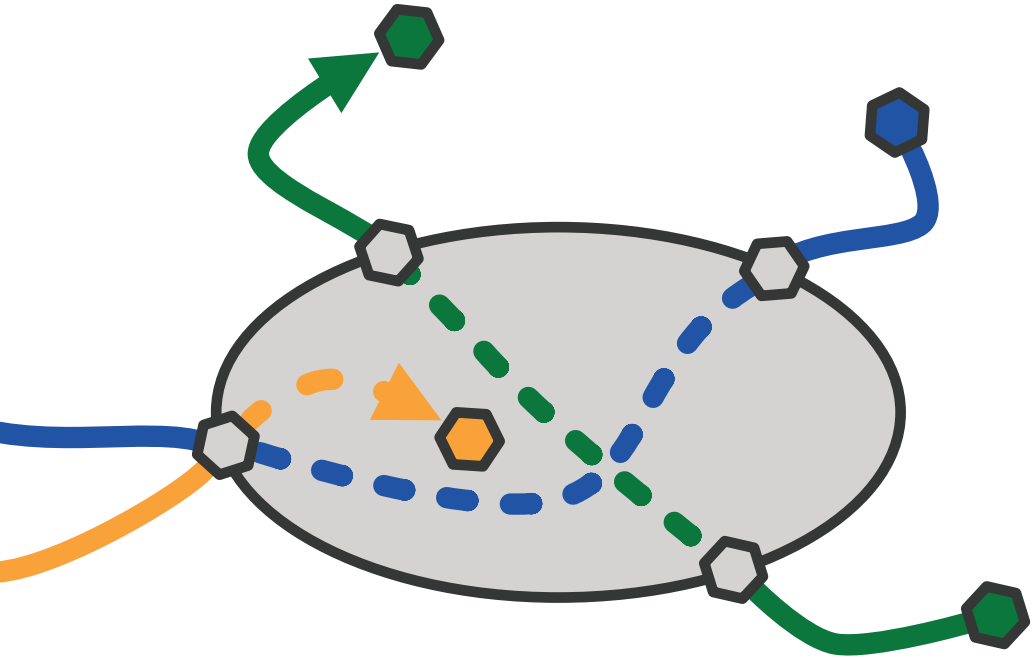
IMPROVING INTERNET PATH PROPERTY INFERENCE

DISS. ETH NO. 29163



TOBIAS MICHAEL BÜHLER

DISS. ETH NO. 29163

IMPROVING INTERNET PATH PROPERTY INFERENCE

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES
(Dr. sc. ETH Zurich)

presented by

TOBIAS MICHAEL BÜHLER
MSc ETH EEIT
ETH Zurich

born on May 15th, 1990

accepted on the recommendation of

Prof. Dr. Laurent Vanbever (Advisor)
Prof. Dr. Marco Chiesa
Prof. Dr. Anja Feldmann

2023

Diss. ETH No. 29163
TIK-Schriftenreihe-Nr. 204

ABSTRACT

The Internet has become an inseparable part of our daily life and work activities. Due to its distributed nature, performant packet forwarding along a given Internet path only works if each individual network operates properly. This leads to an inherent challenge for network operators. They must provide high performance and select appropriate paths towards *external* destinations while limited to *internal* signals and traffic observations.

This dissertation focuses on one key solution, namely the inference of path properties, which supports network operators in monitoring, debugging, and threat detection tasks. To this end, we present two systems that focus on *internal* and *external* path properties, respectively.

First, we present *Magnifier*, a system that enhances existing sampled monitoring data with packet mirroring to produce validated network ingress and egress observations. One of *Magnifier*'s key insights is to mirror traffic where we do *not* expect to observe matching packets. This way, we profit from the advantages of mirroring (precise and fast feedback) without the typical drawbacks (significantly increased traffic amounts).

Second, we present *Oscilloscope*, a system that detects malicious hijacks of network traffic by analyzing changes in locally collected Round-Trip Time signals. Intuitively, a path change leads to an observable difference in a packet's travel time. *Oscilloscope* combines hijack-typical patterns with statistical tests to increase its confidence that detected Round-Trip Time changes belong to hijack events.

Both systems use inferred path properties *reactively*. However, to prevent problems *proactively*, operators need to adapt their forwarding decisions based on inferred path properties.

Third, we explore how adding simple path properties to the existing path selection process improves routing decisions. For example, we prevent unnecessary packet losses by testing the reachability of a new path before blindly trusting it to carry *all* matching traffic. We advocate slowing down the traffic shift towards the new path to achieve that. Although that allows for more control, it poses new convergence and communication challenges.

ZUSAMMENFASSUNG

Das Internet ist ein untrennbarer Bestandteil unseres täglichen Lebens und unserer Arbeit geworden. Da es aus vielen individuellen Netzwerken besteht, kann die effiziente Weiterleitung von Datenpaketen über einen bestimmten Internetpfad nur gelingen, wenn jedes einzelne Netzwerk ordnungsgemäss funktioniert. Dies führt zu einer grundlegenden Herausforderung für Netzwerkbetreiber. Sie müssen gute Leistungen bieten und geeignete Pfade zu *externen* Zielen auswählen, während sie nur *interne* Signale und Paketbeobachtungen zur Verfügung haben.

Diese Dissertation konzentriert sich auf einen zentralen Lösungsansatz, nämlich die Ableitung von Pfadeigenschaften, die Netzbetreiber bei der Überwachung, Fehlersuche und Erkennung von Bedrohungen unterstützt. Zu diesem Zweck stellen wir zwei Systeme vor, die sich auf *interne* bzw. *externe* Pfadeigenschaften konzentrieren.

Zunächst stellen wir *Magnifier* vor, ein System, das vorhandene Überwachungsdaten mit Paketspiegelung erweitert, um validierte Beobachtungen des Netzwerkeingangs und -ausgangs zu erhalten. Eine der wichtigsten Erkenntnisse von *Magnifier* ist die Spiegelung von Datenverkehr an Orten, bei welchen wir *keine* passenden Pakete erwarten. Auf diese Weise profitieren wir von den Vorteilen der Paketspiegelung (präzises und schnelles Feedback) ohne deren typischen Nachteile (deutlich erhöhte Verkehrsmengen).

Zweitens stellen wir *Oscilloscope* vor, ein System, das böswillige Umleitungen des Netzwerkverkehrs (sogenannte "Hijacks") durch die Analyse von Änderungen in lokal gemessenen Paketumlaufzeiten erkennt. Intuitiv führt eine Pfadänderung zu einem beobachtbaren Unterschied in der Paketumlaufzeit. *Oscilloscope* kombiniert Hijack-typische Muster mit statistischen Tests, um die Wahrscheinlichkeit zu erhöhen, dass erkannte Änderungen in der Paketumlaufzeit zu Hijack Ereignissen gehören.

Beide Systeme verwenden abgeleitete Pfadeigenschaften *reaktiv*. Um jedoch Probleme *proaktiv* zu verhindern, müssen die Netzwerkbetreiber ihre Entscheidungen zur Paketweiterleitung auf der Grundlage der abgeleiteten Pfadeigenschaften anpassen.

Drittens untersuchen wir, wie das Hinzufügen einfacher Pfadeigenschaften zum bestehenden Pfadauswahlprozess die Paketweiterleitung verbessern kann. Zum Beispiel verhindern wir unnötige Paketverluste, indem wir die Erreichbarkeit eines neuen Pfades testen, bevor wir ihm blind vertrauen, dass er *alle* entsprechenden Pakete befördern kann. Um dies zu erreichen, schlagen wir vor, die Verlagerung von Paketen auf einen neuen Pfad zu verlangsamen. Dadurch erhalten wir mehr Kontrolle, sind aber auch mit neuen Konvergenz- und Kommunikationsproblemen konfrontiert.

PUBLICATIONS

This dissertation is based on several papers published in conference proceedings presented hereafter.

Sentinels: Guarding ISP Networks from Forwarding Anomalies
Tobias Bühler, Ingmar Poese, Laurent Vanbever.
ACM CoNEXT Student Workshop, Irvine, CA, USA, 2016.

Enhancing encrypted transport protocols with passive measurement capabilities
Tobias Bühler, Mirja Kühlewind, Brian Trammell.
ACM IMC Poster, London, UK, 2017.

Generating representative, live network traffic out of millions of code repositories
Tobias Bühler, Roland Schmid, Sandro Lutz, Laurent Vanbever.
ACM HotNets, Austin, TX, USA, 2022.

Oscilloscope: Detecting BGP Hijacks in the Data Plane
Tobias Bühler, Alexandros Milolidakis, Romain Jacob, Marco Chiesa, Stefano Vissicchio, Laurent Vanbever.
arXiv, arxiv:2301.12843, 2023.

Enhancing Global Network Monitoring with Magnifier
Tobias Bühler, Romain Jacob, Ingmar Poese, Laurent Vanbever.
USENIX NSDI, Boston, MA, USA, 2023.

The following publications were part of my PhD research and are referenced in this dissertation, but they were led by other researchers.

A Path Layer for the Internet:
Enabling Network Operations on Encrypted Protocols

Mirja Kühlewind, Tobias Bühler, Brian Trammell,
Stephan Neuhaus, Roman Müntener, Gorry Fairhurst.

IEEE CNSM, Tokyo, Japan, 2017.

Stroboscope: Declarative Network Monitoring on a Budget

Olivier Tilmans, Tobias Bühler, Ingmar Poese,
Stefano Vissicchio, Laurent Vanbever.

USENIX NSDI, Renton, WA, USA, 2018.

Three Bits Suffice: Explicit Support for Passive Measurement of Internet Latency in QUIC and TCP

Piet De Vaere, Tobias Bühler, Mirja Kühlewind, Brian Trammell.

ACM IMC, Boston, MA, USA, 2018.

Challenges in Network Management of Encrypted Traffic

Mirja Kühlewind, Brian Trammell, Tobias Bühler,
Gorry Fairhurst, Vijay Gurbani.

arXiv, arXiv:1810.09272, 2018.

An Open Platform to Teach How the Internet Practically Works

Thomas Holterbach, Tobias Bühler, Tino Rellstab, Laurent Vanbever.

ACM SIGCOMM CCR, New York, NY, USA, 2020.

Smart BGP hijacks that Evade Public Route Collectors

Alexandros Milolidakis, Tobias Bühler, Marco Chiesa,
Laurent Vanbever, Stefano Vissicchio.

ACM IMC Poster, Virtual Event, 2021.

On the Effectiveness of BGP Hijackers
That Evade Public Route Collectors

Alexandros Milolidakis, Tobias Bühler, Kunyu Wang,
Marco Chiesa, Laurent Vanbever, Stefano Vissicchio.

IEEE Access, Volume 11, 2023.

The following publications were part of my PhD research, but they are not covered in this dissertation.

Mille-Feuille: Putting ISP Traffic under the Scalpel

Olivier Tilmans, Tobias Bühler, Stefano Vissicchio, Laurent Vanbever.

ACM HotNets, Atlanta, GA, USA, 2016.

pForest: In-Network Inference with Random Forests

Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, Laurent Vanbever.

arXiv preprint, arXiv:1909.05680, 2019.

ACKNOWLEDGMENTS

This dissertation was only possible due to the help of many people who made my doctorate journey very interesting and enjoyable. In the following paragraphs, I mention some that deserve a special thank you.

First and foremost, I would like to thank Prof. Laurent Vanbever for his relentless support and guidance. Starting with my Master's thesis up to discussions about plans after my doctorate, I always received great feedback. Laurent did not only teach me a lot about how to perform good research, but I could also significantly improve my writing and presentation skills. Last but not least, I learned good teaching practices. I especially appreciate the enormous trust and freedom I received as a teaching assistant.

I am also incredibly thankful to my second advisor, Dr. Romain Jacob. His inputs and ideas during our weekly meetings were always on point and helped me immensely.

I thank Prof. Anja Feldmann and Prof. Marco Chiesa for being part of my dissertation committee and reading a draft of this dissertation.

I especially thank Dr. Mirja Kühlewind, Brian Trammell, and all the other EU Horizon 2020 MAMI project members. They gave me a great perspective during the start of my doctorate and allowed me to visit many exciting places and universities in Europe.

I would also like to thank all the other members of the Networked Systems Group for all their support, great time, and fun moments we had together: Maria Apostolaki, Ahmed El-Hassany, Thomas Holterbach, Rüdiger Birkner, Roland Meier, Rui Yang, Albert Gran Alcoz, Alexander Dietmüller, Coralie Busse-Grawitz, Edgar Costa Molero, Ege Cem Kirci, Roland Schmid, Theo von Arx, Tibor Schneider, Yu Chen, Georgia Fragkouli and Muoi Tran. A special thank goes to Ahmed, Rüdiger, Edgar, and Roland Schmid, with whom I had the pleasure of handling our server infrastructure. Although stressful at times, we also learned a lot of new skills. Then, Roland Meier, who was an extremely pleasant and helpful office mate during many years of my doctorate. Finally, Rüdiger for his enormous help with teaching activities and invaluable discussions during my doctorate.

Furthermore, I thank all my co-authors and collaborators. Especially Olivier Tilmans, with whom I had the pleasure to work at the beginning of my doctorate. I learned a lot from his coding skills. Then Prof. Marco Chiesa and Alexandros Milolidakis for all the interesting discussions and calls we had together. Thomas Holterbach and Roland Schmid even managed to make the stressful paper deadlines a pleasant time. Finally, Dr. Stefano Vissicchio for his helpful ideas and discussions.

I am also grateful to those who supported my doctorate through other means. Namely, Dr. Bernhard Ager and Dr. Ingmar Poese gave me access to monitoring data from real networks, bridging the gap between research and production networks. Derk-Jan Valenkamp and Paul Stark provided me with various hardware routers and switches. Finally, Beat Futterknecht found solutions to every administrative and organizational problem.

During my doctorate, I had the great opportunity to supervise various students with their Bachelor's, semester, and Master's thesis. A special thank goes to Piet De Vaere, Alexander Dietmüller, Coralie Busse-Grawitz, Hendrik Züllig, Tino Rellstab, and Sandro Lutz. Additionally, I truly enjoyed my years as (head) teaching assistant in the Communication Networks lecture. Only due to the relentless questions from motivated students was I truly able to understand complicated network topics.

Finally, I would like to thank my family and friends who supported me throughout my academic journey. I was always sure to find someone to talk to, especially during stressful times. A special thank goes to my parents.

Tobias Bühler
July 2023

CONTENTS

1

INTRODUCTION

Today, the Internet is essential to nearly every aspect of our life. The economy depends on it; we use the Internet for our work and even during our free time: for entertainment, as news sources, and to socialize. The increased usage inherently leads to higher demands on availability, performance, security, and privacy. Luckily, the Internet keeps delivering as the transition to remote work during the Covid-19 pandemic impressively showed.

The Internet consists of numerous interconnected individual networks, so-called Autonomous Systems (ASes), which forward packets between each other. Every network on the forwarding path must work properly for a packet to reach its destination. As the name suggests, an AS builds an isolated entity. Network operators configure the AS according to the network dimension, provided services, and custom policies. Most of the fine-grained configuration details are kept secret and thus unknown to operators of other ASes. The lack of insights paired with the necessity of working Internet-wide forwarding paths leads to an inherent challenge for operators: they must provide good user performance and make forwarding decisions towards *external* destinations while being limited to local, *internal* traffic observations and signals from their own network.

To overcome this challenge, operators must *infer* – rather than precisely extract – Internet-wide path properties and take appropriate actions. As a first step, operators infer properties of their *internal* network paths, which is essential to *(i)* answer questions from customers who quickly blame [15] operators for Internet problems, even if the corresponding network is not at fault [16]; and *(ii)* detect network outages that can have drastic consequences such as unreachable emergency services [17].

As a second step, operators infer properties of *external* network paths based on locally-collected signals. Besides performance-related properties, such as path latency and current throughput, path inference also allows for detecting malicious attacks, e.g., traffic hijacks [18], which is an increasingly important consideration for network operators [19]. As a final step, operators consider the inferred path properties when deciding between multiple path options. To do so, they must adequately configure the routing

protocols running on local devices. In today's Internet, inter-domain routing is primarily controlled by the Border Gateway Protocol (BGP) [20].

More fundamentally, operators encounter two common difficulties when performing path inference – *first*, different noise sources. Inside a single network, the ever-increasing Internet traffic [21] often requires systems that consider sampled data only. For external path property inference, based on locally-collected signals, operators face a different noise source. An available signal does not directly represent one path property: Multiple properties, and other Internet traffic, commonly affect the same signal.

And *second*, path property inference is only possible with live production traffic. In other words, without corresponding user traffic, operators are "blind" and cannot infer all necessary properties. Naive solutions, such as generating probe packets, either do not scale or are not representative [22].

This dissertation presents new systems and ideas to improve Internet path property inference to help operators with network monitoring, threat detection, and active routing decisions. We first focus on two systems, *Magnifier* and *Oscilloscope*, which infer *internal* and *external* path properties, respectively. In both cases, we find solutions to cope with different noise sources resulting in imperfect property inference.

Magnifier [5] combines existing, sparsely sampled flow data to create so-called sentinels. A sentinel uniquely maps an IP prefix either to a network ingress or egress – the main path property we focus on. However, given that sampled data is incomplete, some of the inferred sentinels are wrong. *Magnifier* validates them by deploying mirroring rules on all *other* ingress or egress points, i.e., locations where no sentinel-related traffic should enter or exit the network. Thus, a lack of mirrored traffic validates the sentinel inference, while mirrored packets quickly inform *Magnifier* about wrong insights. In conclusion, *Magnifier* provides network operators with validated traffic ingress and egress observations.

Oscilloscope [4] focuses on the detection of so-called BGP hijacks. During a hijack, malicious entities intercept traffic that normally does not cross their network. *Oscilloscope* follows a simple yet powerful intuition: once a hijack for an IP prefix starts, some packets towards the prefix suddenly follow a different path, resulting in an increase (or decrease) of the observed Round-Trip Times (RTTs). Therefore, *Oscilloscope* continuously collects RTT samples and detects changes using a moving window. However, not every RTT change originates due to a malicious hijack. For example, regular forwarding adjustments build another source of RTT variations but do not represent

the path property we are looking for. To boost its certainty, *Oscilloscope* uses statistical tests in combination with domain-specific insights. It compares the distribution of RTT samples from a potentially hijacked prefix with samples belonging to close IP space routed equally in the *absence* of a hijack. If the test indicates different distributions, the RTT change likely stems from a BGP hijack. In conclusion, *Oscilloscope* provides network operators with another solution to detect BGP hijacks using locally collected signals.

Although *Magnifier* and *Oscilloscope* improve network monitoring and reveal external threats, they only inform operators *reactively*. For a more *proactive* approach, network operators must adapt their routing decisions based on inferred Internet path properties. In an exploratory study, we discuss how adding path properties in today's primary inter-domain routing protocol (BGP) prevents unnecessary packet losses due to avoiding paths that do not provide reachability. Compared to the immediate and irrevocable traffic shift BGP performs upon selecting a new best path, we advocate for a slower, more spread-out movement. This way, network operators can infer properties of the new path and revert the choice should they detect suboptimal properties, such as unreachable destinations. However, our study shows that a naive integration of path properties quickly leads to wrong inferences or slow routing convergence. We reveal pitfalls, compare design principles, and provide best practices.

Looking back at the overall challenge, this dissertation improves *internal* path property inference with selected packet mirroring and *external* path property inference with statistical tests. Finally, we present new ideas to integrate path properties in routing decisions. However, the two fundamental difficulties remain: *(i)* blindness without matching traffic and *(ii)* wrong inferences due to different noise sources. We conclude the dissertation with a high-level discussion of how our vision of an operator-defined, representative traffic generator called *Dynamo* [3] could tackle *(i)*. *Dynamo* uses the abundance of "orchestrable" open-source projects to generate live network traffic, which follows operator-defined properties and reacts to network events. Our idea of an additional packet header that purely focuses on the Internet path [6] could solve *(ii)*. Using this header, endpoints precisely define which signals they provide to on-path devices and what they expect from them in return.

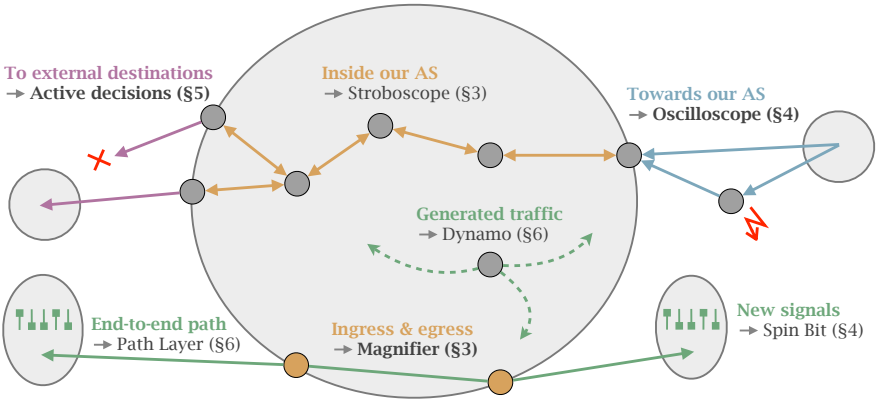


FIGURE 1.1: This dissertation mainly focuses on property inference of paths *inside* our AS (*Magnifier*; Chapter 3), *towards* our AS (*Oscilloscope*; Chapter 4), and to *external* destinations (Active Routing Decisions; Chapter 5).

Dissertation structure Figure 1.1 puts all our contributions into perspective. We focus on one AS that deploys our systems and highlights the path property connected to each contribution. Adding different inferences together, we cover path properties related to the entire end-to-end path between two traffic endpoints.

After a brief background overview (Chapter 2), we first show how *Magnifier* [5] produces *validated* packet ingress and egress points inside an AS (Chapter 3). That allows for precise packet tracing, e.g., using systems such as Stroboscope [7]. Second, we explain how *Oscilloscope* [4] combines locally collected RTT signals with statistical tests to detect BGP hijacks (Chapter 4). This chapter also explores new RTT extraction methods for privacy-aware transport protocols (Spin Bit [8]). Third, we combine path properties with control-plane signals to detect suboptimal forwarding paths (Chapter 5). Finally, we conclude the dissertation and discuss how more fundamental changes, for example, the introduction of a new Path Layer [6] in each packet header or representative traffic generation (*Dynamo* [3]), could lead to a more straightforward path property inference in the future (Chapter 6).

2

BACKGROUND

This chapter introduces the fundamental concepts used throughout the dissertation. We first look at the Internet Protocol (IP) and IP prefixes which identify where a packet is coming from and heading to (Section 2.1). Then we focus on the different entities that form today's Internet and how they are connected (Section 2.2). Afterwards, we introduce the Border Gateway Protocol (BGP), an inter-domain routing protocol that enables Internet-wide packet forwarding (Section 2.3). Finally, we look at the Transmission Control Protocol (TCP), a reliable transport protocol, as well as TCP-based Round-Trip Time (RTT) measurements, which operators often use to monitor and debug networks (Section 2.4).

2.1 INTERNET PROTOCOL AND PREFIXES

A parcel, which transports physical goods, typically contains an address (e.g., name, street, house number, city, country) and the actual content. Digital information transfer in today's Internet is very similar. Endpoints split the digital data into packets containing multiple headers and a payload. One of these headers is the so-called Internet Protocol (IP) header, which includes IP addresses that identify the packet's source and destination.

2.1.1 *IP addresses*

A packet in the Internet contains two IP addresses – a source and a destination IP – and devices, such as routers, use these IPs to figure out where to forward the packet to. We can distinguish between IPv4 and IPv6 addresses.

IPv4 [23] addresses are 32-bit long. That means we can have $2^{32} \approx 4.3$ billion different IPv4 addresses. We express an IPv4 address as four 8-bit decimal numbers separated by a dot. For example, 1.2.3.4 or 52.77.201.34 are two valid IPv4 addresses.

Besides the two IPv4 addresses, the IPv4 header contains additional fields, e.g., a checksum field revealing corrupted packet headers or the so-called Time-To-Live (TTL) field. Each IP-speaking device (for example an IP router) on the path towards the destination decreases the TTL value by one. If the TTL reaches zero, the packet is dropped/destroyed. This mechanism prevents ever-lasting packets in forwarding loops.

Compared to IPv4 addresses, IPv6 [24] addresses consist of 128 bits resulting in $2^{128} \approx 3.4 * 10^{38}$ unique addresses – an enormous number and one main reason for the introduction of IPv6. As more and more devices use the Internet, the available IPv4 address space runs out. In fact, already in 2014, the Internet Corporation for Assigned Names and Numbers (ICANN) announced that they distributed the final remaining blocks of IPv4 addresses [25].

This dissertation primarily focuses on IPv4 addresses. However, all our systems and ideas apply to IPv6 addresses as well. Dedicated paragraphs will highlight possible challenges.

2.1.2 *IP prefixes*

Considering the parcel analogy once more, a post office will initially only look at the city indicated in the address field and then make sure that the parcel reaches this city. The local post office will finally inspect the street name and house number to deliver the parcel to the correct destination. A similar mechanism is used with IP packets. If every forwarding device needed to know how to reach all 4.3 billion IPv4 addresses, we would quickly encounter scalability and memory problems. Therefore, IPv4 (and IPv6) addresses that "belong together" are grouped into so-called IP prefixes.

An IPv4 prefix divides the 32 bits of an IPv4 address into two parts. The first several bits define the prefix or network part. The remaining bits then define the host part [26]. Often, we write the number of bits belonging to the prefix just after the IP address, using a slash to separate them. For example, 1.2.3.0/24 indicates a prefix size of 24: 1.2.3.0. To write the prefix itself, we put all host bits to 0, as in the previous example. The following notation shows the same prefix but addresses a specific host inside the prefix 1.2.3.4/24.

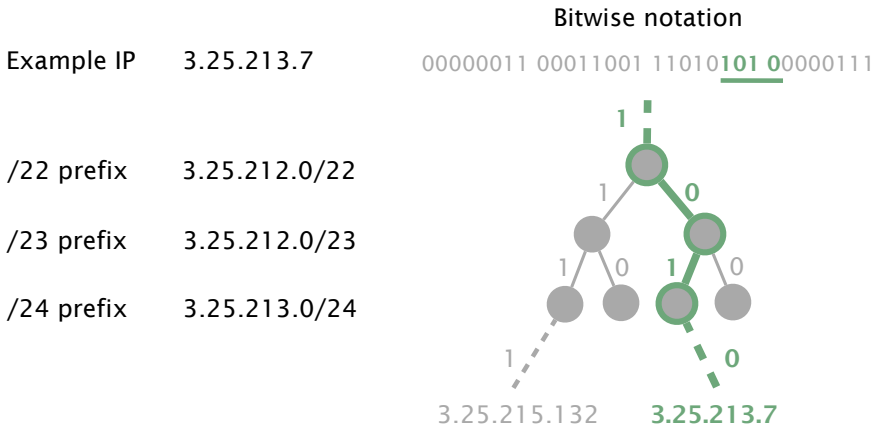


FIGURE 2.1: The IP address 3.25.213.7 belongs to prefixes with different sizes. The figure shows the matching /22, /23 and /24 prefixes and illustrates their relation when considering a bitwise notation (on the right).

Figure 2.1 illustrates three matching IP prefixes for the IPv4 address 3.25.213.7 and how they connect. In our example, the /22 prefix 3.25.212.0/22 contains (besides many more IPs) our example IP 3.25.213.7 as well as the IP 3.25.215.132. However, if we consider, e.g., the /24 prefix 3.25.213.0/24, only 3.25.213.7 belongs to it. We can also consider two extreme cases (not shown in the figure): the prefix 0.0.0.0/0 contains all 2^{32} IPv4 addresses; a /32 prefix only contains a single IPv4 address. The hierarchical structure of the IP space is especially relevant for our *Magnifier* system introduced in Chapter 3.

2.1.3 *IP routers*

In a network, IP routers play the role of post offices. A *router* is a (physical) device that contains interfaces connecting to other network devices. A router (and other network devices) contain three "planes" [27]: the data, control and management plane. The data plane forwards incoming packets to the correct outgoing interface and executes simple but highly optimized operations. It manages the data-plane traffic, i.e., packets containing application traffic such as video or website data. The control plane often runs in general-purpose hardware and can therefore perform more complex algorithms. It mainly updates the data-plane behavior according to incom-

ing control-plane traffic, e.g., from routing protocols (compare Section 2.3). Finally, the management plane allows network operators to configure and monitor the behavior of the control plane.

A fundamental data-plane component of a router is its forwarding table, a long list of IP prefixes together with their corresponding next hop. The next hop indicates where to forward matching packets such that they eventually reach the intended destination. A router performs longest-prefix matching to find the best entry for an incoming packet. In other words, whenever the router receives an IP packet, it searches through the forwarding table and selects the most specific prefix to which the packet's destination IP belongs. Finally, it forwards the packet to the corresponding next hop. The next router on the path repeats this process with its own forwarding table.

We again look at the example in Figure 2.1 and assume that a router has two entries in its forwarding table: one for prefix 3.25.212.0/22 and one for prefix 3.25.213.0/24. If the router now receives a packet with destination IP 3.25.213.7, it will forward it to the next hop belonging to prefix 3.25.213.0/24. Note that the IP would also match the /22 prefix, but the /24 prefix is longer or more specific. If we receive a packet with destination IP 100.5.5.5, for which the router does not have any matching entry, it will drop the packet.

2.2 INTERNET STRUCTURE

The Internet is often called a network of networks. It connects globally distributed entities consisting of various network devices (e.g., routers and switches) and end hosts (e.g., computers, smartphones, or sensors).

2.2.1 *Autonomous Systems and Internet Service Providers*

An important entity in today's Internet is a so-called Autonomous System (AS). An AS owns at least one IP prefix, is operated by a group of network operators, and, importantly, should implement a single routing policy that applies to the entire AS [28]. To identify an AS, each one has a unique number. For example, the SWITCH [29] network, which connects multiple Swiss universities (including ETH Zürich), has AS number 559 [30]. Currently, there are around 74k ASes in the Internet [30] which together advertise nearly one million IPv4 prefixes [31].

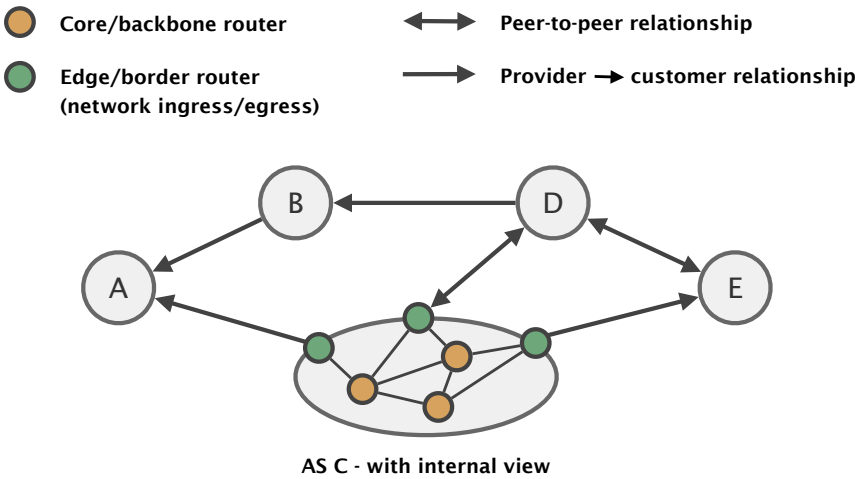


FIGURE 2.2: A simple Internet with five ASes. AS *C* shows an internal view and distinguishes between core and edge routers.

Figure 2.2 shows a simplified Internet consisting of five ASes (*A*, *B*, *C*, *D*, *E*). AS *C* illustrates how the internal structure of an AS could look like. We distinguish between core (or backbone) routers that forward traffic internally and edge (or border) routers that connect to at least one other AS. Edge routers also build the *ingress* and *egress* points for traffic transiting through the AS.

Some ASes actively provide services and Internet connection to other ASes and customers. We call these ASes Internet Service Providers (ISPs). For example, a well-known ISP in Switzerland is Swisscom [32] which has AS number 3303. Customers pay ISPs with the intention that their traffic can reach any destination in the Internet. ISPs are sometimes classified based on a 3-tier model [33]. Important for us are Tier-1 ISPs (Chapter 3) representing the largest ISPs with global presence and infrastructure. However, a single ISP (or AS) can only reach *some* Internet destinations directly and relies on other ASes for global connectivity.

2.2.2 *Peering connections*

To reach external destinations, an AS interconnects with other ASes; it *peers* with them. Not every connection is equal, though, and we can distinguish different peering types [34]. In this dissertation, we are mostly interested in the two main types: peer-to-peer and customer-to-provider connections. Many more exist in the Internet and a categorization to one type is not always obvious.

Peer-to-peer connections In a peer-to-peer connection, both ASes treat each other as equal and agree to forward specific traffic for free over the peer-to-peer connection. In Figure 2.2, we indicate such connections with double-headed arrows. For example, AS C has a peer-to-peer connection with AS D.

Customer-to-provider connections In a customer-to-provider connection, one AS, usually the bigger one (i.e., the AS reaching more destinations), acts as a provider and provides connection to its customer (the smaller AS). In exchange for the increased number of ASes the customer can reach, it pays the provider according to the amount of forwarded traffic. In Figure 2.2, we indicate customer-to-provider connections with single-headed arrows. The arrow points from the provider to the customer. For example, AS C is the provider of AS A.

2.3 BORDER GATEWAY PROTOCOL AND INTER-DOMAIN ROUTING

As we saw in the previous section, each AS owns at least one prefix and distributes the corresponding IPs to its endpoints and network devices. In a first step, network operators must achieve internal connectivity between the different endpoints in one AS. For that, they use intra-domain routing. One commonly used intra-domain protocol (more exist) is Open Shortest Path First (OSPF) [35]. This dissertation mainly focuses on global Internet paths. As such, we will not discuss intra-domain routing in detail.

In a second step, operators of one network need to ensure that they *(i)* can reach prefixes in other ASes; and *(ii)* that other ASes can reach their own prefixes. For that, we need an inter-domain routing protocol. In today's Internet, the *only* inter-domain routing protocol is the Border Gateway Protocol (BGP).

2.3.1 *General functionality*

The Border Gateway Protocol (BGP) [20] is a distributed routing protocol that exchanges advertisements about IP prefixes between ASes (inter-domain). It also distributes the received advertisements inside an AS. BGP is a so-called path-vector protocol and performs its decisions taking the IP prefix, its current AS path, and custom policies into account. Network operators define the routing policies. They filter incoming BGP advertisements and define which advertisements a router propagates to a neighbor.

Before two routers advertise prefixes between each other, they must establish a session. Network operators configure their devices accordingly. Two routers in the *same* AS build an internal BGP (iBGP) session. If the routers belong to *different* ASes instead, they establish an external BGP (eBGP) session. Once the session is established, two BGP routers advertise prefixes via BGP update messages.

BGP update message An update message [20] is part of the control-plane traffic and interacts with a router's control plane. It either *advertises* or *withdraws* one or multiple IP prefixes. Besides the IP prefix(es) and other features, an update message contains: *(i)* the AS path; *(ii)* a next hop; and *(iii)* several BGP attributes.

The AS path *(i)* provides information about the current AS path towards the origin of the advertised IP prefix. A BGP router uses the AS path as part of its decision process (discussed in the following subsection) and to detect forwarding loops [20]. Whenever a router receives a BGP update from a neighboring AS, it first inspects the AS path and looks for its own AS number. If present, the router drops the update message and does not consider the advertised prefix. Otherwise, the router adds its own AS number to the path before advertising the route to a neighboring AS.

The next hop *(ii)* defines the IP address belonging to the next router on the path to eventually reach the origin of the advertised IP prefix(es). Routers store the next hop in their forwarding table together with the matching IP prefix.

Finally, the various BGP attributes *(iii)* interact with the configured policies and influence BGP's decision process (discussed next).

#	criteria	prefer route ...
1	local preference	with higher local preference
2	AS path length	with shorter AS path length
3	MED	with lower Multi-Exit Discriminator (MED)
4	eBGP vs. iBGP	received via eBGP over iBGP
5	IGP metric	with lower IGP metric towards next hop
6	tie break	with lower egress IP address

TABLE 2.1: BGP considers multiple criteria when deciding between two routes for the same prefix. The table orders them from highest to lowest priority. An actual implementation may contain additional or vendor-specific criteria.

2.3.2 *BGP's decision process*

Let us assume a router receives a BGP update for prefix P while it already knows a route towards P, for example, over a different AS. In such a case, BGP always selects a *single* best route which is then stored in the forwarding table and advertised to other BGP neighbors (if the configured policies allow it). To select the best route, BGP uses a "decision algorithm" [20, 36].

Table 2.1 highlights the most critical steps in the decision process. The most important decision feature is the local preference value, configured by network operators inside one AS. If both routes have the same local preference value, BGP will pick the one with the shorter AS path instead. A shorter AS path could (but does not have to) indicate a shorter travel time towards the destination. In case of a tie, the lower MED value is preferred, should both routes come from the same neighboring AS. Afterwards, BGP prefers routes learned via eBGP. As a result, the corresponding traffic will exit the local AS faster, as it does not need to travel internally to another router (i.e., a route learned via iBGP). Similarly, the next step, the lower Interior Gateway Protocol (IGP) metric, will prefer the route over the "shortest" internal path. If both routes still have the same priority at this point, BGP decides based on the lower egress IP address (a random tie-breaking mechanism).

2.3.3 *Common inter-domain routing policies*

Network operators influence BGP's best route selection process in at least two ways. Either they configure filters that allow or prevent specific routes from entering or exiting the network, or a router modifies route attributes (for example, by prepending ASes to the AS path) such that *other* routers in the same or another AS handle them differently. Note that the filtering approach leads to deterministic results but only works locally, while some attribute changes *could* also influence other networks' decisions.

In Section 2.2.2, we introduced the most common peering relationships. Let us look at them from a money perspective. An AS earns money if it receives traffic from a customer, does neither earn nor lose money when receiving traffic from a peer (but still has to forward the traffic in the network), and loses money if it receives traffic from a provider. At the same time, customers pay to ensure their traffic reaches all the advertised destinations, so their traffic should be handled carefully.

ASes usually maximize their earnings while minimizing the amount of forwarded traffic. To this end, operators define filters and rules to achieve specific traffic flows between their customers, providers, and peers [34]. Routes they receive from their customers are advertised to all neighbors, no matter their peering relationships. This way, everyone can reach the customer's prefixes. Similarly, the operators will also advertise their own prefixes to everyone. However, routes coming from peers or providers are *only* advertised to their customers. As a result, customers can forward traffic to all destinations while, e.g., a peer only knows some of the routes. Finally, the operators also configure different priorities (e.g., by using local preferences): customer routes > peer routes > provider routes. That means, should the AS receive a route for the same prefix from multiple neighbors, it will prefer (BGP decision process) the cheapest one.

Figure 2.3 shows two examples. AS E receives a BGP route for a prefix of AS A over C and D. It prefers the route over AS D as it has a peering relationship with D ("cheaper" than the provider relationship with AS C). In the reverse direction, AS A receives a BGP route for a prefix of AS E also over two neighbors, B and C; both are providers. The BGP decision process now prefers the shorter route. The AS path over C [C E] is shorter than the AS path over B [B D E]. It is important to note that: *(i)* the BGP updates/routes and the forwarded traffic flow in opposite directions. For example, AS A receives a route *from* C and therefore forwards matching traffic *to* C (next

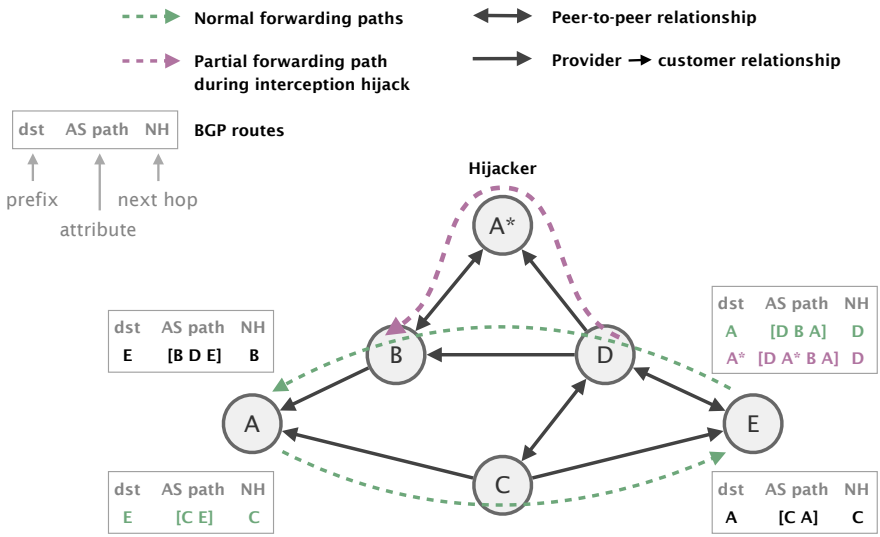


FIGURE 2.3: The green arrows show the forwarding path for traffic between AS A and E according to the selected BGP routes. AS A^*, a malicious entity, starts an interception hijack for a sub-prefix belonging to AS A resulting in a forwarding path change.

hop); *(ii)* the traffic from sources in AS A towards destinations in E does *not* follow the same path as the traffic from AS E to A. Asymmetric traffic forwarding is also commonly observed in the real Internet [37].

2.3.4 *BGP hijacks (malicious attacks)*

As we saw in Section 2.2, each AS owns at least one IP prefix, which it might advertise over BGP. However, the BGP protocol itself does not care about prefix ownership. Any AS can advertise any prefix, even if they do not own it or know how to reach the advertised destination. This can lead to large-scale connectivity issues [38] and, even more problematic, be abused by malicious entities to perform so-called BGP hijacks [39]. Hijacks are especially relevant for our *Oscilloscope* system discussed in Chapter 4 and constitute only one of many attacks on BGP, given that BGP was not designed with security in mind [40].

In a BGP hijack, a malicious AS starts to advertise a prefix that it does not own to attract specific traffic that normally does not reach the malicious AS. Consequently, services using the hijacked destination might be disrupted, or the malicious AS can inspect sensitive data (e.g., traffic towards a financial institute). We identify multiple main components in a BGP hijack: *(i)* a benign AS which rightfully owns and advertises a specific prefix; *(ii)* a malicious AS which starts to hijack this prefix (with the same or a different prefix size); and *(iii)* at least one (often many more) ASes that change their forwarding decisions towards the advertisement of the hijacker (the victims of the hijack).

We distinguish different hijack types. A hijack either blackholes or intercepts traffic:

Blackhole attack In a blackhole attack, the malicious AS drops all the hijacked traffic. This is "easy" to perform but makes the hijack very visible as all corresponding flows start to terminate.

Interception attack A more involved BGP hijack is a so-called interception attack. The goal is that the malicious AS knows at least one path *not* affected by the hijack. It will forward the hijacked traffic over this path back towards the benign AS/destination. As a result, hijack detection is much more complex, as users might not even realize that their traffic is being intercepted. Figure 2.3 shows a simple example. AS A^* performs an interception hijack for a sub-prefix belonging to AS A. However, it crafts the malicious BGP hijacks to only affect AS D and E (the victims). AS A^* can now use the path over AS B to forward the intercepted traffic to its benign destination, AS A.

Similarly, the hijacker can advertise the same or a more-specific prefix to attract traffic from hijacked victims.

Same-prefix attack In a same-prefix attack, the hijacker advertises the same prefix (i.e., same IP and prefix length) compared to what the benign owner advertises. In such a scenario, some ASes will receive both advertisements and use the BGP decision process to select one. As a result, the hijack affects only a part of the Internet.

More-specific prefix attack In a more-specific prefix attack, the hijacker starts to advertise one or multiple prefixes that are more specific (i.e., have a longer prefix size) than what the benign AS currently advertises. Given that this is a new prefix, it should propagate throughout the Internet. Additionally, as introduced in Section 2.1.3, routers perform a longest-

prefix match to select the best-matching route. As such, the hijacked prefix is preferred over the benign one. Note that a less-specific prefix attack, although possible, will typically not result in any hijacked traffic, at least as long as the benign (more-specific) prefix(es) are still advertised.

2.4 TRANSMISSION CONTROL PROTOCOL AND ROUND-TRIP TIME

In this section, we will briefly introduce the Transmission Control Protocol (TCP) and then mainly focus on Round-Trip Time (RTT) measurements on top of TCP. RTT measurements are essential for our *Oscilloscope* system (Chapter 4).

2.4.1 *Transmission Control Protocol*

The Transmission Control Protocol (TCP) [41, 42] is a reliable transport protocol. TCP provides reliable end-to-end delivery over the Internet, which, by design, only provides best-effort delivery. That means packets might be lost, reordered, or duplicated. TCP's logic runs on the two end hosts, which communicate with each other. They exchange related information over the TCP transport header, which follows the IP header discussed in Section 2.1.

SEQ and ACK numbers TCP endpoints send and receive a stream of bytes and exchange data in both directions simultaneously. The SEQuence number (SEQ) keeps track of the currently transmitted byte number and is part of the TCP header. To achieve TCP's reliability guarantees, received data is eventually acknowledged with an ACKnowledgment number (ACK), again part of the TCP header. The receiver sends the ACK back to the sender. TCP transmits the entire header in clear text. As a result, an on-path observer, which sees a data packet and the corresponding ACK, can match SEQ and ACK numbers.

2.4.2 *General RTT introduction*

Compared to one-way delay measurements (how long does it take from point A to B), the Round-Trip Time (RTT) indicates how long it takes for a packet to travel from one endpoint to another *and back*. Given that RTT measurements are closely related to the distance between the two endpoints

and the current path occupancy, they reveal many path-related properties, as discussed in detail in Chapter 4.

Many *end users* perform RTT measurements when they use the well-known ping [43] tool, for example, to estimate their current latency or to determine if a given IP address is reachable. Ping actively sends a new packet towards the indicated destination and triggers a reply, assuming no device on the path blocks the ping packets [44]. We can then compute the time difference between the two packets to estimate the RTT. Note that most ping implementations do not use TCP, but rather the Internet Control Message Protocol (ICMP) [45].

TCP *endpoints* use RTT estimations to compute their retransmission timers [46], i.e., the time to wait before retransmitting a potentially lost packet.

2.4.3 *On-path TCP-based RTT estimations*

Besides active approaches, e.g., the previously mentioned ping tool, network operators perform passive, on-path RTT estimations. The key idea is to compare the observation time of a packet going in one direction with the observation time of a matching reply in the reverse direction. Note that this necessitates the observation of both traffic directions and the ability to identify matching packets. One approach uses the visible SEQ and ACK numbers in TCP headers. We can subtract the two observation times and estimate the RTT by finding a matching ACK for a given data packet. Highly optimized approaches [47] run directly in the data plane at line rate.

Another approach focuses on the TCP timestamp options [48], an optional field in the TCP header. Similarly to the method based on SEQ and ACK fields, it allows on-path observers to find matching packets between the two visible traffic directions, as explained in detail in [49]. Note that these two approaches will not always result in the same RTT estimate for the same TCP flows, for example if a packet is lost.

3

VALIDATED TRAFFIC INGRESS AND EGRESS INFERENCES WITH MAGNIFIER

In this chapter, we introduce *Magnifier* [5], a system that helps network operators to infer and validate *internal* path properties, namely the ingress and egress points of traffic in their network. This knowledge is a fundamental first step to reasoning about the full, internal forwarding path and, more complex, *external* path property inferences. Our poster [1] shows early ideas going in the same direction.

Monitoring transit traffic, for example, 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 support 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; and traffic from the same source prefix often enters via multiple ingresses. More 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. Deutsche Telekom's IP network, for example, reports a transit capacity exceeding 30 Tbps and IP throughput of over 3500 PB per month [50].

Two common techniques to collect data-plane measurements are packet sampling and traffic mirroring. Both have advantages and disadvantages, making them suboptimal for inferring traffic ingresses and egresses.

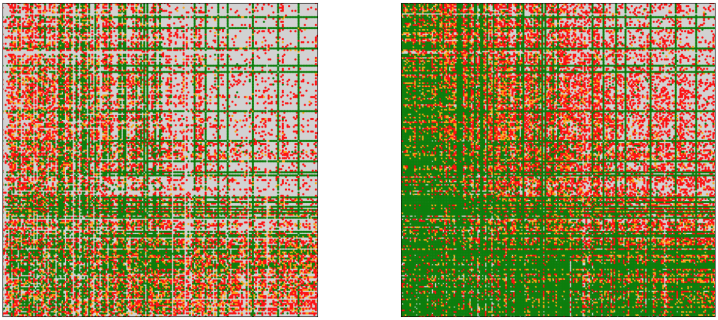
Sampling-based approaches such as NetFlow [51] or sFlow [52] provide good coverage at the expense of precision and correctness. Often only a few flows are sampled, and even fewer are sampled at both the ingress and egress. We confirmed this by analyzing a 5-minute slice of NetFlow

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 Figure 3.1a locates only 22% of sampled flows; we waste the information from all other sampled flows.

Mirroring-based approaches [7, 53] 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 [54] or in-band telemetry [55, 56] also excel at gathering precise information but can only do so for a specific traffic share.

We ask ourselves whether we can combine the benefits of sampling and mirroring to mitigate their respective drawbacks and infer internal path properties. We answer this question positively and present *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 [56]. 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*. Figure 3.1b 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 (Figure 3.1a).

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 as sentinels are most often correct. Thanks



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

FIGURE 3.1: By inferring ingress or egress points of sampled flows, *Magnifier* significantly improves the network-wide coverage (Figure 3.1b) compared to using sampling only (Figure 3.1a). 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.*

to mirroring, *Magnifier*'s ingress/egress inferences are **guaranteed correct**; a key feature of our design and an essential difference from other monitoring and path inference tools. Mirrored traffic reveals inference errors or traffic shifts in sub-seconds, allowing *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., Figure 3.1) while providing correctness guarantees.
- We implement *Magnifier* [57], 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 (Section 3.5).

- We discuss (Section 3.3.2) and evaluate (Section 3.5.2) 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 (Section 3.5.4).

3.1 OVERVIEW

This section introduces the problem statement (Section 3.1.1) and *Magnifier*'s main building blocks (Section 3.1.2). Finally, we illustrate *Magnifier*'s behavior on a simple example (Section 3.1.3).

3.1.1 *Problem statement*

Can we combine the benefits of sampling and mirroring to design an *easy-to-deploy* system that infers *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.

3.1.2 *Building blocks*

Magnifier extends the coverage of ingress/egress observations using a two-step approach (Figure 3.2): based on sampled data, it first *infers* the missing traffic ingress and egress points, then it *validates* them using mirroring.

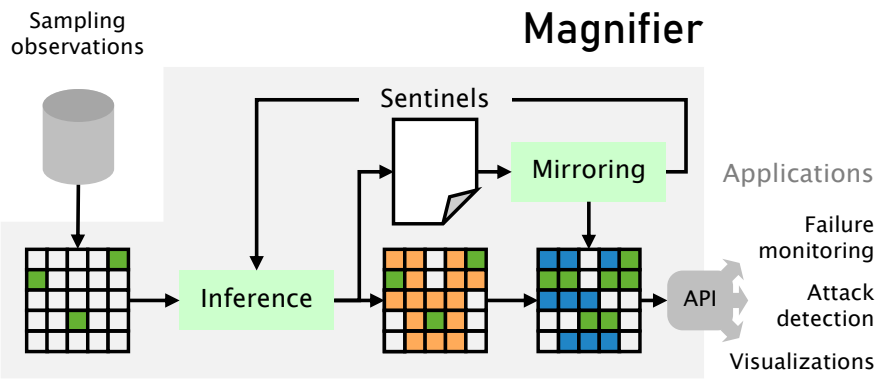


FIGURE 3.2: *Magnifier* uses sampled data to infer sentinels that predict IP subnets' ingress or egress points. *Magnifier* then validates sentinels at runtime using packet mirroring. This way, we can greatly extend the coverage of traffic ingress/egress observations usable by many applications.

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 that *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 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 observations (compare Figure 3.1). 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 an ingress or an egress, or carry no traffic at all. Otherwise, traffic is mirrored, which provides

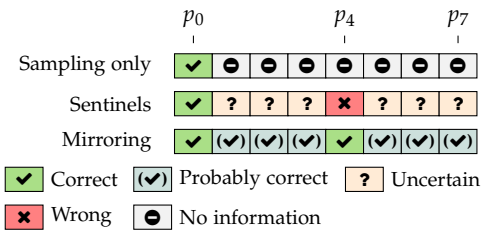


FIGURE 3.3: Sampling provides information about the sampled prefixes only. The sentinel inference extends the coverage, but it is uncertain and can make wrong assumptions without any means to detect them. With mirroring, these inferences can be validated, leading to either correct or probable inferences.

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).

3.1.3 *Illustrative example*

While mirroring rules generate additional traffic, they are essential to *Magnifier*, illustrated with an example (Figure 3.3): 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, 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.

3.2 INGRESS & EGRESS IDENTIFICATION

In this section, we define the notion of "sentinels" and present an efficient algorithm to find them (Section 3.2.1). We then discuss sentinel subnet size tradeoffs (Section 3.2.2) and finally show how *Magnifier* uses these sentinels to match ingress and egress observations in sparsely sampled data (Section 3.2.3).

3.2.1 *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 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

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[IP_new] ← ((IP >> (32 − S)) << (32 − S))
    aggregated ← table.groupby(IP_new)[device]
    result[n] ← nunique(aggregated[device])
    sentinels += (result[n] == 1)
    table −= sentinels
end for
return sentinels
```

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. Current IPv6 allocation strategies, for example explained in [58], are favorable for *Magnifier*. The same AS tends to be allocated large IP blocks that we can use as sentinels.

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.

3.2.2 *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. Figure 3.4 shows a simple example. The net-

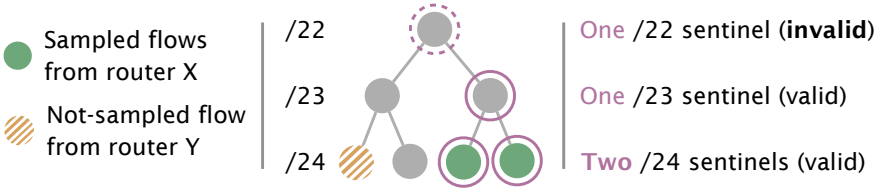


FIGURE 3.4: The sentinel amount and coverage depend on the subnet size. The "largest" /22 sentinel is invalid, whereas one /23 sentinel or two /24 sentinels are valid.

work forwards traffic from three different /24 subnets. We collect samples from the 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 (Section 3.3.1), *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 [59]. As a consequence, the search sizes also influence the number of required validation mirroring rules (Section 3.3.1) and, therefore, the required router resources.

3.2.3 *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. Section 3.5.4 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.

3.3 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 (Section 3.3.1). 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 (Section 3.3.2).

3.3.1 *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 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 (for example ERSPAN [60]) 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., [7]) 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 [61] 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 Section 3.3.2.

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 Section 3.5.2) 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.

3.3.2 *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 Section 3.5.2.

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.

3.4 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 and discussing 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. Figure 3.5 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 Section 3.3.2. While iteration `N-1`

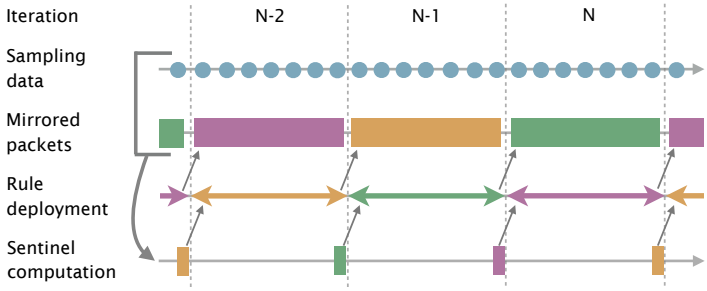


FIGURE 3.5: *Magnifier*'s control flow works in iterations based on the mirroring rule deployment time. Rules for sentinels based on `N-2` are deployed in `N-1` and active in iteration `N`.

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 Section 3.5.4 we use this API call to detect network problems based on data from a real Tier-1 ISP.

***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 to build 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.

3.5 EVALUATION

This section evaluates *Magnifier* in detail. After introducing the evaluation setup (Section 3.5.1), we first focus on *Magnifier*'s performance in simulation and on real hardware devices in our lab (Section 3.5.2). Afterward, we perform a detailed comparison with the Everflow system (Section 3.5.3) before we highlight that *Magnifier* also works with data from a real ISP (Section 3.5.4).

3.5.1 *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) [62], and a larger Nexus 7009 switch (N7K-C7009) [63]: an older (released in 2011 and no longer sold) but more resourceful model that we use for benchmark experiments.

In our labe setup (compare Figure 3.6), 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 with a sampling rate of 1/4096, the highest configurable rate on this model, i.e., we get the *most* samples. 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. However, the four emulated ingresses share the budget of 500 rules, which does not reflect a deployment on four individual devices. Therefore, we use TCAM carving [64] to increase the space for our mirroring rules to 2048 (by taking it from other features), to enable the original budget (512) per ingress. Unless otherwise specified, *Magnifier* prioritizes sentinels according to the `activity` ordering (Section 3.3.2).

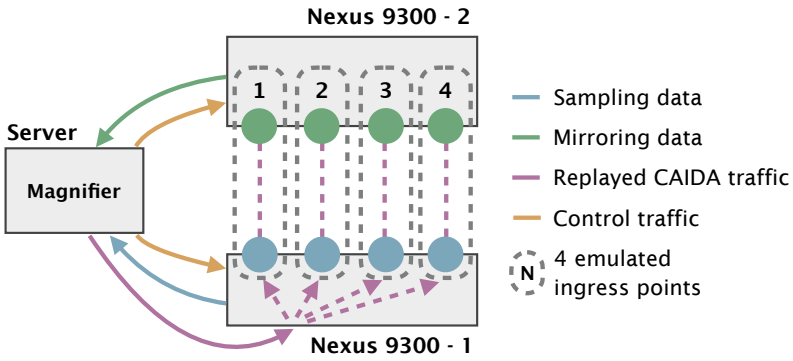


FIGURE 3.6: Two Nexus 9300 switches emulate four network ingress points. The traffic is replayed and sampled on the first switch, then forwarded to the second, which mirrors packets.

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 [57]. Unless specified differently, we always consider *Magnifier*'s iterations to be 60 s long. For the `N`-th sentinel computations, we take sampling and mirroring data from iterations `N-1` and `N-2` (compare Figure 3.5 in Section 3.4).

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 (Section 3.5.4). 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 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. In our lab setup, we use the following four slices: 1st 0.0.0.0/2; 2nd 64.0.0.0/2; 3rd 128.0.0.0/2; and 4th 192.0.0.0/2. 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*

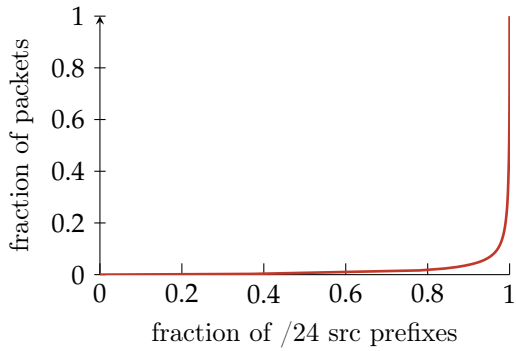


FIGURE 3.7: A CDF plot of the amount of packets observed per source /24 prefix in 60 s (one iteration at real speed) in our CAIDA trace.

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*.

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 [65] (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 [66] 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.

Figure 3.7 shows a CDF of the amount of packets observed per source /24 in 60 s of our CAIDA trace used in the evaluation. 60 s 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.

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 2× 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.

3.5.2 *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 80×) *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.

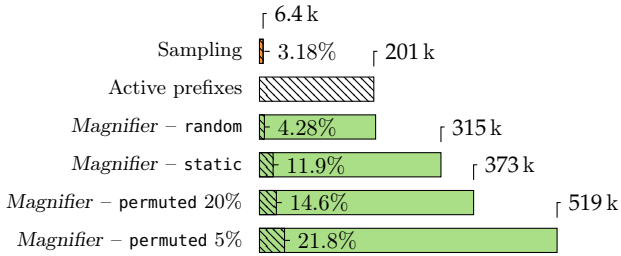


FIGURE 3.8: Amount of covered /24 source prefixes by *Magnifier* and sampled data assuming unlimited mirroring resources. *32 border routers, 1/1024 sampling rate, and real replay speed.*

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

Per-prefix results Figure 3.8 shows the per-prefix coverage with 32 border routers, 1/1024 sampling rate, and real-time replay speed. Sampling covers ≈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 ≈200k (`random`), ≈315k (`static`), ≈370k (`permuted` 20%) and ≈520k (`permuted` 5%) respectively. Second, *Magnifier* covers prefixes that are currently active in the CAIDA traces (dashed boxes). The active prefixes increase from ≈4% (`random`) up to ≈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

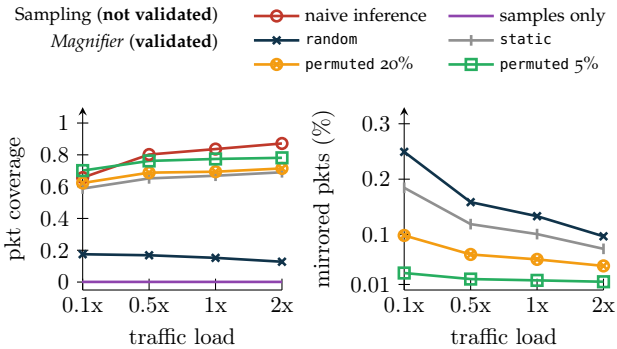


FIGURE 3.9: Amount of covered packets and mirrored traffic for different assignment strategies and inferences based on sampled packets only. *32 border routers and 1/1024 sampling rate.*

prefix space that is not sampled using sentinels of reasonable sizes – hence the small % of active prefix coverage (Figure 3.8, 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 Figure 3.9 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 ≈20% (`random`) up to ≈80% (`permuted` 5%) which can be surprising given the lower active prefix coverage (Figure 3.8). 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 60 s trace data (Figure 3.7). 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, Figure 3.9 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 Figure 3.9 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.

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%.

Additional plots For completeness, we also show additional plots which further evaluate *Magnifier* in terms of packet coverage and mirrored traffic.

Figure 3.10 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 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.

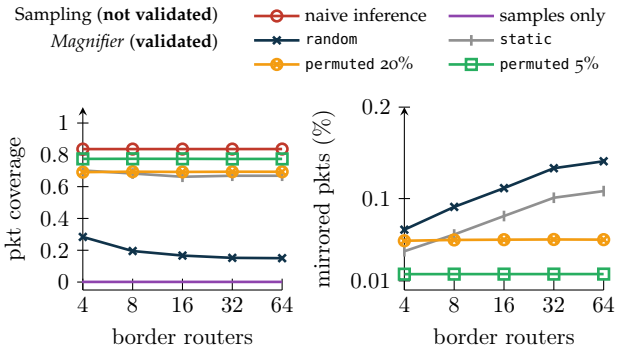


FIGURE 3.10: Simulation results for coverage and mirrored traffic when *Magnifier* runs with different amounts of border routers. CAIDA traces replayed at real speed, sampling rate 1/1024.

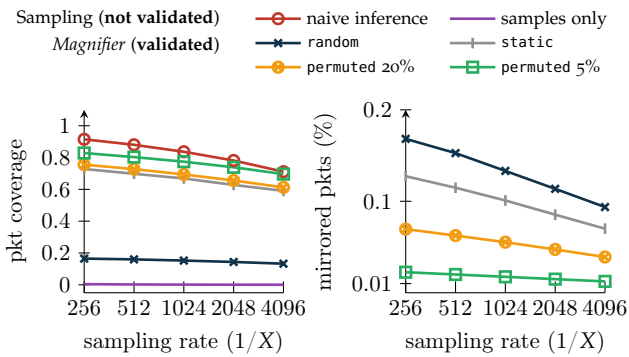


FIGURE 3.11: Simulation results for coverage and mirrored traffic when *Magnifier* runs with different sampling rates. CAIDA trace replayed at real speed, 32 border routers.

Figure 3.11 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.

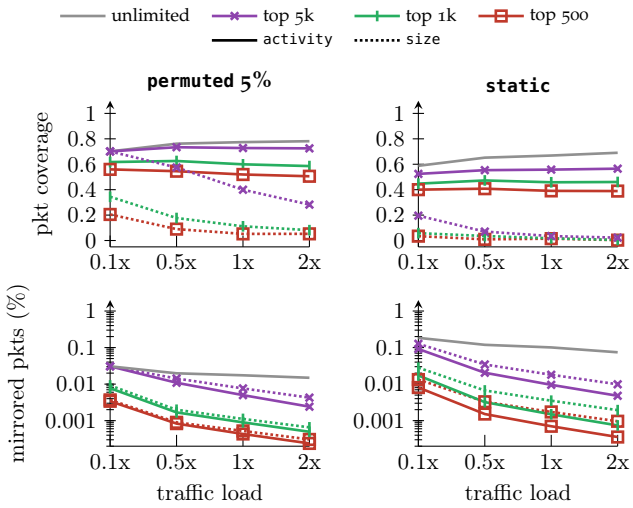


FIGURE 3.12: Coverage and mirrored traffic amount for different top sentinels ordered by `activity` or `size`.

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 (Section 3.3.2): `activity` (covering most sampled packets) and `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 Figure 3.12 compares *Magnifier*'s per-packet coverage achieved with different numbers of validated sentinels: 500, 1k, 5k and unlimited; using the same settings as in Figure 3.9. We show results for the `permuted` 5% (left) and `static` (right) assignment strategy, additional plots can be found towards the end of this subsection.

More validated sentinels achieve a higher coverage and generate more mirrored traffic. The top `size` sentinels have the highest chance of being

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 3.1: 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.

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, 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 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 (Figure 3.12), but in a large per-prefix coverage (Table 3.1). 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 sentinels.

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

Additional plots Figure 3.13 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 Figure 3.12, the

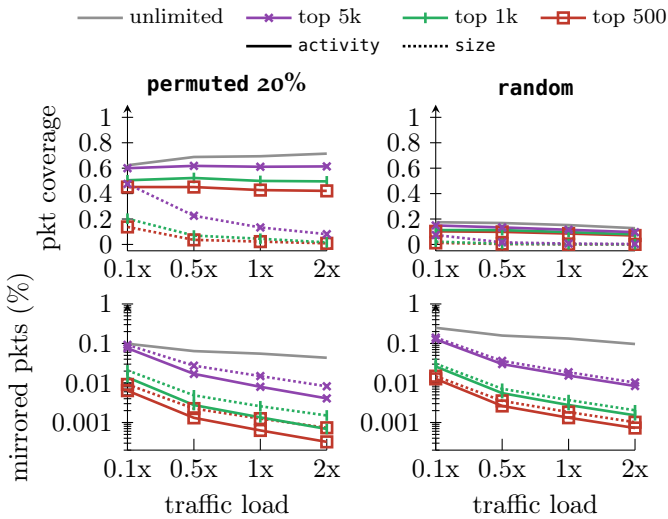


FIGURE 3.13: Simulation results for coverage and mirrored using different amount of top sentinels according to a `activity` and `size` ordering. We show the `random` and `permuted` 20% traffic assignment strategies (compare with Figure 3.12). The plots show values for different traffic replay speeds of the CAIDA trace with 32 border routers and sampling rate of 1/1024.

`activity` strategy provides better coverage than `size` and a lower amount of mirrored traffic.

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.

Results Table 3.2 shows the number 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

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

TABLE 3.2: Number of changed sentinels between iterations for different assignment and sentinel ordering strategies.

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.

Conclusion The top sentinels often change between iterations, however *Magnifier* is not really impacted by that. As we describe in Section 3.3.1, *Magnifier* works with two ACLs and switches between them. While one is active, the other one gets populated. The frequent sentinel changes between iterations are therefore not a big problem as we anyway need to build a completely new ACL.

Comparison with the lab setup

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

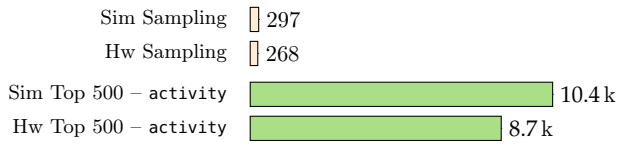


FIGURE 3.14: Covered /24 source prefixes by *Magnifier* and sampled data in simulations and on Nexus 9300 switches. *`random` assignment, 1/4096 sampling rate, and 0.1× replay speed.*

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 60 s. *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× replay speed.

Results Figure 3.14 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 (Section 3.5.1), 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: ≈10.4k prefixes in the simulation, ≈8.7k prefixes on the hardware. We also have to consider that 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 60 s), *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.

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 3.3: Mirroring rule deployment times.

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 [67].

Micro-benchmarks

In the following subsection, we 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 [64]). On the Nexus 7009, we can deploy ≈32k rules using one TCAM bank and ≈128k 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 3.3 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 Section 3.5.2 (500 rules for 4 ingresses on one device). We expect that the rule deployment time will continue to decrease with more powerful/newer devices.

Note that even if we cannot activate 5k+ rules on the Nexus 9300 switch with the current TCAM carving (Section 3.5.1), we can still deploy them. Overall, these results confirm *Magnifier*'s design (Figure 3.5) 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 (Section 3.4). We repeated the experiment ten times. The mean deactivation time is 1.65 s (min: 1.62 s, max: 1.73 s). 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.

3.5.3 *Comparison with Everflow*

We compare *Magnifier* with Everflow [68] 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.

Implementation details Everflow uses packet mirroring to produce its random packet samples. The paper [68] 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 fewer 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 comparison with *Magnifier*, we did not implement this feature in our simulation code. The same holds for Everflow's controller, storage and reshuffler components.

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. Additional plots can be found towards the end of this subsection.

Results Figure 3.15 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 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 Figure 3.15). 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 ≈80% (unlimited) and ≈60% (top 1k) coverage as we only have randomly sampled packets as input. Despite that, *Magnifier* manages to reach good coverage values, with

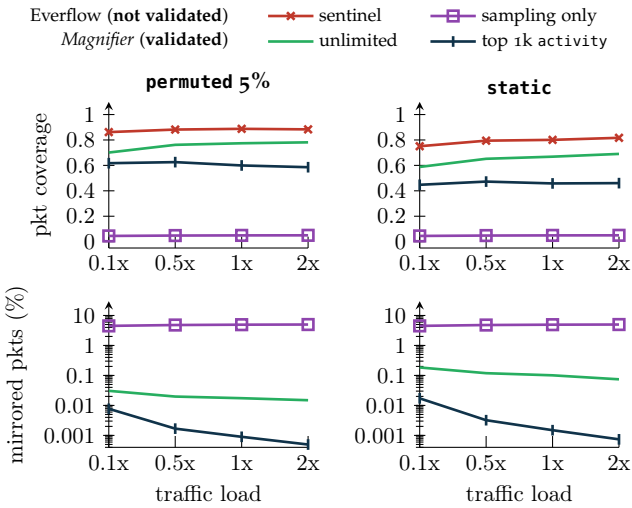


FIGURE 3.15: Comparison of coverage and mirrored traffic for *Magnifier* and Everflow under different traffic loads.

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 Figure 3.9 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 Figure 3.15). 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 ≈5% of all traffic while *Magnifier* is more than one magnitude lower (≈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.

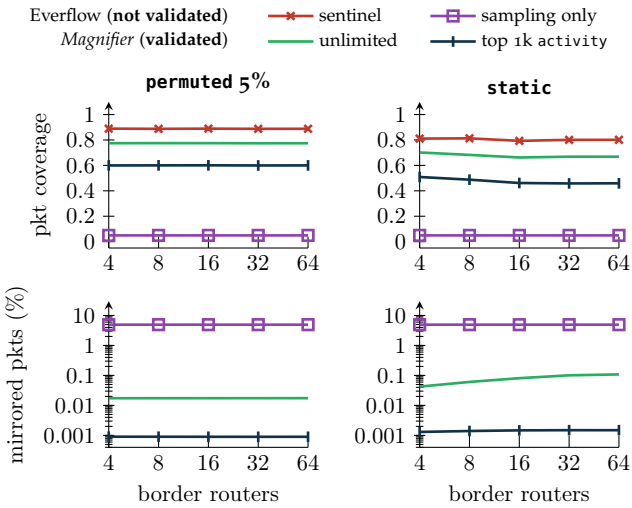


FIGURE 3.16: Comparison of coverage and mirrored traffic for *Magnifier* and Everflow for different amounts of border routers. We show the `static` and `permuted` 5% traffic assignment. We replay the CAIDA trace at real speed and use a sampling rate of 1/1024.

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 corresponding TCP flags. In terms of mirroring rules, Everflow only needs around 20 of them [68]. *Magnifier* needs more mirroring rules but also uses them for *validation* – something that Everflow cannot achieve.

Additional plots In this section we show additional comparison plots between *Magnifier* and Everflow. Figure 3.16 shows different number of ingress routers while Figure 3.17 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 (Figure 3.17) have more impact on *Magnifier*

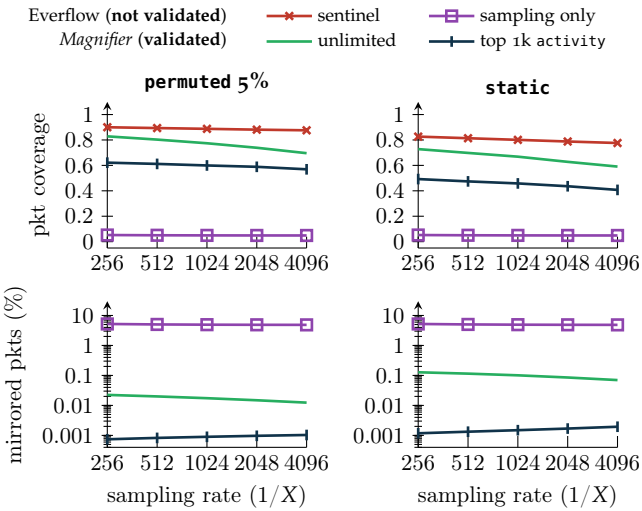


FIGURE 3.17: Comparison of coverage and mirrored traffic for *Magnifier* and Everflow for different sampling rates. We show the `static` and `permuted` 5% traffic assignment. We replay the CAIDA trace at real speed and distribute traffic over 32 simulated ingresses.

than on Everflow. *Magnifier*'s performance is tightly related to the amount and distribution of the randomly sampled packets.

3.5.4 *Sentinels in Tier-1 dataset*

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

Existence of sentinels

Setup We divide our Tier-1 dataset into 30 s 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.

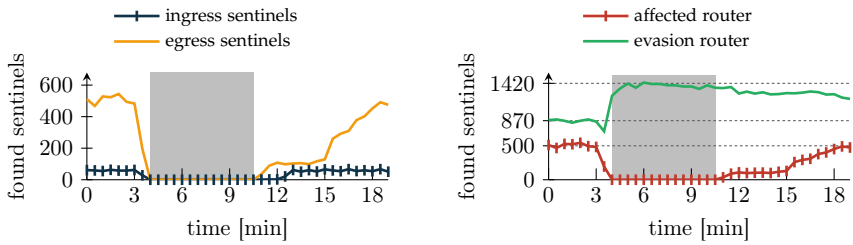


FIGURE 3.18: A temporary router outage (gray block) decreases the number of found sentinels (left) while we see similar increases on a close router (right).

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.

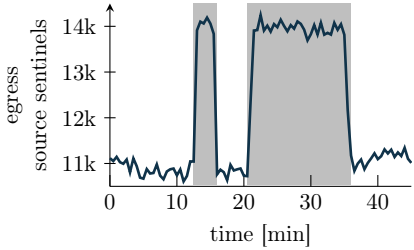
We observe that we find more ingress than egress sentinels. This results from the typical forwarding behavior observed in an ISP: traffic from each of the ISP's customers, which own specific prefixes, tends to enter via a single ingress point, which leads to a high number of ingress sentinels. At the 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.

Per-device sentinel changes

Setup We divide our Tier-1 dataset into 30 s 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.

Results Figure 3.18 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



Normal		DDoS-like	
prefix	flows	prefix	flows
P1	70	DDoS1	3700
P2	63	DDoS2	3560
P3	56	P1	72
P4	55	P2	65
P5	49	P4	47

FIGURE 3.19: A sudden burst of egress source sentinels (left, gray blocks) is likely to result from a DDoS-like event (right).

of egress sentinels of a geographically-close router increases shortly after and closely matches the number of lost sentinels.

Figure 3.19 (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.

3.6 RELATED WORK

Sampling-based network monitoring Many systems use NetFlow [51], J-Flow [69], sFlow [52] or related flow extraction tools for network measurements. Sampling suffers from a fundamental trade-off between coverage and accuracy. For example, Teixeira et al. [70] use NetFlow data to detect egress changes due to BGP hot-potato routing but are limited by the col-

lected ten-minute bins. In addition, Cunha et al. [71] uncover measurement artifacts in two J-Flow implementations.

Consequently, several works aim to improve sampling by optimizing the collection process. Estan et al. [72] propose router software updates to dynamically adapt the NetFlow sampling rate depending on the available traffic and memory amount. FlowRadar [73] uses flow sets to count flow observations in multiple array cells, then combines and decodes these counters centrally. Similarly, Flowyager [74] 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 [7] supports query-based monitoring under a strict budget of mirrored traffic. Everflow [68] provides the possibility to mirror some packets of *every* flow, e.g., by mirroring packets with special TCP flags or debug header bits. Planck [53] 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 [75], SDN monitoring [76], on in-network analysis [77, 78].

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., [55, 56, 79–81]) or sketches (e.g., [54, 82–85]). 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 and egress points, which has been previously studied: Feldmann et al. [86] 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 router forwarding tables, which are both costly to obtain. Mahajan et al. [87] 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. [88] 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. [89] 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. [90] 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 (Section 3.5.4). With mirroring, *Magnifier* also quickly detects changes and updates its traffic matrix estimation. OpenTM [91] 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. [92] addresses the special case of SDN networks, which limits the system's applicability. Pingmesh [93] 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 in various networks.

Monitoring frameworks Several monitoring frameworks support rich sets of queries, e.g., [78, 94, 95]. In particular, Flowyager [74] 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.

3.7 CONCLUSION AND FURTHER USE CASES

Precise observations of traffic ingress and egress points are difficult to infer in large networks. This dissertation chapter showed 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 networks while only generating a small amount of mirrored traffic. The inferred ingress and egress observations are one example of *internal* path properties.

In Section 3.5.4, we already showed two direct use cases of *Magnifier*. Based on sampled NetFlow data from a tier 1 ISP, *Magnifier* detects device outages and possible DDoS attacks. However, *Magnifier*'s precise and validated ingress and egress inferences also constitute valuable inputs to other systems, which either use them as input parameters or perform further analysis on top of them, as we highlight in the following paragraphs.

***Magnifier*'s visual output allows for fast network health checks.** Operators can easily visualize *Magnifier*'s output in a matrix form (e.g., Figure 3.1b) or by filtering the output for specific IPs or prefixes. Discussions with operators and related work (e.g., [93]) show that visual representations are essential to quickly assess a network's health. For example, in a matrix view, *Magnifier*'s focus on ingress/egress observations shows clear patterns should a specific border router fail (i.e., rows or columns with missing/fewer observations). Additionally, we believe a visual representation helps to communicate with customers who might not be network experts. Outputs which purely focus on numbers or convoluted log messages are more difficult to understand.

Stroboscope can use *Magnifier*'s output to answer user queries. *Magnifier*'s ingress and egress observations build a valuable input to our Stroboscope system [7]. Similarly to *Magnifier*, Stroboscope uses short slices of mirrored traffic to answer user queries while adhering to a defined mirroring budget. However, unlike *Magnifier*, Stroboscope infers properties of the *entire* internal packet path. This flexibility, though, comes at the cost of mandatory user queries. As a result, operators face the following challenge: Writing precise queries which adhere to Stroboscope's mirroring budget necessitates existing pre-knowledge about *where* we should observe *which* traffic.

For this task, *Magnifier*'s observations are beneficial as they allow an operator to identify ingress and egress points in order to formulate Stroboscope input queries. One specific use case would be the following: After receiving a ticket from a customer who observes suboptimal network performance,

the operator first looks at *Magnifier*'s output to identify where the customers' packets enter or exit the network. With this knowledge, the operator then formulates an input query to Stroboscope, asking for detailed information about the full internal network path and potential packet drops.

We envision at least two different deployment strategies which allow both systems to share the mirroring resources: *(i)* they operate in turns, i.e., *Magnifier* first builds an updated ingress/egress matrix during time slot N and Stroboscope then answers specific user queries during time slot $N + 1$; or *(ii) Magnifier* permanently runs on all border routers providing up-to-date ingress/egress observations while Stroboscope runs on top of the internal (so-called backbone or core) routers inferring query-specific insights related to internal paths.

More generally, *Magnifier*'s output reveals *where* to collect data-plane signals. Generalizing the previous example, we observe that *Magnifier*'s ingress and egress observations reveal *where* to collect additional data-plane signals using other systems. This is especially important to infer *external* path properties, i.e., related to paths *outside* the network *Magnifier* is deployed in. Even better, *Magnifier*'s focus on the network border implicitly filters out unwanted noise due to paths inside our network. The following chapter shows how we use Round-Trip Time (RTT) signals collected at the network border to detect certain BGP hijacks.

4

PATH-PROPERTY-BASED HIJACK DETECTION WITH OSCILLOSCOPE

In Chapter 3, we saw how *Magnifier* infers and validates ingress and egress points of packets *inside* a single network or AS. That dramatically simplifies the process of assessing *where* operators must collect signals to gather insights about *external* path properties. Additionally, *Magnifier*'s detection of ingress or egress movements (i.e., invalid sentinels) already constitutes an interesting signal in itself. This chapter explores some relevant signals and methods to infer external path properties such as BGP hijacks.

More precisely, we first introduce the high-level idea of how BGP hijacks influence external packet paths and, therefore, one specific signal that operators collect locally, namely the Round-Trip Time (RTT) (Section 4.1). Intuitively, the observable RTT will increase if a packet needs to wait or make a detour before it reaches the intended destination. BGP hijacks might result in packet losses (i.e., lack of RTT signals) in case of a blackhole attack or an increased (decreased) RTT in case of an interception attack. We then extend the well-known RTT extraction methods in TCP flows (compare Section 2.4) with new ideas to extract RTT signals in encrypted, privacy-aware transport protocols.

Finally, we introduce our system called *Oscilloscope* (Section 4.3), which detects BGP hijacks based on changes in the RTT signal. Given that RTT signals are often noisy and a direct mapping from path properties (i.e., a BGP hijack) to an observable signal reaction is not always possible or conclusive, we show two key concepts to reduce the uncertainty: *(i)* signal aggregations based on domain-specific knowledge to strengthen the intended observations (Section 4.4); and *(ii)* statistical tests to validate some of the observations by comparing samples from different signal aggregation sets (Section 4.5). *Oscilloscope* uses both concepts. Afterwards, we evaluate *Oscilloscope*'s performance in various emulated networks (Section 4.6) as experiments in the real Internet are difficult and raise ethical questions. We conclude the chapter with a related work discussion (Section 4.7) and highlight remaining challenges and future work (Section 4.8).

Personal contributions The latter parts of this chapter mainly focus on the *Oscilloscope* system [4]. Two PhD students were working on this project: myself (Tobias Bühler), leading the project, and Alexandros Milolidakis (PhD student at KTH Royal Institute of Technology). This chapter of my dissertation is only based on paper sections that directly reflect my contributions, namely:

- 1. Introduction;
- 3. Overview;
- 4. Large-Scale Data-Plane Change Detection;
- 5. Validation and Analysis of Changes;
- 6.2. Hijack dataset;
- 7.1. *Oscilloscope*'s accuracy;
- 7.2. Classification delay;
- 7.3. Computational overhead;
- and 8. Discussion.

Alexandros Milolidakis was mainly contributing to the following paper sections:

- 2. Background, Related Work, and Motivation;
- 6.1. Mini-Internet emulation;
- 7.4. Availability of buddies;
- and Appendix A. The Stealthy Hijacker.

Results from these parts will only be cited (if needed) and are not directly used in the remaining parts of this chapter (except for the related work subsection). Our IMC poster [11] and journal paper [12] give additional information to Alexandros Milolidakis' main work.

4.1 BGP HIJACKS AND RTT CHANGES

This section highlights the general problem and motivates our BGP hijack detection approach with a simple experiment in a cloud-based setup. We conclude with a discussion of other interesting path signals.

4.1.1 *BGP hijacks are an ongoing problem*

BGP *hijacks* have plagued the Internet over the last decade, targeting critical societal services (e.g., finance services) [18], the core of the Internet infrastructure (e.g., the DNS system) [96], and citizens (e.g., for surveillance purposes) [97]. Even worse, several BGP hijacks have gone unnoticed for a long time [98]. The core problem is the lack of in-built security mechanisms in the inter-domain routing protocol, i.e., the Border Gateway Protocol (BGP), which makes launching a BGP hijack attack a simple task. Any network can falsely announce itself in BGP as the legitimate owner of any IP prefix, and all other networks will trust such an announcement. So, when the attacker's BGP announcement propagates in the Internet, some networks will reroute their traffic towards the attacker's network.

Defenses to BGP hijacks aim to prevent them entirely or detect them reliably. Existing solutions to prevent hijacks incorporate cryptographic operations into BGP to verify the exchanged messages. However, complete prevention is tough to achieve. For example, BGPSec [99] faces insurmountable deployability barriers, while RPKI [100] only protects against limited hijack types. Proposals to detect hijacks, in contrast, are often easy to deploy but have other shortcomings. Control-plane-based solutions are inherently limited by the visibility of their vantage points and cannot detect all attacks (as we show in related work [4, 11]). Data-plane-based systems (e.g., [101]) often require a large number of active probes, which leads to scalability problems, or they can only detect specific hijack types (e.g., blackholes). Instead, we envision a system that uses passively collected Round-Trip Time (RTT) samples, a signal which *(i)* is easy to extract (e.g., shown in [47]), and *(ii)* inherently connects to BGP hijacks, as the following experiment illustrates.

4.1.2 *Motivational experiment*

Performing experiments with BGP hijacks in the real Internet is challenging and quickly raises ethical questions, even if the "hijacked" prefix space is under our control. We, therefore, found the following compromise, which exposes some of the experiment traffic to the real Internet (important as our ideas focus on data-plane signals) while avoiding ethical problems.

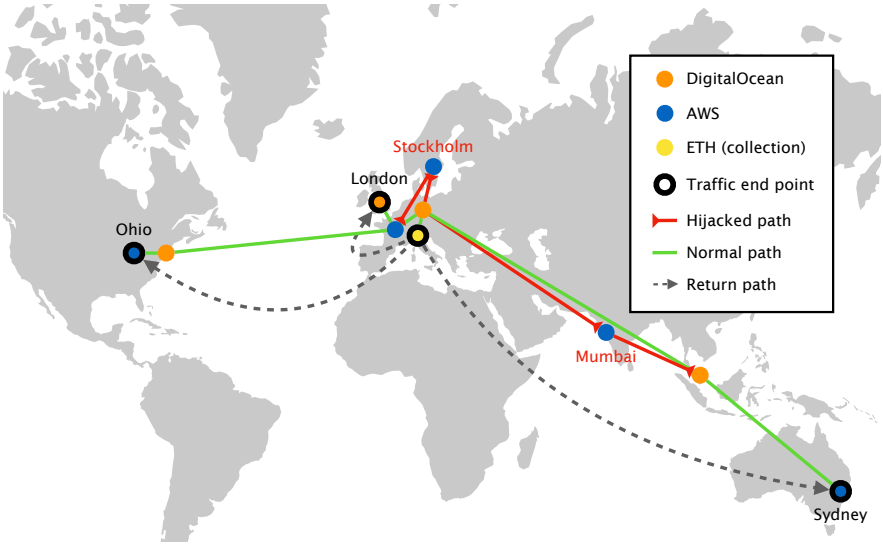


FIGURE 4.1: Multiple VMs in two different cloud providers build a global network to motivate our ideas. The VMs in Mumbai and Stockholm perform BGP interception hijacks.

Setup We deploy nine cloud VMs of two different providers (DigitalOcean [102] and Amazon AWS [103]) around the globe. Figure 4.1 shows the setup in detail. Between the different VMs, we establish so-called Generic Routing Encapsulation (GRE) [104] tunnels which encapsulate all the exchanged IP packets in another IP header. That has two advantages: *(i)* we can select arbitrary IP addresses and prefixes for the applications running inside the VMs. Their traffic is then encapsulated with the IPs given to us by the cloud providers. And *(ii)* the encapsulated packets cross the "real" Internet when they travel between VMs and might be influenced by other Internet traffic or network events.

Each VM represents a different AS, and we assign unique IP prefixes. In addition, we also have a local VM deployed on one of our servers at ETH (yellow dot in Figure 4.1). Each VM runs a FRRouting [105] software router and establishes BGP sessions with other ASes/VMs. This way, we can simulate hijack events. Our ETH VM advertises two /23 prefixes (5.6.0.0/23 and 5.6.2.0/23). Over two "malicious" ASes, in Mumbai and Stockholm, we perform interception hijack attacks for (sub-)prefixes of the IP space belonging to our VM at ETH. We artificially limit the hijacked routes' spread

so that we precisely target traffic from certain VMs (i.e., the "victims" of the hijack). Note that the tunnel-based setup prevents ethical problems. From the outside, it looks like two cloud VMs exchanging packets.

Three different VMs (in Sydney, Ohio, and London) generate application traffic with different patterns (rate-limited streaming, http request/response, telnet sessions, . . .) using the flowgrind [106] measurement tool. Each VM communicates with our VM at ETH (running flowgrind as well). On the ETH VM, we extract the observed RTTs with techniques from Section 2.4.

RTT results Figure 4.2 shows the extracted Round-Trip Time (RTT) measurements (in seconds) over roughly half a minute. Each colored RTT line represents RTT samples from different applications exchanging traffic with the specified prefix belonging to the ETH VM. As expected, we see similar RTTs for all the flows initiated by a single traffic source (labeled on the right side). London, which is (geographically) closest to our server in Zürich (Switzerland), has the lowest RTT.

In some cases (for example, the RTT peak around 27 seconds, indicated with a gray background), we observe RTT patterns visible in all the RTT samples. Most likely, they originate due to problems close to or at our local VM (a clear bottleneck in our setup). However, we also see interesting patterns that only affect selected sources or specific flows from one source. These relate to our simulated interception hijack attacks, as discussed in the following paragraphs.

Key observations During the shown time period in Figure 4.2, we performed two BGP hijacks (annotated in the figure). First ("Hijack 1"), the VM in Mumbai performs a same-prefix attack towards 5.6.2.0/23 to attract traffic from Sydney towards our ETH VM. Later ("Hijack 2"), the hijacker in Stockholm performs a more-specific hijack attack for 5.6.1.0/24 (i.e., a sub-prefix of 5.6.0.0/23) to attract some of the traffic from the Ohio and London VMs.

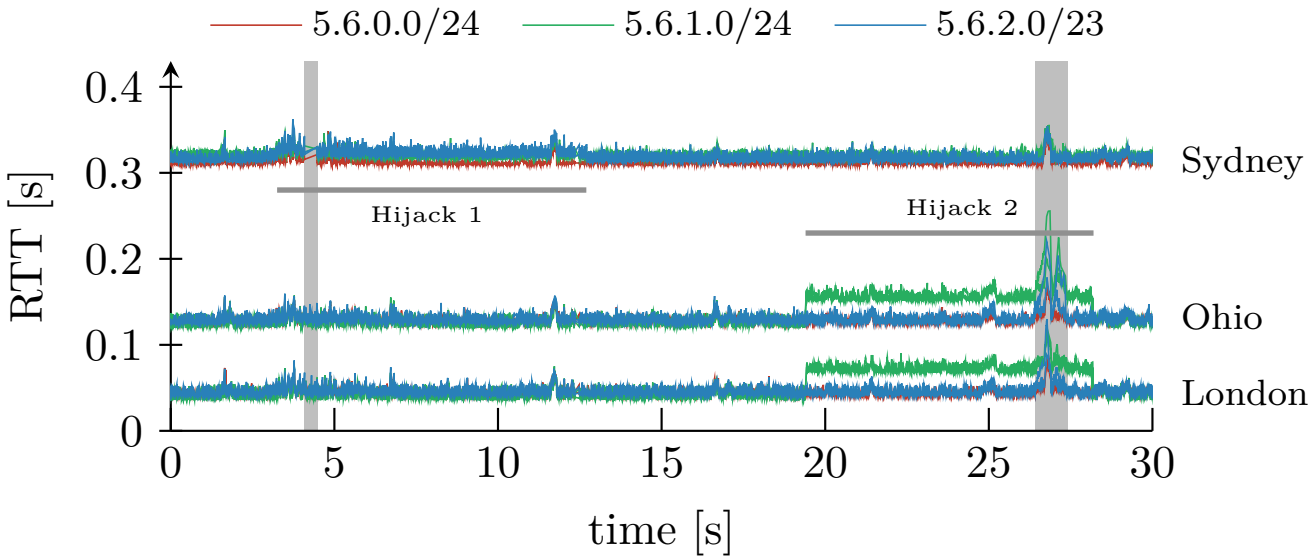


FIGURE 4.2: The figure highlights the RTT reaction to two hijack events and other interesting RTT patterns (gray rectangles). During the first hijack, a VM in Mumbai hijacks traffic towards 5.6.2.0/23 from Sydney. Given that Mumbai is geographically close to the "normal" forwarding path (compare Figure 4.1) the hijack-induced RTT change is smaller than during the second hijack (where traffic makes a detour over Stockholm).

We observe multiple main characteristics which guide our hijack detection approach described in the following sections:

- As expected, different hijacks will result in different RTT magnitude changes. For example, the hijack affecting traffic from Sydney is less pronounced than the hijack affecting traffic from Ohio and London.
- A more-specific prefix attack that only affects a sub-prefix results in distinct RTT patterns. For example, during the second hijack, we see an RTT difference between Ohio flows towards 5.6.0.0/24 and 5.6.1.0/24, i.e., inside the prefix 5.6.0.0/23 advertised by the ETH VM.
- Similar to the previous point, also same-prefix attacks result in a pattern. However, this time we observe the RTT change between different prefixes advertised by our VM, i.e., 5.6.0.0/23 and 5.6.2.0/23.
- A single hijack might affect traffic from multiple ASes (from which we receive traffic/RTT samples), but that does not have to be the case.
- Other network events also lead to RTT changes. The figure highlights two interesting events, first (around 4 seconds) a lack of RTT samples unique to the Sydney traffic. Most likely due to a problem with the local VM. Second (around 27 seconds), an RTT peak common to all RTT samples, indicating a problem close or at our ETH VM.

Limitations of the cloud-based setup Overall, the RTT samples in Figure 4.2 look very "clean" and stable. Detecting the hijack-induced patterns seems trivial with such input data. There are at least three reasons our motivational example behaves this way: *(i)* the amount of generated traffic is relatively low and does not lead to congestion or other problems; *(ii)* we have a single location (our VM at ETH) that receives *all* samples and can easily estimate the corresponding RTTs; and *(iii)* large cloud providers (such as DigitalOcean and AWS) are well interconnected (e.g., shown in [107]). As a result, our traffic mostly flows through over-provisioned cloud networks and is rarely "competing" with other Internet traffic for bandwidth, resulting in a low chance for congestion and/or packet loss.

More precisely, one possibility is an interconnection over so-called Internet Exchange Points (IXPs) [108]. Simplified, an IXP looks like an enormous router or switch connecting with many ASes. These ASes can directly peer over the IXP, even if they usually do not have a direct peering session with each other. As a result, our traffic, e.g., from a DigitalOcean VM to an AWS VM, will first travel inside the DigitalOcean network, then briefly cross the IXP, and afterwards travel inside the AWS network.

That being said, the cloud-based setup also has advantages. For example, we observe realistic delays between the geographically-distributed locations, resulting in hijack-induced RTT *changes*, which we could expect in a real attack scenario. In conclusion, we design our system following the main observations of the VM-based experiment, but we do not expect to observe such clean and stable RTT signals.

4.1.3 *Comparison with other data-plane signals*

The motivational experiment heavily focuses on RTT signals. However, other data-plane signals might also reveal interesting insights and potential hijacks. In this subsection, we summarize properties of RTT signals and then briefly compare other signal candidates.

RTT As we just saw, the RTT signal closely reflects path changes as it is directly connected with the packet travel time. By definition, an (interception) hijack leads to a path change; otherwise, the traffic flowed over the malicious AS to begin with. Hence, we expect an RTT change. Additionally, the RTT is theoretically (Section 2.4) and practically easy to compute (Section 2.4). For example, [47] extracts RTT samples at line rate using programable data-plane devices. As discussed in Section 4.8, it is challenging for a hijacker to modify the observable RTT signal to hide the hijack artificially. Finally, even a few flows can produce enough samples to detect a change.

Besides these advantages, though, RTT signals also come with some disadvantages. First, the observable RTT varies depending on the application behavior. For example, if one endpoint does not immediately acknowledge a received data packet, we might observe an artificially inflated RTT (as an on-path observer). This problem is less pronounced if we combine samples from many different flows. Second, the RTT also changes due to other network events. For example, a congested link will initially lead to an increased RTT (packets have to wait in queues) before some packets are dropped (lack/reduced number of RTT samples). However, long-lasting congestion events, for example, because a link between two ASes is highly under-provisioned, once again result in stable RTT patterns (although on a higher level), as e.g., [109] indicates. And third, RTT extraction is challenging in non-TCP flows, especially if the packet header is encrypted. We present some solutions in Section 4.2.

Throughput Another data-plane signal would be the current throughput from one or multiple ASes towards some of our (sub-)prefixes. A big advantage is the simple assessment of the current value. A counter is already enough. Although an interception hijack might lead to a throughput change, after all, some of the traffic is moved to a new path, a direct connection between the hijack and an expected throughput change is challenging for at least two reasons: *(i)* a hijack rarely changes the entire path. Especially the path segments close to the victim AS or close to the benign owner of the hijacked prefix might still be shared by hijacked and normal traffic (compare to our simple example in Section 2.3.4). Therefore, it is unclear if a hijack leads to a significant throughput change. And *(ii)* the throughput also heavily depends on the current application/endpoint behavior and the traffic volume of cross traffic which shares some of the path segments. Note the clear difference compared to the RTT signal. Even if we only observe a few flows, the (minimum) RTT still heavily correlates with the physical travel time of packets between the two endpoints.

For these reasons, throughput signals are unsuitable for detecting interception attacks. However, a blackhole attack should lead to an abrupt throughput decline towards zero.

Packet loss Similarly, the number of lost packets provides another data-plane signal. Unlike RTT and throughput, packet loss is more challenging to measure at scale. For example, duplicated acknowledgments can, but do not have to, indicate a packet loss. We need additional state to keep track of per-flow packet loss.

Even with precise packet loss signals, it is unclear how they correlate with an interception hijack. Related work, such as [110], shows that routing updates can lead to packet losses. However, the patterns are not uniform, making a clear correlation between packet loss with BGP hijacks difficult. After all, we will also observe similar loss patterns due to "normal" routing updates. More generally, packet loss, together with corresponding patterns of TCP's retransmission mechanism, is a helpful signal to detect complete, remote outages, as e.g., [111] shows. To this end, the loss signal would be useful for blackhole attack detection.

One-way delay Finally, instead of the *Round*-Trip Time, the *one-way* delay, i.e., the delay from the external AS towards our AS, shows interesting properties. As shown in Figure 4.1, our RTT measurements also contain the return paths (i.e., from our VM back to the other traffic endpoint), which are another source of noise or unrelated RTT changes. However, precisely

extracting or estimating the one-way delay from a single observation point is challenging or nearly impossible without additional endpoint or protocol support.

4.2 RTT EXTRACTION IN ENCRYPTED PROTOCOLS

Section 2.4 introduced two techniques for on-path RTT extraction on top of TCP flows. Both approaches use the fact that TCP is a so-called clear-text protocol. Most (all) of the header information is readable by any on-path device. Although this allows network operators to perform helpful monitoring operations, it also limits user privacy and enables possible attacks. One example relates to the TCP timestamp options that the RTT estimation described in Section 2.4 uses. If the timestamp option values are based on the actual uptime of the server initiating the TCP flow, on-path observers can roughly estimate how long a given server is running. This information reveals which security patches might yet *not* be installed, allowing for targeted attacks [112].

For these reasons, newer transport protocol headers are fully or partially encrypted, preventing some of the privacy problems and attack vectors. One prominent example is the recently standardized QUIC [113] protocol which encrypts the entire transport header (as well as the actual payload). Unfortunately, operators lose the possibility to extract RTT samples as they can no longer match packets with each other. In this section, we discuss research ideas on allowing RTT estimations while keeping the privacy benefits of encrypted protocols.

MAMI White Paper During the Management and Measurement Summit – as part of the EU Horizon 2020 Measurement and Architecture for a Middleboxed Internet (MAMI) project [114] – we formulated a white paper [9] that discusses challenges in network management with the increased deployment of fully encrypted protocols. One of the outcomes of the discussion with industry partners was that on-path monitoring devices (sometimes also called middleboxes) should no longer be transparent (i.e., invisible) to the endpoints. Instead, the endpoints should have control over which information they share with such devices, for example, by sending it unencrypted. The following Spin Bit idea follows this principle.

Spin Bit Our IMC poster [2] illustrates the general problem once more. From a user perspective, we hope for privacy and good performance. Operators

need to extract accurate and simple measurements using easy-to-deploy techniques. These goals often contradict each other. As previously described, clear-text transport protocols, such as TCP, can be used to overcome some of these challenges. However, encrypted protocols, such as QUIC, prevent simple matching of data and corresponding acknowledgment packets. RTT measurements are no longer possible. We then sketch an idea involving a single spinning bit: the "Spin Bit".

We explain the Spin Bit idea in detail in [8]. A single unencrypted bit already provides enough information for reasonable RTT estimations. The idea is simple. One endpoint always reflects the current value of the Spin Bit back in the matching packet replies (i.e., acknowledgments). The other endpoint initially sets the Spin Bit to zero and adds this value to all packets it sends. As soon as it receives the first acknowledgment, which also contains a Spin Bit with value zero, it "spins" the bit and now always adds the value one to its packets. Eventually, the first acknowledgment with Spin Bit one is received, at which point the endpoint spins back to adding Spin Bits with value zero, and the process repeats.

On-path observers can measure the time difference between the first packet with bit value zero and the first packet with bit value one to estimate full (or partial) RTTs. We argue that one unencrypted bit alone does not compromise the user's privacy, mainly as the endpoints can prevent RTT estimation at any point, i.e., by always setting the Spin Bit to zero. This opt-out approach does not work with TCP. Sending no or wrong SEQ and ACK numbers would disrupt *reliable* data transport.

Spin Bit's future Newer work proposes slightly adapted Spin-Bit-related techniques which have their own strengths and weaknesses and provide different measurement capabilities [115]. Performant prototype implementations [116] track the Spin Bit in programmable network devices [117] at line rate. Finally, the current QUIC specification [113] includes the Spin Bit idea as "Latency Spin Bit". It defines the usage of a spinning bit in specific QUIC packet types that are exchanged after version negotiation and connection establishment. The specification makes it very clear that the Latency Spin Bit is an *optional* feature; endpoints or network administrators can disable the Spin Bit.

4.3 OSCILLOSCOPE SYSTEM

With a better understanding of how BGP hijacks influence RTT signals (Section 4.1) and the availability of RTT measurements on top of TCP and QUIC traffic (Section 4.2), we now focus on the *Oscilloscope* itself. This section introduces *Oscilloscope*'s main components and describes the detectable hijacks.

4.3.1 *High-level overview*

Oscilloscope detects BGP hijacks by observing and comparing hijack-specific characteristics in passively collected data-plane traffic. *Oscilloscope* is designed to run standalone, analyzing the *existing* traffic crossing the network it is deployed in. *Oscilloscope*'s design leverages two simple yet powerful observations: *(i)* existing data-plane data already carries "signals" which can be linked to an ongoing hijack attack as motivated in Section 4.1 (also shown in [47, 118]); and *(ii)* the often-used per-neighbor routing policies [119] result in forwarding paths that are similar (or equal) for traffic flows between any pair of networks (i.e., ASes) in the Internet. The second observation implies that "normal" forwarding changes (e.g., due to a link failure) will equally affect all the traffic flows from *Oscilloscope*'s network and a specific external AS. In contrast, a hijack targeting a small set of our prefixes will break this similarity and build an exploitable comparison point.

Oscilloscope detects hijacks in three steps, illustrated in the example in Figure 4.3. In the example, AS Z intercepts traffic towards the prefix 212.0.3.0/24, which is announced by our AS where *Oscilloscope* is deployed.

First, *Oscilloscope* combines collected RTT samples to build so-called "combined signals," which contain all the RTT samples from flows that share the same local /24 prefix and target destinations belonging to one external AS (the figure shows three examples). Aggregating measurements this way improves the practicality and scalability of *Oscilloscope*, in terms of both data to maintain and comparisons to perform. We do not lose any information, given that /24 is the smallest prefix size globally advertised by BGP [59], and hence hijacks cannot target prefixes more specific than /24s.

Second, according to observation *(i)*, *Oscilloscope* detects changes in the combined signals. It compares the long-term minimum RTT value against RTT values in consecutive short-term windows and flags changes bigger

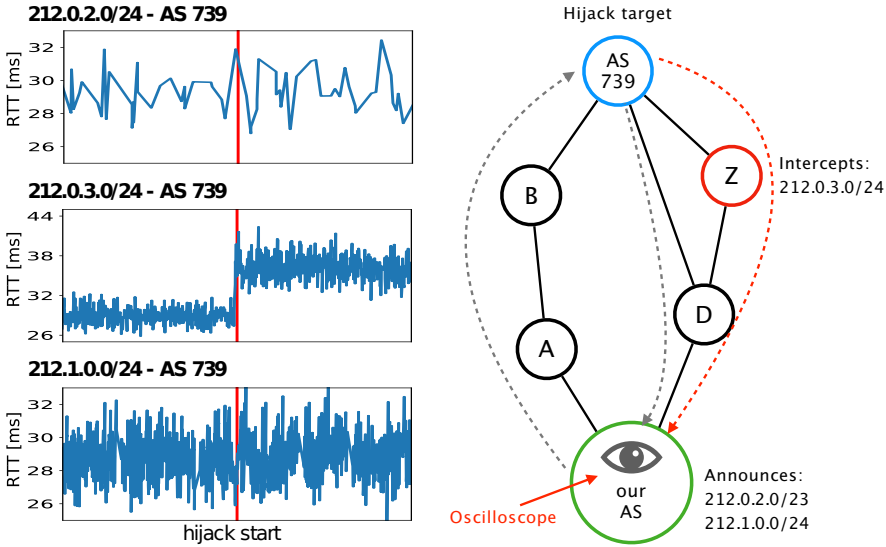


FIGURE 4.3: As Z intercepts traffic from AS 739 towards our prefix 212.0.3.0/24 which leads to an observable RTT change in the corresponding "combined signal". Note that traffic between other prefixes and AS 739 does not show the change which *Oscilloscope* uses as comparison points to increase its confidence.

than a given threshold (see middle plot in Figure 4.3). Comparing the RTT across consecutive short-term values filters out noise. Since the minimum RTT can be computed efficiently, changes for all combined signals are detected in near real-time.

Finally, *Oscilloscope* performs two-sample statistical tests between RTT samples from the combined signal which observes the change and its "buddy" prefixes (i.e., the combined signals *not* affected by the hijack in Figure 4.3) – where buddy prefixes are defined based on our observation *(ii)*. In the case of a hijack-induced, prefix-specific change, the test will indicate that the samples come from different distributions increasing *Oscilloscope*'s hijack evidence.

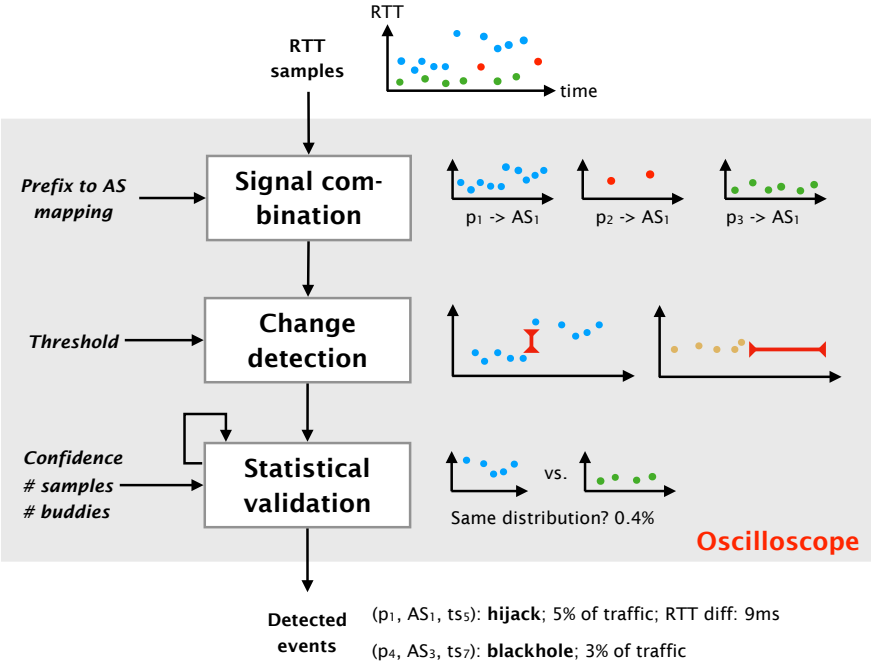


FIGURE 4.4: *Oscilloscope* contains three major functional blocks. It combines RTT samples, then detects changes in the combined signals and performs statistical tests to validate the detected events.

4.3.2 *Oscilloscope's main components*

The *Oscilloscope* pipeline comprises three main functional blocks (Figure 4.4): the combination of RTT signals, the detection of changes in the signals, and the hijack validation.

Input: RTT sample extraction The main input to *Oscilloscope* is a set of RTT samples extracted from the flows that traverse the network *Oscilloscope* operates in. There exist multiple methods to extract RTT signals from network traffic. For example, Google's Espresso [120] provides such RTT samples from their servers, and recent work has shown how to extract RTT samples from programmable data-plane devices [47, 121]. Therefore, we assume that the RTT signals have already been extracted. In order to know *where* to collect the RTT samples in the network, our previously introduced *Magnifier* system provides verified ingress and egress observations. A deployment

at the network border is beneficial as the resulting RTT estimations do not contain potential noise from internal paths inside our own network.

Extracting and gathering RTT signals from all the flows in a large network may become a scalability problem. In such cases, we can feed the RTT signals of only the largest flows to *Oscilloscope*. As we show later, *Oscilloscope* does not need to monitor all the flows from/to an IP prefix to detect a hijack.

Stage 1: Signal combination The first step of *Oscilloscope* is to build a "combined" RTT signal that captures changes at the level of entire IP prefixes and/or ASes. This level of granularity allows us to detect a hijack attack that affects even a single external AS, provided that we have ongoing traffic flows from/to that AS. The challenge is to combine flows *(i)* efficiently while *(ii)* preserving the necessary information to detect potential hijacks.

Oscilloscope takes as input a given set of internal IP prefixes to be monitored and groups the RTT samples at the granularity of /24 IP prefixes. For instance, a /23 IP prefix will result in two combined RTT signals corresponding to the two /24 subnets, allowing us to detect both same-prefix and more-specific attacks. In fact, common BGP filters [59] would block hijacks of a /25 IP prefix or more specific ones. We note that aggregating the RTT signals of all the flows from/to a monitored /24 is, however, still too coarse-grained, as same-prefix attacks or hijacks with artificially-limited spread, only affect certain Internet regions. Thus, we combine RTT signals per *(i)* /24 prefix and *(ii)* per external AS with which a flow is communicating. In other words, each *combined signal* contains RTT samples from traffic exchanged between one of our /24 IP prefixes and all the prefixes belonging to a *single* external AS.

To figure out which IP prefixes belong to which AS, *Oscilloscope* relies on the AS path of the BGP routes received at the local routers. For each IP prefix, the corresponding AS path usually indicates which AS originated the route

Stage 2: Change detection In the second block, *Oscilloscope* analyzes each combined signal to detect sudden RTT changes (increase or decrease). Following, we present a simple overview and give more details in Section 4.4. We perform the change detection by comparing long- and short-term minimum RTT values. Suppose the difference exceeds a user-defined threshold (input parameter) and is ongoing for the entire duration of a user-defined time window. In this case, *Oscilloscope* detects an event and triggers the last operation of the pipeline: It validates whether this event is a BGP hijack or a genuine change in the Internet state. In addition, we also forward

combined signals to the validation stage if they suddenly terminate, which could indicate a blackhole attack.

Stage 3: Hijack validation Several Internet routing events may lead to changes in the RTT of specific flows, e.g., reconfiguration, policy updates, load balancing, or fiber cuts. The last block in *Oscilloscope*'s pipeline is therefore responsible for distinguishing "true" hijacks from normal routing events. We only want to report the hijacks.

Oscilloscope validates hijacks using the concept of "buddy" prefixes which translates insights from [122] into the data plane. We say that a set of IP prefixes originated by the same AS are *buddies* if their flows towards the same external AS destination follow the same path. In the absence of malicious hijack attacks, IP prefixes originated by the same AS propagate through BGP along the same paths, as most ASes do not apply per-prefix policies [123]. Hence, buddies offer comparison points that can identify hijacks: a hijack of one of our prefixes does not affect its buddies, whereas normal routing events affect *all* buddy prefixes similarly as they traverse the same AS path. When the change detection block reports an event for a particular IP prefix, *Oscilloscope* collects RTT samples from the prefix and one of its buddies. Using a two-sample statistical test, *Oscilloscope* checks whether the two RTT samples appear to come from the same *distribution*; if not, we take this as evidence that the flows started to follow different paths, which indicates a potential hijack. If possible, *Oscilloscope* repeats this validation step with additional buddy prefixes to increase its confidence.

4.3.3 *Detectable hijacks and attacker model*

Oscilloscope detects hijacks based on changes in the observed data-plane signals, which reveal blackhole and interception BGP hijacks (as introduced in Section 2.3.4). In a blackhole attack, traffic from one or multiple external ASes targeted to some of our internal prefixes is terminated/reaches a blackhole in an unwanted AS. An interception attack is more refined. Instead of terminating the ongoing flows, an attacker intercepts the traffic and forwards packets over different/additional ASes before they reach their original destination, i.e., our AS. Both hijack types emit data-plane signals, which we can detect by analyzing RTT samples collected in our AS. A blockhole hijack leads to a (one-directional) traffic loss, resulting in an abrupt termination of any observable RTTs. In contrast, an interception hijack leads to a potential change in the RTTs.

We assume that an attacker either performs a same-prefix or a more-specific prefix attack (compare Section 2.3.4). Both lead to data-plane changes and are covered by *Oscilloscope*'s combined signals that focus on a /24 prefix level. However, *Oscilloscope*'s purely data-driven approach fails if hijacks do not generate a representative data-plane signal (no flows or too small of an RTT change), as we show in Section 4.6. Similarly, *Oscilloscope* does not detect misconfigurations which do *not* lead to any newly hijacked traffic. For example, a configuration mistake could lead to an advertisement of a prefix which does not belong to an AS. However, its provider automatically filters out wrong advertisements and stops the spread of the "hijacked" route. The lack of data-plane signals prevents *Oscilloscope* from detecting the misconfiguration. A control-based detection system (compare Section 4.7) might cover these cases.

4.4 SIGNAL AGGREGATION AND CHANGE DETECTION

This section explains how *Oscilloscope* detects changes in the combined RTT signals, filters noise, and reacts to events where no RTT samples are available. We conclude with practical insights gathered while testing existing, well-defined change point detection algorithms with our combined RTT signals.

4.4.1 *Change detection*

Figure 4.5 shows *Oscilloscope*'s workflow to detect changes in a combined RTT signal using simplified examples. As explained in Section 4.3.2, a "combined signal" contains all RTT traffic samples exchanged between one of our /24 prefixes and all the prefixes belonging to a single external AS. The change detection process runs on each combined signal separately once every second. That also means that the number of RTT samples considered each second depends on the amount of traffic in a specific combined signal.

Using a threshold to detect large enough RTT changes. RTT signals are often noisy due to network congestion and end-host processing delays. This noise may even be exacerbated when combining RTT samples from different flows, which is the case when *Oscilloscope* builds combined signals. Therefore, *Oscilloscope* uses a threshold (indicated in Figure 4.5 with the red vertical bars) to dismiss changes in the RTT signal that are too small

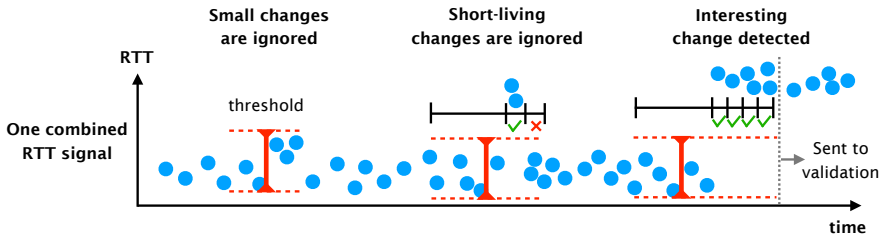


FIGURE 4.5: *Oscilloscope* uses long- and short-term windows to detect minimum RTT changes larger than a given threshold.

(example on the left). The network operator defines the threshold directly impacting the number of detectable hijack events (as empirically validated in Section 4.6).

Using the minimum RTT over time windows to track the RTT. However, we do not apply the threshold for every two consecutive RTT samples but rather use it to detect changes in the *minimum* RTT observed in a long-term and short-term window. *Oscilloscope* uses the minimum RTT for two reasons. First, the minimum is practical when considering RTT measurements, as there is an explicit lower bound on how fast packets can traverse over a given path [124]. As such, a change in the observed long-term minimum RTT indicates a potential path change. Second, *Oscilloscope* can update the current minimum value whenever it receives a new RTT sample and does not need to store all the samples belonging to one combined signal. That improves efficiency and saves storage resources which is essential as the number of combined signals is large depending on the network size.

Example Figure 4.5 shows how *Oscilloscope* uses short- and long-term windows. In the example in the middle, *Oscilloscope* realizes that the short- and long-term RTT difference exceeds the given threshold for a few RTT samples (indicated with the green tick below the short-term window). If *Oscilloscope* would now immediately report the change and continue with the validation step, we would be very susceptible to short-living RTT spikes (e.g., due to queues filling up). For this reason, *Oscilloscope* saves the long-term RTT minimum and repeats the comparison in the next short-term windows. In the example, the RTT difference no longer exceeds the threshold (red cross), and we dismiss the change. *Oscilloscope* only deems a change as significant if the RTT difference exceeds the threshold in as many consecutive short-term windows as the long-term window is wide. In our

implementation, we use a four-second long-term window and a short-term window of one second. For that reason *Oscilloscope* needs to detect the change in four consecutive short-term windows before acknowledging the RTT change (as shown on the right in Figure 4.5). We found out that a long-term window of four seconds is long enough to filter out short delay spikes (due to congestion) while being short enough to allow *Oscilloscope* to operate reactively and detect potential hijacks in a timely manner.

Detecting positive and negative RTT changes. To detect potential hijack events that would decrease the observed minimum RTT (i.e., a hijacker that can provide better connectivity than the original path), *Oscilloscope* compares the absolute value of the minimum difference with the threshold. Also, it checks that the sign of the difference is consistent in the four consecutive short-term windows. If all these conditions hold, *Oscilloscope* continues with the validation stage described in Section 4.5.

4.4.2 *Dealing with a lack of RTT samples*

It is important to note that *Oscilloscope* handles short-term windows without any RTT samples in a unique way. We observe short-term windows without samples if, for example, the combined signal consists of flows with low activity or an event that causes packet loss (which results in the loss of RTT samples). In such a case *Oscilloscope* performs two operations. First, it extends the long-term window by one unit (i.e., keeps track of the existing long-term RTT minimum) rather than decreasing the amount of RTT samples in the long-term window with every empty short-term one. Without that, *Oscilloscope* would eventually lose the long-term minimum RTT value, which an attacker could abuse to evade detection. More precisely, an attacker could drop the initial hijacked packets before starting an interception attack, and *Oscilloscope* would no longer be able to detect the introduced RTT change. Second, *Oscilloscope* starts a blackhole detection process. *Oscilloscope* once again uses a similar approach to track the minimum RTT, but this time it searches for four consecutive short-term windows *without* any RTT samples. However, to accommodate combined signals containing flows with infrequent RTT samples (i.e., some short-term windows without any samples), the number of required consecutive short-term windows without any samples (4) is multiplied by $n + 1$ where n corresponds to the number of empty short-term windows in the currently considered long-term window. With this design choice, we combine the well-working long-term

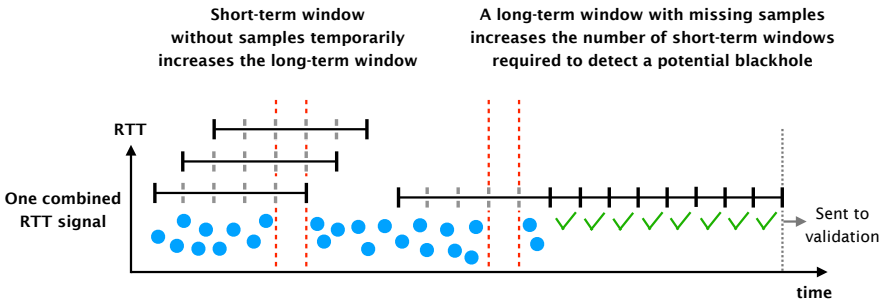


FIGURE 4.6: Short-term windows without any RTT samples increase the corresponding long-term windows and influence the detection process of potential blackholes.

window size of four with an adaptive component (multiplication by $n + 1$) based on the current behavior of the combined signal. If enough consecutive short-term windows contain no samples, *Oscilloscope* forwards the potential blackhole to the validation stage.

Example Figure 4.6 illustrates these two operations. In the example on the left, the current long-term window now contains five instead of four short-term windows, as one does not contain any RTT samples. The same holds for the example on the right, where *Oscilloscope* requires $4 \times (1 + 1) = 8$ consecutive short-term windows before the potential blackhole observation is forwarded to the validation stage.

4.4.3 *Challenges with existing change point detection algorithms*

Change point detection in time series data is a well-studied field. For example, [125] gives an overview of different techniques. We applied combined RTT signals collected in our emulated setup (compare Section 4.6) to various algorithms. We made the following practical observations which eventually led to the development of *Oscilloscope*'s window-based approach.

Online approaches are sensitive to RTT spikes. *Oscilloscope* tries to detect BGP hijacks as fast as possible. Therefore, we first explored so-called online change point detection algorithms that report changes in (near) real-time. Unfortunately, frequent RTT spikes, unrelated to BGP hijacks, lead to false positives. Our motivational experiment (Figure 4.2) shows one example of an RTT spike towards the end of the measurement.

Statistical methods require knowledge of signal or noise distributions. Multiple statistical methods need input parameters related to signal or noise distributions. In most cases, network operators do not know which distributions RTT samples follow. On top of that, we apply the change detection to a *combined* RTT signal containing samples from various flows. We make similar observations with *Oscilloscope*'s verification step, leading us to non-parametric statistical tests (Section 4.5.3).

"Simpler" approaches make assumptions on the *number* of change points. Another set of change point detection algorithms takes the number of expected change points in the given time series as input. Obviously, we do not know a priori if a combined RTT signal covers an expected change, i.e., a possible BGP hijack. As a result, these algorithms did not apply to our use case.

In conclusion, we decided to develop a custom change detection approach for *Oscilloscope*. Our window-based technique is fine-tuned towards the detection of hijack-induced RTT patterns. Additionally, the required parameters, e.g., duration of the short- and long-term windows, are easier to configure for network operators than complicated RTT distributions.

4.5 STATISTICAL TESTS

We now introduce the concept of buddy prefixes and explain how we use statistical tests to validate our hijack assumptions.

4.5.1 *Buddy concept*

Oscilloscope's primary validation strategy is built around the intuition that successful interception (or blackhole) hijacks will introduce a forwarding change reflected in the collected RTT signals. To have a baseline reference for detecting hijack attacks, *Oscilloscope* uses the concept of so-called "buddy" prefixes. In the absence of any hijacks, traffic from two buddy prefixes should follow the same paths when communicating with identical external ASes. The concept of buddy prefixes is not new. Buddyguard [122] first introduced the term to describe prefixes that behave similarly (in the absence of hijacks) when looking at them in the *control plane*. *Oscilloscope* translates this concept to the *data plane*.

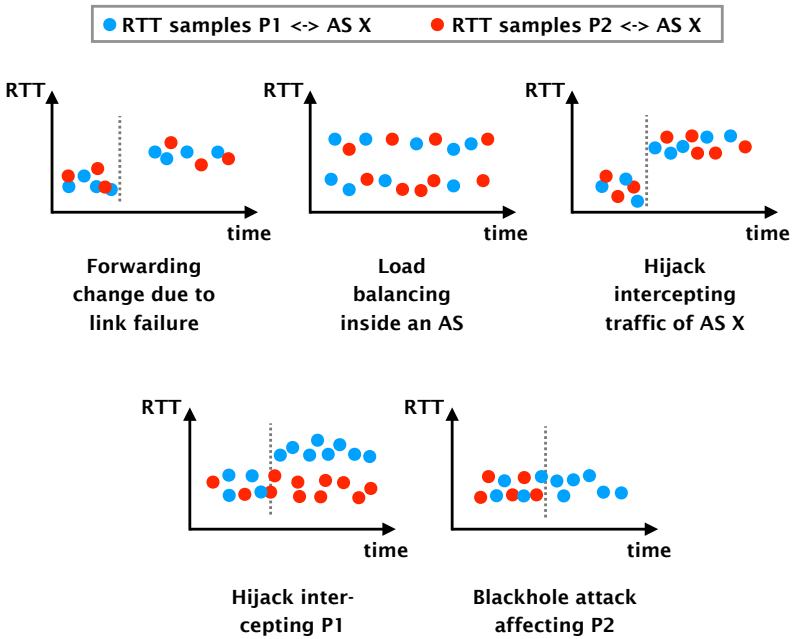


FIGURE 4.7: RTT samples of two buddy prefixes (P1, P2) exchanging traffic with AS *X*. We only observe an asymmetric behavior between buddy RTT samples if a hijack affects P1 or P2.

To build buddy prefixes, we split the IP space advertised by our own AS into /24 prefixes, the smallest prefix size advertised by BGP Internet-wide [59]. As a result, a potential hijacker must create fake advertisements for at least one of these /24 prefixes to influence our traffic. Consequently, the hijack will equally affect traffic inside one buddy prefix. Note that this approach allows to precisely detect more-specific prefix attacks, that affect one or multiple /24 prefixes inside a larger announced BGP prefix. We leave the problem of identifying buddy prefixes in real-time as future work. In [4, § 7.4] an analysis with real BGP data shows that most IP prefixes have multiple buddies.

Figure 4.7 shows simplified RTT distributions for various network events. Most of these examples are based on traces used in our evaluation. We made similar observations in our motivational experiment with cloud VMs (Section 4.1). Buddy prefixes (P1 and P2) exchange traffic with AS *X*. The first two examples (forwarding change due to link failure and

load balancing) show that RTT samples from both buddies are equally affected. That means, eventually, flows from both buddies will use the new forwarding path after the link failure or use the same set of physical paths when load balanced, respectively. The same observation holds in the third case (an interception hijack targeting prefixes of AS X), where traffic from P1 *and* P2 will be affected. Note that *Oscilloscope* does *not* try to detect such hijacks as the hijack does not target one of *our* prefixes. Finally, the interception attack of P1 and the blackhole attack targeting P2 (lower two examples) are the only two events leading to observable asymmetries in the RTT distributions of the two buddy prefixes. *Oscilloscope* is precisely looking for such changes and compares samples between multiple buddy prefixes to validate its hijack assumptions in these cases.

4.5.2 *Sample collection*

As soon as *Oscilloscope* detects a change in traffic belonging to a combined signal (Section 4.4), it collects RTT samples from this combined signal. In addition, we also look for comparison samples from valid buddy prefixes. Initially, *Oscilloscope* assumes that all our /24 prefixes are buddies (Section 4.8). However, *Oscilloscope* cannot blindly collect comparison samples from these buddy prefixes. The problem is that *Oscilloscope* neither knows a priori how many /24 prefixes are affected by the potential hijack (i.e., a hijack targeting a /23 prefix affects two /24 buddies) nor do we want to make any assumptions. Suppose *Oscilloscope* would pick validation samples from a buddy which is also targeted by the hijack. In that case, the validation step might report wrong results because we do not have a valid comparison point.

For this reason, *Oscilloscope* ignores buddies that *(i)* were recently declared as hijacked towards the currently observed external AS (an operator can clear these buddies after investigating the event); or *(ii)* are currently also in the process of being validated as we detected a change in their corresponding combined signal towards the same external AS (remember that we perform the change detection for each combined signal individually). The second criterion includes buddies for which the change is *nearly* confirmed, i.e., we already found the change in three of the required four consecutive short-term windows (see Section 4.4), to prevent strict time synchronization requirements between the change detection processes. From the remaining buddy prefixes, *Oscilloscope* follows a "first come, first served" principle

to pick the required amount of comparison samples from the specified number of buddy prefixes (two operator-defined parameters).

Note that all buddy prefixes are ignored for apparent events (e.g., a forwarding change introducing a significant RTT change). In this case, *Oscilloscope* correctly terminates the validation process and does not report any hijacks. In the more likely cases (e.g., changes detected due to noise, hijacks targeting specific prefixes, ...) *Oscilloscope* applies the statistical test described in the following subsection. It compares the distributions of samples collected from the combined signal that observed the change and its buddy prefixes.

If *Oscilloscope* tries to validate a potential blackhole, it expects to find no samples from the corresponding combined signal. We will immediately discard the blackhole assumption if we still observe a sample. Suppose, in addition, the buddy prefix(es) also do not provide any RTT samples (i.e., no more traffic towards this external AS). In this case, we cannot confirm the blackhole idea. It looks like a total traffic loss, e.g., due to a link failure, and *Oscilloscope* will not report anything.

4.5.3 *Statistical tests*

After collecting samples from the combined signal, which observes a change and at least one buddy prefix, *Oscilloscope* uses the Wilcoxon-Mann-Whitney test (e.g., [126, p. 128+]) to compare the two sample sets. The Wilcoxon-Mann-Whitney test is a non-parametric statistical test that only assumes that the samples from both sets are independent (given as they come from different /24 prefixes) and that we can order them (i.e., one RTT sample is bigger than another one). We selected this test for at least two reasons. First, given that it belongs to the class of non-parametric tests, we do not require that the collected RTT samples follow additional assumptions, e.g., that they belong to a specific distribution. Second, we can also apply the test if the number of found RTT samples is small (compare [126, Appendix Table J]), which is often problematic for comparable parametric tests.

As null hypothesis H_0, we assume that the distributions of both sample sets are equal. In contrast, the alternative hypothesis H_1 states that one sample distribution is stochastically larger (or smaller) than the other. As mentioned before, an interception hijack will most likely increase the observed RTT but could also decrease it. For this reason, we perform a *two-sided* test.

The test first sorts all collected RTT samples in increasing order and then sums up the ranks of samples belonging to the same set (i.e., ranks of all samples coming from the buddy prefix). If both sample sets belong to the same distribution (H_0), the rank sum should roughly be equal.

The network operator defines a confidence level as an input parameter to *Oscilloscope*'s validation step. Whenever the test result falls within the confidence level, we assume that the null hypothesis holds and the two sample sets come from the same distribution, i.e., they follow the same forwarding path. Otherwise, we assume they come from two different distributions, strengthening our confidence that we observe an ongoing hijack. If available, *Oscilloscope* repeats the same test with samples from additional buddy prefixes.

If the validation test indicates that the null hypothesis holds, *Oscilloscope* immediately discards the event and will not report anything. However, if the tests suggest that the RTT samples come from different distributions, we can optionally repeat the validation step once (or multiple times). That means *Oscilloscope* will collect another batch of RTT samples from the combined signal observing the change and its buddy prefixes. Not only will that strengthen our hijack assumption, but it also allows us to compare against other buddy prefixes, diversifying the collected samples. As we show in Section 4.6.3, repeating the validation step once greatly reduces the number of false positives/increases *Oscilloscope*'s precision.

4.6 PERFORMANCE

This section explains our evaluation setup (Section 4.6.1) and the used Hijack dataset (Section 4.6.2). Afterwards, we evaluate *Oscilloscope*'s accuracy (Section 4.6.3), its classification delay (Section 4.6.4), and the introduced computational overhead (Section 4.6.5). The paper [4, § 7.4] also evaluates our buddy concept with the help of control-plane data from the real Internet. As we show there, most observed prefixes have at least three buddies.

4.6.1 *Mini-Internet emulations*

We use the mini-Internet [10] for our evaluation setup. Our group developed the mini-Internet to perform practical projects during our lectures, but it can also help with research. It emulates arbitrary networks consisting of

routers, switches, and hosts. Each device runs in its own Docker [127] container. We then connect the different containers together in order to build various topologies. The routers use the FRRouting [105] protocol suite. As such, they run the most common routing protocols, e.g., BGP or OSPF. Additionally, each link can have custom delays and loss rates configured with `tc-netem` [128].

As explained in [4, § 6.1], we emulate an Internet topology with more than 600 ASes based on CAIDA data [129] and RIPE Atlas [130] measurements for "realistic" inter-domain link delays. "If we do not have enough data to estimate a link delay, we default to a delay of 5 ms and 2 ms jitter" [4, § 6.1].

4.6.2 *Hijack dataset*

We leverage our emulated mini-Internet topology (Section 4.6.1) to generate a synthetic dataset of packet traces containing hijacks. First, we identify a set of possible hijack events for our topology. Then we collect packet traces for simulated traffic from multiple runs with and without the hijack events.

Initially, we select one well interconnected AS as "our AS"; it advertises twelve different /24 prefixes in the entire mini-Internet topology. We then choose 70 other ASes to perform an interception attack towards our prefix space. For each of these ASes, we generate one hijack event where the AS hijacks one or two of our /24 prefixes from one or two of its directly connected neighbors. The hijacking AS uses path poisoning [131]. Therefore, the fake advertisement propagates in a localized part of the mini-Internet, potentially hijacking traffic from additional ASes. Similarly, we generate more than a hundred blackhole events by simply dropping the attracted traffic rather than forwarding it back to our AS. Finally, we also generate *normal* forwarding events, such as link failures, by dropping all the traffic on a given link. The routers run Bidirectional Forwarding Detection (BFD) [132] to detect link failures and trigger corresponding actions. As a result, a link failure either leads to a complete loss of traffic or a benign forwarding change with corresponding increases or decreases in the observed RTT.

For the evaluation, knowing how a specific event influences the observable data-plane signals (i.e., ground-truth behavior) is essential. Therefore, we compare traceroute outputs before and after each event to extract all pairs of /24 prefixes and external ASes that observe a forwarding path change. Additionally, we quantify the induced RTT change. Obviously,

Oscilloscope does not have access to this information. We only used it to label and group the events for the following evaluation.

Finally, we collect packet traces containing hijacks or normal network events. We use Flowgrind [106] to generate traffic mimicking real applications, e.g., video streaming or HTTP-style request-response patterns. For each emulated event, we generate an average of 100 flows resulting in about 200 MBps or 70k packets per second between our /24 prefixes and ASes, that are potentially affected by the event. During the entire run, tcpdump [133] records the packets observed on all interfaces connected to the hosts in our AS. Each experiment runs for two minutes, with the event starting after one minute and staying active until the end of the run. We perform one run for each event in our dataset and 30 runs without any specific events, which results in more than six hours of traces. For each trace, we extract the RTTs of all TCP flows by matching sequence and acknowledgment numbers (similar to the method in [121]).

Equipped with this set of RTT traces, we evaluate the hijack detection performance of *Oscilloscope offline* for different parameter choices, as shown in the following subsections.

4.6.3 *Oscilloscope's accuracy*

The core function of *Oscilloscope* is its ability to detect hijacks and differentiate them from normal network events. Performance is mainly affected by two system parameters: the detection threshold (Section 4.4.1) and the number of repetitions of the validation step (Section 4.5). In the following experiments, we quantify how these affect *Oscilloscope*'s precision (i.e., the rate of true positives among the detected events) and recall (i.e., the rate of true events correctly detected).

Recall vs. magnitude of the RTT change

Setting the change detection threshold defines the detectable events (i.e., achievable recall). Intuitively, hijacks that induce RTT changes smaller than the threshold are hard to detect.

Setup For each interception event in our dataset (Section 4.6.2), we run *Oscilloscope* with change detection threshold values of $\{1, 3, 5, 10\}$ ms and compute the ratio of true events detected by *Oscilloscope* (i.e., the recall).

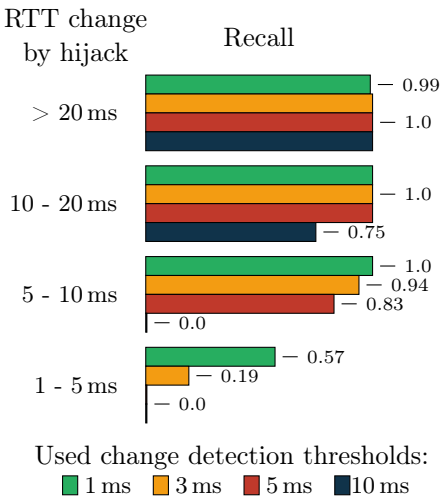


FIGURE 4.8: *Oscilloscope*' recall for different change detection threshold values, binned by the RTT change induced by the interception event. *Oscilloscope* reliably detects hijack events with RTT changes larger than the change detection threshold.

Results Figure 4.8 shows the recall for the different threshold values, binned to the magnitude of the hijack-induced RTT change. As expected, the recall drops when the RTT change gets close to the threshold. The detection becomes unreliable for the smallest changes (1-5 ms) (57% detected with a threshold value of 1 ms). This is expected as the RTT change is smaller than the average RTT signal noise, and *Oscilloscope* cannot detect these hijacks.

Interestingly, in some cases, a threshold of 1 ms fails to detect hijacks inducing more than 20 ms RTT changes. Closer inspection reveals that these cases have only one buddy prefix available. At the same time, this buddy is blacklisted for buddy comparisons due to a wrongly detected hijack event at a previous point in the trace (compare Section 4.5.2). Thus when the actual hijack occurs, *Oscilloscope* does not find any buddy prefix for the validation and hence does not report the hijack.

Conclusion *Oscilloscope* reliably detects hijack events when they induce RTT changes larger than the detection threshold. However, setting a threshold lower than the typical RTT noise is not useful (it does not help to detect events inducing very low RTT changes) and can even be harmful (*Oscilloscope* fails to detect events that can be caught with a larger threshold).

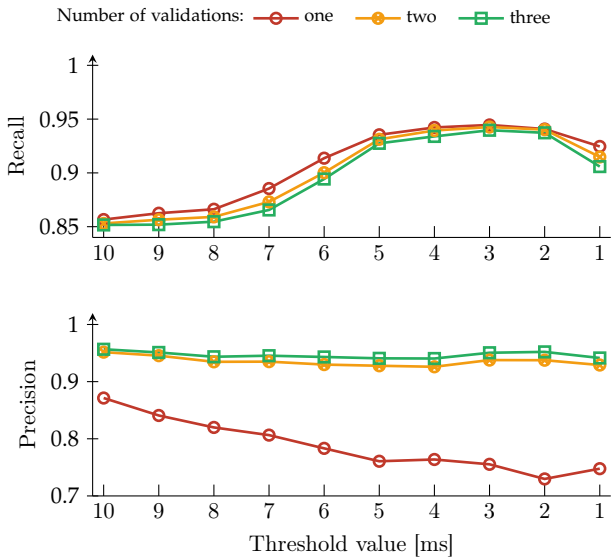


FIGURE 4.9: Precision and recall values for different change detection thresholds and different numbers of validation steps. Performing the validation twice largely improves the precision without hurting the recall much. The third validation has little effect. The best performance is obtained for a threshold of 3 ms, i.e., small but larger than the RTT signal noise.

Precision and recall vs. number of validations

Detecting RTT changes is relatively easy, but differentiating hijacks from normal network events is more challenging. For this purpose, *Oscilloscope* relies on buddy prefixes to validate that an event is a true hijack (Section 4.5). One can perform the validation step multiple times; *Oscilloscope* will report a hijack only if all validations agree that the event is indeed a hijack. Thus, performing more validations will tend to decrease the recall while improving the precision.

Setup We run *Oscilloscope* for all events in our dataset with different change detection threshold values (1 to 10 ms) and one, two, or three validation steps; we compute the resulting precision and recall values.

Results Figure 4.9 shows precision and recall values for the different settings. Looking first at the precision, we observe a significant improvement from one to two validations (up to 20%), then a smaller gain with a third

one. This matches our expectations: since multiple validations must agree to identify an event as a hijack, more validations reduce the number of false positives (i.e., improve precision). Conversely, the recall degrades with more validations. This is also expected since true events have more chances of being wrongly discarded as "non-hijack." Yet, the recall loss is small (less than 2%).

Let us now look at the performance when varying the detection threshold. Generally, smaller threshold values detect more events that either result in true or false positives during the validation step. As a result, smaller thresholds degrade precision and improve recall. However, as discussed in Section 4.6.3, threshold values smaller than the noise level (1-2 ms) yield more false positives and limit the buddy prefixes available for validation, thus hurting the recall as well.

One may notice a tipping point around 5 ms. We explain this by the design of our emulation setup, where we use a link delay of 5 ms as a default value (Section 4.6.1). Our dataset is thus biased towards events inducing a 5 ms RTT change.

Conclusion Overall on our dataset, the best performance is obtained with a threshold of 3 ms and two validations. This results in a 94% recall and 93% precision. A third validation trades slightly better precision for slightly worse recall.

Precision and recall vs. number of buddies

When an event is detected, *Oscilloscope* looks for three buddies to validate whether the event is indeed a hijack. However, we do not always have three buddies available. Thus, we now look at the impact of using fewer buddies on the precision and recall of *Oscilloscope*. Validating with more buddies is expected to decrease the recall while improving the precision.

Setup We run *Oscilloscope* for all events in our dataset with a 3 ms change detection threshold and two validation steps; we compute the resulting precision and recall values depending on the number of buddies used for the validation.

Results Table 4.1 summarizes the precision and recall values depending on how many buddies are used for the validation.

Similarly, as in Figure 4.9, the recall is only marginally affected by the number of available buddies. If the event is indeed a hijack, buddies often

Validation stage	Precision	Recall
With 3 buddies	93.77%	94.26%
With 2 buddies	92.19%	94.18%
With 1 buddy	88.31%	94.24%
No validation	6.11%	96.44%

TABLE 4.1: Precision and recall values for a different number of buddies that have to agree to report a hijack.

validate this correctly (i.e., they do not wrongly discard the event). Interestingly, the recall is only slightly better *without validation at all* and does not reach 100%. This is because some hijack events induce RTT changes that are too small to be detected, given our detection threshold. Conversely, more buddies logically yield better precision (i.e., fewer false positives).

Conclusion *Oscilloscope* achieves good precision and recall (around 90%) even with only one buddy for the validation. Using more buddies results in a slight increase in precision (a few %) for a negligible decrease in recall.

4.6.4 *Classification delay*

Oscilloscope is meant to detect hijacks at runtime. Thus, it is important to quantify how fast potential hijacks are detected, validated, and reported to the operator. The time required for a hijack to propagate through the network is beyond our control and independent of *Oscilloscope*. The delay from detecting an event in the combined RTT signal until the event is reported (or not) by *Oscilloscope* is more relevant. The delay is affected by the type of the observed event and the number of buddies and validation steps that *Oscilloscope* takes into account. Intuitively, it increases with a higher number of performed validation steps.

Setup We run *Oscilloscope* for all events in our dataset with a 3 ms change detection threshold and one, two, or three validation repetitions; we measure the time delay from detecting a change in the RTT signal until the final event classification by *Oscilloscope*.

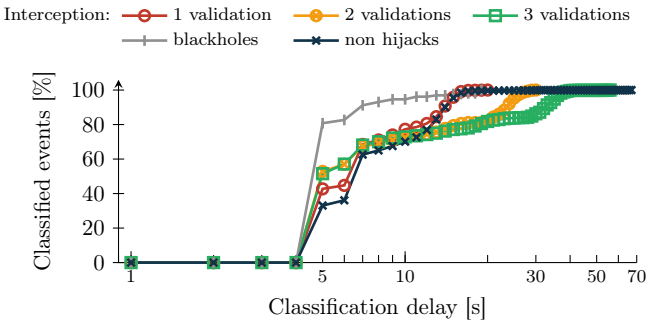


FIGURE 4.10: Delay between the detected change in the RTT signal and the classification of the event by *Oscilloscope*. Most events are classified within the best possible delay (around 5 s). The remaining ones are delayed by the lack of RTT samples from active buddies to perform the validation step(s).

Results Figure 4.10 shows the CDF of the classification delays for different events and validation repetitions. These delays are in the scale of seconds since *Oscilloscope* performs the change detection process once per second. The first observation is that the detection delay is at least 5 s, which comes from *Oscilloscope*'s design decision of waiting for four consecutive change detections before starting the validation (Section 4.4.1).

The blackhole events are the easiest to detect, with more than 80% classification completed after five seconds. Since no validation step is performed for these events, the number of validation repetitions naturally has no effect. For the interception events, around 50% are classified after 5 s; that corresponds to cases where *Oscilloscope* could quickly find buddy prefixes carrying enough traffic for the validation step(s). Cases with higher delays are unavoidable given the data-driven nature of *Oscilloscope*: when there is no traffic, there is no data to work with; one must wait for the data. With two or three validation steps, around 80% of interception events are classified within 20 s. The "non hijacks" represent cases where *Oscilloscope* initially triggers, initiates the validation, and classifies the events as normal routing changes.

Finally, the expected trend when varying the number of validations is most sensible for the tail: for about 70% of events, the classification delay varies by less than 1 s.

Conclusion By design, the classification and reporting of an event by *Oscilloscope* takes at least 5 s. This choice limits the validation step from excessive triggering due to random noise in the RTT signals. Most events are classified within 5 s and 80% to 100% of them within 20 s (depending on the event type and the number of validation repetitions). We argue that this is a reasonable delay for detecting hijacks that would otherwise remain unnoticed for a long time.

If faster detection is desired, an operator can adapt *Oscilloscope*'s design by *(i)* reducing the size of *Oscilloscope*'s long-term window and/or *(ii)* increasing the detection frequency (limited by the computational load – Section 4.6.5). However, this will only help in cases where active buddy prefixes are available for validation. As discussed before around 50% of interception event classifications are delayed by the lack of available RTT samples from buddies in our dataset.

4.6.5 *Computational overhead*

Running *Oscilloscope* requires performing three types of computations: *(i)* extract the minimum RTT values from the combined signals, *(ii)* compare the short- and long-term minimum RTT values to detect changes, and *(iii)* perform the statistical tests to compare the RTT samples between buddies. These computations must be sufficiently fast to run online; in particular, they must scale to the amount of traffic that *Oscilloscope* would have to monitor in a large AS.

Setup Our implementation of *Oscilloscope* is written in Python, similar to common software of programmable control-plane devices. We time our implementation execution for three classes of traffic load, quantified as the average number of RTT samples to process per second.

- low – 600 RTT sample/s
- medium – 3000 RTT sample/s
- high – 5000 RTT sample/s

We measure the timing of the operations mentioned above, labeled as **minimum**, **change**, and **validation**, respectively.

Results Table 4.2 summarizes the results for each operation and traffic load. All times are in μs.

operation	load	min	median	max
	high	0.71	44.1	244.37
minimum	medium	0.71	26.7	304.69
	low	0.71	2.62	111.10
	high	0.23	0.71	16.45
change	medium	0.235	0.95	16.68
	low	0.23	0.95	26.22
	high	1044.98	1152.75	3357.64
validation	medium	382.18	1146.07	2597.33
	low	347.85	459.19	2586.60

TABLE 4.2: Only the time to find the short- and long-term **minimum** is highly influenced by the load (median values). The **validation** operation takes the longest. All values in μs.

As expected, the **minimum** operation is the one most affected by the increasing traffic load, as there are more signals to track, thus, more operations to perform. However, even under high load, this always took less than 250 μs in our experiments. The computation is efficient as *Oscilloscope* can compute the minimum RTT values without storing all the samples, which is an asset of its design. Storing a small number of sample values is only required when the validation step(s) triggers. The **change** operation compares two minimum values; as expected, this operation is independent of the traffic load. Finally, the **validation** operation performs the statistical tests comparing RTT samples among buddies. We observe that higher loads lead to higher computation times (two to three times more between low and high loads). This is explained by the number of statistical tests that are effectively performed: *Oscilloscope* always looks for three buddies in the validation step. However, with low traffic, there are not always enough active buddies available, thus resulting in fewer tests being performed and lower computation times.

Conclusion As expected, the traffic load severely impacts only the **minimum** operation. However, it takes less than 250 μs even for a (high) load of about 5000 RTT samples per second. The validation step is the most computationally expensive, reaching up to 3 ms. Altogether, the total computation time is in the order of ms; in comparison, *Oscilloscope* runs every second, three orders of magnitude higher. Thus, we conclude that the computational

overhead of *Oscilloscope* is limited and, in particular, it is not a limitation for running the system online, even under high traffic loads.

4.7 RELATED WORK

This section introduces related work which focuses on BGP hijack detection. Note that some of the following paragraphs are contributions by the second PhD student (Alexandros Milolidakis) who is also an author of the *Oscilloscope* [4] paper.

Secure routing protocols are either undeployable or have limited effectiveness. Over the years, multiple techniques and modifications to BGP have been proposed to *proactively* secure the Internet against BGP hijacking [40, 134]. However, the lack of global consensus, the ever-increasing size of the BGP tables [135], as well as the extra overhead required to authenticate each entry have led to complex cryptographic signing techniques [136–138], such as BGPSec [99], to be far from deployable in production [134, 139]. As such, simpler forms of crypto-based mitigations have been proposed. For instance, RPKI [100] only stores a digitally-signed mapping between an IP prefix and the AS number allowed to originate a route for it. Unfortunately, previous results have shown that by picking a victim and malicious ASes at random over the Internet, the probability of attracting at least 10% of the traffic, even with RPKI fully deployed, was above 60% [140].

Monitor-based countermeasures have limited visibility. To overcome the limited deployment and effectiveness of crypto-based solutions, today's networks rely on *reactive* approaches that monitor the BGP announcements propagated through the Internet from various vantage points [122, 141, 142]. These systems typically provide real-time detection within 5 s by analyzing BGP announcements through the publicly available BGP monitoring infrastructure, i.e., RIPE RIS [143], BGPStream [144]. An inherent limitation of such solutions is that same-prefix BGP hijacks are only detectable if any monitor receives the announcements. Our paper [4, § 2.3] shows that carefully crafted BGP announcements do not propagate to the monitors.

Probing-based data-plane countermeasures can be evaded. Data-plane approaches rely on active probing measurements (e.g., pings, traceroutes, nmap, ...) often performed between external probe locations and the victim's network. Since traffic is forwarded to the attacker during the hijack period, we may discover hijacks by probes that fail to reach their destinations. The

success of these approaches heavily relies on their ability to distinguish hijack signals from common traceroute problems. As the hijacker can attack any sub-prefix of the victim's prefix, detecting all possible attack scenarios with low false positive rates (or false positives in case of interceptions) is time-consuming and computationally expensive.

For example, iSPY [101] offers a lightweight approach with low false positives that detect hijacks by observing unusual changes to usually stable traceroute paths. However, it protects only against blackholing and same-prefix attacks. Zheng et al. [145] discover interception attacks by observing AS hop count changes in traceroutes. However, an interception hijacker can easily manipulate this information by replying with false IPs. Hu et al. [146] extract "fingerprints" from the devices of the correct origin and compare them with the fingerprints of the potential hijacker. Since the hijacker cannot completely mimic and lie about a network, blackholes become distinguishable, yet interceptions easily evade detection as the traffic reaches the correct origin. As such, systems operating in the data plane have been challenging to commercialize.

Naive RTT-based solutions generate false positives or are challenging to deploy. There exists other work which uses RTT measurements to detect certain BGP hijacks. The RTT-based detection method in [47, 147] uses a simple, threshold-based change-detection algorithm to observe possible attacks. [148] focuses on a crowd-based detection approach which highlights possible routing anomalies. Both methods have difficulties in distinguishing hijacks from other events. *Oscilloscope* can validate some of its observations using statistical tests and buddy prefixes. DARSHANA [149] also uses changes in the RTT to detect hijacks but actively generates RTT samples with a new "cryptographic" ping protocol which requires the exchange of public keys with the destination. In comparison, *Oscilloscope* does not need support from other networks to detect possible hijacks.

4.8 DISCUSSION AND CONCLUSION

This section discusses advanced aspects of *Oscilloscope* regarding deployability and performance under special cases and highlights future work.

Does *Oscilloscope* replace existing systems? No, we aim not to replace existing RPKI and monitor-based solutions but to complement them. For instance, *Oscilloscope* cannot detect attacks where the RTT remains unchanged

or the hijacker intercepts *all* the IP prefixes in the same buddy group. In these cases, one has to resort to monitor-based solutions. Note however, that such attacks come with a cost for the hijacker. Intercepting *all* prefixes belonging to the same buddy group greatly increases the additional traffic that the hijacker needs to handle. *Oscilloscope* also does not *prevent* hijacks, making solutions based on cryptographic functions still beneficial (yet challenging to deploy).

Can *Oscilloscope* detect pre-existing hijacks? If an operator suspects that a specific IP prefix might be hijacked (e.g., based on insights from the control plane), we can manually trigger the validation step, which will compare samples between buddy prefixes and verify the hijack.

Can a hijacker hide from *Oscilloscope*? A hijacker may try to evade detection by tweaking the RTT signal or generating short-time hijacks. *RTT modifications:* It is hard (or impossible in some cases) to evade detection by merely tweaking the RTT signal. First, if the hijacking path has a higher RTT than the legitimate one, there is no way a hijacker can beat the speed of light to pretend the RTT is unchanged. Second, a hijacker that induces a lower RTT may hold packets back but guessing the legitimate RTT distribution would be out of reach since the hijacker has no visibility into those distributions. More precisely, usually, the attacker does not know what the RTT before the hijack was, as the malicious AS is (by definition) *not* on the legitimate path. *Short-time hijacks:* The attack should last long enough to propagate the announcement to the victim ASes. In the meantime, the RTT signal of the intermediate ASes would be affected and could be detected by *Oscilloscope*. High-frequency short-time attacks may be filtered by BGP Flap Damping [20].

Can *Oscilloscope* detect a hijack towards one of its prefixes for which we currently do not observe any RTT samples? In its current form, *Oscilloscope* is entirely data-driven and cannot detect hijacks for which we do not collect corresponding data-plane signals, i.e., RTTs. However, that also means that the hijack is less severe as it does not affect any actual traffic. In Chapter 6 we show a possible solution based on our *Dynamo* [3] system.

Can *Oscilloscope* detect hijacks on IPv6 prefixes? Yes, *Oscilloscope*'s main ideas also translate to IPv6 prefixes. However, operators might need to revisit how they define combined RTT signals given the huge IPv6 address space that is assigned to a single AS [58]. To prevent scalability issues, we can extract combined signals based on the IPv6 address space that is actively distributed to our customers. At least this space might produce user

traffic and corresponding RTT samples. IPv6 also leads to interesting future research questions. For example, can we observe data-plane buddiness between IPv4 and IPv6 prefixes belonging to the same AS?

How can an operator regulate the number of alerts that *Oscilloscope* produces? As we show in Section 4.6, the operator could, for example, increase the change detection threshold, improving the precision and removing some of the noise/false positives, i.e., reported events often indicate a real hijack. Another promising approach extends one of the previous discussion points. An operator could combine *Oscilloscope*'s reports with observations from a control-plane-based detection system. For both systems, we can filter out reports with low confidence unless both systems raise them. For example, we could remove *Oscilloscope*'s reports which are validated with a single buddy prefix only.

Does *Oscilloscope* need to be deployed in multiple ASes to work properly? No, it is enough to deploy *Oscilloscope* in a single AS. It is only designed to report hijacks that affect prefixes originated by this AS. *Oscilloscope* extracts all the required information locally. For example, RTT samples from flows that start/end in the local AS or BGP data from local routers (i.e., to build *combined signals*). However, aggregating reports from multiple *Oscilloscope* instances deployed in different ASes could help reduce false positives or detect large-scale hijacks that might affect prefixes from multiple ASes at once.

The paper [4, § 7.4 & 8] also explains that *Oscilloscope* could overcome *(i)* potentially falsely-classified buddy prefixes by basing its validation step on a majority vote instead; and *(ii)* cope with intra-domain traffic engineering techniques (e.g., hash-based load balancing) which influence single flows but not all flows in a *combined signal*.

4.8.1 *Conclusion*

This chapter introduced *Oscilloscope*, which builds on top of *Magnifier*'s inferred ingress and egress points (Chapter 3) to figure out *where* to collect RTT samples from specific flows passively. *Oscilloscope* leverages the insight that BGP hijacks introduce a forwarding path change which results in an increased or decreased RTT. *Oscilloscope* first detects abrupt changes in the minimum RTT using a window-based approach. Afterwards, it increases the detection confidence using statistical tests.

Oscilloscope highlights that *locally*-collected signals (here RTTs) reveal properties of *external* Internet path segments (i.e., a hijacked path). Similarly to *Magnifier*, *Oscilloscope* only *informs* network operators about potential hijacks but does not actively prevent them. Network operators need to adapt their forwarding decisions based on inferred path properties to achieve more *proactive* systems. We explore the potential of this idea in Chapter 5 and introduce design principles and challenges of active routing decisions based on inferred path properties.

5

PATH-PROPERTY-DRIVEN ROUTING DECISIONS

So far, we have seen how *Magnifier* helps operators identify traffic ingress and egress points in their networks (Chapter 3), enabling precise network monitoring. *Magnifier*'s observations also guide operators on *where* to collect signals to infer external path properties.

In Chapter 4, *Oscilloscope* detects BGP hijacks based on changes in the observed RTT. Given that we use *locally*-collected RTT samples to infer properties of *external* path segments, we also observe more noise and encounter false positives.

Due to their passive nature, both systems only inform network operators *reactively* about unexpected path properties. For example, once *Oscilloscope* detects a hijack, the network's user traffic already crosses the malicious AS. A more *proactive* system necessitates a connection with available tools that operators use to influence path selection. More precisely, inter-domain routing protocols such as the Border Gateway Protocol (BGP) [20].

This chapter explores the opportunity to actively adapt routing decisions based on inferred Internet path properties. Our experience with the *Oscilloscope* system guides us towards a path property that emits a strong signal that we can collect locally, namely the reachability property of a path. In other words, we explore the following research question: Whenever BGP announces a new path for an existing destination, can we confirm that the new path provides reachability *before* forwarding *all* our traffic over it?

Our exploratory study answers the question positively and reveals that BGP's "all-or-nothing" approach hinders a complete path reachability inference. That means if BGP selects a new best route upon receiving a corresponding update message, it immediately and irrevocably forwards all matching traffic over the advertised path. As a result, network operators lack time to infer any path properties of the newly announced path. We advocate spreading the traffic shift over time to *(i)* collect signals for path reachability inference; and *(ii)* allow for reconsideration should the new path reveal unreachable destinations.

We highlight three main insights when assessing, controlling, and integrating path-property-based routing decisions. First, we must decide which IPs we use to test for reachability. Closely connected is the question of which locally-collected signal reveals the *external* reachability path property. Naive approaches, e.g., ping probes, neither provide the required coverage nor the expected precision [22]. Instead, we leverage insights from the *Oscilloscope* system and directly infer reachability based on slowly moved user traffic.

Second, we explore which devices should steer the entire process. We observe a fundamental discrepancy between the network device forwarding traffic to the existing, old path; the device that receives a BGP update announcing a potential new path; and the network ingress points that carry most of the traffic forwarded to the announced destination. We look at different control flow designs, all of which have strengths and weaknesses.

Third, we explore different integration possibilities of our ideas with the existing BGP decision process. We show that network operators can precisely define the importance of path-property-based decisions by deciding *where* to place our operations in the existing decision process, i.e., as a new step. A tight integration also allows for the inclusion of performance-related path properties in the future.

Following, we first introduce the problem in detail (Section 5.1). Then we explain why naive solutions are not good enough to infer path reachability and gradually improve them (Section 5.2). Afterwards, we explore different possibilities to include our path-property-based inference into the existing BGP decision algorithm (Section 5.3), followed by open challenges (Section 5.4) and a comparison with BGP's default behavior (Section 5.5). Finally, we look at related (Section 5.6) and future (Section 5.7) work.

5.1 INTRODUCTION

The Border Gateway Protocol (BGP) [20] is today's only option to perform inter-domain routing between ASes in the Internet. Whenever BGP has to select a new best route towards a destination, it goes through a multi-step decision process taking all available (i.e., according to the router configuration) paths into account. As we saw in Section 2.3, the algorithm only considers control-plane information. It mainly bases its decision on economic reasons, for example, to prefer "cheaper" routes: a route over a peer with higher local preference is better than a route over a provider. Other metrics, such as the AS path length, only loosely represent the physical

distance towards the destination. For example, AS path prepending is a standard tool to deprioritize routes. It leads to artificially inflated AS paths and potential route vulnerabilities [150]. Prepending is commonly observed; origin ASes prepend more than 25% of all IPv4 prefixes [151].

It is striking that simple path properties, for example, if the path provides reachability towards the advertised destination, do *not* play a role in the decision process. Even more concerning, BGP does not provide any feedback loop to validate its path selection. Assuming a newly selected path does *not* provide data-plane reachability, we mainly have two slow and unreliable options to trigger a new decision process: *(i)* some devices close to the problem eventually detect it and withdraw the faulty route; or *(ii)* operators take actions after receiving complaints from customers whose traffic no longer reaches certain destinations. While waiting for a new BGP update, BGP is stuck with a path that does not provide data-plane reachability, and all corresponding traffic is lost. Unfortunately, not only do device or link failures lead to unreachable paths, but also misconfigurations are a problem. Route leaks, for example, are common and can lead to unreachable paths or completely overloaded networks in the worst case [152].

We ask ourselves if we can enhance BGP's decision process with simple data-plane metrics, i.e., *external* path properties. More precisely, we envision a new entry in the existing BGP decision process that selects paths according to inferred path properties. In doing so, we can provide "working" and performant paths whenever possible, even if they would not match the best path selected by today's BGP algorithm.

Working paths We expect that the eventually selected path provides reachability towards the advertised prefixes – something that today's BGP algorithm cannot infer. However, determining reachability is not as easy as just sending a few probes, as we illustrate in Section 5.2 with the help of Figure 5.1. Some reachability problems only surface once a network forwards *enough* traffic over a new path.

To include these scenarios, we define a working path as a path that *(i)* provides reachability towards the advertised prefix(es); and *(ii)*, if possible, can provide the required throughput to carry all our traffic. It is important to note that we only infer these properties when selecting a new path due to an incoming BGP update that reaches our block in the decision process. We do *not* constantly monitor paths and dynamically shift our traffic towards the best option, which could have undesired drawbacks such as frequent

route flapping increasing router load [153] or impacting performance for end users [154].

In the following section, we explore different design ideas *how* we could infer path reachability properties and *which* network device should steer the entire process.

5.2 DESIGN

This section tackles the seemingly effortless problem of detecting whether a path provides reachability towards the advertised prefix. Starting with the most straightforward approach, we iteratively increase the complexity while discussing advantages and drawbacks. We first focus on techniques to assess reachability and then consider how we can deploy our ideas in existing networks, i.e., how the control flow looks.

5.2.1 *Path reachability inference*

We first explore techniques to infer path reachability, focusing on which IPs and signals to select.

Example scenario and terminology Let us assume we are in the following scenario (we discuss other scenarios in Section 5.4). A border router of an AS already knows a best route towards a prefix P, for example, over another local router learned via iBGP. It currently forwards traffic along this route, which we call the old route or path. At some point, one of the eBGP neighbors advertises a new route for prefix P to the border router, i.e., over an eBGP session. We call this route the new route or path. BGP's decision algorithm needs to decide which one is better. From a data-plane perspective, we want to ensure that the selection of the new route provides reachability towards the advertised prefix P; resulting in a working path.

A successful reachability inference necessitates solving two problems: *(i)* which IP addresses can we use to test for reachability, and *(ii)* which technique or path signal do we use to infer the reachability?

IP selection It might be tempting to assume that an operator could randomly pick any IP belonging to prefix P to explore path reachability. However, this approach does not work for multiple reasons. First, we must determine if the random IP belongs to an active end host. If not, nobody

will answer a probe, or we will not observe any corresponding traffic. Therefore, we cannot make a conclusive decision regarding the current reachability towards prefix P. A second problem relates to the coverage of our reachability inference. A random IP will only cover a small part of the overall IP space belonging to P. More precisely, P could, for example, represent a /16 prefix. To reason about the overall reachability of the /16 prefix based on insights from a single IP is a big stretch.

Similarly to the insights we got with our *Oscilloscope* system (Chapter 4), we argue to follow a data-driven approach. To select IPs for reachability tests, we use matching destination IPs of ongoing flows that traverse the *old* path to reach P. That solves the activity problem. Given that we observe ongoing flows, the corresponding end hosts are active. It also (partially) solves the coverage issues. The active flows will cover all the relevant destinations inside P.

As a side note, observe that the old path could exhibit unreachable sub-prefixes belonging to P, which is the reason for a *lack* of ongoing flows. However, even in such a scenario, TCP [42] will periodically retransmit lost packets providing us with new candidate IPs to infer the reachability of the new path.

Signals to use The reachability of an Internet-wide path is an *external* property that network operators infer based on *locally* collected signals. However, unlike *Magnifier* and *Oscilloscope*, two systems that do not know a priori *when* we should expect a change in the inferred path properties, our active combination of routing decisions based on path properties has a significant advantage: We know precisely *when* to expect a potential property change as we actively control the forwarding process.

As such, we can select from various locally-collected signals to infer reachability properties. Similar to *Oscilloscope* when detecting blackholes, a sharp drop in RTT samples reveals reachability problems. Even simpler approaches give us insights into transit traffic, for which we might only observe one traffic direction in our network. For example, we can infer if the traffic "makes progress", i.e., we see increasing TCP SEQ or ACK numbers (Section 2.4). Alternatively, in case of an unreachable path, common TCP retransmission patterns reveal problems (successfully used in [111]). As a result, the main challenge comes from populating the newly announced path with traffic, producing representative signals for reachability inference. The following paragraphs discuss different techniques and ideas.

Different signal generation techniques Akin to common debugging workflows used by network operators and end users alike, we could ping (e.g., with the ping tool [43]) one of the IPs we observe over the old path while directing the ping probe to the new path. Unfortunately, the generated signal does not correctly reflect the reachability property of the new path, as the probe or its reply could be lost.

A better approach sends *multiple* pings; but how many are enough? More problematic, end hosts or on-path devices could actively block ping probes in an attempt to increase security [44]. Under these circumstances, a ping-based reachability inference would lead to erroneous results, even though the new path works without problems. Clearly, more than ping-based probes are required.

To circumvent the ping-related problems, operators could instead move a single user flow (which currently traverses over the old path) to the new path. Real application traffic should not be dropped by on-path devices, thus, resulting in a precise signal for path reachability inference. Assuming the flow continues over the new path (i.e., ongoing traffic), we receive a strong signal that the new path provides reachability.

Unfortunately, the extracted signal still fails to fully reflect path reachability: *(i)* we can only infer the reachability of the IP address belonging to the single, moved flow, and *(ii)* the generated signal heavily depends on the endpoint behavior. What if the application terminates the shifted flow soon after/while we infer reachability? Does that mean the path provides no reachability? Clearly not. In summary, using a single user flow improves the precision of the observable signal. However, it still needs to provide the required coverage to assess a path's reachability property fully.

The logical next step would be to move a *few* user flows to the new path; once again raising many questions. How many flows are enough? How do we select the flows, and at which point are we sure that the destination provides reachability? We first have to broaden the collection of network events that lead to reachability problems to find a technique that answers these questions. More precisely, we must go beyond straightforward outages, such as a link or device failures, that lead to an unreachable path. What if the reachability problems are related to the *amount* of traffic we forward over the new path? Figure 5.1 introduces two scenarios for which we could observe such behavior.

We first focus on the scenario on the left. The new path (in green) provides connectivity towards the dst AS. Probes, e.g., a simple ping [43] probe or

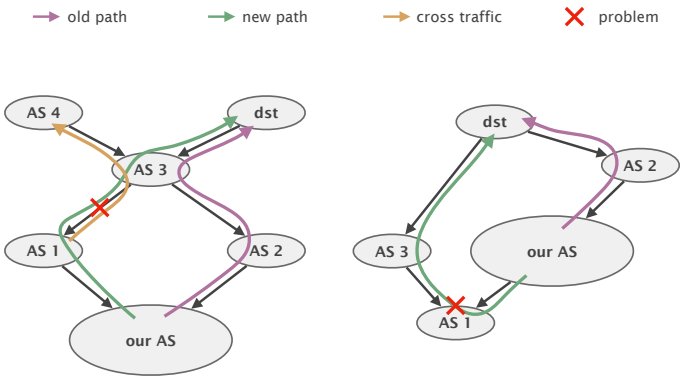


FIGURE 5.1: Two scenarios under which reachability problems are only visible with sufficient enough traffic. Arrows point from providers to customers. On the left, the new path shares a segment with a large amount of cross-traffic. On the right, AS 1 (our customer) leaks a route from another provider (AS 3) to us, i.e., due to a configuration mistake. AS 1 cannot handle all our traffic towards the dst AS.

a moved user flow, confirm that. However, AS 1 sends a large amount of cross traffic (in orange) towards AS 4, partially sharing a path segment with the newly advertised path. Once our AS starts to shift all traffic to the new path, we will immediately encounter congestion events resulting in packet losses for the flows towards the dst AS. The provided throughput is insufficient for our current traffic demand towards the dst AS.

The scenario on the right in Figure 5.1 shows another problem. Our customer (AS 1) *leaks* a route towards the dst AS from its other provider (AS 3). Usually, an AS does not advertise routes between different providers (compare Section 2.3.3). One reason for such a route leak could be a configuration mistake. Assuming our AS does not filter out the wrongly advertised route, BGP will immediately prefer the new path. After all, the new path over our customer (we earn money) is more lucrative than the old path over our provider AS 2 (we have to pay). Similar to the previous example, a simple ping probe or a moved user flow will reach the dst AS over our customer (AS 1). So the new path seems to work. Unfortunately, our customer's network is not provisioned to handle the large traffic amount we send towards the dst AS and might drop most or all of it once we fully commit to the new path. In the worst case, our customer is overloaded and

loses traffic unrelated to the dst AS. Route leaks are common and happen in small and large ASes [152].

In conclusion, moving user flows seems promising, but we cannot limit ourselves to an arbitrary number of flows.

Proposed solution We envision solving the problem by eventually moving *all* the flows from the old path to the new one. Unlike today's BGP algorithm, though, which moves all flows at once as soon as it selects a new best route, we gradually spread the movement over time. The network operator defines the maximal time budget for the traffic shift, which directly impacts how fine-grained we can assess the path reachability. A slow traffic shift enables us to *(i)* significantly reduce the number of lost packets in case the new path does not work; *(ii)* abort the process, i.e., immediately move back to the old path should we infer any reachability problems; and *(iii)* perform the reverse action, i.e., move everything to the new path, in case the old path starts to fail or gets withdrawn by a BGP neighbor during our movement process. Additionally, we cover all the active IP space in P, addressing the initially encountered coverage problem.

Reachability assessment on top of ongoing user traffic requires careful handling. Our actions should not result in reduced customer performance. However, note that our vision does not perform a different operation than what today's BGP algorithm would do. We just distribute the process over time. In fact, given that we can detect reachability problems *before* all traffic is moved to the new path and stop the process accordingly, our approach leads to *fewer* lost packets should the new path not reach P. Additionally, a single flow is only moved *once* (i.e., from the old to the new path) unless we later detect that the new path does not provide reachability.

5.2.2 *Control flow*

To assess reachability properties in an existing network, we have to overcome a fundamental control and communication challenge: We observe an inherent discrepancy between *(i)* the network device which currently connects to the old path and advertises the corresponding route internally, i.e., the egress connected to the old path; *(ii)* the border router which receives the BGP update advertising the new path; and *(iii)* network ingress points which forward most of the traffic to the advertised destination. In other words, who is driving and controlling the entire process of slow traffic

shifts for reachability inference? We explore different design ideas after stating our assumptions and introducing a sample network.

Assumptions Fine-grained traffic shifts necessitate an easy way for border routers to control where specific traffic is forwarded to. Chapter 2 introduced BGP-based forwarding, where every router maintains a forwarding table containing an entry (IP prefix and corresponding next hop) for each advertised destination. In such a setup, individual traffic shifts are difficult to achieve as routers forward packets based on their destination IP address and according to the best matching prefix. As soon as the BGP update for the new path propagates network-wide, routers will immediately move all corresponding traffic, preventing the slow traffic shift we envision.

To this end, we assume that the network either contains programmable hardware devices (e.g., Tofino [117] switches) which allow for flexible per-packet forwarding; or (more likely) uses label-based forwarding in its core. One typical example is Multiprotocol Label Switching (MPLS) [155]. Instead of BGP routes, core routers in an MPLS-based network forward packets according to a label in the packet header, resulting in a more flexible forwarding behavior with less overhead. The edge/border routers define the labels when packets enter the network and remove them at the egress. Therefore, a label change results in individual and targeted traffic shifts. Even newer technologies, such as Segment Routing over IPv6 (SRv6) [156], give network operators precise control over how individual packets flow through their network. We discuss possible deployment concepts *without* these assumptions in Section 5.4.

Sample topology We use Figure 5.2 as a simple example to discuss different system design options and their advantages and disadvantages. Routers $R3$ and $R4$ send different traffic amounts towards the dst AS currently using the old path via $R1$. Router $R2$ receives a BGP update announcing a new route towards the dst AS. $R2$ prefers this route (i.e., higher local preference) and would like to infer the reachability of the new path towards the dst AS. Note that $R2$ does not forward matching traffic to test with.

Our scenario includes the case where the border router for the old and new paths overlap. More precisely, $R1$ could also receive the BGP update, for example, over a different eBGP session. Note that this case is more straightforward to handle as $R1$ already observes the traffic, which might be shifted to the new path.

The following paragraphs explore three different design principles for reachability inferences of the new path by slowly moving traffic: the egress

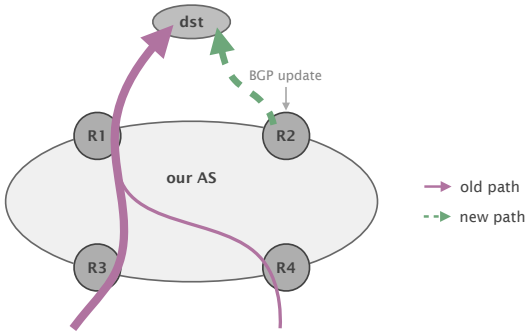


FIGURE 5.2: Two ingress routers ($R3$ and $R4$) send *different* amounts of traffic towards the dst AS over the old path via $R1$. The egress router $R2$ receives a BGP update advertising a new path towards the dst AS.

connected to the *old* path controls the process; the egress connected to the *new* path controls the process; or the *ingresses* control the process.

The old egress controls the process The first idea heavily focuses on the current egress connected to the old path ($R1$). $R1$ seems like the most logical place as it already carries a lot of matching traffic that flows over the old path. Once $R2$ receives the BGP update message, it forwards it to $R1$ over their iBGP session. After that, $R1$ steers the process of validating reachability over the new path by shifting traffic to it, for example, by changing MPLS labels of incoming packets. However, such a control flow drastically affects the overall forwarding behavior of the AS. First, a single egress router ($R1$) decides over all the traffic in the network towards the dst AS, which currently uses the old path. And second, the internal forwarding paths during the reachability check are suboptimal. $R3$ and $R4$ forward the traffic to $R1$, the current best route. At the same time, $R1$ gradually moves some of the traffic to $R2$ to validate the reachability of the new path. As a result, the network performs unnecessary traffic redirections.

The new egress controls the process The second idea gives $R2$ complete control over the process. $R2$ is the first border router receiving the BGP update for the new path, which is a strong argument to coordinate the entire process. However, this opens the question of how $R2$ receives traffic to validate the reachability of the new path. Currently, most/all traffic is forwarded over $R1$ (compare Figure 5.2). So $R2$ initially has no or only very little traffic to test with. We see two different solutions which build on top of each other:

$R2$ can use the existing iBGP sessions in order to receive traffic to test with. More precisely, $R2$ sequentially sends a BGP update for the new path via iBGP to each other ingress, simulating a traffic shift over time. Note that the existing BGP algorithm would send all updates at once. As a result, we slowly receive more traffic at $R2$, which validates the new path's reachability property. If $R2$ discovers that the new path does not provide reachability, it will immediately withdraw the route, and the other ingresses will use the existing old path again.

On top of that, we no longer face the problem of the previous idea, i.e., that a single egress dictates the forwarding behavior for all corresponding traffic. Every ingress could decide on its own to ignore the new path and continue with the current, old path. Section 5.3 explores the integration with the BGP decision process in more detail.

Although the idea seems to work in theory, we still face two problems: *(i)* if we apply such a design to our example in Figure 5.2, $R2$ will receive a massive amount of traffic once $R3$ gets the update and nearly no traffic from $R4$. This does not result in the expected slow traffic shift we had in mind. In an extreme case, one ingress carries all the traffic, resulting in a traffic shift that resembles today's BGP behavior. And *(ii)* after $R2$ sends the update to one neighbor via iBGP, it cannot decide between an ingress router that ignored the new path and an ingress router that currently does not forward matching traffic. As a result, $R2$ encounters difficulties in scheduling the movement process.

A better solution would allow $R2$ to *request* a specific amount of traffic from each ingress router. $R2$ then periodically increases the amount until, eventually, all traffic uses the new path. This results in a smoother traffic shift. However, this approach requires new communication capabilities between $R2$ and the other routers. Additionally, we increase the computation load on all ingress routers as they need to forward specific amounts of traffic and keep the corresponding state. We are unsure if these drawbacks justify the benefits we get. As a result, we explore a third and final control flow design.

The ingresses control the process Finally, the third idea gives the different ingress routers control over the process; in our example, that would be $R3$ and $R4$. They receive the incoming traffic, which should either go to the old or new path. As soon as $R2$ receives the BGP update for the new path, it announces it to $R3$ and $R4$, respectively. Both routers then individually move their traffic from the old to the new path and eventually

make a decision based on the inferred reachability property. That avoids new communication techniques but adds uncertainty to the two egress points ($R1$ and $R2$). Without further notifications, they do not know at which point in time the ingresses made their decisions. Both egresses need to continuously provide the corresponding routes in their forwarding table. Over time, that can lead to a greatly increased number of forwarding entries. Note the difference to today's BGP algorithm. For example, if $R1$ decides that the new path over $R2$ is better, it will replace its forwarding table entry accordingly and send a BGP update to all other routers in the network.

Conclusion Deciding which entity should steer the process is difficult to answer. Our exploration of three different design ideas highlights that we need to find a tradeoff between: *(i)* additional on-device state for ongoing traffic shifts; *(ii)* additional forwarding state to provide the old and new route while we test them; *(iii)* additional communication overhead between the routers; and *(iv)* the achieved smoothness in the performed traffic shift.

5.3 PATH-PROPERTY-AWARE BGP DECISION ALGORITHM

Now that we better understand which devices could steer a path-property-based inference step, we explore how to integrate such operations into the existing BGP decision algorithm. We envision three deployment strategies: *(i)* Verify the reachability properties whenever the existing BGP decision process (compare Table 2.1 in Section 2.3) decides to select a new best path. *(ii)* The network operator gets control over how critical path-property-driven decisions are by freely placing our operations as a new block in the decision process. However, due to the binary outcome (reachability or not), our block will always terminate the decision process should we reach it. Or *(iii)* we define additional criteria which express path-property-based equality, leading to a tighter integration while raising new challenges. The following subsections discuss all approaches in detail.

5.3.1 *Mandatory reachability checks upon the selection of a new path*

The most straightforward integration adds our reachability check when three conditions hold: *(i)* BGP decides that a newly received update provides a better path, i.e., the existing decision process prefers the new path; *(ii)* one of our BGP neighbors continues to advertise the old path; and *(iii)* the

next hop of the new path differs compared to the old one, i.e., at least the beginning of the new path is different to what we currently use.

(i) Ensures that our property inference step activates only when BGP moves all traffic to the new path (Section 5.4.4 discusses different BGP scenarios). Currently, the traffic shift happens at once; in our vision, we will prolong the shift over time. Due to *(ii)*, we provide the additional benefit that we can abort and reverse the traffic shift should the new path lack the expected reachability properties, i.e., the old path still exists. Finally, without *(iii)*, the old and new paths share the next hop, meaning we cannot actively choose between them. Even if the two paths start to diverge later (e.g., different AS paths), BGP lacks a control mechanism to decide which path our packets follow (in remote ASes).

Advantages The tight correlation of the property-based inference step with scenarios that unavoidable lead to a traffic shift (with today's BGP algorithm) strengthens our cause. More precisely, if the new path works well, we *only* add some delay to the process. However, if the new path lacks reachability, our approach actively prevents the complete packet loss we would observe with the existing implementation.

Disadvantages The first condition *(i)* neglects scenarios in which the old path exhibits problems that might be detectable with path-property-based decisions, making the new path the better choice. In other words, given that our decision step is only considered if default BGP prefers the new path, insights based on inferred path properties can never overturn the existing control-plane decisions.

5.3.2 *A deciding step in the existing decision process*

A completely different approach gives network operators full control, given that they might have different preferences on how important path properties are for their network. They decide *where* we consider path attributes in the existing decision process as an additional, new step.

Every existing BGP decision step (with the exception of the last one) has three different outcomes if we compare an old and a new route. First, end the decision process and prefer the old route. Second, end the decision process and prefer the new route. Or third, both routes are equal according to the current criteria and proceed with the next decision step.

Our inference only has two outcomes, either the new path provides reachability or not. Let us assume we place our traffic shift as a new step in the existing decision process. If our inference indicates that the new path provides reachability, we have already moved most/all traffic to it. Although possible, continuing with the following decision step would raise new questions. What happens if a following decision step concludes that the old path is indeed better? Do we move all traffic back? For this reason, our second design idea assumes that our decision step always terminates the process, either preferring the new path if reachability exists or continuing to use the old path instead.

Network operators control *where* the property-based decision happens by placing our step between any of the six existing ones introduced in Table 2.1. We could even place it at position zero, i.e., before all remaining steps. For example, assume an operator places our inference after step three, which means if the BGP decision algorithm finds a unique best route in steps one to three, our reachability inference will *not* happen. However, if the process reaches step four, we will get active. Consequently, the placement of our system in the decision algorithm directly translates to how important path-property-based considerations are for an operator. The following paragraphs describe the characteristics of different placements:

Before step 1 If we place our block before the first step of the original BGP decision process, we skip any existing metrics and always decide based on the reachability of the new path.

Although such a placement has advantages from a data-plane perspective, it ignores any decisions based on the local preference value or other BGP attributes. That means the main monetary-based decision criteria related to customer, peer, or provider peering sessions no longer play a role – a fundamental break with the existing behavior.

We envision two ways to give operators more fine-grained control: *(i)* an operator limits the execution of our block to specific prefixes, for example, destinations with critical customer traffic. Or *(ii)* the operator provides one or multiple local preference *ranges*. Our decision block is only active if the old and new routes belong to the same range. One range could, for example, contain all local preference values which belong to customers. As a result, we only perform a path-property-based decision as long as we have to decide between and old and new route belonging to the same peering category.

Between steps 1 and 2 If we place our block after the first step, we will only consider a new path with the same local preference value as the old one. The most critical BGP decision is already made. For these paths, our system would then perform the reachability inference. We argue that this is a suitable placement if an operator thinks the following step, i.e., AS path length comparison, does not reflect data-plane behavior. Indeed, this is often the case when operators use techniques such as AS path prepending [150] for traffic engineering.

Between steps 2 and 3 Placing our system before step 3 (MED comparison) is interesting as it would allow us to make a more informed decision rather than following the lower MED value, which is set by a neighboring AS, i.e., an external entity.

Between steps 3 and 4 A placement at this point allows us to select an iBGP path even if we have an eBGP one available. That means we would send more traffic through our AS (to reach another exit point) rather than directly sending it to the neighbor over the given eBGP connection. A small price to pay for reaching a path whose reachability is inferred.

Between steps 4 and 5 Similar to the previous point, a placement here would allow us to forward the traffic over a non-shortest internal path to reach the new path.

Between steps 5 and 6 At this point in the decision process, BGP already decided that the two paths are equal from a control-plane perspective and only a tie-breaking step is left. We argue that a path-property-based inference should *always* occur if the BGP decision process reaches this step. The tie-breaking mechanism based on the smaller IP address is entirely random and does not consider any network or path signals.

After step 6 A placement after the final step does not provide any benefits, given that BGP already made a final decision and selected a single path.

5.3.3 *Path-property-based equality*

The final design idea builds on top of the previous one. Once again, we give network operators free control over where they place our path-property-based decision step in the existing process. Additionally, we define equality criteria for path-property-based decisions between the two paths. We explore two options:

Intelligent traffic shifts With this approach, we define a reachable new path as the equality outcome and continue the decision process normally, i.e., the existing, following control-plane-based steps will decide which path is better. If the decision process eventually prefers the new path, we are done. All traffic already follows the new path (as we moved it to infer reachability). However, if BGP instead prefers the old path, we intelligently move the traffic back. More precisely, we keep ongoing flows over the new path while we forward new flows over the old one. Eventually, all flows once again use the old path.

Note that this approach leads to additional: *(i)* state on the devices responsible for the traffic shift; *(ii)* discrepancy between the advertised control-plane behavior and the data-plane forwarding actions; and *(iii)* routing table entries while we still have ongoing flows to forward over the new path. Section 5.4 discusses these problems more generally.

Performance-based path properties A more radical approach extends the inferred path properties. Instead of purely focusing on the reachability aspect, we infer additional path properties related to performance criteria, for example, the minimum RTT or the current throughput. While moving the traffic, we can perform these inferences for the old *and* the new path. As a result, we compare the two paths, which naturally leads to the three expected outcomes of a BGP decision step: prefer the old path, the new path, or both are equal. In case of equality, we continue with the next step in the decision process and eventually move all traffic to the selected, better path. Unlike other performance-based routing systems, which continuously monitor and update their path selections (e.g., [157]), our approach only decides after receiving a BGP update. That limits the number of path changes and events, such as frequent route flapping.

5.4 ADVANCED CHALLENGES

This section introduces advanced challenges and BGP scenarios which we did not address so far.

5.4.1 *Deployment in a network with a full BGP core*

Without programmable devices or label-based forwarding (compare assumptions in Section 5.2.2), we face additional challenges when deploying

our ideas in a physical network: *(i)* moving specific traffic to a dedicated path contradicts the prefix-based forwarding behavior in the network; and *(ii)* each router needs to decide between the two path options, leading to unexpected traffic deflections. For example, the ingress decides to use the new path. However, a router on the internal forwarding path towards the egress point prefers the old path. As a result, the ingress traffic gets deflected to the old path. Note that this can happen as our path-property-related decision block breaks with the currently used decision process and needs additional time to validate the new paths' reachability.

A possible solution to *(i)* involves the internal announcement of new, more-specific prefixes that temporarily move some traffic using the normal BGP forwarding behavior. In an extreme case, temporary /32 prefixes could move traffic on an individual IP basis. However, that increases the size of the local forwarding tables and does not allow for fine-grained traffic shifts.

(ii) heavily depends on which control flow we follow (compare Section 5.2.2). If, for example, the egress router connected to the new path steers the entire process, we can carefully control in which order the new path is announced to other routers via iBGP, preventing some deflection problems. Note that route deflections can happen in networks, even if we do *not* use our property inferences [158]. The following subsection explores the problem more generally.

5.4.2 *Synchronized data and control planes*

In the absence of configuration mistakes and outages, the control and data plane are usually synchronized once BGP converges. That means data packets follow the path specified in the control plane. The recommendations in [159] define different benchmarking methodologies to evaluate data-plane convergence upon control-plane changes and lead the way for a detailed analysis of our idea in future work. In this initial, exploratory study, we only locate the main challenges with our ideas.

The previous subsection introduced problems inside our network. However, the proposed slow traffic shift to infer reachability also impacts consistency between ASes. More precisely, while we infer the path reachability, the currently advertised path via eBGP, e.g., to one of our customers, does not always match the forwarding path their packets take. We distinguish three stages: *(i)* before we start to move any traffic; *(ii)* while we move the traffic to infer reachability; and *(iii)* after we make our decision.

(i) The planes still match before we start to move any traffic. More precisely, we did not yet advertise the new path to our neighbors, and all traffic still follows the old one.

(ii) The planes start to diverge during the traffic shift to infer reachability: We still advertise the old path to our neighbors while we move the traffic. As a result, some of their flows (towards the same destination) might take the old path while other flows already follow the new path.

BGP does not provide a method to express such a traffic shift unless we move the traffic in sub-prefixes (e.g., each /24 individually [59]) and temporarily advertise these prefixes to our neighbors. However, this idea is suboptimal for two reasons. First, temporarily announcing new /24 prefixes, especially if we do not own the prefix space they belong to, quickly looks suspicious (could be seen as a BGP hijack) and leads to additional forwarding entries. And second, if one /24 sub-prefix contains most of the traffic, we will not end up with a smooth traffic shift. Note that we *cannot* simultaneously announce both routes (old and new) to a neighbor. The BGP update for the second route will implicitly withdraw the first route as they belong to the same prefix [20].

For these reasons, and given that we could move back to the old path should we discover reachability problems with the new one, we believe it is best to keep the existing announcements until our reachability inference is complete. If we prefer the new path, our routers immediately advertise the new route to their neighbors, as described in *(iii)*.

(iii) After we move all traffic to the new path and make our decision, the corresponding routers will immediately announce it to their eBGP peers (assuming we prefer a new path). Once a neighbor processes the announcement, the planes match once more. This process does not differ from today's approach after BGP selects a new best route.

In conclusion, during the traffic shift, the planes temporarily misalign for our neighbors (and internally). Section 5.6 lists related work that could solve some of the described problems.

5.4.3 *Lack of traffic*

Similar to our observations with the *Oscilloscope* system, we cannot assess path reachability properties if our network does not carry any user traffic towards the destination for which we receive a BGP update. This is an

inherent problem of data-driven approaches. Interestingly enough, the current lack of traffic could stem from a reachability problem over the existing, old path. We advocate for the following approach: if the BGP update reaches our block in the decision algorithm (gives control to the network operator via its placement), and we realize that we do not carry any matching traffic, we should prefer the new path. Most likely, nothing changes, as we do not forward any traffic. If the old path was *not* reachable, we ended up with a better path by selecting the new one.

5.4.4 *More involved BGP events*

So far, our examples always considered one current path (old) and a single incoming new path. In larger networks with multiple peers, we often have more than two possible path options available to reach a specific destination. Furthermore, BGP updates can withdraw routes or modify existing ones. Let us consider the following scenario: a router knows one current best route and *multiple* non-best routes. We assume that our system would infer reachability whenever BGP selects a new best path (compare Section 5.3.1). This scenario leads to the following more involved cases:

- The router receives a BGP update which modifies a non-best route, but the currently best route is still preferred. In this case, we will not apply any reachability inference as we do not select a new path.
- The router receives a BGP update which modifies a non-best route, making it better than the current best route (the old path). Eventually, the BGP decision process selects this route as best, and we can infer reachability, moving traffic from the existing path to the new one.
- The router receives a BGP update for the currently best route, making it worse than one or multiple non-best routes. One reason could be that the BGP update announces the same prefix but with a longer AS path. This case is interesting as the incoming update does not represent the new path. It rather modifies the old one. Following BGP's existing decision process, one of the other paths will be selected as the new best route. As a result, we can easily apply our reachability inference. We still have a currently used path and a new option.
- The router receives a BGP update which withdraws one of the non-best routes. Given that the router does not use the withdrawn route, we ignore this event and do not perform any path inferences.

- The router receives a BGP update which withdraws the current best route (i.e., the old path). A withdraw message is a strong signal that the existing, old path either no longer works or will soon be disrupted (for example, to perform maintenance work on a router or link). We should react immediately and *not* perform time-intensive reachability checks resulting in slow traffic shifts. In such a case, we follow the existing BGP algorithm to select the new best path, e.g., by skipping our reachability inferences entirely. All traffic moves to the newly selected path at once. Optionally, we start a reachability inference once all traffic follows the new path (e.g., a check if traffic keeps making progress over the new path). But the priority should be to move traffic away from the withdrawn path as soon as possible.

In conclusion, we can also apply our ideas to more involved BGP scenarios. An essential requirement is that we have an existing path and a new one. However, we need to handle withdrawn routes carefully and not artificially prolong the traffic shift. As part of future work (Section 5.7), we also formulate an additional challenge. We consider that we deploy the reachability inference as part of a new step in the decision process (Section 5.3.2) while facing the previously discussed scenarios: More than two paths might reach our decision step. How do we perform reachability inference in this case?

5.4.5 *Communication between devices*

We need additional communication methods between the different routers to realize our ideas. For example, in order to signal *how much* traffic needs to be shifted to *which* egress/path. In the best case, we use existing iBGP methods, for example, by advertising a new route only over selected iBGP sessions. However, that limits us to prefix-based per-router traffic shifts, which are not fine-grained enough. Even more problematic, if we go beyond our assumptions of programmable devices or label-based forwarding (Section 5.2.2), messing with the iBGP update order can lead to route deflections and forwarding loops – problems which we observe even without our changes to BGP [158].

Better solutions include programmable switches, such as the Tofino [117], which communicate over and react to packets with custom headers. Additionally, research work such as [160] introduces new ways for faster routing

protocol innovations and could enable some of the required communication channels with limited overhead.

5.4.6 *Bursts of BGP update messages*

Another challenge comes with BGP update messages which either announce multiple prefixes at once or arrive in entire bursts. As an example, a failure of a link or important router might lead to a burst of BGP withdrawals in a short amount of time. Swift [161] studies such events and uses corresponding BGP update message patterns to predict remote outages. For our reachability inference, many new paths could lead to scalability issues. After all, we suddenly have to perform slow traffic shifts for all corresponding destinations.

We envision two solutions: First, in case of bursts of withdrawn routes, our reachability inference does not operate as we follow BGP's default behavior to move affected traffic to a new path as soon as possible (compare Section 5.4.4). And second, in case of multiple *announcements* of new routes, we can leverage insights from the *Oscilloscope* system and combine prefixes that originate from the same remote AS (i.e., similar to *Oscilloscope*'s combined signals in Section 4.4). As a result, we can scale the reachability inference by only performing a slow traffic shift for one prefix per remote AS. Once we confirm reachability for an AS, we move all traffic belonging to related prefixes at once. Note that such an approach could miss reachability problems located very close to the origin of a prefix, for which we did not perform a slow traffic shift.

5.5 COMPARISON WITH DEFAULT BGP'S BEHAVIOR

This section compares our vision with the default BGP algorithm along four metrics: performance gain, traffic impact, routing table size, and convergence time.

Performance gain We first explore the performance gain of our ideas compared to default BGP operations. We consider multiple cases:

- Both old and new paths provide the expected reachability towards the advertised destinations. In this case, we provide equal performance compared to the default BGP operations; we just spread the traffic shift over time.

- The old path works well, but the new (better) one does not provide reachability. The default BGP algorithm has no insight into path-property signals and will immediately forward all traffic over the broken path. With our approach, though, we will only lose a fraction of the traffic, i.e., the shifted flows, until we detect a lack of reachability. Afterwards, we will immediately move back to the old path. In this case, our approach provides an apparent performance gain.
- The old path lacks reachability, but the new (better) one provides full reachability. If we do not observe any traffic over the old path (e.g., retransmissions due to lost packets), we will immediately fall back to BGP's default behavior, as discussed in Section 5.4. As a result, we perform equally compared to today's BGP algorithm. Should we still observe some traffic, we will gradually shift it to the new path, confirming its reachability property. In this case, we might prolong the traffic shift compared to BGP's default operation. However, note that we will eventually select the working path.
- Both paths lack reachability, for example, due to a problem close to the destination. In this case, neither default BGP nor our system finds an adequate solution. In the future, we envision two interesting extensions to our approach. First, our system detects outage-related traffic patterns, e.g., increasing TCP retransmission times (successfully used in [111]), to identify the lack of reachability over both paths. As a result, we can automatically notify the network operator, which is impossible to achieve with today's BGP algorithm. Second, building on the previous idea, we could instead try a potentially available backup path. That means neither the old, currently used path nor the newly advertised one, but instead, one of the other (non-best) paths that the router might know (compare Section 5.4.4).

In conclusion, with our vision, we (eventually) always select the better path while significantly reducing the number of lost packets should a new best path not provide reachability.

Traffic impact Another metric is our impact on the traffic towards the destination of the incoming BGP update. More precisely, how often do we switch a flow between different paths. As we saw in our *Oscilloscope* work, forwarding paths exhibit varying properties, such as different delays. Changes in the forwarding delay or packet loss rate impact application traffic, for example, video streaming [154].

The following statement describes the behavior of BGP *and* our vision: Every flow towards a destination P is moved between paths at most as often as our network receives BGP updates for P during the flow's lifetime.

It is easy to see for default BGP. In the worst case, every BGP update results in a better path, moving the traffic accordingly. With our vision, we have to consider two additional aspects. First, given that we spread the traffic shift over time, a flow might start *after* we receive a corresponding BGP update. However, the matching portion of the prefix space still needs to be moved to the new path, resulting in an additional movement for this specific flow. That being said, the same holds when a flow terminates. More precisely, the flow could terminate after we receive a BGP update, but we did *not yet* move this flow to the new path. Second, should the new path lack reachability, we might move flows two times, i.e., once more than with default BGP operations. However, thanks to our path property inference, we eventually select the working (old) path, which BGP cannot achieve. Path reachability negates potential drawbacks of the additional traffic shift.

The following traffic impact is more challenging to assess theoretically and leads to interesting future measurement studies: Default BGP moves all flows at once to the new path. Compared to the currently used path, the new one might exhibit different throughput properties, either due to different physical capabilities or ongoing cross-traffic using the same path. As a result, all moved flows compete (with each other and the cross traffic) for a fair throughput share. Mechanisms such as TCP's congestion control algorithm [162] actively increase or decrease the congestion windows, impacting the amount of traffic a single flow exchanges at once.

In our vision, the underlying scenario is the same. However, we *slowly* move the traffic to the new path. Consequently, not all flows start to adapt their congestion windows at the same time. The slower movement could result in better (or worse) user performance, calling for a detailed practical measurement study.

Routing table size Our reachability inference step leads to increased routing table sizes depending on the selected implementation strategy (compare Section 5.2.2). The reasons are twofold: *(i)* while performing the traffic shift, the corresponding router needs to know both routes; and *(ii)* the egresses that connect to the old or new path cannot immediately remove the corresponding routing table entry as other nodes in the network might still perform reachability inference. Problem *(i)* is only temporary and local

to a single router. Once the traffic shift is done and a decision is made, the router removes the second table entry.

To solve *(ii)*, we envision two potential solutions: Either the border routers actively communicate once they complete their inference steps, for example, using ideas discussed in Section 5.4.5. Consequently, the egress router can remove no longer used entries eventually. Or alternatively, the egress routers initiate a "cleaning process" once they no longer receive matching packets for a given destination (according to an operator-defined timeout period). We assess the forwarded traffic amount via custom counters in programmable devices or via existing monitoring solutions on classical routers (e.g., [51]). Once the router removes/cleans a corresponding route, it will send a BGP update via iBGP to actively withdraw the route.

Convergence time Finally, our vision also impacts the convergence speed of BGP. We consider two aspects: convergence time inside a single network and Internet-wide BGP convergence, assuming multiple (all) ASes use our ideas. Generally, we must remember that our inference is only active if we already know a currently used old path. In critical scenarios, i.e., a route withdrawal, we will immediately follow BGP's default behavior and *not* increase the current convergence time.

As discussed in Section 5.2.1, the network operator defines the maximum time budget for the traffic shift. Consequently, the operator controls the convergence time inside a network, and the routers distribute the traffic shift accordingly. More time for the traffic shift, i.e., a *longer* convergence time, allows for a *quicker* reaction in case of an unreachable path and, thus, *fewer* lost packets.

The situation is more interesting in a global deployment scenario. At first, we might think that a BGP update necessarily results in a path reachability validation step in each network it passes through. However, that is often not the case, as we show in Figure 5.3. After a temporary link failure between AS 2 and AS 5, AS 5 once again receives AS 2's prefix over the direct connection. As a result, AS 5 performs a reachability inference operation and eventually forwards the new route to AS 8. Similarly, AS 8 infers the reachability of the new path and further announces it to AS 7 (and AS 6). However, even though AS 1, 3, 4, and 7 also receive a BGP update for the new path (the AS path changes), they do not perform the reachability inference as the old and new path share their next hops. Note that AS 6 does not perform the inference step as it places our decision step towards the end of the decision process. The customer path (via AS 2) is preferred.

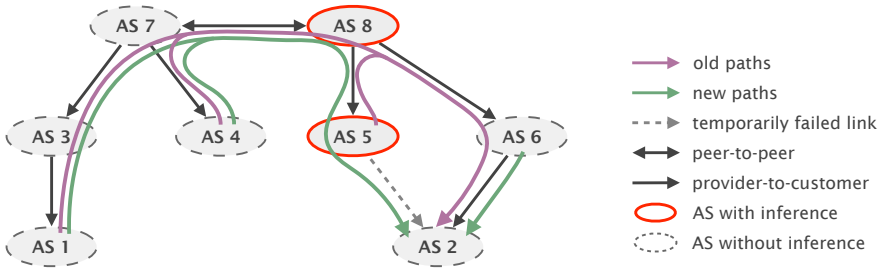


FIGURE 5.3: After AS 8 infers the reachability of the new path via AS 5, all following ASes (1, 3, 4, and 7) can no longer perform a reachability inference of the new path, even though they receive a BGP update with a new AS path (same next hop for old and new path). AS 6 places our inference step after the local preference comparison in the decision algorithm, always preferring customer over provider routes.

More generally, once the old and new paths converge, the following ASes no longer infer the path reachability. Another way to express this behavior is to understand AS 8's BGP update to AS 7 as an implicit withdrawal of the old path. Following our intuition for these cases (Section 5.4.4), we will immediately fall back to BGP's default behavior and *not* perform reachability inference.

Note, once again, that the number of traffic shifts would be the same as with the default BGP behavior. AS 5 and AS 8 move all traffic once the default BGP algorithm selects the new best path. In our use case, we just perform a slower traffic movement.

Conclusion Our exploratory comparison with BGP's default behavior shows that our ideas lead to an apparent performance gain in case a new path does not provide reachability. Other aspects, such as the impact of the slow traffic shift on experienced user performance, require more detailed studies in the future.

5.6 RELATED WORK

Detecting BGP anomalies is well studied, and [163] summarizes different anomalies and detection techniques. For reachability problems, work such as [101, 145] mainly focuses on BGP hijack-induced reachability losses. They use active probing tools, e.g., ping [43] and traceroute [164]. Our ideas

directly infer the reachability property based on user traffic, preventing the generation of additional traffic and artifacts due to blocked probe packets.

Property-driven inter-domain routing decisions In early work, Savage et al. [165] studied performance and reachability problems. They discussed the idea of virtual routers tunneling packets around the detected problems, i.e., the traffic makes a *detour*. Tunneling is another approach to performing fine-grained traffic movements and could complement our ideas to slowly shift traffic to a new path.

Bush et al. [166] explore the reachability of *new* address space, i.e., IP prefixes announced for the first time. They use traceroute and ping packets to infer the reachability of the new prefix space to and from different probe locations. Such measurements complement our effort of reachability inference for new paths, for example, when deciding to use a route for a prefix that our network observes for the first time. However, they need support from external probes to infer Internet-wide reachability precisely.

The Blink [111] and Swift [161] systems detect *remote* outages (resulting in reachability problems) and quickly forward affected traffic to a backup path. More generally, [167] summaries various data-plane fast recovery mechanisms. Blink recognizes typical TCP retransmission patterns, while Swift predicts the overall extent of the outage after receiving a few BGP update messages. However, these systems are another example of a reactive approach. *All* traffic is already lost for an extended time period before they detect the outage. Our ideas take a more proactive approach, inferring the reachability before we move *all* traffic to a new path. Additionally, we are only active once a new path is advertised, reducing the state we must keep for the reachability inference.

RouteScout [157] goes a step further and considers changes in path loss and delay properties to actively move affected traffic to other available paths. Programmable data-plane devices perform the traffic shifts once operator-defined objectives are violated. Similarly, SDNMA [168] heavily focuses on Software-Defined Networking (SDN) to monitor throughput, loss, and delay based on existing monitoring solutions and crafted packets. The SDN controller reroutes traffic to other eBGP peers if the performance falls below a threshold. Our idea focuses on reachability and infers this path property only when receiving new BGP routes. As a result, we can scale better than always-on monitoring approaches and induce fewer route changes for end-user's traffic.

[169] and [170] are two examples of systems that extend BGP with Quality of Service (QoS) features. Both systems include QoS metrics in BGP update messages, impacting the BGP decision process. Our vision goes another way and does not require active additions of path attributes in the BGP update messages. Instead, we infer path reachability properties based on locally collected signals and include these in the BGP decision process. Our approach mitigates problems of lying networks that could include wrong performance metrics in their update messages.

Property-driven intra-domain routing decisions Other related work focuses on intra-domain, rather than inter-domain, performance-based routing. Contra [171] enforces operator-defined performance policies with programmable network devices and custom probe traffic. MATE [172] improves resource utilization inside a network by moving traffic over multiple available paths between an ingress and egress. The strong focus on MPLS-based networks is interesting for our idea, showing that our vision of MPLS-based traffic shifts is feasible.

Network and endpoint coordination Birge-Lee et al. [173] show that collaboration between edge networks can lead to better measurements and more diverse inter-domain paths. Our vision would greatly profit from some of the collected signals and could combine them with incoming BGP updates. However, their approach requires explicit collaboration from edge networks, and how it scales to a large number of networks needs to be clarified. Our ideas only look at locally collected signals.

Big players use their insights in both traffic endpoints, i.e., servers providing content and applications running on end-user's devices, to infer path properties and adapt their forwarding decisions accordingly. Two examples are Google's Espresso [120] and Facebook's Edge Fabric [174] systems. In comparison, our ideas do not require control of both endpoints and infer all signals based on the locally observed traffic in a single network.

[175] highlights that sophisticated, performance-aware routing algorithms of large content and cloud providers often only perform marginally better than BGP when considering low latency. In other words, beating BGP is challenging. These insights confirm our focus to only include simple yet missing properties, such as path reachability, in BGP's decision process.

Routing process coordination Other related work focuses on new ways to coordinate and centralize the routing process allowing for more control. CIRCA [176] offloads inter-domain route computations to the cloud.

CIRCA's global view prevents convergence problems and simplifies certain routing decisions. RCP [177] is a logical, centralized platform that controls the routing decisions of a network. It performs per-router decisions for an entire network communicating via unmodified iBGP sessions with the existing routers. Finally, John et al. [178] propose consensus routing, a new routing technique that heavily focuses on consistency. For our vision, these systems could better coordinate and activate the slow traffic shifts upon receiving new BGP paths. That would simplify some of the discussed control flows and communication challenges.

5.7 CONCLUSION AND FUTURE WORK

This chapter combined our insights from the previous two chapters and explored how inferred Internet path properties can influence active routing decisions. Instead of blindly trusting a new best path – BGP's current default behavior – we first infer the path's reachability property while slowly moving traffic to it. Although a reachability check seems simple, we discussed several challenges and tradeoffs. On the one hand, slowly moving traffic allows path property inference and allows the possibility to abort the process upon encountering unexpected behavior. On the other hand, we also further prolong the BGP convergence process. We additionally explored the discontinuity between border routers observing incoming new routes and routers carrying the matching traffic for reachability inference. This leads to increased communication and synchronization overhead. Overall, our exploratory study only scratches the surface and introduces the general idea, leading to additional measurement and theoretical studies.

5.7.1 *Future work*

To fully embrace the possibility of routing decisions based on inferred path properties, we envision three directions for future work.

Inclusion of additional path properties Continuing our ideas from Section 5.3.3, the potential of path-property-based routing decisions does not stop at reachability inference. Including additional properties could lead to better user performance, reduced congestion, or higher security guarantees. However, inherently pairing routing decisions with path performance

properties or frequently evaluating and changing routing decisions result in new challenges and necessitate tight control and feedback loops.

Inter-AS signaling Figure 5.3 shows one example of how a routing decision based on path properties could influence the overall global BGP convergence speed. Future work needs to explore the impact in detail. However, we also see the potential for additional inter-AS signaling to improve the convergence speed. BGP communities, a BGP attribute distributed via BGP update messages, could play a crucial role. On a high level, BGP communities are custom tags that routers add or remove to advertised BGP routes. Today, network operators use BGP communities to implement internal policies or communicate with customers, providers, or IXPs [179]. In the future, we foresee an additional use case: Specific BGP communities could indicate path-property-related features of the advertised route.

One example, although with the opposite goal than our reachability focus, is studied in [180]: More and more ASes provide specific BGP community values which, once set, indicate that traffic following the announced BGP update should be dropped, i.e., go to a blackhole. Network operators use such services to tackle malicious attack traffic, for example, during a Denial of Service (DoS) attack.

Exploration of non-best path options The previously explored use cases always consider a single old and new path, allowing for a direct comparison. However, in larger networks, connected to multiple customers, providers, and peers, routers know one best route and *multiple* non-best ones. We see potential in inferring and comparing the properties of *all* available paths towards a given destination. Including such an inference operation in the existing decision process raises additional questions. For example, we need to send traffic over the corresponding path for most property inferences. How would we distribute the available traffic to all path options, and what are the implications for the end users?

6

CONCLUSION AND OUTLOOK

In this dissertation, we looked at Internet packet paths and explored how network operators can infer beneficial properties to perform various tasks. *Magnifier* reveals packet ingress and egress points by combining existing packet sampling techniques with targeted mirroring rules. One significant advantage is *Magnifier*'s validation approach, which notifies operators as soon as an ingress or egress observation is no longer valid.

The precise knowledge of *where* packets enter or exit their network allows operators to collect matching signals, such as Round-Trip Time (RTT) samples, which reveal properties of the path segments outside of the local network. *Oscilloscope* continuously collects RTT samples and looks for unexpected changes. By aggregating and validating the observations using statistical tests, we can detect BGP hijacks due to their induced RTT change. Additionally, we explored new TCP estimation techniques for transport protocols with encrypted transport headers.

Finally, we built on top of the concepts of *Magnifier* and *Oscilloscope* to connect the inferred path properties with active routing decisions. We argue that BGP's "all-or-nothing" approach, combined with the lack of path-related insights, leads to suboptimal routing decisions. We advocate adding simple path properties to BGP's existing decision process. For example, by slowly moving traffic towards a new path (assuming the current path still works), we can prevent problems such as a complete traffic loss due to a path that does not provide reachability. However, path-property-based routing decisions also lead to new challenges related to convergence, communication overhead and end user performance.

6.1 OPEN PROBLEMS AND FUTURE SOLUTIONS

This section highlights more fundamental, open problems for better Internet path property inference. We also discuss research ideas and upcoming fundamental design changes which could solve some of them.

6.1.1 *Noise leads to property inference rather than precise extraction*

In a perfect networking world, operators could extract any needed path properties with high precision. However, different noise sources and the limited view of network operators instead lead to path property *inference*. We envision at least two ways to solve the problem in the future.

Protocol improvements One idea focuses on improvements to the existing Internet and transport protocols. Future development and standardization efforts could include additional information for path property inference directly in the packet headers. We highlight one possible design in our Path Layer [6] work which is motivated by two observations:

First, the historical changes in Internet traffic types and usage patterns show an ever-evolving system. A good example is the QUIC [113] transport protocol which made on-path RTT estimation more difficult. Even if we design systems to use path properties that we can easily extract today, it is unclear if they will continue to be helpful in the future.

Second, in Chapter 2 we introduced two Internet packet headers. The IP header represents the network layer, while the TCP header is one example of a transport layer header. Most Internet packets contain headers belonging to additional layers, such as the link or application layer [181]. All these layers either contain information to forward the packet or relate to the endpoint/application logic. However, no layer focuses on information exchange with on-path devices or path signals/property extraction.

A more radical approach would be to handle path monitoring and property extraction as a first-class citizen, for example, by adding a dedicated header/layer to each packet. A path layer allows for the extraction of new path properties and enables adaptions and modifications over time.

Our envisioned path layer explicitly exposes specific data to on-path observers, i.e., the devices on the packet path. More precisely, the endpoints take control over the path layer and can either *(i)* add dedicated signals that on-path devices can read for signal extraction, i.e., explicit sender-to-path signaling; or *(ii)* reserve dedicated, modifiable header space for on-path devices to add information, i.e., explicit path-to-receiver signaling. One use case for *(ii)* would be that on-path devices add timestamps, allowing other on-path devices or endpoints to estimate per-hop delays.

Consequently, we can completely encrypt transport headers and give endpoints full control over which data they share with the network path. For

example, we could deploy our spin bit idea [8] in the path layer. However, note that for path-to-receiver signaling, we cannot guarantee that an on-path device will participate. They could even add wrong information on purpose to hide malicious activities.

Adding an entirely new layer to each Internet packet, though, is a monumental task requiring fundamental changes to all on-path devices, e.g., routers and switches. Every device needs to be able to handle the new layer. Consequently, it is unlikely that we will soon see an Internet-wide deployment of our path layer. However, the benefits of path-to-endpoint signaling are already observed in smaller, deployed ideas. For example, the Explicit Congestion Notification (ECN) bits [182] allow routers (i.e., on-path devices) to set a mark that notifies endpoints of growing path congestion, allowing them to reduce their sending rate. The current ECN implementation re-uses existing bits in the IP or TCP header. Our path layer idea would provide all mechanisms to support ECN immediately. Finally, other research, e.g., the sidecar protocol [183], proposes similar ideas to our path layer vision.

Hardware improvements Another idea focuses on hardware improvements for network devices. The recent trend towards programmable data-plane devices, such as the Tofino switch [117], enables network operators to program the control-plane *and* the data-plane behavior. With more control and packet operations at line rate, operators can extract additional and more precise signals, resulting in reduced noise sources and, thus, improved path property inference.

For example, *Magnifier* could run in a network whose border routers consist of purely programmable devices. In such a deployment scenario, the identification of ingress and egress points could be simplified, e.g., with the help of identifiers in custom packet headers. Nonetheless, the summarization functionality of sentinels (that means we need fewer sentinels than observed unique IPs) continues to be useful as programmable devices contain limited resources.

6.1.2 *Missing traffic leads to unknown path properties*

Another fundamental problem with current path property inference is the inherent requirement of ongoing network traffic to infer interesting path properties. Without matching traffic, network operators are blind and cannot

infer up-to-date properties. For example, *Oscilloscope* cannot detect a hijack if our network does not send any traffic towards a hijack victim. On the one hand, we can argue that this is not a problem: currently, we do not forward matching traffic, so the hijack does not influence customers. On the other hand, the hijack continues to exist and might influence upcoming traffic once we forward matching packets. The following paragraphs highlight two future directions to solve the problem partially:

Representative traffic generation An apparent solution is to artificially generate the required traffic on demand. However, simple blasting strategies (e.g., using tools like iperf [184]) do not represent realistic traffic distributions and might lead to skewed path properties. More involved approaches, e.g., TRex [185], correctly replicate specific traffic features but heavily rely on sound input configuration files. For many traffic features, it is unclear or unknown what a realistic configuration would be. Finally, replaying traces, e.g., CAIDA traces [65], produces traffic that does not react to network events such as packet losses.

Recently, we presented our vision of *Dynamo* [3] (from DYNAmic Mass Orchestration), which follows a different strategy. *Dynamo* combines the "Big Data" available in open-source projects (e.g., on platforms such as GitHub [186]) with automation frameworks such as Docker containers [127] to generate representative, live network traffic.

More precisely, *Dynamo* first builds a database of traffic-generating open-source projects. For that, we iteratively try to run millions of Docker orchestration files and look for network traffic on matching interfaces. In a preliminary evaluation, we could already collect more than 74k traces. We then classify the corresponding Docker container(s) based on the traffic types in the collected traces. Second, network operators specify the traffic they want to generate using a high-level language. Afterwards, *Dynamo* parses the database of traffic-generating projects and selects the best-matching ones. Finally, *Dynamo* re-runs the corresponding docker containers to produce representative, live traffic according to the specifications.

One advantage of this approach is that real applications generate traffic. Therefore, the packets will also correctly react to network events. For example, a lost packet in a TCP flow will be retransmitted. As a result, we can use the generated traffic to extract beneficial path properties should we lack user traffic for specific applications. However, note that this does not solve path property inference problems that require reacting endpoints

in *other* networks. The following paragraphs outline possible solutions for such inference problems.

Internet-wide, automated cooperation In Chapter 5, we showed that the inference of simple path properties (e.g., reachability) still poses many challenges. For example, crafted probes are insufficient as we need to know *which* IP addresses might answer in a remote network.

In the future, we could imagine that every AS provides one network device, e.g., a programmable switch, with a publicly-known IP address. Network operators can use these devices as "beacons" and directly probe them to extract various path properties on top of the generated traffic. However, note that such ideas raise security concerns and might enable malicious ASes to lie by messing with the answer of a beacon.

6.1.3 *Current routing protocols prevent tight actions on inferred path properties*

The system designs in our exploratory study (Chapter 5) highlight that today's inter-domain routing protocol (BGP) does not easily include inferred path properties in its decision process. As a result, network operators – and end users – struggle with tight and frequent path adaptions based on inferred properties. Due to BGP's fundamental integration in nearly all inter-domain routing decisions and network devices, fast future improvements seem unlikely. A new Internet architecture could solve this problem.

A SCION-based Internet SCION [187] is designed to be a completely new, clean-slate Internet architecture that provides high flexibility, control, and isolation while improving issues observed in today's Internet. SCION's design gives endpoints and ASes/ISPs high control over which path their traffic takes. For example, a host can specify certain ASes over which their traffic should *never* flow, and SCION will guarantee the behavior.

An existing prototype implementation [188] shows how end users can specify their preferred traffic paths (according to various criteria or properties) directly in an Internet browser. This flexible design simplifies actions on top of inferred path properties.

OWN PUBLICATIONS

[1] Tobias Bühler, Ingmar Poese, and Laurent Vanbever. "Sentinels: Guarding ISP Networks from Forwarding Anomalies". In: *Proceedings of the ACM International Conference on Emerging Networking EXperiments and Technologies (CoNEXT, Student Workshop)*. 2016.

[2] Tobias Bühler, Mirja Kühlewind, and Brian Trammell. "Enhancing encrypted transport protocols with passive measurement capabilities". In: *Proceedings of the ACM Internet Measurement Conference (IMC, poster)*. 2017.

[3] Tobias Bühler, Roland Schmid, Sandro Lutz, and Laurent Vanbever. "Generating representative, live network traffic out of millions of code repositories". In: *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*. 2022.

[4] Tobias Bühler, Alexandros Milolidakis, Romain Jacob, Marco Chiesa, Stefano Vissicchio, and Laurent Vanbever. *Oscilloscope: Detecting BGP Hijacks in the Data Plane*. `https://arxiv.org/abs/2301.12843`. arXiv preprint. 2023.

[5] Tobias Bühler, Romain Jacob, Ingmar Poese, and Laurent Vanbever. "Enhancing Global Network Monitoring with Magnifier". In: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2023.

[6] Mirja Kühlewind, Tobias Bühler, Brian Trammell, Stephan Neuhaus, Roman Müntener, and Gorry Fairhurst. "A Path Layer for the Internet: Enabling Network Operations on Encrypted Protocols". In: *Proceedings of the IEEE International Conference on Network and Service Management (CNSM)*. 2017.

[7] Olivier Tilmans, Tobias Bühler, Ingmar Poese, Stefano Vissicchio, and Laurent Vanbever. "Stroboscope: Declarative Network Monitoring on a Budget". In: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2018.

[8] Piet De Vaere, Tobias Bühler, Mirja Kühlewind, and Brian Trammell. "Three Bits Suffice: Explicit Support for Passive Measurement of Internet Latency in QUIC and TCP". In: *Proceedings of the ACM Internet Measurement Conference (IMC)*. 2018.

[9] Mirja Kühlewind, Brian Trammell, Tobias Bühler, Gorry Fairhurst, and Vijay Gurbani. *Challenges in Network Management of Encrypted Traffic*. `https://arxiv.org/abs/1810.09272`. arXiv preprint. 2018.

[10] Thomas Holterbach, Tobias Bühler, Tino Rellstab, and Laurent Vanbever. "An Open Platform to Teach How the Internet Practically Works". In: *ACM SIGCOMM Computer Communication Review (CCR)* (2020).

[11] Alexandros Milolidakis, Tobias Bühler, Marco Chiesa, Laurent Vanbever, and Stefano Vissicchio. "Smart BGP hijacks that Evade Public Route Collectors". In: *Proceedings of the ACM Internet Measurement Conference (IMC, poster)*. 2021.

[12] Alexandros Milolidakis, Tobias Bühler, Kunyu Wang, Marco Chiesa, Laurent Vanbever, and Stefano Vissicchio. "On the Effectiveness of BGP Hijackers That Evade Public Route Collectors". In: *IEEE Access* 11 (2023).

[13] Olivier Tilmans, Tobias Bühler, Stefano Vissicchio, and Laurent Vanbever. "Mille-Feuille: Putting ISP Traffic under the Scalpel". In: *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*. 2016.

[14] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. *pForest: In-Network Inference with Random Forests*. `https://arxiv.org/abs/1909.05680`. arXiv preprint. 2019.

REFERENCES

[15] Eliot Miller. *ISPs and Publishers get the blame for video streaming problems*. https://www.mux.com/blog/isps-and-publishers-get-the-blame-for-video-streaming-problems. (Accessed: 2023-07-28).

[16] Robert Grimmick. *Slow Internet? (Maybe) Don't Blame Your ISP*. https://grimmicktechnology.com/slow-internet-maybe-dont-blame-your-isp/. (Accessed: 2023-07-28).

[17] Reuters. *Swisscom boss apologises for massive network outage*. https://www.reuters.com/business/media-telecom/swisscom-boss-apologises-massive-network-outage-newspaper-2021-07-14/. (Accessed: 2023-07-28).

[18] Dan Goodin. *Russian-controlled telecom hijacks financial services' Internet traffic*. https://arstechnica.com/information-technology/2017/04/russian-controlled-telecom-hijacks-financial-services-internet-traffic/. (Accessed: 2023-07-28).

[19] Claudia Glover. *Cybercrime is rampant. ISPs could do more to stop it.* https://techmonitor.ai/future-of-telecoms/cybercrime-is-rampant-isps-could-do-more-to-stop-it. (Accessed: 2023-07-28).

[20] Yakov Rekhter, Susan Hares, and Tony Li. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271. https://www.rfc-editor.org/info/rfc4271. 2006.

[21] Cisco Systems. *Cisco Annual Internet Report (2018-2023) White Paper*. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. (Accessed: 2023-07-28).

[22] T.M. Chen and L. Hu. "Internet Performance Monitoring". In: *Proceedings of the IEEE* (2002).

[23] Jon Postel. *Internet Protocol*. RFC 791. https://www.rfc-editor.org/info/rfc791. 1981.

[24] Bob Hinden and Dr. Steve E. Deering. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460. https://www.rfc-editor.org/info/rfc2460. 1998.

[25] ICANN. *Remaining IPv4 Addresses to be Redistributed to Regional Internet Registries*. https://www.icann.org/en/announcements/details/remaining-ipv4-addresses-to-be-redistributed-to-regional-internet-registries--address-redistribution-signals-that-ipv4-is-nearing-total-exhaustion-20-5-2014-en. (Accessed: 2023-07-28).

[26] RIPE NCC. *Understanding IP Addressing and CIDR Charts*. https://www.ripe.net/about-us/press-centre/understanding-ip-addressing. (Accessed: 2023-07-28).

[27] Hormuzd M. Khosravi and Todd A. Anderson. *Requirements for Separation of IP Control and Forwarding*. RFC 3654. https://www.rfc-editor.org/info/rfc3654. 2003.

[28] John A. Hawkinson and Tony J. Bates. *Guidelines for creation, selection, and registration of an Autonomous System (AS)*. RFC 1930. https://www.rfc-editor.org/info/rfc1930. 1996.

[29] SWITCH. *SWITCH*. https://www.switch.ch/. (Accessed: 2023-07-28).

[30] CAIDA. *CADIA AS Rank*. https://asrank.caida.org/. (Accessed: 2023-07-28).

[31] Geoff Huston. *BGP Routing Table Analysis Reports*. https://bgp.potaroo.net/. (Accessed: 2023-07-28).

[32] Swisscom Ltd. *Swisscom*. https://www.swisscom.ch/en/about.html. (Accessed: 2023-07-28).

[33] ThousandEyes. *ISP 3-Tier Model*. https://www.thousandeyes.com/learning/techtorials/isp-tiers. (Accessed: 2023-07-28).

[34] Lixin Gao and Jennifer Rexford. "Stable Internet routing without global coordination". In: *IEEE/ACM Transactions on Networking (TON)* (2001).

[35] John Moy. *OSPF Version 2*. RFC 2328. https://www.rfc-editor.org/info/rfc2328. 1998.

[36] Cisco Systems. *BGP Best Path Algorithm*. https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html. (Accessed: 2023-07-28).

[37] V. Paxson. "End-to-end routing behavior in the Internet". In: *IEEE/ACM Transactions on Networking (TON)* (1997).

[38] Andree Toonk. *BGP leak causing Internet outages in Japan and beyond.* https://bgpmon.net/bgp-leak-causing-internet-outages-in-japan-and-beyond/. (Accessed: 2023-07-28).

[39] ThousandEyes. *BGP Route Hijacking*. https://www.thousandeyes.com/learning/glossary/bgp-route-hijacking. (Accessed: 2023-07-28).

[40] Kevin Butler, Toni R. Farley, Patrick McDaniel, and Jennifer Rexford. "A survey of BGP security issues and solutions". In: *Proceedings of the IEEE* (2009).

[41] Jon Postel. *Transmission Control Protocol*. RFC 793. https://www.rfc-editor.org/info/rfc793. 1981.

[42] Wesley Eddy. *Transmission Control Protocol (TCP)*. RFC 9293. https://www.rfc-editor.org/info/rfc9293. 2022.

[43] FreeBSD Project. *PING(8) FreeBSD System Manager's Manual*. https://www.freebsd.org/cgi/man.cgi?ping(8). (Accessed: 2023-07-28).

[44] Wayne Tolliver. *Disabling ICMP and SNMP won't increase security, but will impact network monitoring*. https://blog.paessler.com/disabling-icmp-and-snmp-wont-increase-security-but-will-impact-network-monitoring. (Accessed: 2023-07-28).

[45] Jon Postel. *Internet Control Message Protocol*. RFC 792. https://www.rfc-editor.org/info/rfc792. 1981.

[46] Matt Sargent, Jerry Chu, Dr. Vern Paxson, and Mark Allman. *Computing TCP's Retransmission Timer*. RFC 6298. https://www.rfc-editor.org/info/rfc6298. 2011.

[47] Satadal Sengupta, Hyojoon Kim, and Jennifer Rexford. "Continuous In-Network Round-Trip Time Monitoring". In: *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 2022.

[48] David Borman, Robert T. Braden, Van Jacobson, and Richard Scheffenegger. *TCP Extensions for High Performance*. RFC 7323. https://www.rfc-editor.org/info/rfc7323. 2014.

[49] Stephen D. Strowes. "Passively Measuring TCP Round-Trip Times". In: *Communications of the ACM* (2013).

[50] Deutsche Telekom AG. *Internet & Content IP Transit*. https://globalcarrier.telekom.com/business-areas/internet-content/ip-transit. (Accessed: 2023-07-28).

[51] Benoît Claise. *Cisco Systems NetFlow Services Export Version 9*. RFC 3954. https://www.rfc-editor.org/info/rfc3954. 2004.

[52] Sonia Panchen, Neil McKee, and Peter Phaal. *InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks*. RFC 3176. https://www.rfc-editor.org/info/rfc3176. 2001.

[53] 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". In: *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 2014.

[54] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. "One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon". In: *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 2016.

[55] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. "Language-Directed Hardware Design for Network Performance Monitoring". In: *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 2017.

[56] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. "DREAM: Dynamic Resource Allocation for Software-Defined Measurement". In: *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 2014.

[57] Tobias Bühler. *Magnifier GitHub repository*. https://github.com/nsg-ethz/Magnifier. (Accessed: 2023-07-28).

[58] RIPE NCC. *IPv6 Address Allocation and Assignment Policy*. https://www.ripe.net/publications/docs/ripe-738#5. (Accessed: 2023-07-28).

[59] Philip Smith, Rob Evans, and Mike Hughes. *RIPE Routing Working Group Recommendations on Route Aggregation*. https://www.ripe.net/publications/docs/ripe-399. (Accessed: 2023-07-28).

[60] Cisco Systems. *Configuring ERSPAN*. 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: 2023-07-28).

[61] 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: 2023-07-28).

[62] Cisco Systems. *Cisco Nexus 9300-FX*. https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/datasheet-c78-742284.html. (Accessed: 2023-07-28).

[63] Cisco Systems. *Cisco Nexus 7000*. https://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-series-switches/Data_Sheet_C78-437762.html. (Accessed: 2023-07-28).

[64] Cisco Systems. *Nexus 9000 TCAM Carving*. https://www.cisco.com/c/en/us/support/docs/switches/nexus-9000-series-switches/119032-nexus9k-tcam-00.html. (Accessed: 2023-07-28).

[65] CAIDA. *The CAIDA UCSD Anonymized Internet Traces 2018*. https://www.caida.org/catalog/datasets/passive_dataset. (Accessed: 2023-07-28).

[66] Fred Klassen. *Tcpreplay - Pcap editing and replaying utilities*. https://tcpreplay.appneta.com/. (Accessed: 2023-07-28).

[67] Cisco Systems. *Configuring Rate Limits*. 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: 2023-07-28).

[68] 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". In: *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 2015.

[69] Juniper Networks. *Configure J-Flow*. https://supportportal.juniper.net/s/article/SRX-Getting-Started-Configure-J-Flow?language=en_US. (Accessed: 2023-07-28).

[70] Renata Teixeira, Aman Shaikh, Timothy G Griffin, and Jennifer Rexford. "Impact of Hot-Potato Routing Changes in IP Networks". In: *IEEE/ACM Transactions on Networking (TON)* (2008).

[71] Ítalo Cunha, Fernando Silveira, Ricardo Oliveira, Renata Teixeira, and Christophe Diot. "Uncovering Artifacts of Flow Measurement Tools". In: *Proceedings of the International Conference on Passive and Active Network Measurement (PAM)*. 2009.

[72] Cristian Estan, Ken Keys, David Moore, and George Varghese. "Building a Better NetFlow". In: *ACM SIGCOMM Computer Communication Review (CCR)* (2004).

[73] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. "FlowRadar: A Better NetFlow for Data Centers". In: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2016.

[74] Said Jawad Saidi, Aniss Maghsoudlou, Damien Foucard, Georgios Smaragdakis, Ingmar Poese, and Anja Feldmann. "Exploring Network-Wide Flow Data With Flowyager". In: *IEEE Transactions on Network and Service Management (TNSM)* (2020).

[75] Andreas Wundsam, Dan Levin, Srini Seetharaman, and Anja Feldmann. "OFRewind: Enabling Record and Replay Troubleshooting for Networks". In: *Proceedings of the USENIX Annual Technical Conference (ATC)*. 2011.

[76] Ran Ben Basat, Xiaoqi Chen, Gil Einziger, and Ori Rottenstreich. "Designing Heavy-Hitter Detection Algorithms for Programmable Switches". In: *IEEE/ACM Transactions on Networking (TON)* (2020).

[77] Ross Teixeira, Rob Harrison, Arpit Gupta, and Jennifer Rexford. "PacketScope: Monitoring the Packet Lifecycle Inside a Switch". In: *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*. 2020.

[78] 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: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2019.

[79] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. "Sonata: Query-Driven Streaming Network Telemetry". In: *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 2018.

[80] 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: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2019.

[81] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. "PINT: Probabilistic In-band Network Telemetry". In: *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 2020.

[82] 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". In: *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 2017.

[83] Minlan Yu, Lavanya Jose, and Rui Miao. "Software Defined Traffic Measurement with OpenSketch". In: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2013.

[84] Xiaoqi Chen, Shir Landau-Feibish, Mark Braverman, and Jennifer Rexford. "BeauCoup: Answering Many Network Traffic Queries, One Memory Update at a Time". In: *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 2020.

[85] 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: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2021.

[86] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, and Fred True. "Deriving Traffic Demands for Operational IP Networks: Methodology and Experience". In: *ACM SIGCOMM Computer Communication Review (CCR)* (2000).

[87] Ratul Mahajan, Steven M Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. "Controlling High Bandwidth Aggregates in the Network". In: *ACM SIGCOMM Computer Communication Review (CCR)* (2002).

[88] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. "Proactively Detecting Distributed Denial of Service Attacks Using Source IP Address Monitoring". In: *Proceedings of the International Conference on Research in Networking*. Springer, 2004.

[89] 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).

[90] Konstantina Papagiannaki, Nina Taft, and Anukool Lakhina. "A Distributed Approach to Measure IP Traffic Matrices". In: *Proceedings of the ACM Internet Measurement Conference (IMC)*. 2004.

[91] Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. "OpenTM: Traffic Matrix Estimator for OpenFlow Networks". In: *Proceedings of the International Conference on Passive and Active Network Measurement (PAM)*. 2010.

[92] Mehdi Malboubi, Shu-Ming Peng, Puneet Sharma, and Chen-Nee Chuah. "A Learning-Based Measurement Framework for Traffic Matrix Inference in Software Defined Networks". In: *Computers and Electrical Engineering* (2018).

[93] 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". In: *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 2015.

[94] Kentik. *Network Observability, Performance and Security*. `https://www.kentik.com/`. (Accessed: 2023-07-28).

[95] The Zeek Project. *The Zeek Network Security Monitor*. `https://zeek.org/`. (Accessed: 2023-07-28).

[96] Aftab Siddiqui. *What Happened? The Amazon Route 53 BGP Hijack to Take Over Ethereum Cryptocurrency Wallets*. `https://www.internetsociety.org/blog/2018/04/amazons-route-53-bgp-hijack/`. (Accessed: 2023-07-28).

[97] Andree Toonk. *How Hacking Team Helped Italian Special Operations Group with Routing Hijack*. https://www.bgpmon.net/how-hacking-team-helped-italian-special-operations-group-with-bgp-routing-hijack/. (Accessed: 2023-07-28).

[98] Dan Goodin. *How 3ve's BGP hijackers eluded the Internet–and made $29M*. https://arstechnica.com/information-technology/2018/12/how-3ves-bgp-hijackers-eluded-the-internet-and-made-29m/. (Accessed: 2023-07-28).

[99] Matt Lepinski and Kotikalapudi Sriram. *BGPsec Protocol Specification*. RFC 8205. https://www.rfc-editor.org/info/rfc8205. 2017.

[100] Randy Bush and Rob Austein. *The Resource Public Key Infrastructure (RPKI) to Router Protocol, Version 1*. RFC 8210. https://www.rfc-editor.org/info/rfc8210. 2017.

[101] Zheng Zhang, Ying Zhang, Y Charlie Hu, Z Morley Mao, and Randy Bush. "iSPY: Detecting IP Prefix Hijacking on My Own". In: *ACM SIGCOMM Computer Communication Review (CCR)* (2008).

[102] LLC DigitalOcean. *DigitalOcean*. https://www.digitalocean.com/. (Accessed: 2023-07-28).

[103] Amazon Web Services, Inc. *Amazon Web Services*. https://aws.amazon.com/. (Accessed: 2023-07-28).

[104] Tony Li, Dino Farinacci, Stanley P. Hanks, David Meyer, and Paul S. Traina. *Generic Routing Encapsulation (GRE)*. RFC 2784. https://www.rfc-editor.org/info/rfc2784. 2000.

[105] A Linux Foundation Collaborative Project. *FRRouting Project*. https://frrouting.org/. (Accessed: 2023-07-28).

[106] A. Zimmermann, A. Hannemann, and T. Kosse. "Flowgrind - A New Performance Measurement Tool". In: *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*. 2010.

[107] Alexander Marder, K. C. Claffy, and Alex C. Snoeren. "Inferring Cloud Interconnections: Validation, Geolocation, and Routing Behavior". In: *Proceedings of the International Conference on Passive and Active Network Measurement (PAM)*. 2021.

[108] Nikolaos Chatzis, Georgios Smaragdakis, Anja Feldmann, and Walter Willinger. "There is More to IXPs than Meets the Eye". In: *ACM SIGCOMM Computer Communication Review (CCR)* (2013).

[109] David D. Clark, Steven Bauer, William Lehr, KC Claffy, Amogh D. Dhamdhere, Bradley Huffaker, and Matthew Luckie. "Measurement and Analysis of Internet Interconnection and Congestion". In: *Research Conference on Communication, Information and Internet Policy (TPRC)*. 2014.

[110] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. "A Measurement Study on the Impact of Routing Events on End-to-End Internet Path Performance". In: *ACM SIGCOMM Computer Communication Review (CCR)* (2006).

[111] Thomas Holterbach, Edgar Costa Molero, Maria Apostolaki, Alberto Dainotti, Stefano Vissicchio, and Laurent Vanbever. "Blink: Fast Connectivity Recovery Entirely in the Data Plane". In: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2019.

[112] Veit Hailperin. *Misusing TCP Timestamps*. `https://www.scip.ch/en/?labs.20150305`. (Accessed: 2023-07-28).

[113] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. `https://www.rfc-editor.org/info/rfc9000`. 2021.

[114] MAMI contributors. *Measurement and Architecture for a Middleboxed Internet (MAMI)*. `https://doi.org/10.3030/688421`. (Accessed: 2023-07-28). 2016.

[115] Ike Kunze, Klaus Wehrle, and Jan Rüth. "L, Q, R, and T: Which Spin Bit Cousin is Here to Stay?" In: *Proceedings of the Applied Networking Research Workshop (ANRW)*. 2021.

[116] Ike Kunze, Constantin Sander, Klaus Wehrle, and Jan Rüth. "Tracking the QUIC Spin Bit on Tofino". In: *Proceedings of the Workshop on Evolution, Performance and Interoperability of QUIC (EPIQ)*. 2021.

[117] Intel Corporation. *Intel Tofino*. `https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html`. (Accessed: 2023-07-28).

[118] Töma Gavrichenkov and Artyom Gavrichenkov. "Breaking HTTPS with BGP Hijacking". In: *Proceedings of the Black Hat Conference*. 2015.

[119] Wolfgang Mühlbauer, Steve Uhlig, Bingjie Fu, Mickael Meulle, and Olaf Maennel. "In Search for an Appropriate Granularity to Model Routing Policies". In: *ACM SIGCOMM Computer Communication Review (CCR)* (2007).

[120] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, Victor Lin, Colin Rice, Brian Rogan, Arjun Singh, Bert Tanaka, Manish Verma, Puneet Sood, Mukarram Tariq, Matt Tierney, Dzevad Trumic, Vytautas Valancius, Calvin Ying, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. "Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering". In: *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 2017.

[121] Xiaoqi Chen, Hyojoon Kim, Javed M. Aman, Willie Chang, Mack Lee, and Jennifer Rexford. "Measuring TCP Round-Trip Time in the Data Plane". In: *Proceedings of the ACM SIGCOMM Workshop on Secure Programmable Network Infrastructure (SPIN)*. 2020.

[122] J. Li, T. Ehrenkranz, and P. Elliott. "Buddyguard: A buddy system for fast and reliable detection of IP prefix anomalies". In: *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*. 2012.

[123] Phillipa Gill, Michael Schapira, and Sharon Goldberg. "A Survey of Interdomain Routing Policies". In: *ACM SIGCOMM Computer Communication Review (CCR)* (2014).

[124] Waleed Reda, Kirill Bogdanov, Alexandros Milolidakis, Hamid Ghasemirahni, Marco Chiesa, Gerald Q. Maguire, and Dejan Kostić. "Path Persistence in the Cloud: A Study of the Effects of Inter-Region Traffic Engineering in a Large Cloud Provider's Network". In: *ACM SIGCOMM Computer Communication Review (CCR)* (2020).

[125] Samaneh Aminikhanghahi and Diane J. Cook. "A survey of methods for time series change point detection". In: *Knowledge and Information Systems* (2017).

[126] Sidney Siegel and N. John Castellan. *Nonparametric Statistics for The Behavioral Sciences*. Second edition. McGraw-Hill, 1988.

[127] Docker Inc. *Docker*. `https://www.docker.com/`. (Accessed: 2023-07-28).

[128] Stephen Hemminger. *NetEm - Network Emulator*. `https://man7.org/linux/man-pages/man8/tc-netem.8.html`. (Accessed: 2023-07-28).

[129] CAIDA. *The CAIDA AS Relationships Dataset, <2019-08-01>*. `http://www.caida.org/data/active/as-relationships/`. (Accessed: 2023-07-28).

[130] Ripe NCC. *RIPE Atlas platform*. https://atlas.ripe.net/. (Accessed: 2023-07-28).

[131] Ethan Katz-Bassett, Colin Scott, David R. Choffnes, Ítalo Cunha, Vytautas Valancius, Nick Feamster, Harsha V. Madhyastha, Thomas Anderson, and Arvind Krishnamurthy. "LIFEGUARD: Practical Repair of Persistent Route Failures". In: *ACM SIGCOMM Computer Communication Review (CCR)* (2012).

[132] Dave Katz and David Ward. *Bidirectional Forwarding Detection (BFD)*. RFC 5880. https://www.rfc-editor.org/info/rfc5880. 2010.

[133] The Tcpdump Group. *TCPDUMP*. https://www.tcpdump.org/. (Accessed: 2023-07-28).

[134] Cecilia Testart. "Reviewing a Historical Internet Vulnerability: Why Isn't BGP More Secure and What Can We Do About it?" In: *Proceedings of the Research Conference on Communication, Information and Internet Policy (TPRC)*. 2018.

[135] Geoff Huston. *BGP in 2019 - The BGP Table*. https://blog.apnic.net/2020/01/14/bgp-in-2019-the-bgp-table/. (Accessed: 2023-07-28).

[136] Stephen Kent, Charles Lynn, and Karen Seo. "Secure border gateway protocol S-BGP". In: *IEEE Journal on Selected areas in Communications (J-SAC)* (2000).

[137] Paul C Van Oorschot, Tao Wan, and Evangelos Kranakis. "On inter-domain routing security and pretty secure BGP (psBGP)". In: *ACM Transactions on Information and System Security (TISSEC)* (2007).

[138] Russ White. "Securing BGP through secure origin BGP (soBGP)". In: *Business Communications Review* (2003).

[139] Meiyuan Zhao, Sean W Smith, and David M Nicol. "The performance impact of BGP security". In: *IEEE network* (2005).

[140] Sharon Goldberg, Michael Schapira, Peter Hummon, and Jennifer Rexford. "How Secure Are Secure Interdomain Routing Protocols". In: *ACM SIGCOMM Computer Communication Review (CCR)* (2010).

[141] Pavlos Sermpezis, Vasileios Kotronis, Petros Gigis, Xenofontas Dimitropoulos, Danilo Cicalese, Alistair King, and Alberto Dainotti. "ARTEMIS: Neutralizing BGP hijacking within a minute". In: *IEEE/ACM Transactions on Networking (TON)* (2018).

[142] Xingang Shi, Yang Xiang, Zhiliang Wang, Xia Yin, and Jianping Wu. "Detecting Prefix Hijackings in the Internet with Argus". In: *Proceedings of the ACM Internet Measurement Conference (IMC)*. 2012.

[143] Chris Amin. *RIS Live BGP Message Stream*. https://labs.ripe.net/Members/chris_amin/ris-live-bgp-message-stream/. (Accessed: 2023-07-28).

[144] Chiara Orsini, Alistair King, Danilo Giordano, Vasileios Giotsas, and Alberto Dainotti. "BGPStream: a software framework for live and historical BGP data analysis". In: *Proceedings of the ACM Internet Measurement Conference (IMC)*. 2016.

[145] Changxi Zheng, Lusheng Ji, Dan Pei, Jia Wang, and Paul Francis. "A light-weight distributed scheme for detecting IP prefix hijacks in real-time". In: *ACM SIGCOMM Computer Communication Review (CCR)* (2007).

[146] Xin Hu and Z Morley Mao. "Accurate real-time identification of IP prefix hijacking". In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. 2007.

[147] Daniel Jubas. *Detecting BGP Interception Attacks using RTT Measurements*. https://www.cs.princeton.edu/~jrex/papers/rtt-change-point.pdf. Student thesis (Accessed: 2023-07-28).

[148] Rahul Hiran, Niklas Carlsson, and Nahid Shahmehri. "Crowd-based Detection of Routing Anomalies on the Internet". In: *Proceedings of the IEEE Conference on Communications and Network Security (CNS)*. 2015.

[149] Karan Balu, Miguel L. Pardal, and Miguel Correia. "DARSHANA: Detecting route hijacking for communication confidentiality". In: *Proceedings of the IEEE International Symposium on Network Computing and Applications (NCA)*. 2016.

[150] Doug Madory. *Excessive BGP AS-PATH prepending is a self-inflicted vulnerability*. https://blog.apnic.net/2019/07/15/excessive-bgp-as-path-prepending-is-a-self-inflicted-vulnerability/. (Accessed: 2023-07-28).

[151] Pedro Marcos, Lars Prehn, Lucas Leal, Alberto Dainotti, Anja Feldmann, and Marinho Barcellos. "AS-Path Prepending: There is No Rose without a Thorn". In: *Proceedings of the ACM Internet Measurement Conference (IMC)*. 2020.

[152] Alex Shapelez. *BGP Route Leak prevention and detection with the help of the RFC9234*. https://habr.com/en/company/qrator/blog/710420/. (Accessed: 2023-07-28).

[153] Curtis Villamizar, Ravi Chandra, and Dr. Ramesh Govindan. *BGP Route Flap Damping*. RFC 2439. https://www.rfc-editor.org/info/rfc2439. 1998.

[154] Jaroslav Frnda, Miroslav Voznak, and Lukas Sevcik. "Impact of packet loss and delay variation on the quality of real-time video streaming". In: *Telecommunication Systems* (2016).

[155] Arun Viswanathan, Eric C. Rosen, and Ross Callon. *Multiprotocol Label Switching Architecture*. RFC 3031. https://www.rfc-editor.org/info/rfc3031. 2001.

[156] Clarence Filsfils, Pablo Camarillo, John Leddy, Daniel Voyer, Satoru Matsushima, and Zhenbin Li. *Segment Routing over IPv6 (SRv6) Network Programming*. RFC 8986. https://www.rfc-editor.org/info/rfc8986. 2021.

[157] Maria Apostolaki, Ankit Singla, and Laurent Vanbever. "Performance-Driven Internet Path Selection". In: *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*. 2021.

[158] Marc-Olivier Buob, Mickael Meulle, and Steve Uhlig. "Checking for optimal egress points in iBGP routing". In: *Proceedings of the International Workshop on Design and Reliable Communication Networks (DRCN)*. 2007.

[159] Rajiv Papneja, Bhavani Parise, Susan Hares, Dean Lee, and Ilya Varlashkin. *Basic BGP Convergence Benchmarking Methodology for Data-Plane Convergence*. RFC 7747. https://www.rfc-editor.org/info/rfc7747. 2016.

[160] Thomas Wirtgen, Tom Rousseaux, Quentin De Coninck, Nicolas Rybowski, Randy Bush, Laurent Vanbever, Axel Legay, and Olivier Bonaventure. "xBGP: Faster Innovation in Routing Protocols". In: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2023.

[161] Thomas Holterbach, Stefano Vissicchio, Alberto Dainotti, and Laurent Vanbever. "SWIFT: Predictive Fast Reroute". In: *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 2017.

[162] Ethan Blanton, Dr. Vern Paxson, and Mark Allman. *TCP Congestion Control*. RFC 5681. https://www.rfc-editor.org/info/rfc5681. 2009.

[163] Bahaa Al-Musawi, Philip Branch, and Grenville Armitage. "BGP Anomaly Detection Techniques: A Survey". In: *IEEE Communications Surveys & Tutorials* (2017).

[164] FreeBSD Project. *traceroute – print the route packets take to network host*. `https://man.freebsd.org/cgi/man.cgi?query=traceroute`. (Accessed: 2023-07-28).

[165] Stefan Savage, Thomas Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker, and John Zahorjan. "Detour: Informed Internet Routing and Transport". In: *IEEE Micro* (1999).

[166] Randy Bush, James Hiebert, Olaf Maennel, Matthew Roughan, and Steve Uhlig. "Testing the reachability of (new) address space". In: *Proceedings of the SIGCOMM workshop on Internet network management*. 2007.

[167] Marco Chiesa, Andrzej Kamisiński, Jacek Rak, Gabor Retvari, and Stefan Schmid. "A Survey of Fast Recovery Mechanisms in the Data Plane". In: (2020). TechRxiv Preprint.

[168] Rahil Gandotra and Levi Perigo. "SDNMA: A Software-Defined, Dynamic Network Manipulation Application to Enhance BGP Functionality". In: *Proceedings of the IEEE International Conference on High Performance Computing and Communications (HPCC)*. 2018.

[169] Li Xiao, King-Shan Lui, Jun Wang, and K. Nahrstedt. "QoS extension to BGP". In: *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*. 2002.

[170] A. Beben. "EQ-BGP: an efficient inter-domain QoS routing protocol". In: *Proceedings of the International Conference on Advanced Information Networking and Applications (AINA)*. 2006.

[171] Kuo-Feng Hsu, Ryan Beckett, Ang Chen, Jennifer Rexford, Praveen Tammana, and David Walker. "Contra: A Programmable System for Performance-Aware Routing". In: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2020.

[172] A. Elwalid, C. Jin, S. Low, and I. Widjaja. "MATE: MPLS Adaptive Traffic Engineering". In: *Proceedings of the IEEE INFOCOM conference on Computer Communications*. 2001.

[173] Henry Birge-Lee, Maria Apostolaki, and Jennifer Rexford. "It Takes Two to Tango: Cooperative Edge-to-Edge Routing". In: *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*. 2022.

[174] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V. Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. "Engineering Egress with Edge Fabric: Steering Oceans of Content to the World". In: *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 2017.

[175] Todd Arnold, Matt Calder, Italo Cunha, Arpit Gupta, Harsha V. Madhyastha, Michael Schapira, and Ethan Katz-Bassett. "Beating BGP is Harder than We Thought". In: *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*. 2019.

[176] Shahrooz Pouryousef, Lixin Gao, and Arun Venkataramani. "Towards Logically Centralized Interdomain Routing". In: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2020.

[177] Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. "Design and Implementation of a Routing Control Platform". In: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2005.

[178] John P. John, Ethan Katz-Bassett, Arvind Krishnamurthy, Thomas Anderson, and Arun Venkataramani. "Consensus Routing: The Internet as a Distributed System". In: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2008.

[179] Philip Smit and Barry Greene. *Using BGP Communities*. `https://nsrc.org/workshops/2021/riso-pern-apan51/networking/peering-ixp/en/presentations/BGP-Communities.pdf`. (Accessed: 2023-07-28).

[180] Vasileios Giotsas, Georgios Smaragdakis, Christoph Dietzel, Philipp Richter, Anja Feldmann, and Arthur Berger. "Inferring BGP Blackholing Activity in the Internet". In: *Proceedings of the ACM Internet Measurement Conference (IMC)*. 2017.

[181] Robert T. Braden. *Requirements for Internet Hosts - Communication Layers*. RFC 1122. `https://www.rfc-editor.org/info/rfc1122`. 1989.

[182] Sally Floyd, Dr. K. K. Ramakrishnan, and David L. Black. *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC 3168. `https://www.rfc-editor.org/info/rfc3168`. 2001.

[183] Gina Yuan, David K. Zhang, Matthew Sotoudeh, Michael Welzl, and Keith Winstein. "Sidecar: In-Network Performance Enhancements in the Age of Paranoid Transport Protocols". In: *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*. 2022.

[184] ESnet. *iperf3: A TCP, UDP, and SCTP network bandwidth measurement tool*. `https://github.com/esnet/iperf`. (Accessed: 2023-07-28).

[185] Cisco Systems. *TRex - Realistic Traffic Generator*. `https://trex-tgn.cisco.com/`. (Accessed: 2023-07-28).

[186] GitHub inc. *GitHub - Let's build from here*. `https://github.com/`. (Accessed: 2023-07-28).

[187] Laurent Chuat, Markus Legner, David Basin, David Hausheer, Samuel Hitz, Peter Müller, and Adrian Perrig. *The Complete Guide to SCION. From Design Principles to Formal Verification*. Springer, 2022.

[188] A. Davidson, M. Frei, M. Gartner, H. Haddadi, A. Perrig, J. Subirà Nieto, P. Winter, and F. Wirz. "Tango or Square Dance? How Tightly Should We Integrate Network Functionality in Browsers?" In: *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*. 2022.