	7.5s	21.7s	74.9s	475s

Table 2: Mirroring rule deployment times.

Note that even if we cannot activate 5k+ rules on the Nexus 9300 switch with the current TCAM carving (§ 6.1), we can still deploy them. Overall, these results confirm *Magnifier*’s design (Fig. 5) which aligns the iterations to the rule deployment time. Especially as the sentinel computation time is negligible (≈ 1 s on the CAIDA trace.)

Results—Rule deactivation time We finally measure the rule deactivation time on the Nexus 9300 switch. We generate 100 ping probes per second and deactivate a matching mirroring rule as soon as we receive the first mirrored probe. The deactivation time is the difference between the timestamp of the first and last mirrored probe, including the round-trip (≈ 0.5 ms) and session setup time between switch and controller. This setup is representative of an ISP deployment where close, dedicated control servers could quickly deactivate rules (§ B). We repeated the experiment ten times. The mean deactivation time is 1.65 s (min: 1.62s, max: 1.73s). In the worst case, we would receive a burst of traffic for ≈ 1.7 s. The amount of mirrored packets can be further reduced by rate-limiting the mirrored traffic directly on the switch; we expect this would *not* affect *Magnifier*’s performance, as a single mirrored packet is enough to invalidate a given sentinel.

Conclusion Our tests show that hardware switches can contain thousands to tens of thousands of mirroring rules, which is more than sufficient for *Magnifier*. Mirroring rules can be deactivated quickly (≈ 1.7 s), which limits the risk of bursts of mirrored traffic. The rule deployment is the most time-consuming operation (≈ 20 s for 2k rules). As a result, we can adjust the number of deployable mirroring rules (number of validated sentinels) by changing *Magnifier*’s iteration time.

6.3 Comparison with Everflow

We compare *Magnifier* with Everflow [45] which is a monitoring tool designed for debugging datacenter networks. Like *Magnifier*, Everflow randomly samples packets (using mirroring rules). In addition, it also mirrors *all* TCP SYN, FIN, and RST packets. As far as we know, the Everflow code is not available. Therefore, we reimplemented the relevant features and integrated them into our simulation framework (see § C).

Setup We use our simulation setup and the CAIDA dataset, 32 border routers, a sampling rate of 1/1024 (for both systems), and we vary the trace replay speed. We compare the performance of *Magnifier*, and Everflow on the *static* and *permuted 5%* traffic-to-ingress mappings.

Results Fig. 10 shows the per-packet coverage and mirrored traffic of both systems. We consider three different approaches: (i) “Everflow sampling only”, where we rely only on Everflow’s

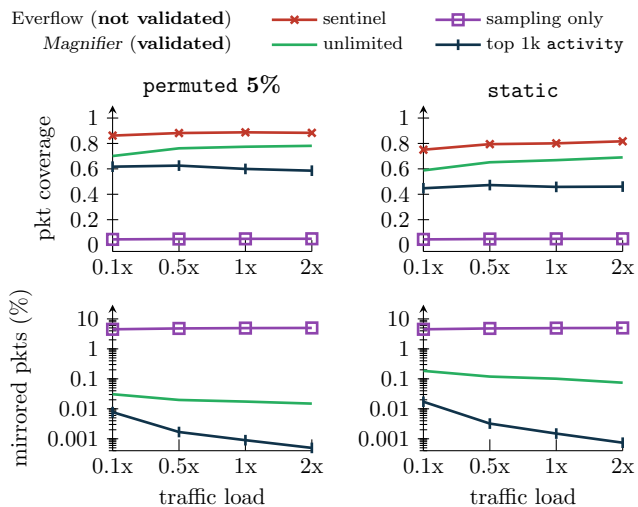


Figure 10: Comparison of coverage and mirrored traffic for *Magnifier* and Everflow under different traffic loads.

sampled packets to compute the ingress points; (ii) “Everflow sentinel”, where we use the sentinel idea on top of Everflow’s sampled packets; and (iii) “*Magnifier* unlimited” and “top 1k activity”, where we report *Magnifier*’s coverage for all and the 1k most active sentinels.

We first look at the coverage values (top plots in Fig. 10). Everflow’s sentinel approach shows the best—although not validated—coverage values with up to 88% in the permuted 5% case. This is due to Everflow’s sampled TCP flag packets. We do not reach 100% as traffic in some /24 prefixes is neither randomly sampled nor does it contain any TCP flags. These prefixes can invalidate found sentinels. Note again that the ground-truth data from the CAIDA trace allows us to compute these values. Everflow does not deploy any validation mirroring rules and does *not* know about the sentinel’s validity. *Magnifier* follows closely with $\approx 80\%$ (unlimited) and $\approx 60\%$ (top 1k) coverage as we only have randomly sampled packets as input. Despite that, *Magnifier* manages to reach good coverage values, with validation from mirroring. Both systems’ coverage values decrease in the more difficult *static* approach. For completeness, we also show Everflow’s coverage if we only consider the sampled packets. This results in a poor packet coverage, although on a higher level than the “sampling only” line in Fig. 7 given that Everflow additionally also samples all TCP packets with SYN, FIN, and RST flags. These coverage values are constant between both assignment strategies as we observe the same TCP flag packets and roughly the same random samples.

Everflow’s increased coverage has a high cost in the amount of mirrored traffic (lower plots in Fig. 10). Everflow generates the randomly sampled and TCP flag packets as mirrored traffic by design. *Magnifier* however, only generates targeted mirrored packets to validate found sentinels. If a sentinel is valid, it does not mirror any traffic. This is visible in the corresponding fraction of mirrored traffic.

Everflow constantly mirrors $\approx 5\%$ of all traffic while *Magnifier* is more than one magnitude lower ($\approx 0.1\%$ of all traffic at real-time replay speed in the static case). This value decreases even further if we only consider the most active sentinels. We again observe that Everflow mirrors roughly the same amount of packets for both assignment strategies.

Conclusion Everflow yields better coverage but generates more mirrored traffic, which is more than one order of magnitude higher than *Magnifier*. Unlike Everflow, *Magnifier* validates the inferred ingress points, informing the controller as soon as a sentinel is no longer valid. In contrast, Everflow might need to wait a long time before receiving a mirrored packet indicative of an ingress point change, especially for long-running flows that do not often have TCP flags. In terms of mirroring rules, Everflow only needs around 20 of them [45]. *Magnifier* needs more mirroring rules but also uses them for *validation*—something that Everflow cannot achieve.

6.4 Sentinels in Tier-1 dataset

We now validate the practicality and benefits of sentinel-based monitoring by evaluating *Magnifier* on Tier-1 ISP data.

6.4.1 Existence of sentinels

Setup We divide our Tier-1 dataset into 30s slices over which we compute sentinels and report the number of found ingress and egress sentinels. We only have sampling data available. Thus, we can only *approximate* the number of sentinels that would be found if *Magnifier* was deployed with mirroring.

Results We find a median of 145k egress sentinels and a median of 174k ingress sentinels. The lower and upper quartiles are within 1.4k around the median values in both cases.

We observe that we find more ingress than egress sentinels. This results from the typical forwarding behavior observed in an ISP: traffic from each of the ISP customers, which own specific prefixes, tends to enter via a single ingress point, which leads to a high number of ingress sentinels. At the same time, most ingress traffic goes to few popular destinations, which leads to few egress sentinels. We also see that the number of (ingress and egress) sentinels is stable over time, as shown by the small quartile ranges.

Conclusion We confirm that we find sentinels based on real sampling data from a Tier-1 ISP network. Furthermore, the number of sentinels is stable over time; this suggests that large changes in sentinel numbers can be used as a signal to detect various network events, which we discuss next.

6.4.2 Per-device sentinel changes

Setup We divide our Tier-1 dataset into 30s slices over which we compute sentinels using *Magnifier*, focus on the number of sentinels found per border router, and search for large changes in the number of sentinels over consecutive slices.

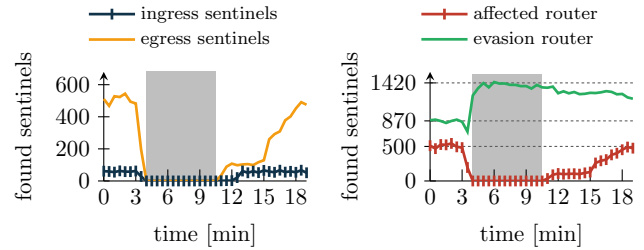


Figure 11: A temporary router outage (gray block) decreases the number of found sentinels (left) while we see similar increases on a close router (right).

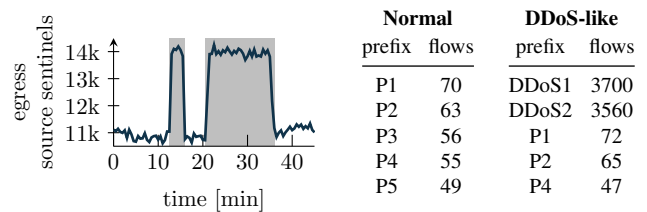


Figure 12: A sudden burst of egress source sentinels (left) is likely to result from a DDoS-like event (right).

Results Fig. 11 shows the number of sentinels found following a single border router outage. As expected, *Magnifier* finds no more sentinels for the affected router. More interestingly, *Magnifier* also detects where the affected traffic was re-routed during the outage, as shown on the right: the number of egress sentinels of a geographically-close router increases shortly after and closely matches the number of lost sentinels.

Fig. 12 (left) shows a router with a burst of *egress source* sentinels (traffic from a given subnet exiting via a unique egress point) while no other router shows a matching decrease. Thus, we observe a sudden burst of packets from “new” source IPs towards a few destinations (table, right), indicating a possible Distributed Denial of Service (DDoS) attack. During this event, the egress traffic volume increased by less than 8%, which is less pronounced than the clear increase in sentinels. *Magnifier* also identifies the ingress of more than 75% of the “attack” flows via their ingress sentinels. Existing volumetric DDoS detection systems could use this information to block the DDoS traffic at the network ingress.

Conclusion Changes in the number of found sentinels reveal interesting network events. Operators could analyze the collected sampling data this way, even if they do not have the resources to deploy mirroring. With mirroring, *Magnifier* detects such changes in sub-seconds, long before similar events are visible in sampled flow data or SNMP counters.

7 Related work

Sampling-based network monitoring Many systems use NetFlow [12], J-Flow [27], sFlow [30] or related flow extraction tools for network measurements. Sampling suffers from a fundamental trade-off between coverage and accuracy. For

example, Teixeira et al. [37] use NetFlow data to detect egress changes due to BGP hot-potato routing but are limited by the collected ten-minute bins. In addition, Cunha et al. [13] uncover measurement artifacts in two J-Flow implementations.

Consequently, several works aim to improve sampling by optimizing the collection process. Estan et al. [14] propose router software updates to dynamically adapt the NetFlow sampling rate depending on the available traffic and memory amount. FlowRadar [21] uses flow sets to count flow observations in multiple array cells, then combines and decodes these counters centrally. Similarly, Flowyager [34] introduces Flowtrees, an efficient data structure to store flow information. These approaches are all limited by the sampling information. A key difference of *Magnifier* is to further improve the network visibility by leveraging mirrored traffic.

Mirroring-based systems Several monitoring systems use mirroring, which provides accurate visibility over a subset of the traffic, flows, or devices. Stroboscope [39] supports query-based monitoring under a strict budget of mirrored traffic. Everflow [45] provides the possibility to mirror some packets of every flow, e.g., by mirroring packets with special TCP flags or debug header bits. Planck [33] takes a radical approach and mirrors all traffic over a single router port, which provides detailed insights but can also overload the network devices. Mirroring has also been used for troubleshooting [41], SDN monitoring [1], on in-network analysis [38, 42].

Mirroring suffers from three problems: (i) the flows of interest must be known in advance; (ii) it is limited by the routers' mirroring capacity, and (iii) it generates a potentially high volume of traffic. *Magnifier* mitigates these problems by leveraging sampling to derive the mirroring rules to deploy and uses negative mirroring to limit the traffic overhead.

In-network monitoring There has been many recent proposals for performing in-network monitoring based on in-band telemetry (e.g., [2, 18, 25, 26, 31]) or sketches (e.g., [5, 19, 22, 43, 44]). Both approaches boil down to implementing highly efficient data structures to gather traffic statistics, e.g., packet counts. The main limitation is that these approaches depend on software-defined or P4-programmable hardware, which is not commonly deployed in ISP networks nowadays. Moreover, these approaches provide precise information, but over specific queries only; setting and collecting counters to track ingress and egress points of an arbitrarily large number of IP prefixes is hard to scale. Negative mirroring addresses this: while *Magnifier*'s inferences are correct, there is no traffic nor compute overhead—only TCAM usage. Packets that do get mirrored provide exact information—i.e., source and destination IP—which allows for quick and precise reactions.

Detection of ingress/egress *Magnifier* is designed to detect traffic ingress/egress points, which has been previously studied: Feldmann et al. [15] provide foundation work for detecting different flow types in ISP networks as well as the ingress and egress of observed flows. To achieve good results, they

need per-flow measurements on the ingress and up-to-date forwarding tables of the routers in the network, which are both costly to obtain. Mahajan et al. [23] use algorithms similar to our sentinel idea to build so-called “aggregates”, a collection of packets with a common property, to free congested links. However, it is unclear how they extract the traffic to build the aggregates or validate their assumptions. Peng et al. [29] run a change point detection algorithm to detect changes in the number of new IP addresses, which is a good metric to detect (the ingress) of DDoS attacks. Most of these systems lack the global ingress/egress view that *Magnifier* provides.

Traffic matrix estimation Soule et al. [36] compare different techniques based on bias and variance properties. They show that direct measurements are required to reduce bias, which is an expensive process. Papagiannaki et al. [28] observe that the node fanout, e.g., how traffic from an ingress is distributed towards different egresses, is stable over time. *Magnifier* confirms and leverages this behavior: sentinels are stable over time, which creates a valuable monitoring signal (§ 6.4). With mirroring, *Magnifier* also quickly detects changes and updates its traffic matrix estimation. OpenTM [40] uses a different approach, based on active polling of every source-destination pair, which is very precise but does not scale to large networks. Malboubi et al. [24] addresses the special case of SDN networks, which limits the system's applicability. Pingmesh [17] frequently generates pings to compute latency matrices. By contrast, *Magnifier* does not require active measurements and runs on traditional routers, which makes it easy to deploy.

Monitoring frameworks Several monitoring frameworks support rich sets of queries, e.g., [20, 32, 42]. In particular, Flowyager [34] is similar to *Magnifier* as it builds primarily on sampling. The downside of these frameworks is their complexity and extensive storage and computational resource requirements. By contrast, *Magnifier* focuses on performing ingress/egress monitoring with little overhead.

8 Conclusion

Precise observations of traffic ingress and egress points are difficult to generate in large ISP networks. In this paper, we show how *Magnifier* combines the global view of sparsely-sampled flow observations with precise, targeted information from mirrored traffic. *Magnifier* enhances observed flows with validated ingress and egress points and scales to the largest ISP networks while only generating a small amount of mirrored traffic. *Magnifier*'s outputs can also help monitor outages or detect volumetric DDoS attacks.

Acknowledgments

We would like to thank Paul Stark and Derk-Jan Valenkamp for their great help with the hardware-based evaluation. Many thanks as well to Tibor Schneider for his support with TikZ.

References

- [1] Ran Ben Basat, Xiaoqi Chen, Gil Einziger, and Ori Rottenstreich. Designing Heavy-Hitter Detection Algorithms for Programmable Switches. *IEEE/ACM Transactions on Networking*, 2020. doi:10.1109/TNET.2020.2982739.
- [2] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. PINT: Probabilistic In-band Network Telemetry. SIGCOMM'20. ACM, 2020. doi:10.1145/3387514.3405894.
- [3] Tobias Bühler. Magnifier GitHub repository, 2022. <https://github.com/nsg-ethz/Magnifier> Accessed: 2022-08-01.
- [4] CAIDA. The CAIDA UCSD anonymized internet traces 2018. https://www.caida.org/catalog/datasets/passive_dataset Accessed: 2022-08-01.
- [5] Xiaoqi Chen, Shir Landau-Feibish, Mark Braverman, and Jennifer Rexford. BeauCoup: Answering Many Network Traffic Queries, One Memory Update at a Time. SIGCOMM'20. ACM, 2020. doi:10.1145/3387514.3405865.
- [6] Cisco Systems. Cisco Python API. <https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus3600/sw/93x/programmability/guide/b-cisco-nexus-3600-nx-os-programmability-guide-93x/m-3600-python-api-93x.pdf> Accessed: 2022-08-01.
- [7] Cisco Systems. Nexus 9000 TCAM Carving, 2016. <https://www.cisco.com/c/en/us/support/docs/switches/nexus-9000-series-switches/119032-nexus9k-tcam-00.html> Accessed: 2022-08-01.
- [8] Cisco Systems. Cisco Nexus 7000, 2021. https://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-series-switches/Data_Sheet_C78-437762.html Accessed: 2022-08-01.
- [9] Cisco Systems. Configuring ERSPAN, 2021. https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus7000/sw/system-management/guide/b_Cisco_Nexus_7000_Series_NX-OS_System_Management_Configuration_Guide/b_Cisco_Nexus_7000_Series_NX-OS_System_Management_Configuration_Guide_chapter_010101.html Accessed: 2022-08-01.
- [10] Cisco Systems. Cisco Nexus 9300-FX, 2022. <https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/datasheet-c78-742284.html> Accessed: 2022-08-01.
- [11] Cisco Systems. Configuring Rate Limits, 2022. https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus3000/sw/security/92x/b-cisco-nexus-3000-nx-os-security-configuration-guide-92x/b-cisco-nexus-3000-nx-os-security-configuration-guide-92x_chapter_010000.html Accessed: 2022-08-01.
- [12] Benoit Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), 2004. <https://datatracker.ietf.org/doc/html/rfc3954> Accessed: 2022-08-01.
- [13] Ítalo Cunha, Fernando Silveira, Ricardo Oliveira, Renata Teixeira, and Christophe Diot. Uncovering Artifacts of Flow Measurement Tools. In *International Conference on Passive and Active Network Measurement*. Springer, 2009. doi:10.1007/978-3-642-00975-4_19.
- [14] Cristian Estan, Ken Keys, David Moore, and George Varghese. Building a Better NetFlow. *ACM SIGCOMM CCR*, 2004. doi:10.1145/1030194.1015495.
- [15] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, and Fred True. Deriving Traffic Demands for Operational IP Networks: Methodology and Experience. *ACM SIGCOMM CCR*, 2000. doi:10.1145/347059.347554.
- [16] Fred Klassen. Tcpreplay - Pcap editing and replaying utilities, 2020. <https://tcpreplay.appneta.com/> Accessed: 2022-08-01.
- [17] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi-Wei Lin, and Varugis Kurien. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. SIGCOMM'15. ACM, 2015. doi:10.1145/2785956.2787496.
- [18] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-Driven Streaming Network Telemetry. SIGCOMM'18. ACM, 2018. doi:10.1145/3230543.3230555.
- [19] Qun Huang, Xin Jin, Patrick P. C. Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang. SketchVisor: Robust Network Measurement for Software Packet Processing. SIGCOMM'17. ACM, 2017. doi:10.1145/3098822.3098831.
- [20] Kentik. Network Observability, Performance and Security, 2022. <https://www.kentik.com/> Accessed: 2022-08-01.







































- [21] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. FlowRadar: A Better NetFlow for Data Centers. In *NSDI'16*. USENIX Association, 2016. <https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/li-yuliang> Accessed: 2022-08-01.
- [22] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. *SIGCOMM'16*. ACM, 2016. doi:10.1145/2934872.2934906.
- [23] Ratul Mahajan, Steven M Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Controlling High Bandwidth Aggregates in the Network. *ACM SIGCOMM CCR*, 2002. doi:10.1145/571697.571724.
- [24] Mehdi Malboubi, Shu-Ming Peng, Puneet Sharma, and Chen-Nee Chuah. A Learning-Based Measurement Framework for Traffic Matrix Inference in Software Defined Networks. *Computers & Electrical Engineering*, 2018. doi:10.1016/j.compeleceng.2017.11.020.
- [25] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. DREAM: Dynamic Resource Allocation for Software-Defined Measurement. *SIGCOMM'14*. ACM, 2014. doi:10.1145/2619239.2626291.
- [26] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. Language-Directed Hardware Design for Network Performance Monitoring. *SIGCOMM'17*. ACM, 2017. doi:10.1145/3098822.3098829.
- [27] Juniper Networks. Configure J-Flow, 2017. https://supportportal.juniper.net/s/article/SRX-Getting-Started-Configure-J-Flow?language=en_US Accessed: 2022-08-01.
- [28] Konstantina Papagiannaki, Nina Taft, and Anukool Lakhina. A Distributed Approach to Measure IP Traffic Matrices. *IMC'04*. ACM, 2004. doi:10.1145/1028788.1028808.
- [29] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Proactively Detecting Distributed Denial of Service Attacks Using Source IP Address Monitoring. In *International Conference on Research in Networking*. Springer, 2004. doi:10.1007/978-3-540-24693-0_63.
- [30] P. Phaal, S. Panchen, and N. McKee. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. RFC 3176 (Informational), 2001. <https://datatracker.ietf.org/doc/html/rfc3176> Accessed: 2022-08-01.
- [31] Salvatore Pontarelli, Roberto Bifulco, Marco Bonola, Carmelo Cascone, Marco Spaziani, Valerio Bruschi, Davide Sanvito, Giuseppe Siracusano, Antonio Capone, Michio Honda, Felipe Huici, and Giuseppe Siracusano. FlowBlaze: Stateful Packet Processing in Hardware. In *NSDI'19*. USENIX Association, 2019. <https://www.usenix.org/conference/nsdi19/presentation/pontarelli> Accessed: 2022-08-01.
- [32] The Zeek Project. The Zeek Network Security Monitor, 2020. <https://zeek.org/> Accessed: 2022-08-01.
- [33] Jeff Rasley, Brent Stephens, Colin Dixon, Eric Rozner, Wes Felter, Kanak Agarwal, John Carter, and Rodrigo Fonseca. Planck: Millisecond-scale Monitoring and Control for Commodity Networks. *SIGCOMM'14*. ACM, 2014. doi:10.1145/2619239.2626310.
- [34] Said Jawad Saidi, Aniss Maghsoudlou, Damien Foucard, Georgios Smaragdakis, Ingmar Poese, and Anja Feldmann. Exploring Network-Wide Flow Data With Flowyager. *IEEE Transactions on Network and Service Management*, 2020. doi:10.1109/TNSM.2020.3034278.
- [35] Philip Smith, Rob Evans, and Mike Hughes. RIPE Routing Working Group Recommendations on Route Aggregation, 2006. <https://www.ripe.net/publications/docs/ripe-399> Accessed: 2022-08-01.
- [36] Augustin Soule, Anukool Lakhina, Nina Taft, Konstantina Papagiannaki, Kave Salamatian, Antonio Nucci, Mark Crovella, and Christophe Diot. Traffic Matrices: Balancing Measurements, Inference and Modeling. In *ACM SIGMETRICS Performance Evaluation Review*, 2005. doi:10.1145/1071690.1064259.
- [37] Renata Teixeira, Aman Shaikh, Timothy G Griffin, and Jennifer Rexford. Impact of Hot-Potato Routing Changes in IP Networks. *IEEE/ACM Transactions On Networking*, 2008. doi:10.1109/TNET.2008.919333.
- [38] Ross Teixeira, Rob Harrison, Arpit Gupta, and Jennifer Rexford. PacketScope: Monitoring the Packet Lifecycle Inside a Switch. *SOSR'20*. ACM, 2020. doi:10.1145/3373360.3380838.
- [39] Olivier Tilmans, Tobias Bühler, Ingmar Poese, Stefano Vissicchio, and Laurent Vanbever. Stroboscope: Declarative Network Monitoring on a Budget. In *NSDI'18*. USENIX Association, 2018. <https://www.usenix.org/conference/nsdi18/presentation/tilmans> Accessed: 2022-08-01.

- [40] Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. OpenTM: Traffic Matrix Estimator for Open-Flow Networks. In *International Conference on Passive and Active Network Measurement*. Springer, 2010. doi:10.1007/978-3-642-12334-4_21.
- [41] Andreas Wundsam, Dan Levin, Srini Seetharaman, and Anja Feldmann. OFRewind: Enabling Record and Replay Troubleshooting for Networks. In *ATC'11*. USENIX Association, 2011. <https://www.usenix.org/conference/usenixatc11/ofrewind-enabling-record-and-replay-troubleshooting-networks> Accessed: 2022-08-01.
- [42] Da Yu, Yibo Zhu, Behnaz Arzani, Rodrigo Fonseca, Tianrong Zhang, Karl Deng, and Lihua Yuan. dShark: A General, Easy to Program and Scalable Framework for Analyzing In-network Packet Traces. In *NSDI'19*. USENIX Association, 2019. <https://www.usenix.org/conference/nsdi19/presentation/yu> Accessed: 2022-08-01.
- [43] Minlan Yu, Lavanya Jose, and Rui Miao. Software Defined Traffic Measurement with OpenSketch. In *NSDI'13*. USENIX Association, 2013. <https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/yu> Accessed: 2022-08-01.
- [44] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, and Nicholas Zhang. LightGuardian: A Full-Visibility, Lightweight, In-band Telemetry System Using Sketchlets. In *NSDI'21*. USENIX Association, 2021. <https://www.usenix.org/conference/nsdi21/presentation/zhao> Accessed: 2022-08-01.
- [45] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y. Zhao, and Haitao Zheng. Packet-Level Telemetry in Large Datacenter Networks. *SIGCOMM'15*. ACM, 2015. doi:10.1145/2785956.2787483.

A CRediT statement

This section lists the author’s contributions to this work. The contributions are described using CRediT, the Contributor Roles Taxonomy, an ANSI/NISO standard.

All authors agree with this declaration of contributions.

Tobias Bühler	0000-0002-8876-1546	
Conceptualization	Equal	
Data curation	Lead	
Formal analysis	Lead	
Investigation	Lead	
Methodology	Lead	
Project administration	Lead	
Software	Lead	
Validation	Equal	
Visualization	Equal	
Writing – original draft	Lead	
Writing – review & editing	Supporting	
Romain Jacob	0000-0002-2218-5750	
Conceptualization	Supporting	
Data curation	Supporting	
Formal analysis	Supporting	
Methodology	Supporting	
Project administration	Supporting	
Software	Supporting	
Supervision	Equal	
Validation	Equal	
Visualization	Equal	
Writing – original draft	Supporting	
Writing – review & editing	Lead	
Ingmar Poese		
Data curation	Supporting	
Investigation	Supporting	
Resources	Equal	
Supervision	Supporting	
Laurent Vanbever	0000-0003-1455-4381	
Conceptualization	Equal	
Funding acquisition	Lead	
Investigation	Supporting	
Methodology	Supporting	
Project administration	Supporting	
Resources	Equal	
Supervision	Equal	
Writing – original draft	Supporting	
Writing – review & editing	Supporting	

B Magnifier’s controller placement

Magnifier needs a central controller to build its network-wide ingress/egress view. As we heavily depend on sampled flow observations, it makes sense to co-locate *Magnifier* with the e.g., already existing, central collector of the sampling data. In large ISP networks, with routers around the globe, we can deploy additional sub-controllers that start and stop the mirroring rules and collect mirrored packets. More precisely, the main controller is needed to compute new sentinels and builds the final ingress/egress observations. It delegates mirroring to the sub-controllers which autonomously handle the deployment, activation and deactivation of rules while reporting back any mirroring-based observations.

C Everflow implementation

Following a few more details to our Everflow reimplementation in our simulation setup.

Everflow uses packet mirroring to produce its random packet samples. The paper [45] explains that Everflow mirrors based on a fixed number of bits in the IP identification header field (IPID). As an example, selecting 10 random bits in the IPID field will result in random packet sampling of 1 out of $2^{10} = 1024$ packets. However, this assumption is only true if the values in the IPID fields are more-or-less uniformly distributed. Taking our CAIDA trace as an example, we see that we have a huge number of packets which set the IPID field to zero. Depending on how we select the bits in the IPID field, we might get way more or less sampled packets than expected. For this reason, we implemented the random packet sampling aspect of Everflow in our simulation code by taking every n-th packet observed on a device, e.g., every 1024th packet in the previous example.

Additional to the implemented mirroring techniques (random packet sampling and TCP flag packets), Everflow also supports mirroring of packets with a special debug bit. As this was not relevant for a direct comparison with *Magnifier*, we did not implement this feature in our simulation code. The same holds for Everflow’s controller, storage and reshuffler components.

D Additional evaluation results

This appendix section first illustrates the lab setup used to evaluate *Magnifier*. We then analyze the used CAIDA trace in more detail. Afterward, we show additional evaluation results focused on *Magnifier*’s performance. We conclude with additional plots comparing *Magnifier* with Everflow.

D.1 Magnifier lab setup

Fig. 13 illustrates the lab setup we used to evaluate *Magnifier*. We establish four parallel connections between the two Nexus

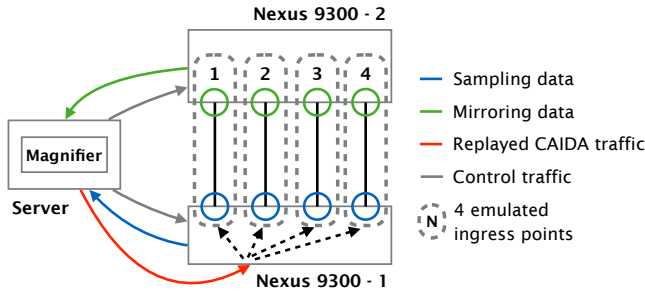


Figure 13: Two Nexus 9300 switches emulate four network ingress points. The traffic is replayed and sampled on the first switch, then forwarded to the second, which mirrors packets.

9300 switches, each emulating a network ingress. The first switch receives and samples the traffic using sFlow (sampling rate $1/4096^7$). It then forwards to the second switch, which mirrors the traffic according to the configured rules. *Magnifier*'s controller runs on a server and collects sampling and mirroring data. As these switches are limited to 512 mirroring rules, we used a fixed budget of 500 rules per emulated ingress point.⁸

D.2 CAIDA data analysis

Fig. 14 shows a CDF of the amount of packets observed per source $/24$ in 60s of our CAIDA trace used in the evaluation. 60s represent one iteration at real-time replay speed. As we can see we have a very small number of heavy hitters which carry most traffic as well as a huge number of $/24$ prefixes which only contain a few packets. Roughly 10% of all $/24$ prefixes contain more than 90% of all the packets.

A lot of the $/24$ prefixes with very low packet counts are most likely DDoS attack traffic (e.g., TCP SYN packets). We decided to keep these packets in the trace as a real ISP network could also observe similar packet distributions in their transit traffic.

D.3 Additional Magnifier plots

This section contains additional plots which evaluate *Magnifier* in terms of packet coverage and mirrored traffic.

Fig. 15 shows the performance results if we consider an increasing number of border routers (from 4 to 64). For *random* and *static* traffic assignment we notice that the coverage slightly drops while we see an increased amount of mirrored traffic. However, this is not true for the *permuted* assignment strategies. *random* and *static* distribute the packets to their ingress points based on equal slices of the *destination* IP space. If we have more border routers, we also have additional slices

⁷The highest configurable rate on this model; we get the *most* samples.

⁸The four emulated ingresses share the budget. We use TCAM carving [7] to increase the space for our mirroring rules to 2048 (by taking it from other features), to enable the original budget (512) per ingress.

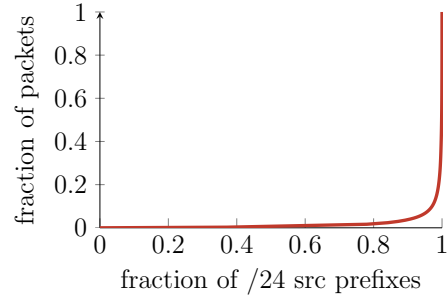


Figure 14: A CDF plot of the amount of packets observed per source $/24$ prefix in 60s (one iteration at real speed) in our CAIDA trace.

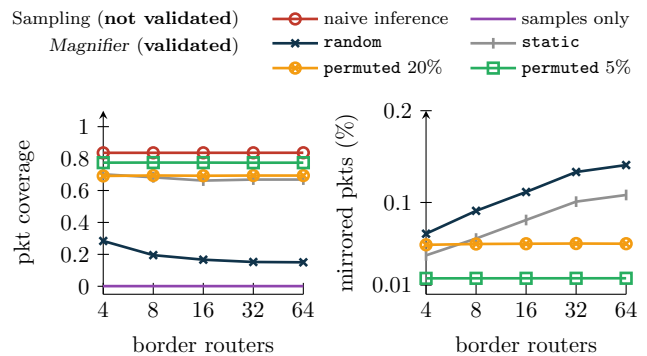


Figure 15: Simulation results for coverage and mirrored traffic when *Magnifier* runs with different amounts of border routers. CAIDA traces replayed at real speed, sampling rate $1/1024$.

and close IP space is distributed over multiple ingresses which leads to the observed drop in coverage. This is not true for the *permuted* cases, where we always permute a fixed number of source $/24$ prefixes to different ingresses.

Fig. 16 considers different sampling rates. As expected, if we have fewer samples as input *Magnifier* can cover fewer packets and also produces fewer mirrored packets as it finds fewer sentinels to begin with. We observe this behavior for all traffic assignments.

Finally, Fig. 17 shows the missing assignment strategies (*random* and *permuted 20%*) if we consider different amounts of top sentinels (*activity* and *size* ordering). Following the results in Fig. 8, the *activity* strategy provides better coverage than *size* and a lower amount of mirrored traffic.

D.4 Stability of sentinels

Following, we evaluate how many sentinels change between simulation iterations.

Setup We use the results from our simulations with 32 border routers, real traffic speed and various traffic-to-ingress assignments (sampling rate $1/1024$). The results show mean values over 60 iterations.

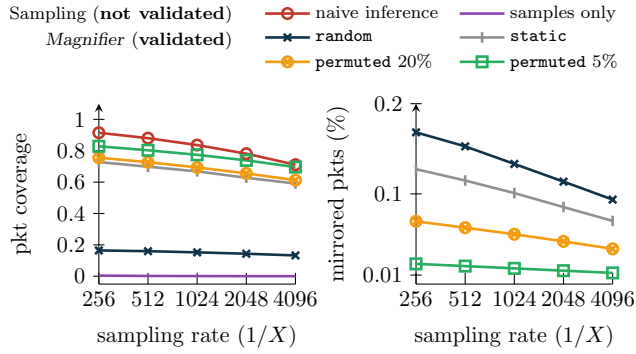


Figure 16: Simulation results for coverage and mirrored traffic when *Magnifier* runs with different sampling rates. CAIDA trace replayed at real speed, 32 border routers.

Results Table 3 shows the amount of changed sentinels for different amounts of deployed sentinels (based on activity and size ordering) for random, static and permuted 5% traffic assignment. We first observe that we have to change fewer sentinels if we base the ordering on sentinel activity. More active sentinels are often also stable over longer periods of time which means that we find them consistently. We see a different behavior for the ordering based on size. Here nearly all sentinels change between iterations. The largest sentinels are often based on sparse samples located in empty prefix space. That means, we might not be able to find the same big sentinel between multiple iterations if the covered flows are no longer visible (e.g., in the sampled data).

As expected, the number of changed sentinels also depends on the difficulty of the traffic assignment. In the *random* case, ingress assignment changes frequently even during a single iteration. That means we often find new sentinels in the following iteration. For *permuted 5%*, the assignment is much more stable and we can always keep around 50% of all sentinels between iterations.

	# sentinels	random	static	permuted 5%
activity ordering	Top 100	84	50	35
	Top 500	406	292	220
	Top 1000	836	610	466
	Top 5000	4487	3662	2769
size ordering	Top 100	94	87	45
	Top 500	472	453	224
	Top 1000	950	918	461
	Top 5000	4817	4698	2776

Table 3: Number of changed sentinels between iterations for different assignment and sentinel ordering strategies.

Conclusion The top sentinels often change between iterations, however *Magnifier* is not really impacted by that. As we describe in § 4.1, *Magnifier* works with two ACLs and switches between them. While one is active, the other one gets populated.

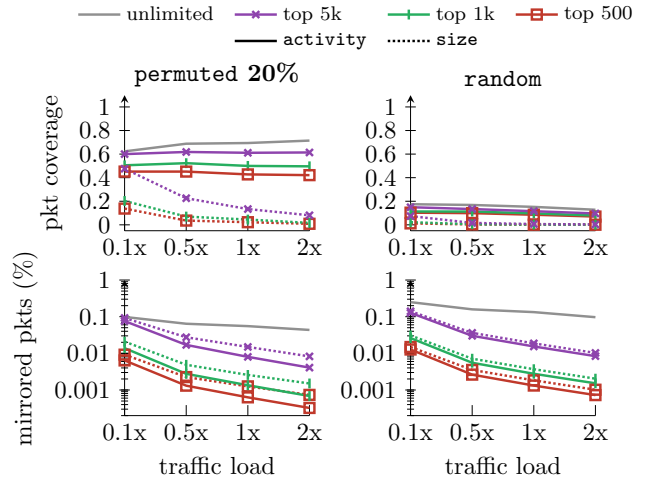


Figure 17: Simulation results for coverage and mirrored using different amount of top sentinels according to a activity and size ordering. We show the *random* and *permuted 20%* traffic assignment strategies (compare with Fig. 8). The plots show values for different traffic replay speeds of the CAIDA trace with 32 border routers and sampling rate of 1/1024.

The frequent sentinel changes between iterations are therefore not a big problem as we anyway need to build a completely new ACL.

D.5 Additional comparison with Everflow

In this section we show additional comparison plots between *Magnifier* and Everflow. Fig. 18 shows different number of ingress routers while Fig. 19 considers varying sampling rates. For both figures we show the results for *permuted 5%* and *static* traffic assignments. Everflow’s packet coverage and amount of mirrored packets show only small reactions to the different ingress routers and/or sampling rates. Everflow’s mirrored packets mainly contain packets due to TCP SYN, FIN or RST flags. The randomly sampled ones contribute only in a small amount. As a result, changes in the sampling rate (Fig. 19) have more impact on *Magnifier* than on Everflow. *Magnifier*’s performance is tightly related to the amount and distribution of the randomly sampled packets.

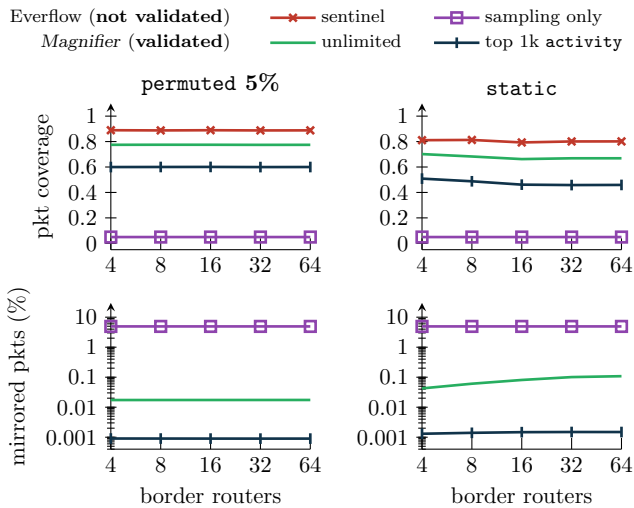


Figure 18: Comparison of coverage and mirrored traffic for *Magnifier* and *Everflow* for different amounts of border routers. We show the *static* and *permuted 5%* traffic assignment. We replay the CAIDA trace at real speed and use a sampling rate of 1/1024.

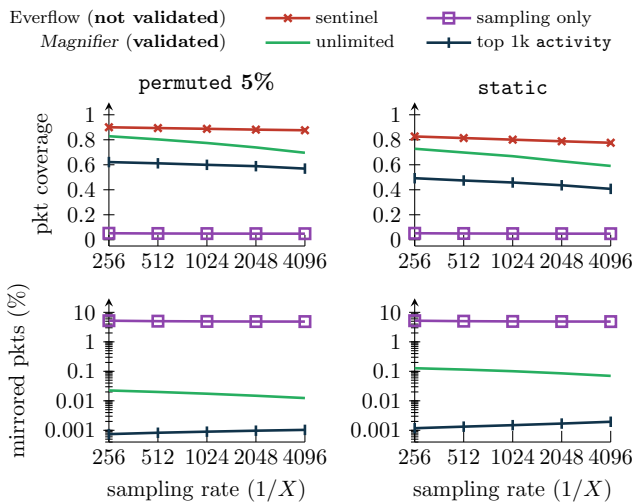


Figure 19: Comparison of coverage and mirrored traffic for *Magnifier* and *Everflow* for different sampling rates. We show the *static* and *permuted 5%* traffic assignment. We replay the CAIDA trace at real speed and distribute traffic over 32 simulated ingresses.