Data-Plane Telemetry to Mitigate Long-Distance BGP Hijacks

Satadal Sengupta
Princeton University
satadals@princeton.edu

Hyojoon Kim
University of Virginia
tcr5zr@virginia.edu

Daniel Jubas*
Five Rings LLC
danieljubas@gmail.com

Maria Apostolaki
Princeton University
apostolaki@princeton.edu

Jennifer Rexford
Princeton University
jrex@princeton.edu

Abstract

Poor security of Internet routing enables adversaries to divert user data through unintended infrastructures (hijack). Of particular concern—and the focus of this paper—are cases where attackers reroute domestic traffic through foreign countries, exposing it to surveillance, bypassing legal privacy protections, and posing national security threats. Efforts to detect and mitigate such attacks have focused primarily on the control plane while data-plane signals remain largely overlooked. In particular, change in propagation delay caused by rerouting offers a promising signal: the change is unavoidable and the increased propagation delay is directly observable from the affected networks. In this paper, we explore the practicality of using delay variations for hijack detection, addressing two key questions: (1) What coverage can this provide, given its heavy dependence on the geolocations of the sender, receiver, and adversary? and (2) Can an always-on latency-based detection system be deployed without disrupting normal network operations? We observe that for 86% of victim–attacker country pairs in the world, mid-attack delays exceed pre-attack delays by at least 25% in real deployments, making delay-based hijack detection promising. To demonstrate practicality, we design *HiDe*, which reliably detects delay surges from long-distance hijacks at line rate. We measure *HiDe*'s accuracy and false-positive rate on real-world data and validate it with ethically conducted hijacks.

1 Introduction

BGP hijacks are a well-known threat, where attackers exploit the poor security of BGP—the Internet's default routing protocol—to redirect traffic through their own infrastructure. These attacks are dangerous, allowing attackers to eavesdrop and steal sensitive information from unsuspecting users [1, 3, 5, 7, 9, 16, 30, 33]. Despite years of research aimed at addressing this issue, BGP hijacking continues to pose a significant threat [34]. Solutions like BGPsec [8] require ubiquitous adoption to be truly effective, which is a significant challenge given the decentralized nature of the Internet. On the other hand, mechanisms like RPKI [15], while beneficial, are not bulletproof against all types of attacks.

Of particular concern, and the focus of this paper, are hijacks where domestic traffic is rerouted through a foreign location before reaching its intended destination, i.e., *long-distance interception attacks*. Such attacks are especially troubling because they—unbeknownst to the user—expose the user's traffic to different jurisdictions and, consequently, to different privacy and surveillance laws. These rerouting incidents have significant implications [19, 20, 40]. Yet, existing monitoring solutions [4, 25, 37, 38] rely almost exclusively on control-plane signals. Concretely, they aim at finding anomalies in BGP route updates and are thus limited to what is visible from their vantage points or monitors [10, 12, 28, 29, 40]. Such an approach is also constrained by the slow convergence of BGP that can take minutes [22].

In this paper, we investigate the usefulness of propagation delay, a data-plane signal, in detecting BGP hijacks and in particular *long-distance* hijacks. Propagation delay is a promising yet underexplored signal for detecting such long-distance attacks, especially compared to well-studied control-plane signals. Unlike control-plane signals, propagation delay cannot be hidden from the victim, and in the case of *long-distance* hijacks, the increase in delay is significant. For instance, for a UK-based source-destination pair, the traffic must cover an additional 15,025 kilometers at least to travel via North Korea, causing a minimum additional round-trip time (RTT) of 75 ms[1]—a latency the victim will directly experience. A change in propagation delay is also immediately observable, enabling a faster detection and mitigation opportunity compared to existing control-plane approaches.

However, building an effective delay-based BGP hijack detector is challenging. First, the expected increase in propagation delay—which is at the heart of a delay-based approach—is highly *location-dependent*. For example, a delay-based detector might successfully flag traffic between hosts in the U.S. being diverted through the UK but may fail to do the same if traffic is diverted through Canada. Additionally, delays can be caused by many other factors such as network congestion, host processing times, and noise in access networks. Second, even if we accurately detect long-distance BGP hijacks, doing so in a scalable, real-time manner remains challenging. Per-packet round-trip time calculation is expensive at line rate. Furthermore, maintaining state and monitoring every flow is intractable. Thus, we pose the following research questions: *(1) What fraction of possible BGP hijacks can a delay-based approach detect? (2) Can we design a practical, always-on monitoring system to detect and mitigate hijacks in a scalable manner without excessive cost?*

To these ends, we design *HiDe*, a practical system for detecting hijacks using propagation delay, which relies on three key insights. First, BGP hijacks occur at the IP-prefix level and affect all traffic routed to the targeted prefix, meaning that during a real hijack, no packet to the targeted prefix can have a delay less than the minimum required for the attacker's route. By passively measuring

*Work done as a Master's student at Princeton University.

[1]This example assumes the speed of data transmission to be the speed of light in optical fiber, given by $c_f = 2c/3$ (approx.), where c is the speed of light in vacuum [24]. Hereafter, in this paper, *speed of light* refers to c_f, which is approx. 200 km per millisecond (more precisely, 199.86 km/ms).

the delays experienced by as many packets as possible and relying on the minimum per prefix, *HiDe* can reliably and scalably detect spurious delay surges. Second, we observe that a BGP hijack induces a distinct pattern in the *denoised* delay over time, clearly differentiating it from other events, such as congestion. Concretely, a hijack causes a *sharp* surge with location-dependent but calculable *minimum height*, which one can detect using a changepoint detection algorithm. Third, implementing *HiDe* can be made practical by deploying it on high-speed programmable switches. This is possible, despite the rigid computation and memory constraints of such devices, thanks to our switch-native implementation of changepoint detection and scalable latency measurements.

Our comprehensive evaluation demonstrates that *HiDe* is highly reliable (zero false negatives by design), minimally disruptive to real traffic, and implementable on commodity hardware. To assess *HiDe*'s effectiveness, we tested it against ethically-conducted real-world hijacks[2] and found that it detects them within 0.5 second. Additionally, to evaluate its impact on regular operations, we run *HiDe* on campus network traces (19 billion packets, 5.3 TB bytes). The results show that its combination of algorithms effectively minimizes false alarms (<0.012%), even in the presence of highly noisy real-world RTT signals. Furthermore, *HiDe* reduces the impact of such false alarms by identifying and correcting them within a median of 0.75 seconds without human intervention. We implement *HiDe* entirely on a programmable switch, showcasing its potential for seamless deployability on a network's border gateway with minimal hardware cost and no delay overhead to normal traffic[3].

2 Background

In this section, we describe BGP hijacks, present our threat model, and explain the limitations of existing strategies.

2.1 BGP-based attacks

BGP is the primary protocol that connects Autonomous Systems (ASes) by enabling them to exchange and forward route announcements for IP prefixes. Each AS advertises routes for the prefixes it owns, including an AS path indicating the sequence of ASes to traverse to reach it. Routers independently select the best route for each prefix based on attributes like path length and routing policies.

BGP hijacks. A BGP hijack occurs when a malicious or compromised AS falsely advertises routes to IP prefixes it does not own or cannot reach, misleading other ASes into rerouting traffic through its infrastructure. Suppose *AS*100 legitimately owns the IP prefix 1.1.1.0/24. A malicious AS, *AS*200, falsely announces ownership of 1.1.1.0/24 to its BGP peers. These peers may accept the announcement as valid and propagate it to their own peers, spreading the false route across the network. As a result, traffic destined for 1.1.1.0/24—for instance, originating from another prefix like 2.2.2.0/24 owned by *AS*300—may get misrouted to *AS*200 instead of reaching *AS*100. This enables the attacker to eavesdrop on, fingerprint, manipulate, or drop this illegitimately-obtained traffic. In some cases, the attacker may even serve malicious content by impersonating the legitimate destination (e.g., by rapidly acquiring a "valid" certificate first by exploiting the weakness in the certificate issuance verification process [11]).

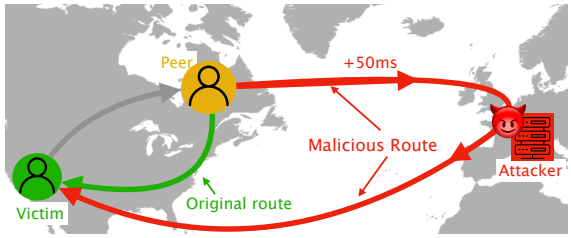


Figure 1: An attacker in the UK exploits the lack of routing security to redirect traffic from a *peer* host in the US—originally destined for a *victim* host in the US—through the attacker's own infrastructure. The *mid-attack* path (in red) from the peer to the victim is longer than the original *pre-attack* path (in green), adding an extra 50 ms of propagation delay.

BGP interception attack. A BGP *interception attack* (Figure 1) is a specific type of BGP hijack where the attacker intercepts traffic but forwards it to the original destination, enabling analysis or manipulation while remaining undetected by the end hosts. Of particular concern is a stealthy subset of these attacks where a sophisticated attacker employs techniques like *AS-path poisoning* and the manipulation of *BGP communities* to limit the propagation of malicious announcements in a bid to evade detection by BGP monitors near the victim. For instance, in the example above, *AS*200 could manipulate its announcements to propagate only to routers near *AS*300 while suppressing those to routers near *AS*100. This could cause traffic from 2.2.2.0/24 to 1.1.1.0/24 to get misrouted via *AS*200, while *AS*100 remains unaware as BGP monitors near it never observe the malicious route. Such an attack has been demonstrated by Birge-Lee et al. [12], which we ethically reproduce in §7.2.

2.2 Threat model

We consider an adversary performing a *stealthy* BGP interception attack—such as the one described above—to reroute traffic destined for a victim through distant infrastructure before forwarding it back to the victim. This infrastructure is located in a faraway country, potentially under different privacy and security laws. The adversary is sophisticated, aware of detection systems, and employs evasion techniques to suppress forged advertisements [12]. By rerouting traffic back to the victim, the attacker keeps connections alive, enabling traffic analysis while evading detection at the application layer. Such attacks can serve as tools in cyber warfare or surveillance. Real-world examples of long-distance interceptions include (among numerous others) the rerouting of US-based traffic via the UK to enable surveillance (Figure 1) [19], rerouting of US-based traffic managed by China Telecom via China undetected over 2.5 years [20], and rerouting of traffic between two hosts in Denver, USA via Iceland [40].

2.3 Limitations of existing approaches

Protocol enhancement-based approaches are expensive or inadequately deployed. Multiple protocols have been proposed

[2]Ethical issues are discussed in detail in our *Ethics* section (Appendix A).
[3]We will make our artifacts (analysis code, prototype, anonymized data, ethical hijack steps) publicly available upon acceptance (see Appendix B).

to secure routing on the Internet using cryptography to validate routing paths, including BGPSec [8], RPKI [15], and even future Internet architectures such as SCION [32]. The key drawback of such solutions is that they require widespread deployment to be effective, which is hindered by the lack of incentives among independent parties. BGPSec requires online cryptographic signing and validation, which incurs high overhead, hindering adoption. RPKI [15] is less expensive and has seen more deployment, but only supports origin validation and not path validation. This makes RPKI unable to detect BGP interception attacks, which maintain the origin of the prefix (to allow traffic to reach its destination). Similarly, SCION requires worldwide deployment, extra infrastructure, and processing overhead on end-host software.

Control plane-based monitoring approaches are slow and susceptible to evasion. There is an abundance of research and industrial tools for detecting BGP hijacks [4, 25, 37, 38]. These approaches analyze the BGP route advertisements that are captured by monitors in different locations on the Internet to find anomalies. However, the detection may take too long. As BGP advertisements can take a long time to propagate to all locations, the detection speed depends on the location of the monitor and its distance from the victim. This can critically delay the reaction to a hijack, allowing the attacker enough time to achieve their goal. Furthermore, while effective against hijacks caused by human error, such mitigation strategies can be evaded by sophisticated attackers. For example, previous works show that it is possible to avoid the monitors by carefully manipulating BGP communities to localize advertisements [12, 28].

Data-plane approaches lack practicality in real-world deployments. While some RTT-based detection methods exist, they fall short in production. Dart [36] introduces real-time RTT measurement in the data plane to address scalability challenges. While it employs min-filtering over windows and thresholds in an interception attack experiment, it cannot differentiate natural RTT variations (e.g., due to congestion), from malicious diversions—essential for effective hijack detection. Oscilloscope provides a more advanced methodology for hijack detection [14], but is based on idealized conditions, since it is evaluated only on emulated data. Even under these controlled scenarios, it suffers from high false-negative and false-positive rates. Moreover, it does not operate directly in the data plane hardware, further limiting its applicability in production.

3 Feasibility of delay-based detection

HiDe relies primarily on propagation delay for real-time BGP hijack mitigation. In this section, we examine the *feasibility* of using propagation delay alone to defend against long-distance interception attacks. In particular, we consider *cross-country* attacks—where both the victim and its peer reside in a *victim country* C_V, while the attacker operates from a different *threat country* C_T. Such attacks are common in nation-state cyber warfare and surveillance.

3.1 Key Questions and Observations

We ask the following questions about cross-country attacks:

(1) Does traffic within a single country have significantly lower propagation delay than traffic across countries? If so, can we use this difference to detect cross-country interception attacks?
(2) Are all countries equally *defendable* using propagation delay-based detection? If some countries are more defendable, what geographic factors drive this difference?
(3) What fraction of cross-country attacks can, in principle, be detected by comparing propagation delays—both under idealized (speed-of-light) assumptions and real-world measurements?

Our analysis in this section reveals the following:

(1) Across 258 countries, the max. distance within a country (*max. intra-country*) is typically far smaller than the min. distance from that country to any other country (*min. inter-country*). The $25^{th}/50^{th}/75^{th}$ percentiles of max. intra-country distances are 49 km, 413 km, and 1,129 km, respectively; the corresponding percentiles for min. inter-country distances are 4,027 km, 7,689 km, and 11,420 km. Propagation delays follow a similar trend.
(2) Some countries are inherently more defendable using propagation delay-based detection than others. For example, Russia ranks among the least defendable, whereas New Zealand is one of the most defendable. More generally, larger countries with many nearby neighboring countries are less defendable, while smaller or more geographically isolated countries are more defendable.
(3) Considering the worst-case (least defendable) attack paths between every pair of countries, we find that 97% cross-country attacks can be detected assuming speed-of-light RTTs, and 91% and 86% respectively, using real-world measurements from two production datasets.

3.2 Intra- vs. Inter-Country Distances

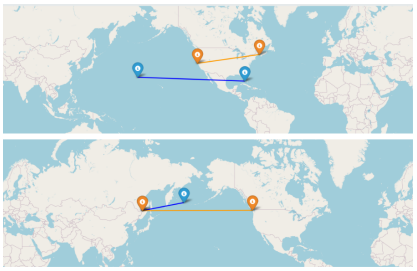
Goal. Our first goal is to assess whether the great-circle distance (shortest distance along Earth's curvature) between two hosts in the same country is significantly smaller than between hosts in different countries, so a cross-country detour would induce a detectable increase in propagation delay.

Dataset. We use the *Natural Earth Admin-0 Countries* dataset—one of the most popular boundary datasets in the Geographic Information System community. It provides high-resolution boundary coordinates (avg. 2.5 km) for 258 countries (Figure 2a) [18]. Since some countries consist of multiple disconnected regions (e.g., contiguous US plus Alaska and islands), we distinguish each country's *mainland* (largest contiguous landmass) from *entire country* (all regions combined).

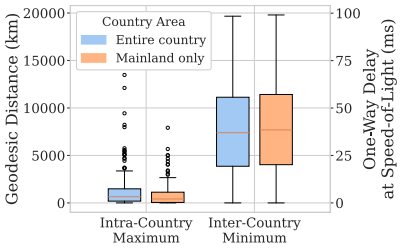
Method. For each country (entire country and mainland), we calculate: (1) Max. intra-country distance: the largest pairwise great-circle distance among all boundary points of the country, and (2) Min. inter-country distance: the smallest great-circle distance from any point in this country to any point in another. Figure 2b illustrates these distances within the US (top) and between the US and China (bottom).

Observations. Figure 2c shows: (1) min. inter-country distances far exceed max. intra-country distances, and (2) these distances for entire countries vs. mainlands are similar: we proceed with mainlands in the rest of this paper for better interpretability. At the speed of light, the $25^{th}/50^{th}/75^{th}$ percentile max. intra-country one-way delay (OWD) are 0.2 ms, 2.1 ms, and 5.6 ms, respectively; the corresponding values for min. inter-country OWD are 20 ms, 38 ms, and 57 ms.

(a) Centroids (orange icons) of the mainlands of 258 countries in the Natural Earth Admin-0 dataset [18].



(b) Top: Max. distance within country. Bottom: Min. distance between countries. (Blue: Entire country, orange: mainland).

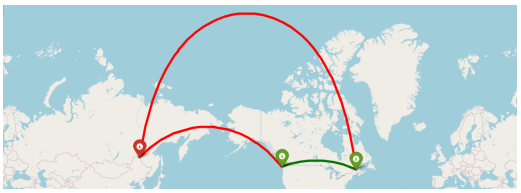


(c) Distribution of max. within and min. between countries: distance (left y-axis) and one-way delay at c_f (right y-axis).

Figure 2: For all 258 countries (Figure a)—using both entire country areas and mainlands only (Figure b)—we compute each country's maximum internal distance and its minimum distance to every other country, then plot both distributions (Figure c). Typically, a country's foreign neighbors are more distant than its own farthest points.

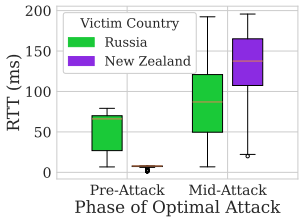
(a) Distance before attack: $\delta_{pre} = 2\delta(S, D)$.

(b) Distance in the middle of the stealthy interception attack: $\delta_{mid} = \delta(S, D) + \delta(S, A) + \delta(D, A)$.

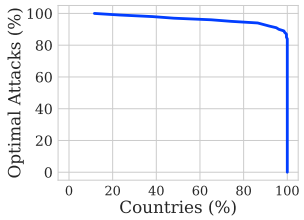


(c) Locations of source, dest. in the US and attacker in China such that $\delta_{mid} - \delta_{pre}$ is minimized.

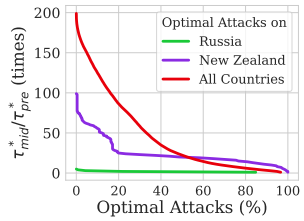
Figure 3: In this example, source S and destination D lie in mainland US and attacker A in mainland China. Figure (a) shows the *pre-attack* round-trip distance δ_{pre} and (b) the *mid-attack* round-trip distance δ_{mid}, leading to the deviation $\delta_{deviation} = \delta_{mid} - \delta_{pre} = \delta(S, A) + \delta(D, A) - \delta(S, D)$. Figure (c) shows the *most optimal attack* on the US from China, with curved lines indicating shortest great-circle paths. (Green: Original path, red: diversion due to attack.)



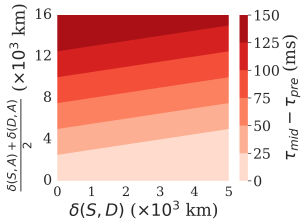
(a) Pre- and mid-attack RTTs during all possible optimal attacks on 2 example countries.



(b) Percentage of countries vs. percentage of defendable optimal attacks.



(c) Attack coverage vs. mid-attack RTT (as a multiple of pre-attack RTT).



(d) Deviation (z) as a function of dist. b/w hosts (x) and avg. dist. b/w hosts & attacker (y).

Figure 4: Defendability against optimal attacks assuming speed-of-light RTT: With Russia and New Zealand (NZ) as example victim countries, (a) shows mid-attack RTT is typically much higher than pre-attack RTT; size and proximity of victim country to other countries determine the extent. Figure (b) shows that for 86% countries can be defended against 94% optimal attacks. Figure (c) shows Russia's post-attack RTT peaks at 4x its pre-attack RTT (corresponding to 110 ms absolute difference), NZ at 100x (190 ms), and all countries combined at 198x (200 ms). Figure (d) shows that, when the victim and peer are co-located, the attacker must be 2,500 km away to induce a deviation of 25 ms; as the victim and peer separate, the attacker must be more distant to induce the same deviation.

3.3 Identifying least defendable attacks

In this subsection, we describe how we identify the least defendable attack given a C_V and a C_T; later, we analyze defendability under ideal (§3.4) and realistic (§3.5) conditions. Figures 3a, 3b show the paths before and during the attack: with source S and destination D in C_V and attacker A in C_T, the round-trip distance changes from

$\delta_{pre} = 2\delta(S, D)$ to $\delta_{mid} = \delta(S, D) + \delta(S, A) + \delta(D, A)$, resulting in a deviation of $\delta_{deviation} = \delta(S, A) + \delta(D, A) - \delta(S, D)$. In the worst-case (least defendable) attack scenario, $\delta_{deviation}$ is minimized by having S and D on C_V's border and as far apart from each other as possible, and A on C_T's border and as close to S and D as possible—we call this an *optimal attack*. Under such an attack, we denote the pre-, mid-attack distances, and deviation as δ^*_{pre}, δ^*_{mid}, and $\delta^*_{deviation}$, respectively, and the corresponding RTTs as τ^*_{pre}, τ^*_{mid}, and $\tau^*_{deviation}$. Figure 3c shows the optimal attack from China on the US, as an example. We compute the optimal attack for each ordered pair of victim and threat countries (258 x 257 attack scenarios).

3.4 Defendability under ideal conditions

Goal. In this subsection, we analyze the feasibility of detecting cross-country optimal attacks under *ideal conditions*.

Assumptions. "Ideal conditions" include: (1) Ideal network — data travels at the speed of light, and (2) Ideal measurements — our measurements capture the actual distance-based propagation delay. Under these assumptions, propagation delay = minRTT = RTT. Therefore, in this subsection, *RTT* refers to propagation delay. We relax these assumptions in §3.5.

Most and least defendable countries. Optimal attacks determine the lower bound of our defense capabilities. To assess defendability under optimal attacks, we compute propagation delay as the round-trip distance (δ^*_{pre} or δ^*_{mid}) divided by speed of light. To illustrate the difference in defendability across countries, we select two examples: Russia, which has among the smallest median $\tau^*_{deviation}$, and New Zealand (NZ), which has one of the largest median $\tau^*_{deviation}$. Figure 4a shows their pre-attack and mid-attack RTT distributions. NZ's RTTs increase significantly mid-attack, making it more defendable; Russia's RTT changes are smaller making it less defendable. In general, small countries with distant neighbors (low $\delta(S, D)$, high $\delta(S, A) + \delta(D, A)$) are the most defendable, while large countries with many close neighbors are the least defendable.

Attack coverage. To quantify defendability, we define *attack coverage* as the percentage of optimal attacks that can be detected under some given condition. To compute overall coverage, we set the condition $\tau^*_{deviation} \geq 5$ ms because: (1) 5 ms far exceeds typical noise in measurements (e.g., due to coarse-grained timestamps, rounding off errors, approximations in distance calculations, etc.), and (2) At speed of light, 5 ms corresponds to approx. 1,000 km of extra path length, so it captures meaningful geographic detours. The attack coverage for Russia (over 257 optimal attacks) is 85% (the minimum for any country), while the same for NZ is 100%. When expanded to all countries (i.e., 258 x 257 optimal attacks), the coverage is 96.6%. Figure 4b shows that 100% (i.e., all) countries can be defended against 84% attacks, 75% against 95%, 23% against 99%, and 11% against 100%. These results illustrate the promise and generality of propagation delay-based detection (in ideal conditions).

Attack coverage at given RTT deviations. To analyze the extent to which optimal attacks increase RTT in ideal conditions, we plot the attack coverage (x-axis) given the ratio between mid- and pre-attack RTT (Figure 4c) (absolute difference shown in appendix–Figure 13). For NZ (purple), this ratio ranges from 1-100x (5-190 ms absolute difference), while for Russia (green), due to its large pre-attack RTTs, it is 1-4x (5-110 ms). For attacks on all countries (red), it is 1-198x (5-200 ms). The ratio is 2x (8 ms) for 95% attack coverage on all countries, and 4x (23 ms) for 85% coverage.

Deviation as a function of distances. The analysis of cross-country attacks does not inform us directly about the relationship between distance and RTT deviation. To bridge this gap, Figure 4d shows RTT deviation (z-axis)—as a function of pre-attack distance ($\delta(S, D)$) (x-axis), and average distance between S-A and D-A (y-axis)—in a heatmap. The x- and y-axis are capped at the maximum intra- and minimum inter-country distances. The $85^{th}/95^{th}$ percentile pre-attack distances are 1,090 km and 1,872 km; to induce an RTT deviation of 25 ms, the attacker needs to be at an average distance of 3,045 km and 3,436 km, respectively, from S and D.

3.5 Defendability *in the wild*

Goal. While our observations under ideal conditions are promising, defendability may differ in the real world because: (1) the actual propagation delay may be larger than the speed-of-light RTT due to longer physical paths, and (2) RTT measurements may not capture the actual propagation delay due to congestion or poor channel conditions (in wireless networks). In this subsection, we evaluate defendability in realistic conditions using two production datasets.

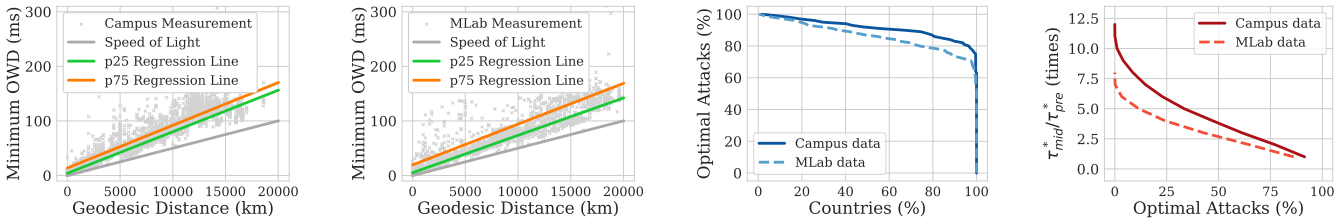
Datasets. Our datasets are outlined below:

- **Campus dataset:** We collect traffic from 7.5 M TCP flows on our campus over 12h on a weekday in May '22, and compute RTTs by matching data packets with ACKs [17].
- **MLab dataset:** We collect minRTTs of 4.3 M TCP flows from NDT7-based measurements over 5 days in Dec. '24 [27].

For each flow in each dataset, we collect geolocations of the source and destination. We discard flows whose minRTT indicates shorter distances than those permitted by their reported geolocations, which is physically impossible. Finally, we group remaining measurements by source-destination /24 prefixes, because (1) a /24 prefix is the smallest unit on which a BGP hijack can be launched, and (2) aggregating by prefix improves the chance of measuring true minRTT (§5.2).

Estimating propagation delay from distance. To quantify defendability in realistic settings, we estimate real-world propagation delay between any two hosts from their great-circle distance d. This real-world propagation delay may vary across host pairs separated by the same distance d depending on: (1) Geolocation: Some regions in the world have denser network connectivity than others, (2) Routing policies: Some providers make shorter paths available than others, (3) Stable queues: Some paths experience consistent queuing delay due to deep buffers, etc. Furthermore, even if the true propagation delay is same, we may measure different minRTTs due to transient congestion. It is infeasible to collect reliable minRTTs from all possible attack locations in all 258 countries. Instead, we apply the following method to both our campus dataset (215 countries) and Google MLab (234 countries) to capture variability:

(1) **Bin distances:** Divide all distances up to 20,075 km (Earth's diameter) into 200 km bins (≈1 ms at c_f).
(2) **Assign prefixes to bins:** For each source–destination prefix pair, compute its great-circle distance, assign it to the appropriate bin, and record its minOWD (minRTT/2).
(3) **Percentile computation:** Within each bin, compute the pth percentile of these minOWDs for $p = 1, \ldots, 100$.



(a) Campus measurements over 12 hours in May '22.

(b) Google MLab measurements over 5 days in Dec. '24.

(c) % countries vs. % defendable optimal attacks.

(d) % attacks vs. ratio of mid-attack to pre-attack RTT.

Figure 5: Defendability against optimal attacks based on real measurements: We estimate (using linear regression) the $p25$ and $p75$ OWD for each 200 km distance bucket of our campus dataset and the Google MLab dataset, in (a) and (b) respectively. Using $p75$ OWD to estimate pre-attack and $p25$ to estimate mid-attack RTT, 85% countries can be defended against 85% attacks based on the campus dataset, and 85% against 78% based on MLab (Figure (c)). Figure (d) shows that the mid-attack RTT peaks at 12x pre-attack RTT in the campus dataset, and 7.5x in MLab.

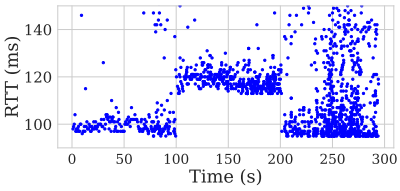


Figure 6: *Abrupt* and *significant* rise and fall in RTT due to interception attack launched (ethically) at 100^{th} second and withdrawn at 200^{th}.

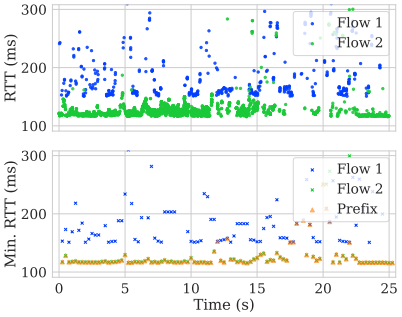


Figure 7: Top: Flow 1 (blue) with noisy RTTs and flow 2 (green) with stable RTTs. Bottom: Aggregating by prefix stabilizes minRTTs (orange).

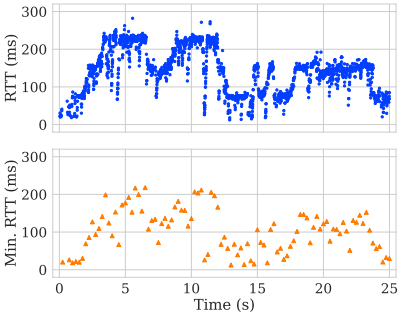


Figure 8: Prefix with noisy RTTs (top) that produce noisy minRTTs (bottom) despite windowing. Such prefixes are less defendable.

(4) **Regression fitting:** Fit one linear regression per percentile across all bins (e.g., a p=1 line through every bin's 1-percentile).

Figures 5a and 5b plot the p=25 and p=75 regression lines for the campus and MLab datasets, respectively. The campus data shows a narrower inter-quartile range—likely because the location of one end is fixed (on campus) and the entire variability is due to the remote host—whereas for MLab, the servers and clients are in different locations. With these delay estimates, we proceed to evaluate defendability against optimal attacks under realistic conditions.

Estimating pre- and mid-attack minRTTs. The variability of minOWDs for the same distance poses a challenge: if our detection is unlucky, it could measure a higher percentile minRTT before the attack and a lower percentile during the attack, causing the deviation in minRTT to be much lower than speed-of-light deviation. To model such a scenario, we use the upper quartile (75^{th} percentile) minOWDs to estimate pre-attack minRTTs, and the lower quartile (25^{th} percentile) minOWDs to estimate mid-attack minRTTs. Then, we evaluate defendability using the same metrics as before.

Observations. Using the condition $\tau^*_{deviation} \geq 5$ ms, the overall attack coverage is 91% on the campus dataset and 86% on MLab. Figure 5c shows that in the campus data, 100% countries can be defended against 63% attacks, 75% against 89%, and 2% against 100%. In MLab, 100% countries can be defended against 53% attacks, 75% against 80%, and <1% against 100%. Real-world defendability is therefore less than in the ideal case—especially in MLab, where minRTT variability is higher. Figure 5d plots coverage versus the mid-/pre-attack minRTT ratio (absolute diff. in appendix–Figure 14). The ratio ranges from 1–12x (5–290 ms) for campus; 1–7.5x (5–250 ms) for MLab. Appendix-figures 15, 16 present corresponding distance-deviation heatmaps, which follow similar trends.

3.6 Takeaways

Our analysis shows that propagation-delay measurements offer a highly effective signal to defend against cross-country interception, primarily because diverted paths almost always incur substantially greater delays than pre-attack paths. Although real-world variability degrades detection coverage compared to ideal conditions, our focus on worst-case (optimal) attacks means these results are conservative—actual detection performance will often exceed our current estimates. We believe our findings are sufficiently strong

to justify designing an interception-detection based solely on propagation delay. At the same time, a range of factors influences how accurately any given attack can be detected: the victim country's size and distance from potential adversaries; the exact geolocations and separation of victim and peer hosts; the true lengths of the pre-attack and mid-attack network paths; transient or persistent congestion along those paths; and the precision of our measurement and aggregation techniques. With these insights in mind, in the next section, we present *HiDe*—a scalable, always-on, data-plane system for detecting and mitigating interception attacks.

4 *HiDe*: Overview

HiDe is a BGP-interception mitigation system that runs entirely on a programmable switch and uses real-time minRTT measurements for detection and mitigation. In this section, we present the key insights that drive *HiDe*.

Converting noisy RTT into a reliable detection signal. During a hijack, all traffic to a victim prefix must traverse the longer path via the attacker, so no RTT sample can be shorter than the minimum propagation delay via the attacker. *HiDe* exploits this by passively collecting RTT samples for every TCP data-ACK pair at the network border (thereby avoiding noise from the internal network), aggregating samples per prefix, and tracking the minRTT per time window. By monitoring these minRTTs, *HiDe* detects hijacks as sudden, sustained spikes in delay. For example (Figure 6), hijacking Bitcoin traffic ethically from a Stockholm client via Amsterdam causes the minRTT to jump by about 20 ms at attack start and to fall back when the hijack ends. A changepoint detection algorithm can reliably identify such shifts.

Prioritize guaranteed protection over broad coverage. We posit that operators prefer a system that reliably defends a well-defined subset of prefixes rather than a best-effort approach that "covers" everything but floods them with false alarms. Section 5.3 shows how *HiDe* restricts its scope to prefixes it can protect with high confidence. When false positives do occur, *HiDe* continues measuring RTT and automatically rolls back its mitigation if the spike proves transient.

Optimize for commodity hardware. *HiDe* stores only per-prefix state—the running minRTT and sample count per time window—instead of expensive per-flow or per-packet state. It employs a lightweight, two-window, threshold-based changepoint detector that is hardware-amenable. On Tofino2, *HiDe* uses native primitives (*mirror* and *packetgen*) to generate occasional packet replicas for RTT measurement, false-positive correction, and (optinal) user alerting—while forwarding all other traffic at line rate with zero additional latency.

5 *HiDe*: Methodology

5.1 Compute location-based lower bound

Translating user input into geographic locations. The user provides *HiDe* with the IP prefix of the home network and the threat regions they want to protect their data from, based on policy decisions or anticipation of threats. The threat regions are either names of countries or enclosed polygons of geographic coordinates. *HiDe* also obtains from its data plane the destination prefixes observed by it. The user can, *optionally*, set threat regions *per destination prefix*. Eventually, the control plane converts all the information into triplets of geolocation information: {source_coordinates, destination_coordinates, threat_coordinates_list} using public geolocation services (*IPinfo*, *MaxMind*) and public geographic datasets (*Natural Earth Admin-0*) [18, 26, 39].

Computing lower bound of mid-attack RTT. For each location triplet, we first identify the *optimal attack*, i.e., the attacker's location in the threat region that minimizes mid-attack round-trip distance. Note that this distance can be much higher than in the optimal attacks in §3 because there, source and destination were always on the victim country's border whereas here, they are almost always inland. Next, we compute the minimum possible mid-attack RTT for this optimal attack (τ^*_{mid}), based on the speed of light. We designate this *lower bound* RTT as the *absolute threshold* of our changepoint detector: *HiDe* flags an attack whenever the observed minRTT reaches τ^*_{mid}, guaranteeing *zero false negatives* (see §5.5 for false positives). While we could choose a less conservative bound—e.g., the 25^{th}-percentile estimated latency in 200 km buckets (§3.5)—we opt for the most conservative threshold to *guarantee* protection, at the expense of coverage, as is our design goal (§4).

5.2 Reduce noise in the RTT signal

Aggregating by prefix to reduce impact of noisy flows. BGP attacks target prefixes, with a /24 prefix being the smallest possible target. All flows to an attacked prefix experience the same underlying change in propagation delay, but noisy RTTs in individual flows can obscure this change. We aggregate RTT samples by prefix before computing the minimum RTT per window (discussed next), as at least one flow per window is likely to produce a sample representative of the true propagation delay. Figure 7 demonstrates this for a US-based destination prefix with one noisy and one stable flow. Also, prefix-level aggregation reduces switch memory requirements from per-flow to per-prefix, which is significant.

Windowing to discard short-term fluctuations. We divide streams of per-prefix RTT samples into non-overlapping time windows of a fixed size (i.e., *tumbling* windows) and compute the minRTT in each window. This helps filter out short-term spikes in RTT due to benign confounding factors like queuing delay from short-lived congestion, end-host processing delays, and TCP oddities like *delayed ACKs* [36]. We select a sub-second (e.g., 0.25- or 0.5-second) time window—while minRTT can be measured more reliable by with longer time windows, it would delay mitigation allowing an attacker more time to complete their attack. Finally, tracking minRTTs in *non-overlapping* tumbling windows requires only per-flow state, as opposed to *overlapping* sliding windows, making it more suitable for a switch implementation.

5.3 Cover vulnerable & defendable prefixes

Identifying vulnerable prefixes. BGP interception attacks primarily target prefixes that host sensitive services—government sites, banking portals, cryptocurrency nodes, and the like—because such websites handle sensitive data from users around the world. The attacker places itself between the user and the server—intercepting "valuable" data. *HiDe* prioritizes these vulnerable server prefixes for protection. By default, it excludes prefixes used exclusively by

WiFi or cellular access networks—since they rarely host critical services—unless the operator explicitly includes them.

Identifying defendable prefixes. Some destination prefixes have RTTs that are *consistently* noisy or high even under benign conditions, making it nearly impossible to detect interception attacks on them from certain threat regions without excessive false positives. For example, Figure 8 shows a prefix where the minRTT often exceeds 100 ms, making it impractical to defend against a threat region that causes a small deviation in comparison. Further analysis of such prefixes reveals that often, they tend to be associated with client-side access networks, such as cellular or WiFi, which *HiDe* does not defend by default anyway. Concretely, during a *profiling* phase independent of the detection phase, we monitor the *max. of min. RTTs* in tumbling time windows for each destination prefix. Later, we defend a prefix against a threat region only if $\tau^*_{mid} - max(RTT_{min}) > \lambda$, where λ is called the *surge threshold*. λ defines the min. increase in RTT_{min} required between two consecutive windows to flag an attack, and can be set to a constant (e.g., 10 ms) or a fraction of $max(RTT_{min})$ (e.g., 10%). A lower λ provides broader coverage but increases susceptibility to false positives, and vice-versa. Users can adjust λ based on their desired trade-offs. We evaluate the false positive rate for different values of λ in §8.

5.4 Switch-amenable changepoint detection

We implement changepoint detection directly in switch hardware by combining the techniques described so far in an approach called the *two-window algorithm*. This involves tracking the per-prefix min. RTT (RTT^i_{min}) for each tumbling window i. Once the i^{th} window completes ($i > 0$), we compare RTT^{i-1}_{min} and RTT^i_{min} and mark the prefix as *attacked* if both the following *surge* conditions are met:
(1) $RTT^{i-1}_{min} < \tau^*_{mid}$ and $RTT^i_{min} > \tau^*_{mid}$: The minRTT crosses the absolute threshold between two consecutive windows.
(2) $RTT^i_{min} - RTT^{i-1}_{min} > \lambda$: The minimum RTT surges by at least the surge threshold in consecutive windows.

Valid windows. Only time windows that contain at least 5 samples are considered valid and used for detection—windows with fewer samples do not necessarily benefit from min-filtering and could lead to false positives.

5.5 Minimize impact of false positives

Despite reducing false positives, our detection algorithm is not foolproof and may occasionally generate them. To minimize their impact and to eliminate the need for human intervention, *HiDe* employs an automatic *false positive correction* mechanism. When an attack is detected, *HiDe* blocks the affected prefix and simultaneously initiates active probing by sending ICMP echo packets to the most recently active IP address in the prefix at each time window. It monitors the corresponding RTT, and if the RTT falls below τ^*_{mid}, the prefix is unblocked, and detection resumes, minimizing disruption to regular operations. To limit probe traffic, *HiDe* reduces the probe rate to one per minute after five minutes of attempts. These parameters are user-adjustable for flexibility.

6 *HiDe*: System

Figure 9 presents an overview of *HiDe*'s end-to-end workflow. *HiDe* comprises a control plane, implemented in software on a server, and a data plane, operating in high-speed hardware on a programmable switch. *HiDe* is deployed at the edge of a production network, and can observe all or most of its traffic depending on the network topology (appendix E).

6.1 Control Plane

User Input. The user configures the control plane by providing the network prefix of their home network (source prefix) and specifying the threat regions (optionally, per prefix).

Auto-Tuning. The auto-tuning component translates the user inputs and destination prefixes read from the data plane into corresponding geographic coordinates and computes the τ^*_{mid}. It also retrieves traffic statistics (specifically, $min(RTT_{min})$ and $max(RTT_{min})$) from the data plane for each prefix. Combining this information, the component identifies which prefixes can be effectively protected and generates changepoint detection parameters for those prefixes, which it then sends to the switch controller. For prefixes that cannot be protected, it provides the user with a summary listing each prefix and their corresponding RTT_{min} statistics.

Switch Controller. The switch controller translates the received parameters into corresponding match-action rules and installs them on the data plane. It also receives attack alarms from the data plane and (optionally) notifies the user.

6.2 Data Plane

RTT Computation. We leverage Dart, an existing system, to generate accurate RTT measurements per flow from all the traffic observed by the switch. Dart achieves this at scale, handling a large number of flows without missing any RTT samples by efficiently managing switch resources [36].

Min. RTT Aggregation. The next component aggregates RTT samples per destination prefix, breaks them down into time windows, and calculates the minimum RTT per window. Additionally, it computes traffic statistics like $min(RTT_{min}$ and $max(RTT_{min})$ per prefix to share with the control plane.

Changepoint. The data plane performs changepoint detection using our two-window algorithm, minRTTs per window per prefix, and parameters installed by the control plane.

Attack Mitigation. When an attack is detected, the data plane blocks the corresponding prefix and raises an alarm.

False Positive Correction. The data plane then crafts and sends active probes periodically to determine whether the detection was a false positive. If so, it unblocks the prefix.

6.3 Hardware Switch Prototype

We implement our prototype in $P4_{16}$, and deploy it on the Intel Tofino2 high-speed programmable switch, which supports up to 12.8 Tbps of traffic at line rate [6, 13]. Our prototype does not depend on any specific features available on the Tofino, and can be ported readily to other programmable packet-processing hardware including other switches (e.g., Juniper Trio) and SmartNICs (e.g., Nvidia BlueField3).

Switch control plane. The switch control plane installs per-prefix match-action rules specifying the window size (W), absolute threshold (τ^*_{mid}), and surge threshold (λ). If enabled, it listens on the CPU port for packets from the data plane containing information about

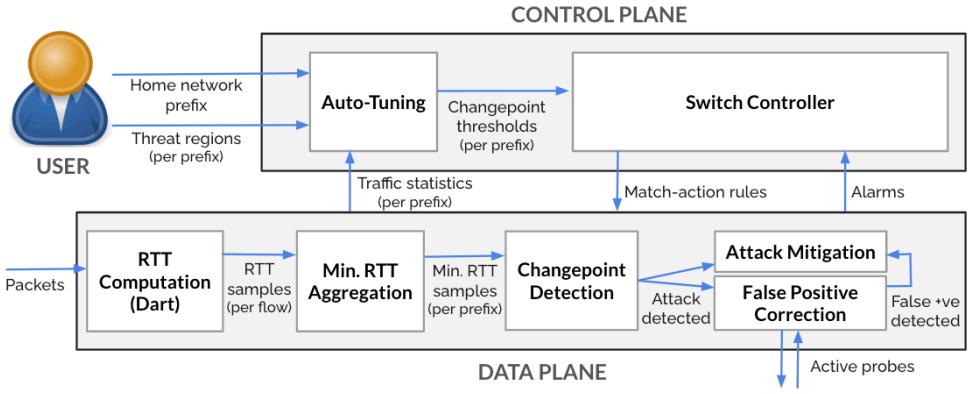


Figure 9: *HiDe* consists of a software control plane and a hardware data plane. The control plane auto-tunes per-prefix parameters for changepoint detection based on user inputs and traffic statistics from the data plane, and installs those parameters as match-action rules on the data plane. The data plane computes RTT samples, aggregates them by prefix, computes minRTT per window, and performs changepoint detection. Upon detecting an attack, the data plane blocks the corresponding prefix and triggers active probing to correct false positives.

either attack detections or non-coverage, and notifies the user. Additionally, it configures the Tofino's *packet generator* to send active probes during the *false positive correction* phase.

RTT computation. We leverage Dart for continuous and accurate per-flow RTT computation [36], and utilize *packet mirroring*, a native feature that replicates packets, to enhance its functionality. First, the original packet is forwarded without added latency, with the mirrored copy used for RTT computation. Second, RTT samples generated by Dart are passed to *HiDe*-specific data-plane components.

Per-prefix state. We maintain a *prefix table* in register memory to support changepoint detection. The table uses a *prefix signature*, derived by hashing the first 24 bits of the external IP, as the key. The stored values include the prefix's start timestamp, timestamp of most recent RTT, start timestamp of current window, number of RTT samples in current window, minRTTs for the current and previous windows, attack status, $max(RTT_{min})$, and $min(RTT_{min})$. The table accommodates up to 65,536 active prefixes, significantly exceeding the peak observed in our 12h campus trace (approx. 5K assuming a 5-second timeout), minimizing hash collisions. For collisions, we use *cuckoo hashing* [31]: the new prefix replaces the old one, which is loaded into memory, checked for timeout, and recirculated to a new index using a different hash seed if still valid. Each insertion allows up to 3 recirculations.

Changepoint detection. When an RTT sample is generated for a prefix, *HiDe* checks the status of the corresponding time window. If the window is not full, it updates the most recent timestamp, increments the RTT count, and replaces the current minimum RTT if the new sample is smaller. If the window is full, the current minimum RTT replaces the previous window's minimum, and $max(RTT_{min})$ and $min(RTT_{min})$ are updated. If the window is valid (i.e., it has enough samples), *HiDe* evaluates the surge conditions and, if those are satisfied, starts the mitigation process by blocking all non-ICMP packets from/to the prefix by adding it to a *block table*.

Active probing. Simultaneously, we start crafting and sending ICMP echo packets to the latest IP seen from the prefix and listening to responses to monitor its RTT. If a false positive is detected, the prefix is removed from the block table.

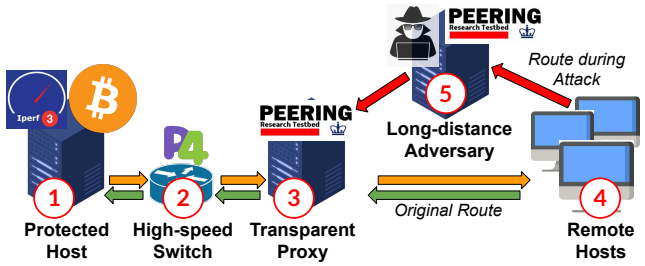


Figure 10: Experimental setup for our live experiments. The orange and green arrows indicate the original "protected host" to "remote hosts" route and back, respectively. The return path (green) is intercepted by the *long-distance adversary*—the diverted portion of the route is shown with red arrows.

Resource usage. We analyze the resource usage of our prototype by function and find that its low resource consumption leaves ample resources for other concurrent switch functions (Table 1 in Appendix E).

7 Experimental Setup

In this section, we outline our experimental setup: first, to demonstrate live detection of ethically launched interception attacks on controlled *iperf* traffic; second, to showcase *HiDe*'s effectiveness *in the wild* by detecting attacks on real *Bitcoin* traffic; and third, to collect a 12-hour campus trace highlighting *HiDe*'s low false positive rate and minimal impact on regular operations.

7.1 Passive Capture of Production Traffic

Data collection. As outlined before, we captured 12h of production traffic—covering 1 pm to 1 am local time to include global working hours—at the edge of our US-based campus network using

a TAP device near the gateway router. Packets—only TCP headers, anonymized at source in a prefix-preserved manner (Appendix A)—from selected subnets were mirrored and recorded on a collection server with *tcpdump*.

Dataset overview. The dataset comprises 1.1 TB of trace data representing 5.32 TB of packet bytes, encompassing 19 billion packets, 7.5 million flows, and 238 million RTT samples. It includes 12K unique internal IPs and 324K unique external IPs, distributed across 183K external prefixes, 23.2K of which are based in the US.

7.2 Live Experiments

Figure 10 shows the experimental setup for our live experiments involving active traffic from the iperf3 and Bitcoin apps. Each component is marked with a number in red.

Deploying *HiDe* to protect experimental traffic. We set up our experiment using three key components: (1) a host on our campus running the application (*iperf3* or *Bitcoin*), (2) a high-speed programmable switch on campus where *HiDe* is deployed to monitor traffic, and (3) a transparent TCP proxy on an Amazon AWS instance. The proxy, which doubles as a *PEERING* node, advertises a /24 prefix allocated to our experiment. PEERING provides distributed ASes for controlled, real BGP announcements [35]. We use one IP address from the /24 pool, applying *iptables* rules on components 1 and 3 to masquerade it as the application's IP. This setup enables *HiDe* to monitor all experimental traffic while allowing external adversaries to ethically launch BGP interception attacks on the /24 prefix. The transparent proxy ensures the application on component 1 sees the remote host's true IP, essential for applications like Bitcoin.

Setting up remote hosts. For our experiments with *iperf3*, we deploy AWS instances in geographically diverse locations, including the US east and west coasts, Europe, and Asia. The *iperf3* server runs on the protected host across multiple ports, while clients on the distributed AWS instances connect to the PEERING IP of the transparent proxy, which forwards traffic to the server. For Bitcoin experiments, we run a node on the protected host, allowing random nodes worldwide to connect. Across different runs, we observe peers from the US, various European countries, and parts of Asia. We collectively refer to these remote hosts—used in both *iperf3* and *Bitcoin* experiments—as component 4.

Launching ethical routing attacks. The final step in our setup is launching ethical BGP interception attacks on the PEERING IP. We designate the PEERING node in Amsterdam as the attacker (component 5) and implement a stealthy interception attack using the technique by Birge-Lee et al., which employs BGP communities to control the blast radius of the attack [12]. The attacker advertises the same /24 prefix as the transparent proxy (an *equally specific* attack), redirecting traffic from nearby nodes to Amsterdam. The attacker then forwards the intercepted traffic to the transparent proxy, leaving both sender and receiver unaware of the attack.

8 Evaluation

In this section, we present our evaluation results. In the first part (Section 8.1), we run faithful simulations of *HiDe* on our campus dataset and report coverage, false positive rate, and downtime due to false positives. In the second part (Section 8.2), we present results from live experiments where the *HiDe* prototype defends the protected host (in Figure 10) when a subset of connections are impacted by an ethically conducted long-distance BGP interception attack.

8.1 Trace-based Evaluation

We evaluate *HiDe* using three metrics: (1) False positive rate or FPR (measures reliability/usability/practicality), (2) Coverage (measures the trade-off with low FNR and FPR), and (3) Downtime (measures impact of false positives on regular operation). We perform this evaluation using a faithful simulation of *HiDe* written in Python on real latency data obtained from production traffic on our campus (Section 7.1). We operate with the goal of protecting US-based prefixes from long-distance interception attacks from the mainlands of other countries in our dataset. For each prefix, we divide into two equal parts the total time during which the prefix was active: the first half is used for *profiling* while the second half is used for *detection*.

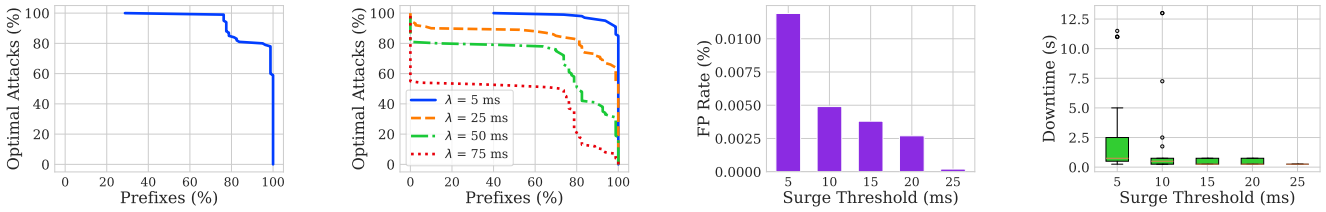
Vulnerable prefixes. In accordance with *HiDe*'s coverage strategy, we only defend vulnerable prefixes, i.e., external prefixes associated with a server. We identify such prefixes using a TCP port number-based heuristic: external host's port < 1024 and internal host's port >= 1024. 16.8K US-based external prefixes match this condition.

Profiled prefixes. From external server prefixes, we further select those that were active for at least 10 minutes out of 12 hours (so we profile on at least 5 mins of data). We determine this by dividing the 12h period into buckets of 1 min, and checking which prefixes generated an RTT sample in at least 10 such buckets. 6K US-based external server prefixes are retained after this step. In a real network, the operator could profile prefixes for as long as needed to ensure it covers typical variation of RTT during benign operation—without any excess overhead since the profiling happens directly in the switch. For the following analysis, we make the assumption that the campus data captured during the 12-hour period did not experience any long-distance interception attacks (i.e., no true positives were present), so if *HiDe* detects a prefix it must be a false positive. We report our results based on optimal attacks from all 257 non-US threat countries.

Coverage impact of theoretical lower bound. Some US-based prefixes are not defendable against certain threat regions because during the profiling phase, they exhibit a $min(RTT_{min})$ larger than the corresponding τ^*_{mid} (i.e., measured delay without diversion is always higher than the lower bound with diversion). This could be because the threat region is geographically too close or because the network is always congested. Figure 11a shows that, based on this condition, *HiDe* can cover 99% prefixes against 78% attacks and 75% prefixes against 99% attacks. The covered prefix-threat country pairs are considered in the subsequent experiments.

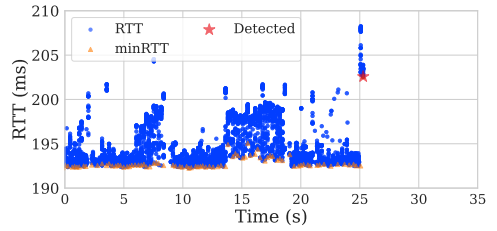
Coverage impact of defensability analysis. For different values of the surge threshold (λ), Figure 11b shows *HiDe*'s coverage based on defendability—i.e., whether the mid-attack lower bound RTT clears the pre-attack $max(RTT_{min})$ by at least λ ms. At λ = 5 ms, 25 ms, 50 ms, and 75 ms respectively, *HiDe* can cover 99% prefixes against 91%, 64%, 31%, and 7% attacks, respectively. This illustrates the trade-off between surge threshold and coverage.

False positive rate. By focusing on defendable prefixes, we achieve a false positive rate of approximately 0.012% at worst, as shown in

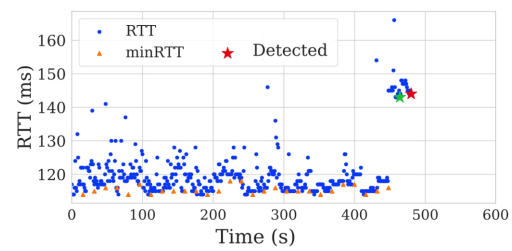


(a) Coverage (lower bound) **(b) Coverage (defendability)** **(c) False positive rate** **(d) Downtime**

Figure 11: Faithful simulation on campus data illustrates that *HiDe* can defend most prefixes from optimal attacks from most countries, incurs low false positives (<=0.012%) and low downtime due to false positives (median<=0.75s).



(a) Detecting attack on iperf3 traffic.



(b) Detecting attack on Bitcoin traffic.

Figure 12: *HiDe* (immediately) detects interception attacks ethically launched by us on iperf3 and Bitcoin traffic.

Figure 11c. For higher surge thresholds (i.e., 30 ms), the rate drops to zero.

Downtime due to false positives. We estimate the likely downtime from a false positive by measuring how long (in multiples of time window size) it takes for the minRTT to return to normal for a falsely detected prefix. Since our active probing sends one probe per window, we expect similar results in reality. The median downtime is only 0.75 seconds.

8.2 Live Interception Attack Detection

Mitigating attacks on *iperf3* traffic:

Setup. We run the iperf3 server on our campus and the transparent proxy in Ireland, who forwards all traffic it receives to our campus via our prototype. The iperf3 clients are in Virginia (2 flows from the same prefix), Ohio, and Mumbai. The prefix in Virginia is hijacked from Amsterdam, causing Ireland to send traffic to Amsterdam instead of Virginia. Amsterdam then forwards the traffic to Virginia. The traffic takes the following round-trip route before the attack: *Virginia (via PEERING infra.) to Ireland to our campus to Ireland to Virginia (via PEERING infra.)* and the following one during the attack: Virginia (via PEERING infra.) to Ireland to our campus to Ireland to Amsterdam (via. PEERING infra.) to Virginia (via PEERING). Due to limitations of where we can deploy a Tofino switch on live traffic and a lack of diversity in the PEERING topology, we are restricted to this complex setup. The attack takes effect at 25 seconds, as can be observed from the abrupt rise in RTT (blue dots) in Figure 12a.

Interception detection. Using multiple runs of traceroute, we estimate the lower bound of RTT as approx. 190.5 ms before attack and 199 ms during attack (absolute threshold). We set the window size to 0.25 sec and the surge threshold to 5 ms. Based on the minimum RTTs (orange triangles), we detect the attack almost immediately (red star).

Mitigating attacks on *Bitcoin* traffic:

Our setup is similar to the previous experiment with a Bitcoin application running on our campus being proxied to the host in Ireland. However, unlike the controlled iperf3 traffic in the previous case, real Bitcoin nodes from around the world—not controlled by us—connect to our application (ethical considerations are discussed in §A). We identify the nodes connecting from Europe using geolocation and install rules to monitor them. Then, we launch the interception attack ethically from Amsterdam. We set the absolute threshold to 135 ms based on our calculations and the surge threshold to 6 ms. Figure 12b shows the effect of the attack on RTT samples followed by the detection and mitigation.

9 Discussion

Faster data transmission. Certain networks, like free-space communication (e.g., microwave links) and satellite systems, can transmit data faster than speed of light in optical fiber. While not covered here, free-space networks are typically short-range and therefore irrelevant to long-distance rerouting, and satellite ASes can be easily identified by looking up their prefixes and excluding from our coverage.

Non-TCP traffic: We do not need to explicitly monitor non-TCP flows (e.g., RTP or QUIC-over-UDP) to protect them from BGP interception attacks, provided there is at least one TCP flow within the same prefix generating RTT samples. These TCP flows, which share the same network path as the non-TCP flows, provide RTT measurements that can be used to detect potential attacks affecting the entire prefix.

10 Related Work

Control-plane based detection: Control-plane approaches detect BGP hijacks by monitoring route advertisements [25, 37, 38], but they are slow, with BGP convergence taking up to 30 seconds [22, 23]. These methods can also be evaded by limiting route advertisements or targeting victims using BGP community manipulation [12].

Detection via active probing: Active probing methods use tools like ping, traceroute, and nmap to detect BGP hijacks. For example, iSPY detects hijacks in near real time by analyzing traceroutes from multiple vantage points [41]. However, these approaches are vulnerable to surgical attacks [12] and incur higher overhead compared to passive methods.

Detection via passive measurements: Passive measurements, such as RTT, have been used for BGP anomaly detection. Hiran et al. utilized crowd-sourced RTT data to detect attacks [21], though their method only addresses detection, not mitigation. Oscilloscope [14] offers advanced hijack detection but relies on emulated data, suffers from high false-positive and false-negative rates, and lacks data plane implementation, limiting its scalability.

11 Conclusion

We present *HiDe*, a system that detects and mitigates long-distance BGP interception attacks—where an adversary in another country hijacks traffic through its own infrastructure to eavesdrop before forwarding it to the victim. By leveraging propagation-delay measurements that attackers cannot conceal, *HiDe* delivers high-accuracy defense at line rate on a programmable switch (Tbps). Our analysis of worst-case attacks across 258 countries confirms its effectiveness, and we validate *HiDe* through simulations on anonymized campus traces and ethically conducted real-world hijacks, achieving robust mitigation with low false-positive rates.

References

[1] [n. d.]. KlaySwap crypto users lose funds after BGP hijack. https://therecord.media/klayswap-crypto-users-lose-funds-after-bgp-hijack/.

[2] [n. d.]. PEERING: The BGP Testbed. https://peering.ee.columbia.edu/about/.

[3] [n. d.]. Russian-controlled telecom hijacks financial services' Internet traffic. https://arstechnica.com/information-technology/2017/04/russian-controlled-telecom-hijacks-financial-services-internet-traffic/.

[4] [n. d.]. Thousandeyes. https://www.thousandeyes.com/resources/detecting-hijacks-and-leaks-webinar.

[5] Aftab Siddiqui. [n. d.]. Public DNS in Taiwan the latest victim to BGP hijack. https://manrs.org/2019/05/public-dns-in-taiwan-the-latest-victim-to-bgp-hijack/.

[6] Anurag Agrawal and Changhoon Kim. 2020. Intel tofino2–a 12.9 tbps p4-programmable ethernet switch. In *2020 IEEE Hot Chips 32 Symposium (HCS)*. IEEE Computer Society, 1–32.

[7] Andree Toonk. [n. d.]. How Hacking Team Helped Italian Special Operations Group with Routing Hijack. https://bgpmon.net/how-hacking-team-helped-italian-special-operations-group-with-bgp-routing-hijack/.

[8] Rob Austein, Steven Bellovin, Russ Housley, Stephen Kent, Warren Kumari, Doug Montgomery, Chris Morrow, Sandy Murphy, Keyur Patel, John Scudder, Samuel Weiler, Matthew Lepinski, and Kotikalapudi Sriram. 2017. *BGPsec Protocol Specification*. RFC 8205.

[9] bgphijack [n. d.]. $83k in bitcoins 'stolen' through BGP hijack. https://www.virusbulletin.com/blog/2014/08/83k-bitcoins-stolen-through-bgp-hijack/.

[10] Henry Birge-Lee, Maria Apostolaki, and Jennifer Rexford. 2024. Global BGP Attacks that Evade Route Monitoring. *arXiv preprint arXiv:2408.09622* (2024).

[11] Henry Birge-Lee, Yixin Sun, Anne Edmundson, Jennifer Rexford, and Prateek Mittal. 2018. Bamboozling certificate authorities with {BGP}. In *27th USENIX Security Symposium (USENIX Security 18)*. 833–849.

[12] Henry Birge-Lee, Liang Wang, Jennifer Rexford, and Prateek Mittal. 2019. Sico: Surgical interception attacks by manipulating BGP communities. In *ACM SIGSAC Conference on Computer and Communications Security*. 431–448.

[13] Mihai Budiu and Chris Dodd. 2017. The p416 programming language. *ACM SIGOPS Operating Systems Review* 51, 1 (2017), 5–14.

[14] Tobias Bühler, Alexandros Milolidakis, Romain Jacob, Marco Chiesa, Stefano Vissicchio, and Laurent Vanbever. 2023. Oscilloscope: Detecting BGP Hijacks in the Data Plane. *arXiv:2301.12843* (2023).

[15] R. Bush and R. Austein. 2013. *The Resource Public Key Infrastructure (RPKI) to Router Protocol*. RFC 6810.

[16] celer [n. d.]. Celer Bridge incident analysis. https://www.coinbase.com/blog/celer-bridge-incident-analysis.

[17] Xiaoqi Chen, Hyojoon Kim, Javed M Aman, Willie Chang, Mack Lee, and Jennifer Rexford. 2020. Measuring TCP round-trip time in the data plane. In *ACM SIGCOMM Workshop on Secure Programmable Network Infrastructure*. 35–41.

[18] Natural Earth Data. 2011. Natural Earth Data-Free vector and raster map data. *http://www. naturalearthdata. com (accessed 10.12.2024)* (2011).

[19] Sharon Goldberg. 2014. Why is it taking so long to secure internet routing? *Commun. ACM* 57, 10 (2014), 56–63.

[20] Dan Goodin. 2018. Strange snafu misroutes domestic US Internet traffic through China Telecom. *Ars Technica* 6 (2018).

[21] Rahul Hiran, Niklas Carlsson, and Nahid Shahmehri. 2015. Crowd-based detection of routing anomalies on the Internet. In *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 388–396.

[22] Thomas Holterbach, Edgar Costa Molero, Maria Apostolaki, Alberto Dainotti, Stefano Vissicchio, and Laurent Vanbever. 2019. Blink: Fast connectivity recovery entirely in the data plane. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 161–176.

[23] Thomas Holterbach, Stefano Vissicchio, Alberto Dainotti, and Laurent Vanbever. 2017. Swift: Predictive fast reroute. In *ACM SIGCOMM*. 460–473.

[24] Dave Levin, Youndo Lee, Luke Valenta, Zhihao Li, Victoria Lai, Cristian Lumezanu, Neil Spring, and Bobby Bhattacharjee. 2015. Alibi routing. In *ACM SIGCOMM*. 611–624.

[25] Jun Li, Toby Ehrenkranz, and Paul Elliott. 2012. Buddyguard: A buddy system for fast and reliable detection of IP prefix anomalies. In *2012 20th IEEE International Conference on Network Protocols (ICNP)*. IEEE, 1–10.

[26] LLC MaxMind. 2006. GeoIP.

[27] Measurement Lab. (2024-12-01 – 2024-12-05). The M-Lab NDT Data Set. https://measurementlab.net/tests/ndt. Bigquery table `measurement-lab.ndt.download`.

[28] Alexandros Milolidakis, Tobias Bühler, Kunyu Wang, Marco Chiesa, Laurent Vanbever, and Stefano Vissicchio. 2023. On the Effectiveness of BGP Hijackers That Evade Public Route Collectors. *IEEE Access* 11 (2023), 31092–31124.

[29] Alexandros Milolidakis, Tobias Bühler, Kunyu Wang, Marco Chiesa, Laurent Vanbever, and Stefano Vissicchio. 2023. On the Effectiveness of BGP Hijackers That Evade Public Route Collectors. *IEEE Access* 11 (2023), 31092–31124. https://doi.org/10.1109/ACCESS.2023.3261128

[30] Shaun Nichols. [n. d.]. AWS DNS network hijack turns MyEtherWallet into ThievesEtherWallet. https://www.theregister.com/2018/04/24/myetherwallet_dns_hijack/

[31] Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo hashing. *Journal of Algorithms* 51, 2 (2004), 122–144.

[32] Adrian Perrig, Pawel Szalachowski, Raphael M. Reischuk, and Laurent Chuat. 2017. *SCION: A Secure Internet Architecture*. Springer Verlag.

[33] RIPE NCC. [n. d.]. YouTube Hijacking: A RIPE NCC RIS case study. https://www.ripe.net/publications/news/youtube-hijacking-a-ripe-ncc-ris-case-study/.

[34] Andrei Robachevsky. 2019. "Routing security getting better, but no reason to rest!

[35] Brandon Schlinker, Todd Arnold, Italo Cunha, and Ethan Katz-Bassett. 2019. PEERING: Virtualizing BGP at the Edge for Research. In *ACM CoNEXT*. Orlando, FL.

[36] Satadal Sengupta, Hyojoon Kim, and Jennifer Rexford. 2022. Continuous in-network round-trip time monitoring. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 473–485.

[37] Pavlos Sermpezis, Vasileios Kotronis, Petros Gigis, Xenofontas Dimitropoulos, Danilo Cicalese, Alistair King, and Alberto Dainotti. 2018. ARTEMIS: Neutralizing BGP hijacking within a minute. *IEEE/ACM Transactions on Networking* 26, 6 (2018), 2471–2486.

[38] Xingang Shi, Yang Xiang, Zhiliang Wang, Xia Yin, and Jianping Wu. 2012. Detecting prefix hijackings in the internet with argus. In *Proceedings of the 2012 Internet Measurement Conference*. 15–28.

[39] IP Info Team. 2017. IP Info.

[40] Dan York. 2014. BGP Hijacking In Iceland And Belarus Shows Increased Need for BGP Security. https://www.internetsociety.org/blog/2014/02/bgp-hijacking-in-iceland-and-belarus-shows-increased-need-for-bgp-security/.

[41] Zheng Zhang, Ying Zhang, Y Charlie Hu, Z Morley Mao, and Randy Bush. 2008. iSPY: Detecting IP prefix hijacking on my own. In *ACM SIGCOMM*. 327–338.

A Ethics

This research study was reviewed and approved by our Institutional Review Board (IRB). All packet-trace data come from our university network and were anonymized at the point of collection by network engineers who are expressly authorized to handle private data. Anonymization followed the exact procedures laid down by the IRB—anonymizing all IP and MAC addresses, and stripping all payloads. Researchers never had access to any raw or deanonymized data.

To validate *HiDe* 's detection and mitigation capabilities in a live Internet environment, we performed two controlled BGP hijacks using prefixes assigned to us by the PEERING testbed [2]. We temporarily announced these prefixes from our own hosts under testbed guidelines, ensuring no impact on any external networks or clients. All BGP announcements and withdrawals adhered to PEERING's guidelines, and only our own test prefixes were affected.

For our Bitcoin experiments, we operated a dedicated experimental node connected to public Bitcoin clients using Bitcoin's peer-to-peer protocol. We diverted only the traffic destined for our test node, maintaining the protocol's standard multi-peer connectivity to ensure that no client experienced service interruption or security degradation. The increase in induced latency was temporary (under two minutes per peer) and well below the threshold required to compromise transaction privacy or network health.

B Open Science

We will release to the public—in a public GitHub repository—the source code of the *HiDe* Tofino2 application written in P4, the complementary control plane written in Python, and the Jupyter notebooks used to conduct the analysis and evaluation presented in this paper. Additionally, we will also provide the exact steps to setup the *Bitcoin* and *iperf3* experiments used in this paper. For each component, we will also provide detailed documentation on how to install, deploy, and execute it. This work also utilizes *anonymized* campus packet traces for part of the evaluation and analysis. These traces are protected by an Institutional Review Board (IRB) protocol and hence cannot be shared as-is. Instead, we will extract in a CSV file *only the per-flow latency time-series* from these traces with the *flow ID*, *ACK timestamp*, and *RTT* columns. The flow ID will be a unique identifier (e.g., f123) provided to each unique flow having no relation to the actual 5-tuple to preserve strict anonymity. We will release this CSV in the same repository to aid in the reproduction of the results and analysis presented in this paper.

We will publish all our code and materials in a public GitHub repository, including:

- The *HiDe* Tofino2 P4 program;
- The complementary Python control-plane code; and
- The Jupyter notebooks and C++ code used for our analysis and evaluation.

We will also include step-by-step instructions for setting up and running the Bitcoin and iperf3 experiments from the paper, along with full installation, deployment, and execution guides for every component.

Because our evaluation uses IRB-protected, anonymized campus packet traces, we cannot share the raw data. Instead, we will provide a CSV containing only per-flow RTT time series, with three columns: (1) Flow ID: a unique label (e.g., f123) unlinked to any actual 5-tuple, (2) ACK Timestamp, and (3) RTT in ms. This CSV will appear alongside the code so that readers can reproduce our results without compromising privacy.

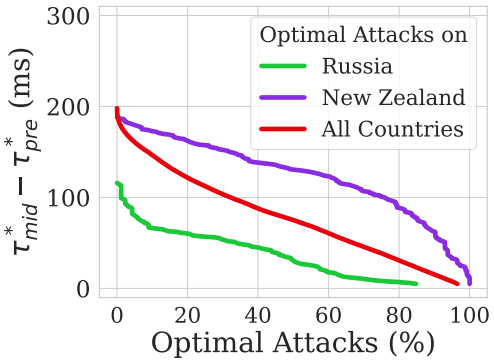


Figure 13: Attack coverage vs. minimum deviation (raw) for Russia, NZ, and all countries: With Russia and New Zealand (NZ) as example victim countries, the figure shows that the highest $\tau^*_{deviation}$ for Russia is 110 ms while it is 190 ms for NZ. For all countries combined, it is 200 ms.

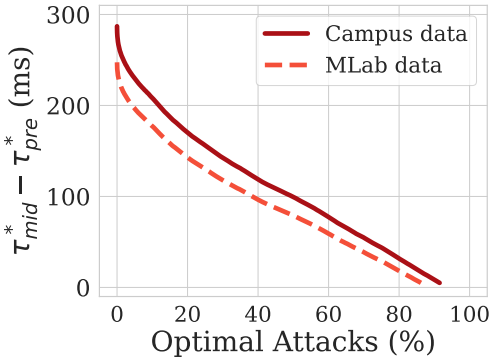


Figure 14: Attack coverage vs. minimum deviation (raw) for campus and MLab datasets.

C Defendability based on speed-of-light RTTs

Figure 13 shows the absolute difference between mid-attack and pre-attack RTTs vs. attack coverage in ideal conditions.

D Defendability based on measured RTTs

Figure 14 shows the absolute difference between mid-attack and pre-attack RTTs vs. attack coverage in real-world conditions. Figures 15 and 16 show the relationship between distance and minimum deviation under optimal attacks in the campus dataset and the MLab dataset, respectively.

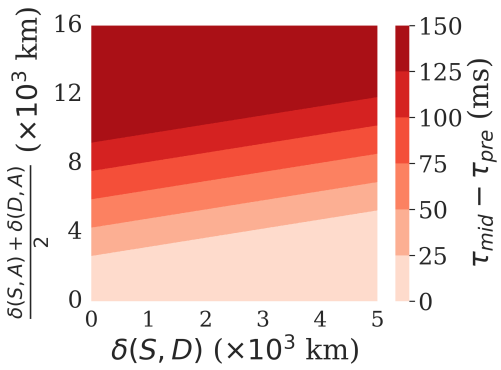


Figure 15: Campus dataset: Relationship between distances and minimum deviation.

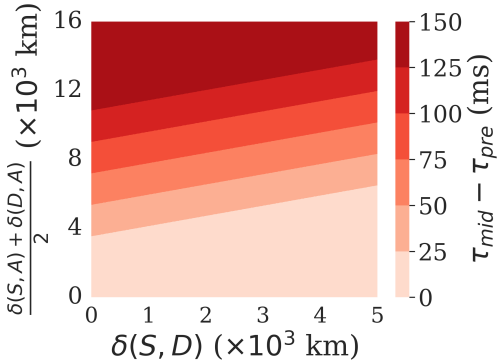


Figure 16: MLab dataset: Relationship between distances and minimum deviation.

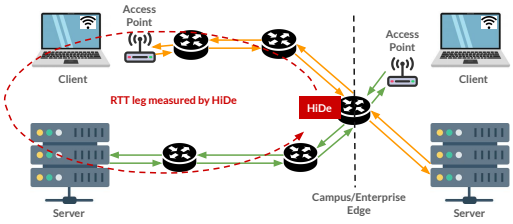


Figure 17: *HiDe*—deployed at the edge of a production network—defends servers and clients inside it by measuring the *external leg* of RTT from itself to external hosts.

E Deployment

HiDe is deployed at the edge of a production network (Figure 17), protecting clients within the network by monitoring the external leg of RTTs (*HiDe* to external hosts) rather than the internal leg (*HiDe* to internal hosts) [36]. We denote connections with clients inside the defended network as *Client-In-Server-Out* (CISO) and connections with servers inside the network as *Server-In-Client-Out* (SICO). We observe that the primary source of noise in RTTs is typically the access link near the client. For CISO connections, the access link is part of the internal leg and does not affect the monitored RTTs, resulting in less noise. In contrast, SICO connections experience higher noise levels, as the access link is external. In our campus data, we apply a TCP port number-based heuristic to distinguish CISO connections from SICO connections: if the port number used by the campus-internal host is < 1024 and the one used by the external host is > 1024, we consider it a SICO connection, and vice-versa.

Resource Type	Compute RTT [36]	Track Min. RTT	Detect Change	Mitigate Attack
Stages	7	2	4	3
TCAM	2.9%	0.0%	1.1%	0.0%
SRAM	4.5%	4.0%	2.4%	3.6%
Instructions	3.6%	2.4%	1.0%	1.1%
Hash Units	35.8%	12.5%	2.8%	5.6%
Input Crossbars	10.1%	3.0%	1.6%	1.9%

Table 1: Hardware resource usage of the Tofino2-based prototype, divided by functional component.