KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Information and Communication Technology

Second cycle, 30 credits

Investigating the Effectiveness of Stealthy Hijacks against Public Route Collectors

Is AS-Path Prepending Enough to Hide from Public Route Collectors?

KUNYU WANG

Stockholm, Sweden, 2023

Investigating the Effectiveness of Stealthy Hijacks against Public Route Collectors

Is AS-Path Prepending Enough to Hide from Public Route Collectors?

KUNYU WANG

Master's Programme, Communication Systems, 120 credits
Date: February 24, 2023

Supervisor: Alexandros Milolidakis
Examiner: Marco Chiesa
School of Electrical Engineering and Computer Science
Swedish title: Undersökning av effektiviteten hos smygande kapningar mot offentliga ruttinsamlare
Swedish subtitle: Är AS-Path Prepending tillräckligt för att dölja från offentliga ruttinsamlare?

Abstract

BGP hijacking is a threat to network organizations because traditional BGP protocols were not designed with security in mind. Currently, research to combat hijacking is being done by detecting hijacking in real time from Public Route Collectors. However, by using AS-Path Prepending, a well-known traffic engineering technique, hijackers could adjust the influence scope of hijacks to potentially avoid Public Route Collectors. This thesis investigates fist, whether AS-Path Prepending is sufficient to hide from Public Route Collector, and second whether the hijacker can predict its hijack's stealthiness by simply comparing the AS path length with the victim. Last, we investigate the non-hijacker-controlled parameters, which are the geographical locations and victim prepending times if the victim also enable AS-Path Prepending for traffic engineering in our study. Our results show that on one hand, AS-Path Prepending benefits stealthy hijacks to route collectors. While on the other hand, it is not sufficient to completely hide from route collectors only using it. By simply comparing the AS paths length, the hijacker's prediction is constructive but not practical. And non-hijacker-controlled parameters indeed can significantly affect the stealthiness of hijacking.

Keywords

BGP, BGP Hijack, Stealthy IP prefix hijacking, AS-Path Prepending, BGP monitoring

Sammanfattning

BGP-kapning är ett hot mot nätverksorganisationer eftersom traditionella BGP-protokoll inte har utformats med säkerheten i åtanke. För närvarande bedrivs forskning för att bekämpa kapning genom att upptäcka kapning i realtid från offentliga ruttinsamlare. Genom att använda AS-Path Prepending, en välkänd trafikteknik, kan kapare dock justera kapningarnas inflytande för att eventuellt undvika offentliga ruttinsamlare. I den här avhandlingen undersöks för det första om AS-Path Prepending är tillräckligt för att dölja sig för Public Route Collector och för det andra om kaparen kan förutsäga hur smygande kapningen är genom att helt enkelt jämföra AS Path-längden med offrets. Slutligen undersöker vi de parametrar som inte kontrolleras av kaparen, dvs. geografiska platser och offrets prependingtider om offret också aktiverar AS-Path Prepending för trafikteknik i vår studie. Våra resultat visar att AS-Path Prepending å ena sidan gynnar smygande kapningar av ruttinsamlare. Å andra sidan räcker det inte för att helt och hållet dölja sig för ruttinsamlare om man bara använder det. Genom att helt enkelt jämföra AS-vägarnas längd är kaparens förutsägelser konstruktiva men inte praktiska. Parametrar som inte kontrolleras av kaparen kan faktiskt påverka kapningens smygande på ett betydande sätt.

Nyckelord

BGP, BGP Hijack, Stealthy IP prefix hijacking, AS-Path Prepending, BGP-övervakning

Acknowledgments

I would like to express my gratitude to my examiner Marco Chiesa for the opportunity to work on this interesting project. Also I would thank my supervisor Alexandros Milolidakis for his patient guidance and valuable comments on my thesis.

I would also like to thank my opponent Yulian Luo for her written review, and my friends and family for their support.

Additional thanks to PEERING for their experiment approval, and thanks to the researchers and developers who care about Internet security and are passionate about open source.

Stockholm, February 2023
Kunyu Wang

Contents

List of Figures

List of Tables

Listings

List of acronyms and abbreviations

AS	Autonomous System
ASN	Autonomous System Number
ASPP	AS-Path Prepending
BGP	Border Gateway Protocol
BMP	BGP Monitoring Protocol
CDN	Content Delivery Network
FPR	False Positive Rate
IP	Internet Protocol
ITE	Inbound Traffic Engineering
MED	Multi Exit Discriminator
P2P	Peer-to-Peer
PKI	Public Key Infrastructure
PRC	Public Route Collector
RIB	Routing Information Base
ROC	Receiver Operating Characteristic
ROV	Route Origin Validation
RPKI	Resource Public Key Infrastructure
TPR	True Positive Rate
VP	Vantage Point

Chapter 1

Introduction

On the Internet, there are currently thousands of Autonomous Systems (ASs) that are interconnected and constantly exchange traffic. Between the various ASs, the Border Gateway Protocol (BGP) is used to advertise IP prefixes and so construct inter-domain routes. Each AS is able to announce its own prefixes to its neighbors, as well as routes to the prefixes of other connected networks. However, BGP is not a protocol designed with security in mind; thus, any AS may accidentally or deliberately advertise bogus routes (*i.e.,* non-existent routes) for any prefix. Such attacks, known as BGP hijacking, pose a potential threat to network security. Other ASes in the Internet would mistake the hijacker as the victim, or follow a bogus path to the victim, so the traffic from affected ASs is redirected towards the hijacker.

Currently, to defend against BGP hijacking, networks usually use reactive mechanisms [1]. Those mechanisms consist of two steps: detection and mitigation. Mitigation is to use some methods to reduce the impact and damage caused by hijacking. Before taking steps on mitigation, detection is essential which mainly relies on some third-party distributed data sources. These data sources usually rely on route collectors, which collects the information distributed by multiple monitors on other networks. This infrastructure of Public Route Collector (PRC) is a collection of multiple distributed route collectors that allow public access to the Internet routes for prefixes they collect. Hijacked victims can retrieve the data of their own prefixes from these collectors, detect the presence of a hijack, and produce alert notifications. Therefore, public collectors can, in practice, notice changes in BGP routes and detect hijacking if the monitors record the hijacked routes. However, hijackers focusing on counter-reconnaissance may carefully design the attack to avoid being caught by the public collectors. After all, these data

sources are also visible to the hijackers. By observing the routes collected by public collectors for their own legitimate prefixes, hijackers may be able to evaluate the visibility of their hijacks.

In this thesis, we investigate how effective route collectors can be in observing BGP hijacks from the routes that their monitors' report. Meanwhile, we also illustrate the viability of developing stealthy hijacks that can conceal themselves from PRC. The Peering Testbed [2] will be used in this thesis to ethically announce hijacks so that results from the real Internet are obtained.

1.1 Background

The BGP protocol itself lacks authentication and security mechanisms. BGP hijacking occurs when any AS announces bogus routes for any IP prefix that it does not own, therefore stealing the traffic destined for that prefix. As a result of such a BGP hijacking, attackers can sniff the traffic from hijacked destinations, use it to phish, or even simply drop the traffic.

Numerous hijackings have occurred in the real world. For example, on 1st April 2020, AS12389 in Russia announced an unusual Content Delivery Network (CDN) prefix that belongs to Facebook. And as a result, more than 8,000 CDN prefixes have been affected and traffic has been directed to Russia. Although the incident lasted for 5 minutes, many CDN including Facebook, Google, Amazon, Line, and Cloudflare were affected [3].

The fundamental cause of such problems is because the original BGP protocol does not guarantee Route Origin Validation (ROV). Various research approaches and techniques have been proposed to combat hijackers. Some proactive designs has been implemented to ensure the trustworthiness of BGP updates by means of cryptography [4]. Furthermore, some enhanced protocols such as Resource Public Key Infrastructure (RPKI)[5] have also been proposed to improve security. However, not all ASs have adopted these recommendations, possibly due to the complexity of these procedures [6]. As of September 2022, less than 40% of the prefixes are RPKI protected.[7]

Therefore, the main method of defending BGP is still reactive detection and mitigation. Hijackers may employ interdomain traffic engineering techniques, such as AS path prepending or AS path poisoning, to limit the scope of hijacking breadth among ASs, and therefore potentially escape collector surveillance. In such circumstances, it is important to investigate the real-world effectiveness of collector monitoring and to investigate whether it is or not possible for hijackers to build such stealthy hijacks.

1.2 Problem

Third-party PRCs play a significant role in the detection of potential BGP hijackings. Collectors record information from monitors and disclose it to some public platform at specific intervals for research. However, according to the work of Enrico et al.[8] public route collectors have a relatively constrained and skewed vision of the Internet topology in BGP. The top-down collection structure is constrained by the relationships between peers, and the number of routes collected is limited across the complex real Internet. As a result, the messages collected are more indicative of the network topology as seen by some of the biggest ISPs in the Internet, and it is hard to display the substantial number of Peer-to-Peer (P2P) links at the bottom. Geographical factors also play a role in this limitation. For example, P2P links in Africa are totally invisible for PRC as of 2012 [8]. All these facts mean that the route collectors face more complex and variable situations in a real network than it does in a simulation, which can lead to different results.

To make matters worse in hijacking, hijackers can adjust the affected area to some extent by modifying the hijacks they announce. Since hijackers also have access to these PRCs, they can observe their relative position to the victim. Therefore, the visibility of hijacking can be inferred to some extent, which helps hijackers to better implement stealthy attacks.

In particular, as described above and in Section 2.3, *PRC are commonly not able to collect all BGP messages. PRCs are accessible to hijackers. And hijackers could adjust the routes that route collectors observe using traffic engineering methods, such as AS-Path Prepending (ASPP), AS-path poisoning, etc.* These three facts could make it possible to design stealthy hijacks in the real Internet that escape from PRCs.

1.3 Purpose

The purpose of this thesis is to evaluate the ability of hijackers to improve the stealthiness of hijacking by some methods in order to hide from the PRC. The stealthiness described here implies visibility to PRC, which can be quantified as the number of monitors observed hijacking collected by PRC.

Since we built a standard experiment for the BGP hijacking test on a real network, our research is likely to be of interest to researchers interested in the visibility of BGP hijacks. We restricted the visibility of hijacking to the collector's field of view, so this study may also be useful for researchers and

operators of PRC. In addition, we tested the effect of different geographical locations of the hijacker and the victim on the collector's field of view, which may shed light on the placement of new monitors or route collectors.

1.4 Goals

The basic goal of the project is to evaluate the capability of hijackers to avoid PRC. Past work by Milolidakis A. [9] used Traffic Engineering methods, such as AS-path prepending and AS-path poisoning, to evaluate the capability of hijackers to avoid PRC in simulations. In this thesis, we focus in the real world to understand how capable hijackers would be to avoid PRC using AS-path prepending methods. To answer this goal, we divide it into the following three subgoals.

1. Verify how helpful ASPP is in generating stealthy attacks. As mentioned in Section 2.4.2, the hijacker can increase the AS-Path in its own hijacking message through ASPP, theoretically causing the hijacking to become less-preferred to some other ASes. The longer the AS-path length, the stealthier the hijack should be. The first goal is to demonstrate that the hijacker is capable of narrowing the observable range of the hijack via ASPP.

2. Verify the accuracy of a simple algorithm used to infer the stealthiness of the hijack prior to its announcement. As mentioned in Section 2.5.2, the PRC data are publicly available. This means that the hijackers are able to observe both their own and the victims' BGP prefix propagation on the Internet. Theoretically, the hijacker could be able to infer whether monitors would report or not a planned hijack.

3. Verify the impact of non hijack-controlled parameters on the stealthiness of hijacking. There are many parameters beyond the hijackers' control that may influence the impact and stealthiness of an attack, such as the proximity of the hijacker to the victim and to the monitors, the geographical location of the hijacker and the victim, and whether or not the victim prepends its prefixes.

   As described in Section 2.4.1, there are many ASes that themselves use ASPP as a means of traffic engineering. The victim's use of ASPP to extend the length of its own AS path is beyond the hijacker's control. Additionally, the geographical location of the hijacker and the victim

affects the overall visibility of the hijack. Theoretically, hijackers closer to the monitor are more likely to be exposed. Therefore, in this thesis, we also consider as an important sub-goal to study the stealthiness impact of these “meta-parameters" that are beyond the control of the hijacker.

1.5 Research Methodology

The research done in this thesis is accomplished using a mix of qualitative and quantitative research methods. Qualitative methods are mainly used for sub-goal 1 and sub-goal 2. Through qualitative comparisons, we can mainly identify a trend and determine whether the relevant parameters have an impact on the final result. Quantitative methods are mainly used for sub-goal 1 and sub-goal 2. The specific number of monitors in the results allows us to judge the performance of each experiment set and the accuracy and the feasibility of the designed algorithm.

1.6 Delimitations

We study the role of PRC in BGP hijacking in real networks, and the possibility of improving stealthiness. Therefore, our study is limited to the field of view of collectors. Furthermore, we focus on the range of hijacking reported by PRC instead of hijacking mitigation strategies performed by the victims.

To verify the stealthiness of the hijacking, our hijackings are limited to exact-prefix hijacking, which is described in Section 6.2. Meanwhile, we only care about the hijack in the control plane instead of the intent of the hijacker in the data plane. The purpose of the hijacker, *i.e.,* traffic black holes, man-in-the-middle attacks, etc, are outside the scope of our evaluation.

BGP Monitoring Protocol (BMP) [10] is a recent enhancement to the BGP protocol that enables an AS to pass its entire Adj-RIB-In table to route collectors, rather than just the best route in the Loc-RIB table. This new protocol improves the ability to look for BGP hijacking, but we consider that the monitors’ messages collected do not support BMP in the design.

1.7 Structure of the thesis

Chapter 2 presents relevant background information on BGP hijacking and ASPP. Chapter 3 presents the methodology and method used to solve the problem. Chapter 4 presents some technical details and data analysis in the

experiments. Chapter 5 shows our results and gives our interpretation through discussion. Chapter 6 summarizes the whole text, gives our conclusions, and suggests some directions for future work.

Chapter 2

Background

This chapter provides a overview of some basic background about BGP selection , BGP hijacking and AS-path prepending. In addition, this chapter presents some relevant platform we use in this project. Section 2.1 describes BGP path selection and why the hijackers could hide from monitors through this process. Section 2.2 describes the BGP hijacking classification in detail and which type of hijacking should be if the hijacker wants to build up a stealthy hijack. Section 2.3 describes what is PRC and how the monitors work with PRC. Section 2.4 describes what is ASPP and how to use ASPP in traffic engineering and hijacking. Section 2.5 describes the platform we use to experiment and fetch data. Section 2.6 describes some related work against BGP hijacking.

2.1 BGP Route Delivery Process

BGP is one of the primary decentralized autonomous routing protocols of the Internet. It utilizes a rule set based on paths and each AS's network policies to establish routes, hence enabling inter-AS connectivity. According to the current BGP protocol [11], a BGP speaker stores routing information received from inbound neighbors into the Adj-RIB-In table, and then selects the ideal route to be stored in the Loc-RIB table following the local BGP speaker's decision process. Through the Adj-RIB-Out table, the routing information in the Loc-RIB can then be transmitted to outbound peers. The main process can be shown in Figure 2.1.

BGP requires each AS to inform its peers of only of the optimal (best) path it knows of each prefix. This allows hijackers to remain hidden.

There are several rules for BGP selection and are prioritized in a specific

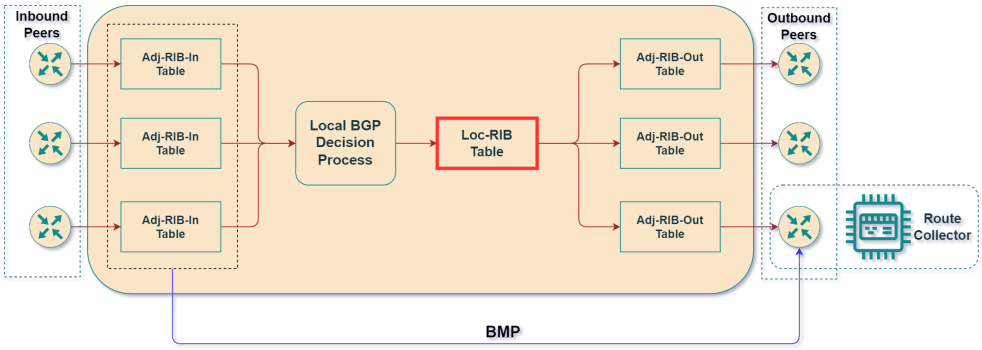


Figure 2.1: Example of BGP Route Delivery Process

order. Path selection for BGP tables generally occurs in the following order. [11]

1. highest local-preference value
2. shortest AS-path length
3. lowest origin value
4. lowest MED value
5. routes from EBGP over IBGP
6. best exit from AS
7. routes from peers with lowest Router ID

The hijacker can simply modifies its message's attribute to make it more enticing in this selection. Among these attributes, changing the second and fourth rules are the most direct and efficient methods, since they can be controlled by BGP speakers. While the MED value is a non-transitive optional attribute, we focus primarily on hijacking via the AS path.

Each BGP message must carry the AS-path attribute, and by actively changing it, hijackers could win the BGP selection against victim in some ASes. However, to hide from the collectors, what hijackers need is to lose the selection for some specific ASes, at least, before propagating to the monitors. The hijacker can change the length and content of AS path to lose this selection. For example, by prolonging the length of the advertised AS-path so that it is eliminated by the BGP selection law prior to being transmitted to the collector. An ideal scenario is that the propagation range of the hijacked route

can be controlled and predicted by adjusting the length of the AS path. In actual networks, however, the path selection laws within each AS are trade secrets and typically do not fully comply with the selection provisions of the protocol. According to the Gao-Rexford model [12], an AS's BGP strategy may prioritize commercial profitability over network connectivity. Thus, many ASs in the network will prioritize routing information from customers who pay him over information from P2P neighbors, which could be more useful but free. And this option is merely a conceivable desire, not a strict rule. As a result of these nontechnical factors, the effect of AS-path prepending on real networks can differ from its simulation performance, making it hard for a hijacker to forecast and organize a flawless hijack.

Consequently, and as explained in Section 1.6, we don't consider the BMP in our design. Since requiring all ASs in the actual network to pass the Adj-RIB-In table is impractical and goes against the original intent of the BGP protocol design. The hijacker has no way of knowing which ASes on the network have BMP turned on. Furthermore, if the hijacker considers that all monitors in PRC enable BMP, the difficulty can be shifted from deceiving collectors to deceiving their monitors, which has little effect on the hijacker's structure. Therefore, we do not consider the collector's peer to have enabled BMP in the current experiments.

2.2 BGP Hijacking Classification

BGP hijacking occurs when a hijacker redirects network traffic in the wrong direction by manipulating BGP messages. Typically, a hijacker accomplishes this by falsely claiming ownership of a hijacked Internet Protocol (IP) prefix which it does not own or route to. Because the widely used BGP protocol lacks a mechanism to verify the legitimacy of messages, BGP hijacking for network control is possible and difficult to manually prevent.

According to Sermpezis et al., BGP hijacking can be modeled into three dimensions which are the affected prefix, the AS path, and the data plane levels [1]. To better understand, let's consider such a scheme shown in Figure 2.2. We assume that victim possesses prefix $10.0.0.0/23$ with Autonomous System Number (ASN) *V*, while hijacker legally administers $10.1.0.0/23$ with ASN *H*. We use *{AS-path - Prefix}* to represent the information in a BGP message that is valuable to us. The current valid path to AS V from the standpoint of AS *Z* should be $\{C, B, A, V - 10.0.0.0/23\}$, where C, B, A, and V represent the ASs on the path for Z to reach V.

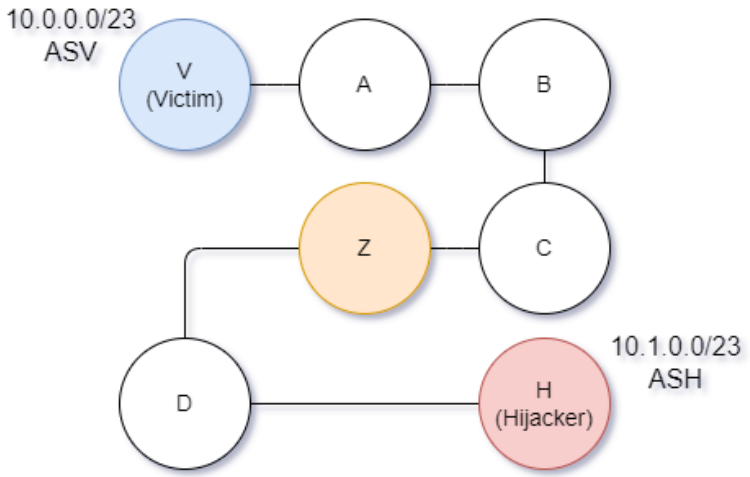


Figure 2.2: Example of a BGP hijack where the nodes V, A, B, C, Z, D, and H represent different ASes.

2.2.1 Affected Prefix

Based on the announced prefix of the hijacker, hijacking can be categorized as either exact-prefix hijacking or sub-prefix hijacking.

Exact-prefix hijacking: In this case, the hijacker announces the exact same prefix as announced by the victim. This means that the hijacker's announcement has to compete with the victim's prefix for AS-path length in the network. Therefore, the Internet will be separated between regions affected either by the hijacking or that remain unaffected continuing to forward traffic to the victim's prefix.

Sub-prefix hijacking: Hijackers can declare a smaller range of IP addresses than the victim to achieve hijacking. For example, hijacker H can announce a message like $\{H - 10.0.0.1/24\}$ while the victim announces $\{V - 10.0.0.0/23\}$. In such cases, the entire Internet will prefer the hijacker's choice, since BGP favors a more specific prefix.

2.2.2 Announced AS-path

Hijacking can also be classified by the way the hijacker manipulates the AS path that it announces to the rest of the Internet. N below denotes the number of hops required to reach the victim through this phony path.

Type-N: $N > 0$ signifies that a hijacker trys to insert its own AS into a path to the victim. This means that the hijacker is prepending to be one participant

in a fake path and then malforms the AS-path. $\{ASh, ASd, ASa, ASv - 10.0.0.0/23\}$ is an example of type-3 hijacking, whereas $\{ASh, ASv - 10.0.0.0/23\}$ is an example of type-1 hijacking. In both false routes, the hijacker is simulated to reach the victim.

Type-0: $N = 0$ signifies that the hijacker has posed as the prefix owner, which means that the hijacker exists in the last portion of the AS path. As an illustration, $ASh - 10.0.0.0/23$ is a type-0 hijacking.

Since our experiment seeks to conceal hijacking as much as possible using the AS-path prepending and is constrained by platform limits, we explore the case where N is between 1 and 4.

2.2.3 Data plane

BGP hijacking can also be classify according to the way the hijacker diverts the hijacked traffic. The primary goal of traffic diversion is to steer traffic to the hijacker. After acquiring the traffic, a hijacker may operate as an intermediary to eavesdrop, corrupt existing connections, or establish new connections. It is also feasible to simply drop traffic and create a black hole for traffic.

2.2.4 Methods against BGP Hijacking

As we described in Section 1.1, some researches trying to block hijacking proactively like RPKI [5] and some cryptographic methods [4] are facing with issue to fully deploy into all the ASes on the Internet.

Therefore, the traditional procedure is to first identify and then mitigate hijacking. Various BGP hijacking analysis algorithms and protocol enhancement have been presented. Some examine the Routing Information Base (RIB) history they have collected to recognize fraudulent routing paths [13]; while others enhance the BGP protocol to ensure that BGP messages are trustworthy [14]. However, these solutions which require large-scale deployment to be effective, are also challenging apply in practice as it requires the global coordination of multiple ASes to effectively stop hijacks from propagating.

Another idea is to detect hijacking using path-related algorithms [15] or by analyzing large amounts of data using machine learning [16]. As not all ASs can deploy a large number of probes throughout their networks, data sources represent a significant obstacle for these systems. A third-party data collector becomes an indispensable service.

Also, there are some other issues, such as false positives in the actual

Internet. Due to network engineering and other factors, the routing information of each AS is non-uniform, and the traffic design within each AS is considered confidential internal information. For these reasons, our investigations are limited to a situation in which a AS collects public routing information from a third-party collector to identify whether its own prefixes have been hijacked.

2.3 PRC and Monitors

In this section we introduce the PRC and the structure of PRC to collect BGP information from monitors.

Whether for monitoring, troubleshooting, or academic research, it is necessary to collect BGP routes in the network. An effective way to do this is through BGP looking glasses, where users can access the current state of a router's RIB through the looking glass interface provided by the AS. However, this approach from the command line is not suitable for large systematic collection and is more suitable as an interactive way of obtaining some targeted data. Another way is through some protocols designed for monitoring, such as BMP [10], which allows the user to obtain the Adj-RIB-In Table inside the router and thus monitor its peering sessions. Unfortunately, as far as we currently know, there are no major projects involving BMP that are open to the public.

PRC is another method of collecting BGP routing data. There are many mature technical implementations and public projects involving PRC. A route collector is a host in the network that has a process such as Quagga [17] running to collect routing information. By emulating a virtual router, the PRC establishes peering sessions with real routers on the network. A PRC may establish connections with multiple peers, which are generally called Vantage Points (VPs) [18]. As shown in Figure 2.3, the PRC implements the collection of BGP routes by aggregating the Adj-RIB-Out tables of the many VPs collected.

Unfortunately, PRCs are commonly unable to collect all the routing information of the VPs they are peering with. Some VPs are able to pass all their routing information in Loc-RIB to the PRC by establishing a customer-provider relationship with the PRC. However, some VPs do not provide transit service, limiting the PRC to access only some of the information on their Loc-RIB. As establishing a connection with the PRC and providing routing information is usually voluntary, PRC do not enforce VPs to provide their transit routes neither they actively analyze how this information changes over

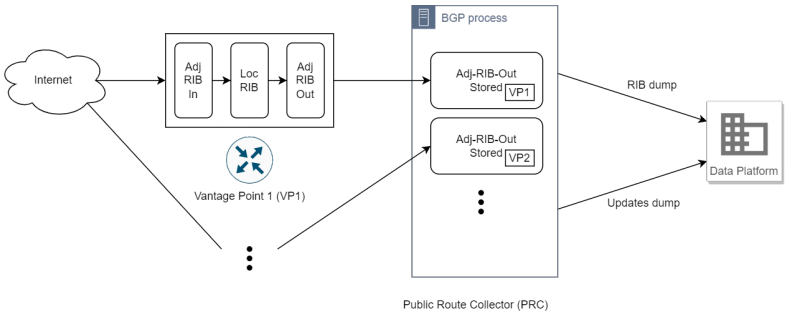


Figure 2.3: PRC collects BGP information from peering VPs and output two formats of data

time – although researchers could document and infer this information from the collected routes.

PRC will periodically provide two data collection formats. Every few hours, a currently retained federated snapshot of Adj-RIB-Out tables fetched from all VPs is dumped, called an RIB dump. Every few minutes, a federated union of update messages is fetched from all VPs since the last dump, called an update dump. The RIB dump provides a coarse-time granularity analysis tool, while the Updates dump makes it possible to analyze routing changes in a short period of time.

In our project, the focus on whether VPs can report or not a hijacking to PRC, so we call these VPs as **monitors** in the thesis. Also, we use the Update dumps rather than the RIB dumps, which is limited by the time span of our experiments. A more specific discussion of the choice between the two will be given in Section 3.3.

2.4 AS-Path Prepending

Inbound Traffic Engineering (ITE) is a technique for optimizing traffic and maximizing benefits by modifying stated routing information. Numerous ASs that receive more traffic than they send frequently employ ITE to affect the link selection of their incoming traffic. Among the numerous ITE techniques, such as BGP communities and Multi Exit Discriminator (MED) selection, ASPP is one of the most straightforward to use. Before announcing or passing the BGP message to its neighbor, AS can manually increase its own ASN for n times to regulate traffic. This n is *prepend-size* ($n \in \mathbb{N}^+$). ASPP as a means of traffic engineering can control the distribution of traffic, yet the hijacker can

also use it to control the scope of the hijack, thus increasing the stealth of the hijack. Below we first discuss how ASPP simply works as traffic engineering to control the distribution of traffic, and then describe how it helps to hide for hijackers.

2.4.1 ASPP in Traffic Engineering

Figure 2.4 shows a basic schematic of ASPP. In the illustration, it is assumed that each AS prioritizes the shortest AS path over other commercial factors. In this case, AS *E* has two neighbors, AS *B* and AS *E*. The ASs in the blue region will spread its traffic along the blue path, while the ASs in the orange goes along the orange path. Since the orange path is shorter than the blue path, if no ASPP is deployed, the traffic from AS *G* will propagate down the orange path to AS *A*. However, if AS *A* deploys ASPP as depicted in the image of *G*, the blue path is logically shorter and *G* will be divided into the blue zone.

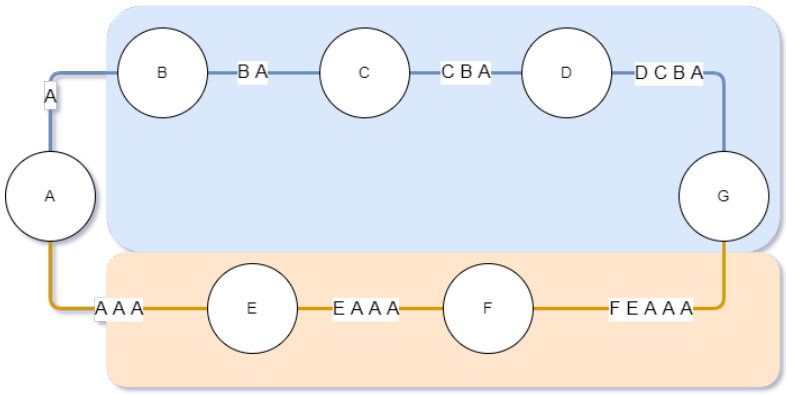


Figure 2.4: A simple ASPP schematic

Since each AS can have multiple different peers, ASPP can be divided into three categories using ASPP strategies per AS. According to Pedro et al. [19], for a prefix-origin pair, we can divide ASPP into three categories: *no-prepend*, *uniform* and *diverse*. If there is no prepending happening, we say it is *no-prepend*. If this AS deploys the same policy for all peers it has, we consider it as *uniform*. And if for different peers the prepending is different, we define it as *diverse*. In Figure 2.4, AS *A* doesn't prepend for AS *B*, and it prepends twice for AS *E*. Therefore, it is a *diverse* strategy in this case.

ASPP is common phenomenon in the Internet. The research by Pedro et al. [19] found that nearly 30% of AS and a quarter of prefixes have ASPP

enabled. Among these ASes, the use of policies has been stable and more prepending occurs during COVID-19. Furthermore, contrary to intuition, a number of ASes use the *uniform* strategy, although it is not able to make their neighbors different.

The function of ASPP is to be able to adjust the area responsible for each link to achieve an ideal traffic distribution. Although the effectiveness of ASPP is difficult to be evaluated and depends largely on the location of AS, in the majority circumstances ASPP can move traffic more efficiently. This has made this simple technology widespread. Notice that ASPP is a method to adjusting the weight of traffic on different paths. This method can be used to balance links in the absence of a hijacker. However, this may present a chance for the hijacker to take advantage of it if ASPP is present. Apparently, ASPP regulates traffic distribution by making some of its routes "weaker" in the competition, which in turn could make it easier for the hijacker's fake routes to win.

2.4.2 ASPP in Hijacking

ASPP is a means of the weakening routes, which causes a proportion of routes to not be preferred by BGP's best path selection. However, hijackers can also use it to weaken their hijacking if they want to increase the stealthiness of the hijacking. We can represent this process more visually by constructing the topology as a tree. Figure 2.5 shows a slightly more complex topology. Victim AS V adopts *diverse* strategy for ASPP. Fake nodes can be used to represent the prepend size for different neighbors. In this case, we can clearly see that AS E has chosen the yellow route over the closer blue route because the fake nodes have extended its path.

However, when the hijacker is involved by announcing the attack, the topology, in fact, takes the shape of a spindle. Figure 2.6 shows if hijacker AS H announces a simple type-1 hijacking for all its neighbors. We can see that in this case, AS F and AS G are hijacked by the hijacker, while AS E and AS H face paths that have the same length as the hijacker and the victim. However, this is only the most basic type-1 hijacking. If the hijacker decides to use ASPP in Figure 2.7, the scope of the hijacker's hijack becomes smaller. Due to the longer path announced with the peer AS G, AS H will choose the shorter route to the victim. If AS H is a monitor (*i.e.,* a VP) of PRC, ASPP will then help the hijacker increase its stealthiness.

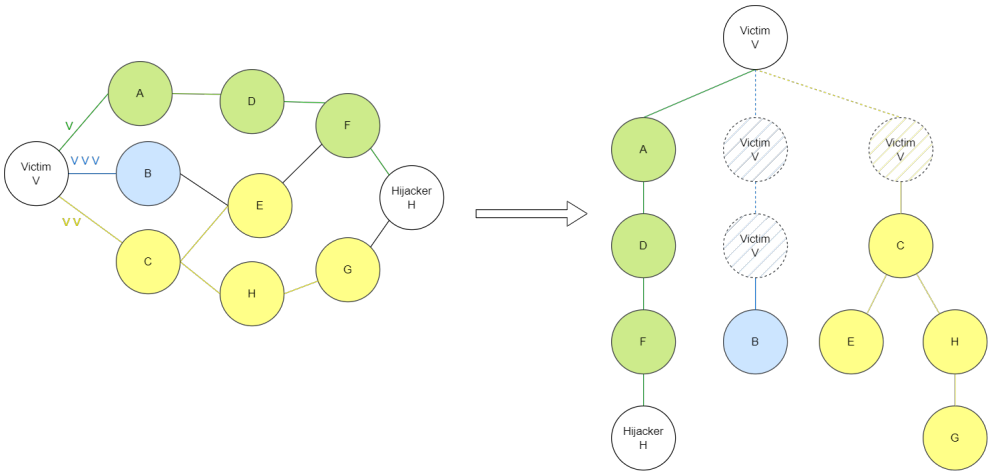


Figure 2.5: A simple BGP topology tree

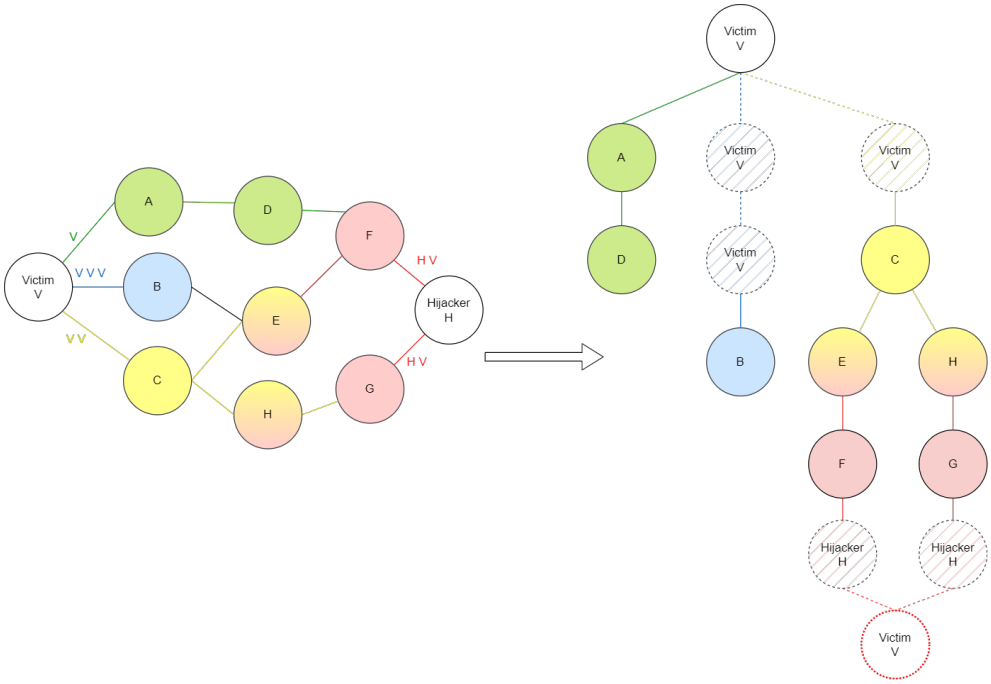


Figure 2.6: A simple BGP topology tree with simple type-1 hijacking

2.5 PEERING and BGPStream

Throughout the experiment, we need an experimental platform that could provide simulated hijackers and victims in a real network. Furthermore,

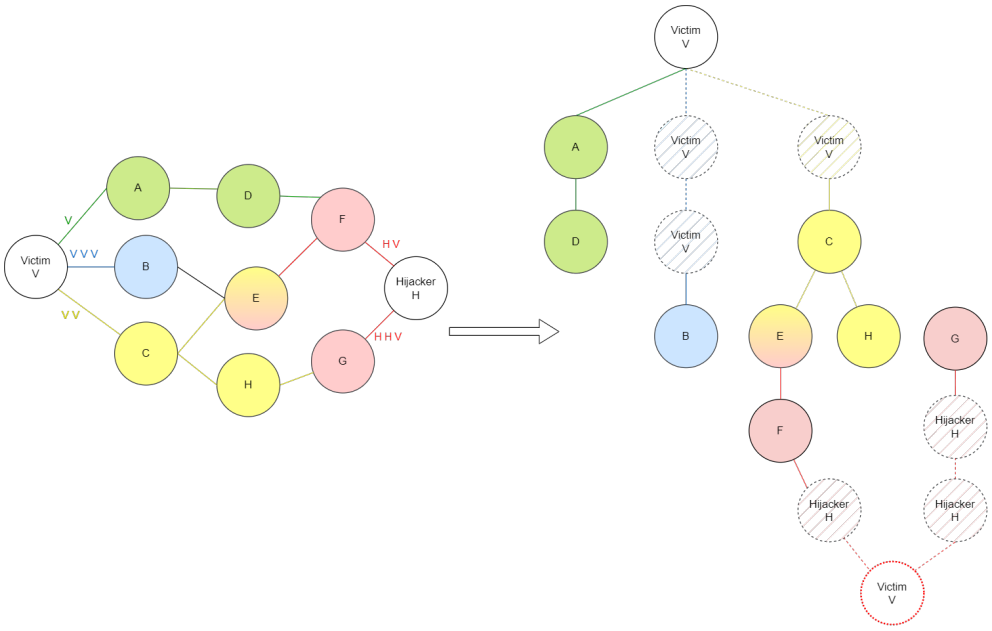


Figure 2.7: A simple BGP topology tree with hijacking using ASPP

we need data platforms that can access PRC information. In this project, the PEERING testbed [2] is used to simulate victims and hijackers, and the BGPStream [20] is used to obtain BGP messages in the real network.

2.5.1 PEERING Testbed

PEERING is a community platform built on top of vBGP [21]. vBGP is a framework for virtualizing the control-plane and the data-plane. On this platform, the operation is equivalent to operating AS directly. It allows several parallel experiments, which applies to our need to simulate both the victim and the hijacker at the same time. In addition, it offers more than one ASN and prefixes for free combinations, which provides convenience for experimentation.

It officially supports some ITE and hijacking operations. Operators can declare different messages to different peers simply via the command line and can also experiment with ITE using ASPP. In addition, some BGP hijacking methods can be enabled such as BGP poisoning. The platform officially supports customizing the BGP messages sent by an AS by loading configuration files. PEERING also provides different locations as BGP upstream, making it easy to evaluate the impact of the hijacker and the victim

at different locations on the network.

2.5.2 BGPStream

BGPStream is a software framework to analyze historical and real-time BGP messages [20]. The focus of BGPStream is to help operators build complex network monitoring applications, such as monitoring network outages and BGP hijacking, by providing a unified API to handle large amounts of historical and real-time data [18]. It takes information from data sources and provides a unified tool for users to access information from these data sources. There are two popular projects which provide data for BGPStream which are RouteViews [22] and RIPE RIS [23]. These data sources provide over 900 monitors, and the number is still increasing.

Just as we introduced in Section 2.3, there are two data formats provided: *RIB dump* and *Updates dump*. *RIB dumps* provides a general summary of the changes in the BGP routing table over a long period of time, and *Updates dump* gives a complete illustration of all the changes in the BGP tables over a short period of time. Therefore, *Updates dumps* is well suited to analyze the visibility of hijacking at a small granularity.

2.6 Related Work

This section introduces some related work including research in BGP security, hijacking countermeasures, and more. The first is to enhance the security aspects of BGP. There has been a long history of work to improve BGP security, and there are many achievements and measures in place. For example, adding sequence numbers to BGP messages [24], adding authentication to BGP messages [25], and implementing encryption of BGP messages between peers [26]. Stephen Kent *et al.* [4] built on these studies by combining techniques such as Public Key Infrastructure (PKI) and IPsec to construct an S-BGP architecture that addresses most of the vulnerabilities.

Second, continuous monitoring of BGP data is important to detect hijacking. Some researchers wanted to gain continuous access to the router's BGP RIBs. BMP protocol [10] was thus born. With this protocol, users can monitor or dump Adj-RIB-In from a peer for further analysis. Some software implementations based on the BMP protocol have been put into use, such as OpenBMP [27].

Finally, several studies focusing on the problem of hijacking were presented. These studies focus on how to detect and mitigate hijacking.

Several algorithms applying anomaly detection [28] [29] were proposed to identify spurious routes. Josh Karlin et al. [13] took a different approach, proposing to delay the update of suspicious routes by constructing trusted lists, reducing the likelihood of hijacking causing damage. Some hijacking experiments in real networks have also been numerous. The effect of ASPP in traffic engineering is studied in detail in [19] through the Peering Testbed platform [2], which also presents the fact that ASPP has an impact on hijacking. ARTEMIS [1] proposed by Pavlos Sermpezis *et al.* provides a set of solutions to hijack discovery and mitigation. It should be noted that ARTEMIS relies on data from public BGP monitoring infrastructure (e.g. RIPE[23] and routeviews[22]) studied in this thesis.

Chapter 3

Methods

The purpose of this chapter is to provide an overview of the research method used in this thesis. Section 3.1 describes the research process, including how we designed the experimental structure and data collection. Section 3.2 details the construction of each experiment unit. Section 3.3 details the thought on data collection. Section 3.4 presents shows how we intend to analyze the final results of the experiment to achieve our project goals.

3.1 Research Process

The research process of this project is divided into two parts: the design of the structure of the BGP hijacking experiment and the collection of BGP information to get the result of hijacking.

Figure 3.1 illustrates the experimental process we have designed. We can see the experiment structure where an experiment set contains five experiment units, and each of the units is collected via BGPStream. We automated the experimental and data collection process for each experiment set through scripting. And this hierarchical structure of the experiments was designed to help us analyze the data.

3.1.1 Experiment Structure Design

For the first part, we will outline how we implemented our three sub-goals when designing the experimental BGP hijacking method.

The first sub-goal is to confirm that using ASPP in hijacking improves stealthiness. The primary idea is to compare the changes in the number of monitors that observe hijacks collected by route collectors by increasing the

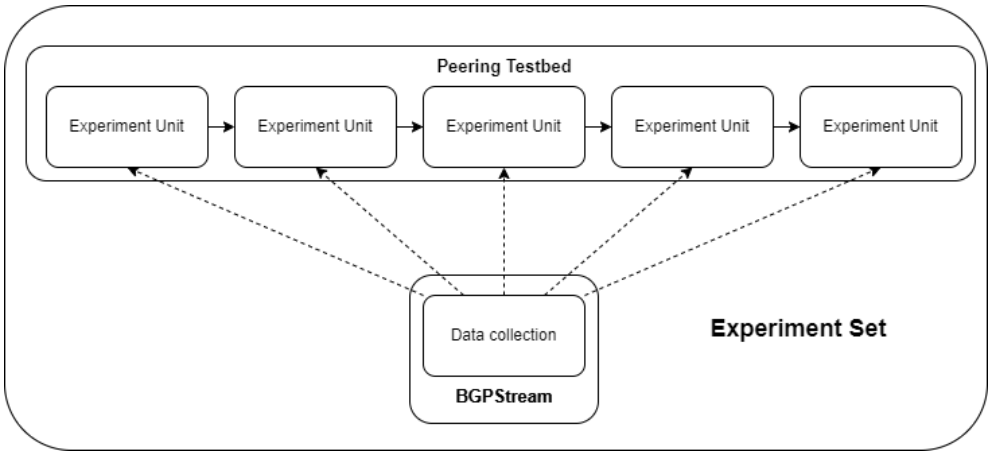


Figure 3.1: Process of one experiment set. One set contains five experiment units.

hijacking type number. Theoretically, as the hijacking type number increases, the number of monitors observing the hijacking should decrease. This would mean that hijacker-initiated ASPP routes improve the stealthiness of the hijack.

To achieve this objective, we need to create a fundamental experimental unit. Each experiment unit contains a particular hijacking type number ranging from 1 to 4. After announcing those four types of experiment units, the results can be analyzed by comparing the AS-paths that monitors report. Experimentally, we name the process of announcing a BGP route and withdrawing it as one **experiment unit**. For each experiment unit, we configure the hijacking type number, *i.e.,* the path length of the announced route. For sub-goal 1, this involves the four experiment units discussed above with hijacking type numbers ranging from 1 to 4. In Section 3.2 and Section 4.1, respectively, the configuration procedure and the particular implementation of each experiment unit are described in greater detail.

The second sub-goal is to explore whether the hijacker could infer the monitors that would observe the hijack using publicly available data reported by route collectors. This requires the hijacker to announce a legal, victim-independent prefix. When comparing this prefix with the victim's information reported by the PRC, the hijacker may be able to make a prediction. Consequently, we must also design an experiment in which the hijacker announces a prefix that is entirely unrelated to the victim. For this, another type of experiment unit similar to sub-goal 1 is utilized to implement this procedure. We achieved our objective by configuring a victim-independent,

non-hijacked experiment unit. In this distinct unit, the hijacker does not initiate a hijack, but rather announces a legitimate prefix-ASN pair. The hijacker predicts the hijack's visibility from this valid unit's visibility if it had been declared as a type N hijack based on an analysis of this unit's results. This is then compared to the results of the real hijacking experiment performed in sub-goal 1, which was actually declared as a type N, and thus the feasibility of the hijacker to predict the visibility of the hijack is determined.

The third sub-goal is to investigate the impact of non-hijacker controlled parameters on the stealthiness of hijacking. In order to study this sub-goal, we need to determine which parameters are non-hijacker controlled from those parameters that fall within the scope of this project. Specifically, we examine two non-hijacker controlled parameters: the geographical location of the victim and the hijacker*, and the impact of ASPP done by the victim e.g., for traffic engineering reasons. Section 3.2 describes how we designed our experiments and section 3.3 further explains why we choose to investigate these two non-controlled parameters. From now on, these non-hijacker controlled parameters will be referred to as **meta parameters**. Clearly, the dimensions of the hijacking type number and meta parameters are not the same. This is because, to study ASPP in hijacking, we need to change the hijacking type number while leaving the meta-parameter unchanged. However, if we want to study non-hijacker controlled parameters, we need to change the meta-parameters while implementing the hijacking type number within this set of meta parameters. Clearly, meta-parameters are one dimension higher than hijacking type numbers, and a hierarchical experimental structure should be designed to match these hierarchical research parameters.

Therefore, to answer our three sets of goals. we define a set of experiment units, consisting of *five* experiment units with the same meta parameters, the hijacking type number 1 to 4, and the hijacker declaration of the legitimate BGP information, as an **experiment set**. The data from each set of experiments are analyzed to answer sub-goal 1 and sub-goal 2, while multiple experiment sets are analyzed to answer sub-goal 3. A set of meta-parameters can be used to create a set of experimental sets, and when we need to study a meta-parameter, we can simply alter it and compare the two experimental sets. For instance, if we want to study the effect the geographical locations of the victim and the hijacker have to hijacking, we can construct two experiment sets; one for a hijacker in location A and a victim in location B, and

* While with sufficient time hijackers could deploy routing devices closer to victims, we assume hijacker locations are fixed in this thesis.

another for a hijacker in location B and a victim in location A. We can analyze then the effect of geographical location by comparing the outcomes of the two experiment sets. Clearly, the construction of an experiment set enables us to more clearly organize our data, focusing on the number of hijacking types within an experiment set and between different experiment sets when analyzing meta-parameters.

3.1.2 Data Collection Design

The second part of the research method is the collection of data via BGPStream, first decribed in section 2.5.2. Actually, to answer the research goals, we need to collect all the data from the experiments as part of the results. Therefore, in each experiment unit, we record the announcement time and the withdrawal time of the BGP messages during the announcement of the hijacking experiment. Using these recorded timestamps, we use BGPStream to collect all BGP messages recorded by route collectors for our prefixes within the recorded time period. In this way, we obtain the BGP data required for each of our experiments. After this, we create our own local database to collate the information we have obtained from each experiment. For more details on this step, see section 3.3.

3.2 Experiment Unit design

In this section, we discuss the design of the experiment unit. As mentioned in Section 3.1, in an experiment set, we need two kinds of experiment units: hijacking and non-hijacking ones. We need to make a hijacking announcement to fetch the hijacking result for sub-goal 1, and also need to make a legitimate announcement from the hijacker to infer the monitors which may observe the attack for sub-goal 2. This requires the design of a different structure of experimental units than sub-goal 1. As can be seen from the perspective of the code in Section 4.1.1, the two types of experiment unit differ only in the value set of parameters, using essentially the same framework. These two types of experiment unit are designed for different goals, so we have divided them into two types: **the hijacking experiment unit** and **the base experiment unit**. In the hijacking experiment unit, we have the hijacker announcing a type N hijack to learn the effect of ASPP. After this, in the base experiment unit, we have the hijacker announcing its own legitimate BGP information to investigate the visibility of the hijack. These five experimental units form an experiment set.

Figure 3.2 specifically describes the composition of an experiment set from an experimental perspective.

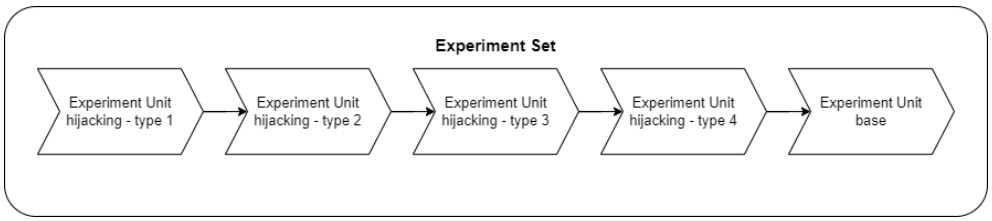


Figure 3.2: General structure of one experiment set: four hijacking experiment units from type 1 - 4 and one base experiment unit

However, there are three challenges. First, it takes time for BGP messages to propagate to other ASes on the Internet. This means that from the time we announce and withdraw BGP prefixes, it takes sometime for those BGP UPDATE messages to be delivered to route collectors. Second, because the delivery of messages is not instantaneous, it is possible for a monitor to report multiple paths during the time it receives the announcement and withdrawal. This means that the monitor may be constantly optimizing the path based on newly received messages. Third, we must consider that withdrawal delivery is also not instantaneous; therefore, we need to wait sometime before announcing the next experiment, otherwise it will collide with the previous one. The first and third challenges can be solved by carefully designing the experiment unit process, while the second challenge requires careful design of the data collection process, which will be described in section 3.3.

Here, we describe our design for the internal processes of the two experiment units and how this process deals with these challenges. First, we discuss the hijacking experiment unit. In one experiment set, we only change the type number among the hijacking experiment units. The platform restricts our hijacking type number to types 1-4, that is, four types of hijacking. In the hijacking experiment units, we first announce the victim. The victim announces its prefix and ASN on the Internet, and then after 20 minutes, the hijacker announces the hijack on the Internet. After waiting forty minutes to allow BGP to stabilize, we withdraw both the hijacker's and victim's announcement simultaneously. This provides ample time for the BGP messages to propagate before withdrawal to solve the first challenge. Similarly, we waited another 40 minutes to allow the network to stabilize and prevent any impact on the next experiment unit to solve the third challenge. The entire time frame that we need for the data collection, starting with the

victim declaration and ending with the withdrawal of both declarations, is 60 minutes. This is because BGPStream's data sources provide data at 5 and 15 minute intervals, respectively, as described in Section 3.3, and we want to fetch as few and synchronized packets as possible while keeping the experiment time in a reasonable range. One experiment unit takes 1 hour and 40 minutes, and with five experiment units in an experiment set, we cannot provide an unlimited amount of time for an experiment unit. As shown in the left part of the Figure 3.3, the blue bar represents the time of the announcement until the withdrawal of the victim, while the red bar represents the time of the announcement until the withdrawal of the hijack. Although there may still exist BGP messages on the Internet for the last 40 minutes (because withdrawal takes time to propagate), we do not log them. This is because for data collection, we care for the data between the victim's announcement to the withdrawal.

In the base experiment unit, things get easier for we do not need to care about the sequence of victim and hijacker's announcements. Since both announce their own BGP prefixes, which are unrelated to each other, we simply treat them the same way by announcing both for an hour then withdrawing them waiting for 40 minutes for BGP to converge. As shown in the right part of Figure 3.3, time of the victim and hijacker announcement and withdrawal is equal in this unit.

In the following, we discuss how the impact of ASPP and meta-parameters can be studied through the experiment unit and experiment set. As discussed, there are two meta-parameters that we study in this project: victim prepending time and geographic locations. victim prepending time is the amount of time required to simulate the victim's traffic engineering using ASPP. The number of geographic locations refers to the relative positions of the victim and the hijacker.

As shown in Figure 3.4, we have three sets of experiments, each with hijacking units of type 1-4. In red in the figure, we illustrate our study on the impact of ASPP on hijacking. This refers to an experiment set. Each experimental set (three shown in the figure) focuses on different meta-parameters. When we study victim prepending, we keep the geographic locations constant, as shown in the green block in the figure. And, as shown in the blue block, when we study geographical locations, we leave victim prepending unchanged. This allows the better organization of the meta-parameters.

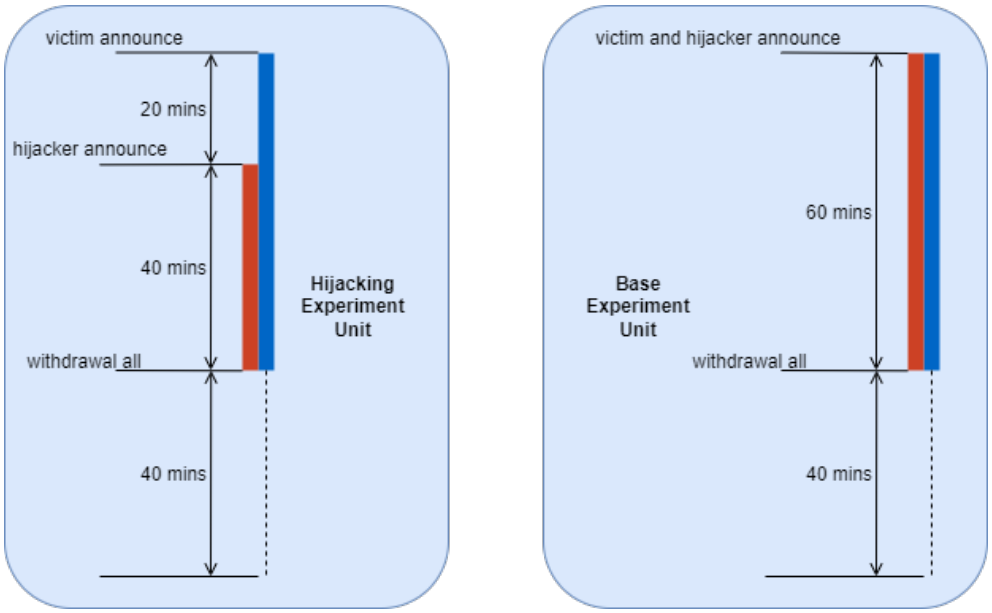


Figure 3.3: Internal process of the hijacking experiment(left) unit and base experiment unit (right).

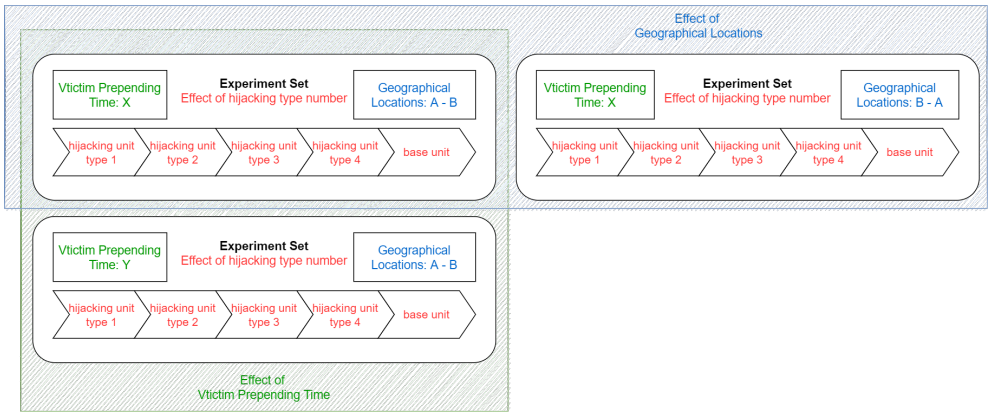


Figure 3.4: A example of how to use experiment unit and set to analyze ASPP and meta parameters

3.3 Data Collection

We use the libraries provided by BGPStream, and on top we build our own scripts and collect the data.

As Section 2.5.2 described, BGPStream fetches BGP data from two

projects: Route Views and RIPE RIS. Both data sources have two data formats: RIB dumps and update dumps. Route Views discloses RIB dumps every 2 hours, and RIPE RIS RIB discloses RIB dumps every 8 hours. RIB dumps contain a complete routing table with prefixes and AS paths. Obviously, such a long interval for RIB dumps is detrimental to our experiment unit. We cannot make a unit last for 8 hours. In contrast, Updates dumps provide collected BGP messages within shorter time period; therefore, they are more suitable for data collection, and so we fetch data from those Updates dumps.

BGPStream provides the appropriate interface, which allows us to get all the detected BGP Update dumps for the period given a prefix and a pair of timestamps. And these data are collected by BGPStream from its two data sources for the period. Due to fluctuations in the real network, we may see messages from multiple timestamps from the same monitor. Sometimes, as is often the case when BGP is unstable, one monitor may choose to report an AS path and then choose another. So, there are some route stability issues here that arise from our choice of tracking BGP updates instead of the actual RIB for the sake of experimental time granularity. The first issue is when we fetch one record, how we can ensure that it is the last stable BGP state to be maintained across networks. The second issue is when we fetch another record which has some same attribute as a previous one, how we can judge whether this new record should replace or not the previous state, or consider it as a new one to allow both to co-exist in an up-to-date state. Obviously, all these problems arise because we chose to fetch Update dumps rather than RIB dumps for a more reasonable experimental time, which led to the need to manually infer the final network state based on the update records.

Since we need to parse records one by one and there is no guarantee that the parsed data will be the last state of the network, we have to keep all the records in our database. With this in mind, we have added a field called *latest* to our database to mark whether or not a record is the most recent record. Specifically, when two conflicting records appear in the database at the same time, we make a judgment based on their timestamps - whether to give the new record latest state and revoke the old one's state, or just to leave the old record latest unchanged. So, here is the only question that remains: How can we judge whether two records are conflicting? More specifically, when a new record is parsed, based on what criteria are all records that conflict with it retrieved from the database?

Figure 3.5 briefly shows how we retrieve the BGP data and how we build the database. As mentioned above, we need to establish a criterion for determining whether two records are conflicting, i.e., whether the two records

are updates of the same AS. If they are, we do an **update**, otherwise we do an **addition**. So, we set a criterion in the script by converting the data we get through the interface into the data structure we need, using the quintet $(project, collector, peerasn, peeraddress, router)$ as the criterion for each entry. We consider this quintet to uniquely identify a BGP path reported by the same monitor. Therefore, we consider all different records that match the same quintet to be updates. Finally, we set the field $latest$ as a Boolean value. In all data with the same criterion, only the record with the latest timestamp can have the $latest$ field $True$. Some more technical details of database construction can be found in Section 4.2.

Figure 3.5: Parsing data from BGPStream and saving them into database.

3.4 Planned Data Analysis

In this section, we discuss how we plan and analyze the collected data. It is important to note that our experiment plan restrict us from conducting multiple experiments simultaneously in parallel. Each experiment unit is conducted one after the other, therefore a buffer time must be reserved between experiments

for the previous experiment to completely withdraw from the Internet (*i.e.,* the 40 minutes discussed before). So we make the following assumptions, which are necessary for us to analyze the experimental data.

- We assume that within the duration of the same experiment set, the "active" monitors collected by the PRC do not change in the network or change so little that they have a negligible impact on the collected data observations of the experiments.
- We assume that the variation in monitors collected among different experiment sets in our experiment is small and has negligible impact on the experiment analysis of data at the experiment set scale.

As described in Section 3.2, we control several experiment variables through the experiment units and experiment sets, and organized the data from different perspectives. Our approach for analyzing the experiment data will be based on this. Our planned data analysis will be carried out from the following perspectives: between experiment units within an experiment set, between experiment sets, and the presentation of a single experiment unit. We will then describe in detail the planned analysis process and its implications.

3.4.1 The Effect of Type Number and the Accuracy of Inference - Between Experiment Units

When we analyzed the data between the experiment units of the same experiment set, we kept the meta parameters unchanged except for the hijacking type number. This is to verify the effect of hijacking type number on the visibility of hijacking. Theoretically, the larger the type number is, the fewer networks will prefer the hijacker in the Internet, and therefore, the more covert the hijacking will be, i.e., fewer monitors will observe the hijacking. By taking the collector as the horizontal coordinate and the number of monitors observing the victim and hijacker as the vertical coordinate, we can make a plot of the results for each hijacking experiment unit, and so one experiment set can make four such plots (type 1-4). By comparing these plots, we can verify that increasing the type number of the attack is beneficial for hiding the hijacking, which in turn supports the idea that hijackers can improve hijacking concealment by designing higher type number attacks.

Additionally, we performed a base experiment to simulate the process of a hijacker announcing its own legitimate message and then evaluating the impact of the hijack. We devised a simple algorithm that takes the BGP messages

received by each monitor and calculates the number of ASes in the BGP path between the two legitimate BGP messages. This number of ASes is called the *distance* of this monitor from the hijacker and the victim (respectively). We then subtract these two distances to decide whether the monitor is closer to the hijacker or closer to the victim. Theoretically, any hijack with a greater hijacker-monitor distance than a victim-monitor distance should be able to hide from the monitor. Based on this simple algorithm, the hijacker can infer the possible visibility of the hijacking based on its own and the victim's legitimate BGP messages.

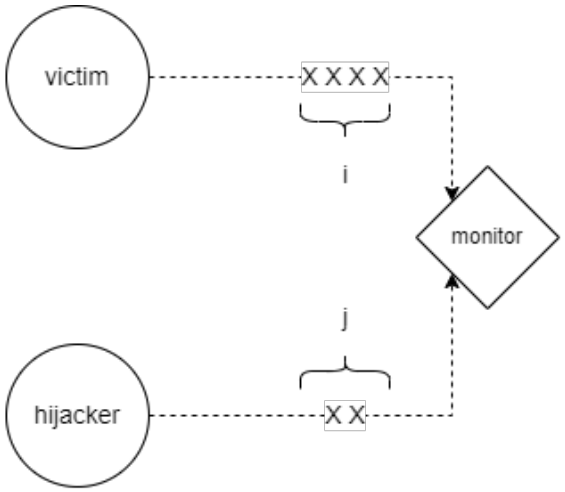


Table			
Victim Distance	Hijacker Distance	Proximity	Safe Hijacking Type Number
i	j	i - j	> i - j

Figure 3.6: A simple algorithm to judge a hijack's stealthiness.

For example, in the process shown in Figure 3.6, the hijacker can infer from the base experiment unit about which monitors in types 1-4 would not observe the attack and what is the ideal minimum Type Number for complete invisibility. Then, obtain from the hijacking experiment unit data on how many monitors are indeed not observing the attack, what the accuracy of the inference is, and what the probability of false negatives and false positives

Table 3.1: Meta parameters between experiment sets

	Prefix	ASN	Victim Prepending Number	Locations
Modifiable			✓	✓

is. These data allow for a simple assessment of the likelihood of the hijacker predicting the possible stealthiness of the attack based on legitimate data.

3.4.2 The Impact of Meta Parameters - Between Experiment Sets

We refer to parameters other than Type Number as meta-parameters, as these are variables that are fixed in one of our experimental sets. Specifically, there are the following kinds of meta-parameter during this experiment.

As Table Table 3.1 shows, we did not modify the Prefix and ASN of the victim and hijacker in our experiments. We modified the Victim AS prepending Number and the geographic locations of the hijacker and the victim. As mentioned in the background, a significant proportion of ASes use ASPP as part of traffic engineering, so the impact of different Victim Prepending Numbers on hijacking concealment is also worth investigating. Theoretically, the higher the Victim Prepending Number, the easier it is for the hijacker to hijack the victim however the more likely for the hijacker would be to be exposed as more monitors would observe the attack. Additionally, information on the geographical location of the hijacker and victim is also an important aspect to study. Although from the hijacker's point of view, the geographical location of both is not controllable. From the a researcher's point of view, a change in geographic location is worth studying as it could make a considerable difference.

It is important to note that, from the perspective of experiment sets, we still need to control other variables. For example, when looking at the Victim Prepending Number, we want to control for the geographic location of both to be constant, and vice versa.

3.4.3 Structure of the Topology and the Presence of Key Nodes - In Experiment Unit

Finally, we focus on a single experiment unit. The single experiment unit has all parameters fixed and is therefore not the focus of our study. Here we simply

try to combine the AS-Path information from all networks into a tree and network diagrams, as described in Section 2.4.2. We speculate that there may be some so-called key nodes present. By key nodes, we mean that there are multiple child nodes under a single node in the tree structure translated from the topology diagram of the AS paths. These nodes appear in the AS-path of many monitors. It is also possible that some nodes exist in both the hijack path and the victim path; they play different roles in the paths of different monitors. There are significant differences between the real network and the idealized model. The hijacker may also be able to design some advanced algorithms (e.g. using machine learning, etc.) based on this information. Such studies are not in our focus, and some conjectures will be given in Section Section 6.3.

Chapter 4

Technical Details

In this chapter we will describe some technical details of our experiments. Section 4.1 describes the API that the Peering Testbed platform provides us with to release BGP messages into the network and how to organize the experimental code structure. In this chapter, we briefly describe how we modified the parameters in the BGP messages (such as hijacking type number and locations) through the configuration file to simulate both hijackers and victims. Additionally, we create automated scripts in each experiment set to ensure that the experiment units are ordered and serial. Section 4.2 describes the process of building our database. In this section, we give a brief introduction to the format of the data obtained through the BGPStream API, and also create scripts to collate BGP information into a local database. The selection of the database and the format in which the data is stored will be described in detail in this chapter. Section 4.3 details how we analyze the experiment data and propose solutions to some problems in the analysis.

4.1 Experiment Design

In this section we will briefly introduce the scripting interface provided by Peering Testbed from a code level, and our design for automating each experiment set.

4.1.1 Peering Testbed

Peering Testbed provides a client [30] for individual researchers. Through the Python interface provided by this client, authorized users are able to publish BGP messages for authorized prefixes and ASN.

This client exists to use the command line method of passing the parameters directly via the command line to send BGP messages. However, the command line approach does not allow the use of ASPP to hide the hijacking while typing the parameters, and it is also difficult to integrate into an automated script. So, we use another use which is to configuration files to set the BGP messages. For the client, each BGP message (including announcement and withdrawal) is a configuration file. The configuration file is a JSON file, and the following Listing 4.1 is an example of this JSON configuration.

Listing 4.1: One example of JSON format configuration

```
{
  "184.164.236.0/24": {
    "announce": [
      {
        "muxes": [
          "grnet01"
        ],
        "origin": 61576,
        "prepend": [61575, 61575, 61575],
      }
    ]
  }
}
```

This profile is an example of type-3 hijacking for the prefix 184.164.236.0/24. The victim's ASN is 61576 and the hijacker's ASN is 61575. The hijacker is sending the messages to the network via grnet. Obviously, we can control the parameters of each message by changing this configuration file. We can change the ASN and hijacking type number of the hijacker, as well as the mux issued by the hijacker, which identifies the network provider or organization with which Peering Testbed peers. By changing this identifier, we can change the geographical location of the hijacker and the victim.

4.1.2 Experiment Set and Experiment Units Scripts

An experiment set contains hijacking experiment units of type 1-4 and one base experiment unit. Since the functions of the experiment unit interface to Peering Testbed API and the configuration file we provide, there is no need to distinguish between the two types of experiment unit in experiment unit script. We have unified the two types of experiment units into one single Python class at code level. In this class, specific methods are exposed for the upper layers to call (e.g. announce and withdraw). The code snippet Listing 4.2 shows some methods used to control the experiment unit process, with some of the methods collapsed.

Listing 4.2: Some methods in the Python class of experiment unit

```
"""
Experiment class to control the experiment unit process
"""
class Experiment():
    def __init__(self, *exp_confs):
        self.announce_exp_confs_vic = [utils.load_json(conf) for conf
            in exp_confs if 'announce' in conf and 'victim' in conf]
        self.announce_exp_confs_hij = [utils.load_json(conf) for conf
            in exp_confs if 'announce' in conf and 'hijacker' in conf]
        self.announce_exp_confs_bas = [utils.load_json(conf) for conf
            in exp_confs if 'announce' in conf and 'base' in conf]
        ...
        self.muxes = set()
        self.controller = utils.BGPController()
        self.deploy_timestamp = dict()
        self._init()

    def _init(self):
        ...

    def _open_client(self):
        ...

    def _deploy_one_conf(self, conf, cooling_time):
        self._open_client()
        self.controller.deploy(conf)
        self.deploy_timestamp.setdefault(str(conf), int(time.time()))
        time.sleep(cooling_time)

    def deploy_victim_announcement(self, cooling_time=COOLING_TIME):
        ...

    def deploy_hijacker_announcement(self, cooling_time=COOLING_TIME):
        ...

    def deploy_base_announcement(self, cooling_time=COOLING_TIME):
        ...

    def deploy_base_withdrawal(self, cooling_time=COOLING_TIME):
        ...
```

The type of configuration file (hijacking or base unit) in this code is determined by the name of the configuration file. So when writing one script to call this class (i.e. the experiment set), we do not need to care about its internal details. We only need to name the configuration file correctly and then call the instance method with the corresponding name according to the order of the experiments. Correspondingly, since the configuration files for our multiple experiment sets exist as templates and only specific parameters need to be modified between different sets, we also wrote a script to generate the corresponding configuration files by type some parameters we care. The code for the entire experiment set can thus be structured as shown in Figure 4.1.

In fact, the only difference between our different experiment sets is the

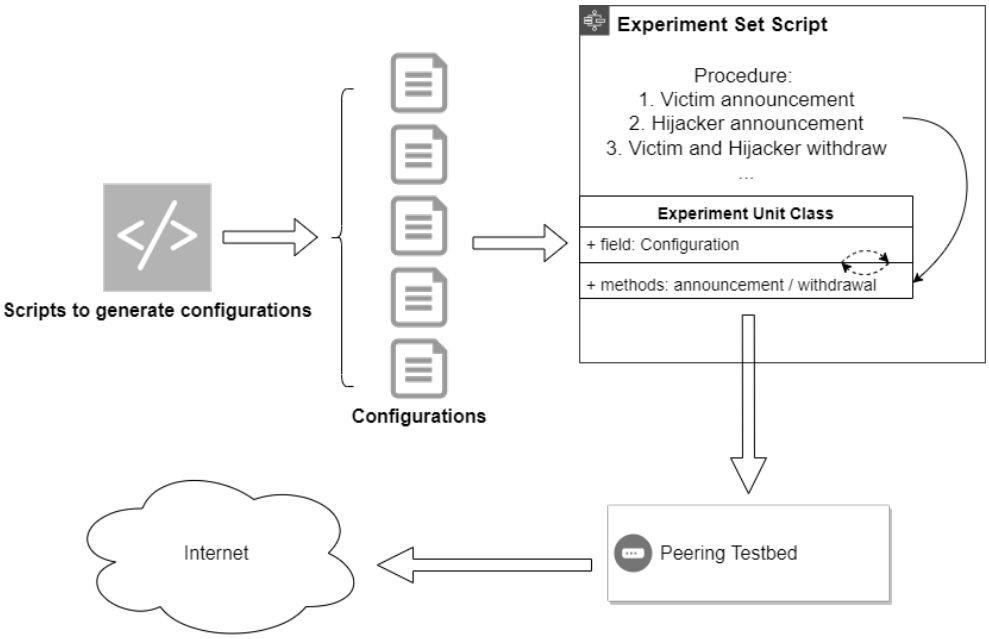


Figure 4.1: Experiment set structure in code level

location of the configuration files, so it is only necessary to generate the configuration files for many groups of experiment sets in advance to automate the sequential execution of the different experiment sets.

4.2 Data Collection

In this section we will briefly introduce the API and data formats provided by BGPStream from a code level, as well as the details of our own database build.

4.2.1 BGPStream

As Figure 4.2 shows, BGPStream has been designed as a multi-layer structure. [31] At the core of this is libBGPStream, a library written in C that can be installed through the package management tools of several Linux distributions. This library provides the ability to collect data from its data providers and a unified API to fetch these data. In addition, BGPStream provides a command-line tool, BGPReader, and a Python library, PyBGPStream. We use the API provided by PyBGPStream in our scripts to fetch data.

By giving a time range, target prefix and ASN, PyBGPStream can return

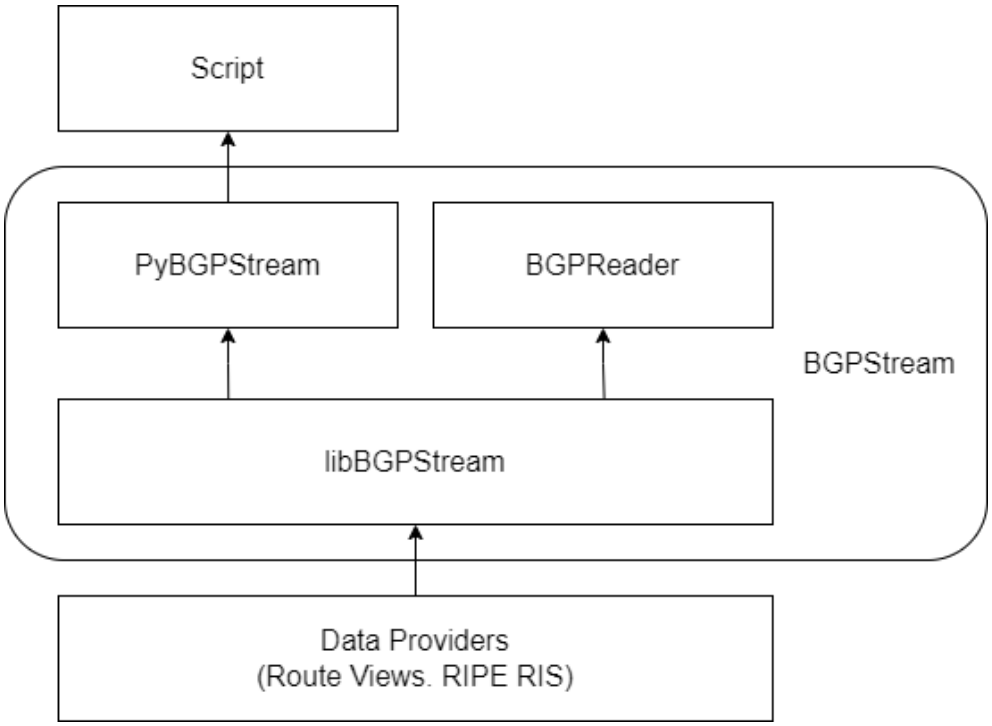


Figure 4.2: BGPStream framework structure

the parsed records in its given format. Listing 4.3 is a snippet of our script to invoke the PyBGPStream API.

Listing 4.3: A snippet of invoking PyBGPStream API

```
"""
Invoking PyBGPStream API with our database
"""
stream_type = StreamType.UPDATE
prefix = "184.164.236.0/24"

for index, time_pair in enumerate(time_pair_list):
    start, end = time_pair
    _, mycol = collect_db(stream_type=stream_type, prefix=prefix,
        name=f'184.164.236.0/24_{index+1+start_index}-{num}')

    stream = pybgpstream.BGPStream(
        from_time=start, until_time=end,
        record_type=stream_type.value,
        filter="prefix exact " + prefix
    )

    main(mycol=mycol, stream_type=stream_type, stream=stream)
```

In this snippet, PyBGPStream returns a stream, and the stream can be traversed by a number of records. PyBGPStream calls these records

elems [32], and each elem as an instance has its own properties. Function *parse_to_dict* in code snippet Listing 4.4 shows what properties we used in elems.

In fact, the PyBGPStream API is so simple that all we need to do is provide the correct parameters, and we can iterate through the records that match the conditions without caring about the details of the internal implementation.

4.2.2 MongoDB

We used MongoDB [33] when building our own database. MongoDB is a document database, and we chose MongoDB because we wanted to use a flat way of storing data. MongoDB also supports saving fields as arrays, which is a great formal advantage for saving AS paths. In addition, each database collection can be imported and exported in json format, which is very attractive for our data migration. Taking many judgements into account, we finally chose MongoDB as our database.

In fact, interfacing with BGPStream and MongoDB is not difficult, as there is a Python library named PyMongo [34] that allows one to connect to the database. This code snippet Listing 4.4 shows how we combine it with BGPStream.

Listing 4.4: Two typical funtions of combine PyMongo and PyBGPStream

```
"""
Parse the data fetched into our database using our format
"""
def parse_to_dict(elem: pybgpstream.pybgpstream.BGPElem):
    returned_dict = {
        "project": elem.project,
        "collector": elem.collector,
        "time": int(elem.time),
        "peer_asn": elem.peer_asn,
        "second_asn": None,
        "ori_asn": None,
        "peer_address": elem.peer_address,
        "router": elem.router,
        "as_path": None,
        "latest": True
    }

    if elem._maybe_field("as-path") is None:
        returned_dict["as_path"] = []
    else:
        returned_dict["as_path"] = elem._maybe_field("as-path").split(" ")
        if len(returned_dict["as_path"]) > 2:
            returned_dict["second_asn"] = returned_dict["as_path"][1]
        returned_dict["ori_asn"] = returned_dict["as_path"][-1]

    return returned_dict

```

```
def parse_update_into_db(db_collection: pymongo.collection.Collection,
    stream: pybgpstream.BGPStream):
    for elem in stream:
        parsed_dict = parse_to_dict(elem)
        query_dict_gt = {"project": parsed_dict["project"], "collector":
            parsed_dict["collector"], "peer_asn": parsed_dict["peer_asn"],
                    "ori_asn": parsed_dict["ori_asn"],
                    "peer_address": parsed_dict["peer_address"],
                    "router": parsed_dict["router"],
                    "time": { "$gt": parsed_dict["time"]}, "latest": True}
        if db_collection.count_documents(query_dict_gt) > 0:
            parsed_dict["latest"] = False
            db_collection.insert_one(parsed_dict)
        else:
            query_dict_lte = {"project": parsed_dict["project"],
            "collector": parsed_dict["collector"],
            "peer_asn": parsed_dict["peer_asn"],
                    "ori_asn": parsed_dict["ori_asn"], "peer_address":
                        parsed_dict["peer_address"],
                    "router": parsed_dict["router"],
                    "time": { "$lte": parsed_dict["time"]},
                    "latest": True}
            db_collection.update_many(query_dict_lte,
                {"$set": {"latest": False}})
            db_collection.insert_one(parsed_dict)
```

The function *parse_to_dict* extracts the attributes we need from the data format provided by PyBGPStream, and *parse_update_into_db* implements the function to update the records according to the criteria we described in Section 3.3. Note that we implement the judgement and latest field updates through MongoDB's query and aggregation statement at the time of each record fetched. The performance of the database statement to check and update is clearly superior to simply saving all the data into one JSON file and then traversing it, which is another important reason for our choice of using the database.

4.3 Data Analysis

In this section, we will describe some details in how we achieve our data analyses. We will follow the sequence of how to implement our three sub-goals. This is also the order in which we will express the results of our experiments in Chapter 5.

4.3.1 Analysis of ASPP in Hijacking

The analysis of ASPP is straightforward. We just need to be given a set of meta parameters and then compare the results of the hijacking type number

growing from 1 to 4. We can then determine whether ASPP is conducive to improving the stealthiness of hijacking.

Our quantitative metric for measuring stealthiness of a hijack is the number of monitors per PRC collected that reported the hijack. We will give a stacked bar chart plotted in absolute numbers of monitors. Also, we will also make a stacked bar chart with the percentage of monitors reporting hijacking. Since the total number of monitors reported by different PRCs can vary considerably, a comparison of the above absolute and percentage plots will allow better conclusions to be drawn. We plan to look at all the data we have collected to confirm that it is a general phenomenon that ASPP is helpful to hijack stealthiness. We also plan to show several sets of bar plot under typical meta-parameters in the thesis.

4.3.2 Analysis of the Accuracy of Hijacking Inference

As Section 3.4.1 described, we have made a simple algorithm to infer stealthiness of a hijack given the hijacking type number. To achieve this, we need to gather the paths reported by PRC, both for the hijacker's and the victim's independent prefix announcement.

4.3.2.1 Complete Stealthy Indicator

To compute the potential stealthiness of a hijack, we plot two kinds of result graphs based on a quantitative indicator which is called as *complete stealthy indicator*.

As described in Section 3.4.1, given a potential prepending number for the hijacker, we could infer whether the hijack will be stealthy or not to a specific monitor. By analyzing all the monitors of a PRC, a *minimum prepending number* could be inferred that has the potential to make the hijack *completely* invisible to this PRC. We define this minimum prepending number as the *complete stealthy indicator* for this PRC. However, before we talk about this indicator, we need to first talk about a special situation when we try to infer the hijacking stealthiness.

In Section 3.4.1, we define that for a single monitor, the proximity to infer whether the hijacking will be stealthy or not is the difference in the distance in AS path between victim and hijacker. However, since in base experiment units, the hijacker announces its own legitimate BGP prefix which independent of the prefix announced by the victim, we cannot guarantee that the monitors that observe both are exactly the same. In fact, for more than six hundred monitors reporting their prefixes to these PRCs, it is possible for the hijacker

to be observed without the victim being observed in a monitor's report, and vice versa. And these monitors observed only one side may vary with the geographical location of victim and hijacker.

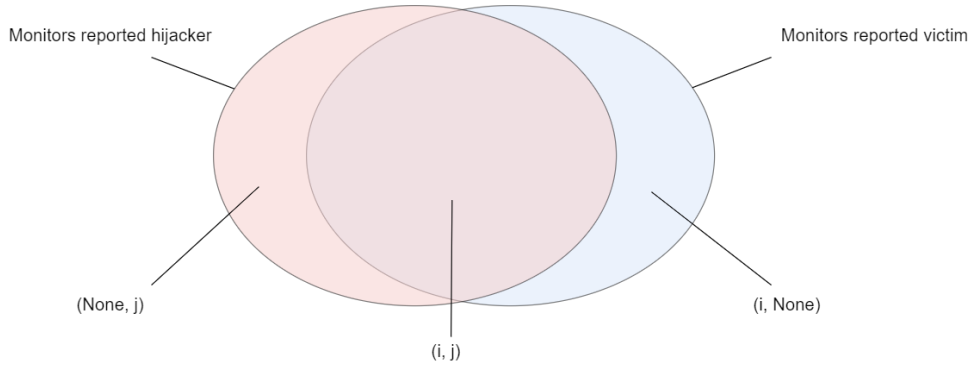


Figure 4.3: A graph to show the sets of monitors in PRC

As shown in Figure 4.3, all monitors under a PRC can be divided into two sets: monitors reporting the victim and monitors reporting the hijacker. However, the two sets are not identical as there may exist monitors in one set that do not exist in the other. In this figure, the blue set represents monitors reporting the victim and the red set represents monitors reporting the hijacker. As shown, some monitors report the victim without hijacker (blue part), and vice versa (red part).

In the algorithm presented in Section 3.4.1, for each monitor we have two distances: Fist, the distance of the monitor to the hijacker (j variable), and second the distance of the monitor to the victim (i variable). To calculate the proximity used to compute the stealthiness of the hijack, we need both distances (tuple (i, j)). For those monitors that do not report the hijacker, the tuple becomes $(i, None)$. Meanwhile, for those monitors that do not report the victim the tuple becomes $(None, j)$. When we want to calculate the proximity with $None$, we will represent $None$ as $+\infty$ as $+\infty$ represents an unreachable distance for the proximity formula is $i - j$.

Figure 4.4 shows the possible calculation outcomes of the proximity formula. Column labels represent the hijacker distance observed by each monitor. Row labels represent the victim distance. The red $+\infty$ at $(None, j)$ indicates a monitor that is not reporting the victim. As such monitors may always report the hijacker no matter the hijack Type, we say that we cannot design a stealthy hijack against this monitor (for the purposes of this work). This is a rational argument as the hijacker cannot simple hide from a monitor

Hijacker Distance / Victim Distance	j	None
i	i - j	$-\infty$
None	∞	

Figure 4.4: A table to show a monitor's proximity in different situation

that only reports the hijacker without reporting the victim.*

Unfortunately, this infinity happens sometimes and will contaminate all inferred results in a PRC, since no Type hijack exists that is bigger than infinity. Although these monitors' appearance of an infinity would signal that we would not be able to design a completely stealthy hijack for this PRC via ASPP, we present a graph of all PRC's complete stealthy indicator after removing these monitors in Section 5.3. We will also show a graph before the removal. The difference in the length of the prepending number required by the different PRCs can be clearly analyzed in these two graphs.

4.3.2.2 TPR and FPR

For a given prepending number, the hijacker would infer whether this potential hijack will be stealthy or not for a monitor. We define a monitor as *dangerous* if the monitor observes the hijack. The hijacker's goal (second sub-goal of this thesis) is to infer whether a monitor is dangerous. Thus, the algorithm which computes the dangerous monitors is actually a classifier for dangerous monitors. A traditional tool to analyze the true positive rate and false alarm rate is the Receiver Operating Characteristic (ROC) graph [35]. The ROC graph has been used in a wide range of areas, such as diagnostic systems [36].

For a binary classification problem, the outcomes have two labels which are positive (p) and negative (n). Given an instance's ground truth and the

* This is the case for this thesis' methods. For clarity, we note there may exist other methods (not used in this thesis) that may allow a hijacker to hide from such monitors).

classifier's output of this instance, if both the instance and output are positive, then this is a *true positive*. While, if the output of classifier is negative, then this is defined as a *false negative*. In our case, the instance is a monitor and the label is whether the monitor is dangerous.

By counting the true positive and false positive instances, we obtain the two components of the ROC curve: the True Positive Rate (TPR) and the False Positive Rate (FPR). [35]

$$TPR = \frac{TruePositives}{TotalPositives}, FPR = \frac{FalsePositives}{TotalNegatives} \tag{4.1}$$

As shown in Figure 4.5, once we get a group of TPR and FPR, we can find a point in the ROC graph. If the output of a classifier lies at the top left corner, we can conclude that the classifier is very effective. For example, in Figure 4.5 classifier A is better than classifier B. Classifier C is very close to the diagonal, which indicates that the classifier results are closer to the results of random assignment. An interesting fact is that while classifier D has worse results than a random classifier, it is not bad. In fact, classifiers with points at the bottom right corner are also effective, as they tend to correctly make the opposite judgment of what they were designed.

Researchers with experience in machine learning will be more familiar with the ROC curve. However, our study analisys of results stops at the point in the ROC space. For this project, we did not train our algorithm and we did not have a redundant dataset. In fact, each PRC is different, and each monitor under a PRC is also different. Our (FPR, TPR) results are only used to show the performance of this simple algorithm in a coarse-grained way, as well as to try to visualize the differences in performance under different prepending number and meta-parameters. The design of a more advanced algorithms could be the purpose of future work.

4.3.3 Analysis of Meta Parameters

For the meta parameters, we have used a similar visualization approach as the analysis in Hijacking Type Number. When analyzing the impact of the victim's prepending number, we did so with the help of a stacked histogram of the percentage of hijacked monitors. And when analyzing the impact of geographic locations of the hijacker and the victim, we observed the changes in visibility by changing only one position of the two, and also by swapping the two positions (*i.e.,* the hijacker and the victim).

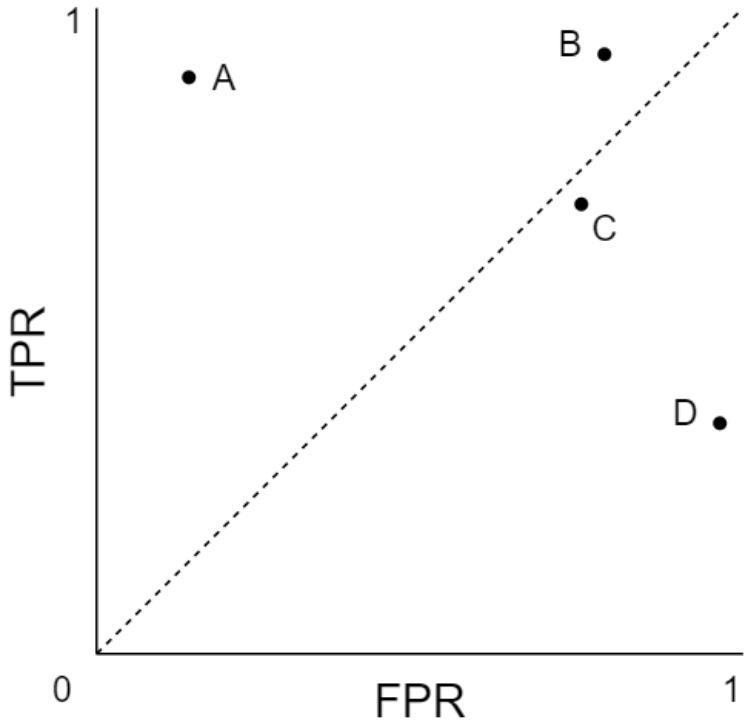


Figure 4.5: An example of analyze points in ROC graph

Chapter 5

Results and Analysis

In this chapter, we present the results and discuss them. First, We will present the data we have collected and then describe the major results of the experiments according to the order of the sub-goals defined in a previous section. To answer the sub-goals, Chapter 3 has described the set of experimental data and parameters that we research. Those are: *(hijacker location, victim location, victim prepending number)*. In fact, these three set of variables describe the meta-parameters of an experiment set, i.e., the geographical location of the hijacker, the geographical location of the victim, and the number ASNs prepended when the victim uses ASPP for traffic engineering. In the figures that follow, we use the abbreviation *rv.xxx* to refer to *routeviews.xxx* when displaying the PRC names for aesthetics reasons.

5.1 Data Collected and Statistical Information Analysis

Here we show the data we have collected and some statistics. We use MongoDB to build the database with each record fetched stored as a document, as shown in Figure 5.1 and Figure 5.2.

Before proceeding with our main analysis, we briefly analyze the database in a statistical perspective unrelated to a specific monitor. There are two interesting set of features. The first is the density of the crawled data. We looked at all data with the *latest* field set as $true$, analyzed the distribution of their timestamps in the database, and analyzed them according to the density of their occurrence shown in Figure 5.3. This graph is generated by the database, and its X-axis represents the whole time of one experiment unit. The green

```
{
    "_id" : ObjectId("62bad86b4cde4c2e584f8ff8"),
    "project" : "routeviews",
    "collector" : "route-views.linx",
    "time" : NumberInt(1655836064),
    "peer_asn" : NumberInt(13030),
    "second_asn" : "21320",
    "ori_asn" : "61576",
    "peer_address" : "195.66.224.175",
    "router" : null,
    "as_path" : [
        "13030",
        "21320",
        "5408",
        "47065",
        "61576",
        "61576",
        "61576",
        "61576",
        "61576"
    ],
    "latest" : false
}
{
    "_id" : ObjectId("62bad86b4cde4c2e584f8ff9"),
    "project" : "routeviews"
```


Figure 5.1: Each record is stored as a document in MongoDB

_id	project	collector	time	peer_asn	second_asn	ori_asn	peer_address	router	as_path
62bad86b...	ris	rrc20	1655836065	29691	13030	61576	91.206.52.9	null	[10 elements]
62bad86b...	ris	rrc20	1655836065	196621	8758	61576	91.206.52.103	null	[11 elements]
62bad86b...	ris	rrc20	1655836065	196621	8758	61576	91.206.52.103	null	[11 elements]
62bad86b...	ris	rrc20	1655836065	21232	13030	61576	91.206.52.32	null	[10 elements]
62bad86b...	ris	rrc20	1655836065	51786	51786	61576	91.206.52.246	null	[14 elements]
62bad86b...	ris	rrc20	1655836065	51786	34549	61576	91.206.52.246	null	[11 elements]
62bad86b...	ris	rrc20	1655836065	51786	34549	61576	91.206.52.246	null	[11 elements]
62bad86b...	ris	rrc20	1655836065	51786	34549	61576	91.206.52.246	null	[11 elements]
62bad86b...	ris	rrc20	1655836065	20612	8758	61576	91.206.52.127	null	[11 elements]
62bad86b...	ris	rrc20	1655836065	20612	8758	61576	91.206.52.127	null	[11 elements]
62bad86b...	ris	rrc20	1655836065	8298	34549	61576	91.206.53.12	null	[11 elements]
62bad86b...	ris	rrc20	1655836065	8298	34549	61576	91.206.53.12	null	[11 elements]
62bad86b...	ris	rrc20	1655836065	8298	34549	61576	91.206.53.12	null	[11 elements]
62bad86b...	ris	rrc20	1655836065	8218	21320	61576	91.206.52.116	null	[9 elements]
62bad86b...	ris	rrc20	1655836065	58299	13030	61576	91.206.52.130	null	[10 elements]
62bad86b...	ris	rrc20	1655836065	58299	2603	61576	91.206.52.130	null	[10 elements]
62bad86b...	ris	rrc20	1655836065	24482	2603	61576	91.206.52.205	null	[10 elements]
62bad86b...	ris	rrc20	1655836065	210312	8298	61576	91.206.53.32	null	[12 elements]

Figure 5.2: All documents form a collection in MongoDB

vertical line represents the frequency of recorded occurrences. The records here are the final data we included in our analysis whose *latest* field is *True*. And the more pronounced the green line is at a given point in time, the more records are reported at that point in time. An interesting result was that in an experiment unit where most monitors could observe hijacking, there was a high-density spike in the amount of data observed at one third of the total time, which is when we announce the hijack. In most of the experiment units where little hijacking was observed, this peak was absent or insignificant. A typical experiment set is *(amsterdam01, wisc01, 0)*. In this experiment set, as the hijacking type number increases, the number of monitors in which hijacking

is observed changes from a majority to a smaller minority. We also illustrate this example in terms of the number of observers in Section 5.2. And where the temporal plots of the four experiment units are done, we can clearly see that as the hijacking type number increases, the wave at the third of the total time (hijacker announcement time) fades. Meanwhile the wave at the beginning of the experiment (victim announcement time) gradually deepens as shown in Figure 5.3. This is in line with our logic.

Figure 5.3: Record density of experiment set *(amsterdam01, wisc01, 0)*

The second interesting feature is the results AS path length. We perform a macroscopic analysis from the resulting AS path length. Instead of distinguishing whether a record was observed to be hijacked, we just performed a statistical analysis of the AS path length for all valid records where the *latest* field was *True*. Here we still choose *(amsterdam01, wisc01, 0)* as a typical analysis to show the result. As can be seen in fig. 5.4, the AS path length of majority of records increases as the hijacking type number increases. In Figure 5.4, from the statistical point of view of AS path length alone, the AS path length of most records concentrates towards 7 as the hijacking type number increases. This is in line with our expectations. Because as the hijacking type number decreases, more monitors will be attracted to hijackers with shorter paths, leading to a fraction of AS path lengths. In theory, each monitor tends to choose the shorter AS path between the hijacker and the victim. Monitors who choose the hijacker do so mainly because the hijacker's path is shorter for them. The increase of hijacking type number will cause AS path lengths in the records of such monitors to increase. On the other hand,

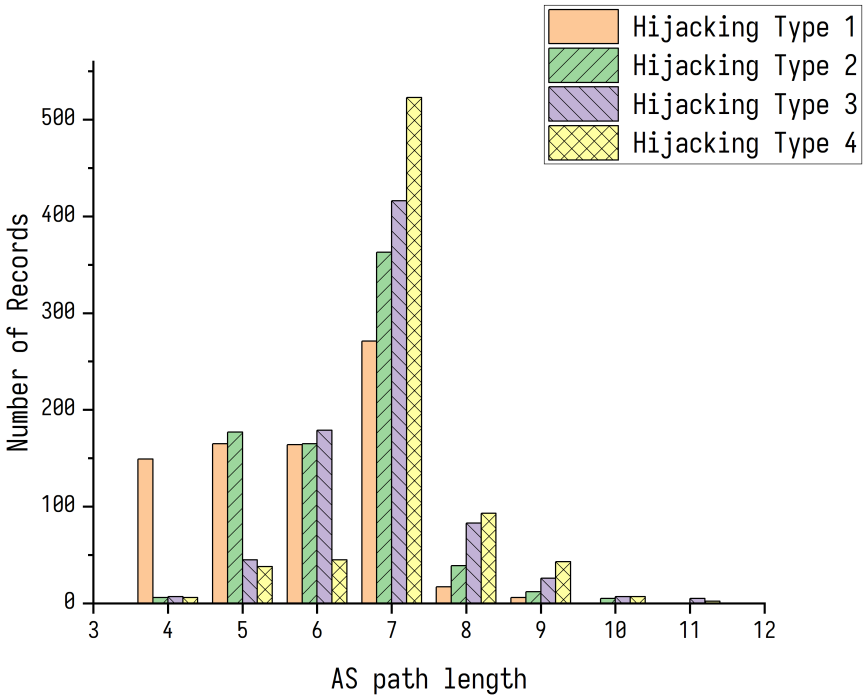


Figure 5.4: AS path length distribution of experiment set *(amsterdam01, wisc01, 0)*

this inflation is limited since large hijacking type numbers could also causes some monitors to choose the victim, thus at some point reaching an upper limit of inflation.

5.2 The Impact of ASPP in Hijacking

Our first sub-goal is to verify how helpful ASPP would be for the hijacker for generating stealthy attacks. In sum, from our experimental results, it appears that prepending the AS path is helpful for achieving stealthier hijacks. The degree to which ASPP helps such attacks may vary for different hijackers and victims. However, as the number of hijacking types increases, the stealthiness of hijacking also commonly increases. Here, we show plots of the results for two experimental sets where the results are intuitive.

In the first experiment set, we announce the victim's prefix through *grnet01* and then the hijacker's prefix through *wisc01*. The experiments are identified

as *(wisc01, grnet01, 3)*, with Y-axis showing the number of monitors per route collector (X-axis) that either observe or do not observe the hijack. The results are shown in Figure 5.5, where blue bars of each stacked bar shows the monitors that do not observe the hijack, while red bars show the number of monitors that observe the hijack. The hijacker type number increases from 1 to 4, from the top to the bottom of the four subplots.

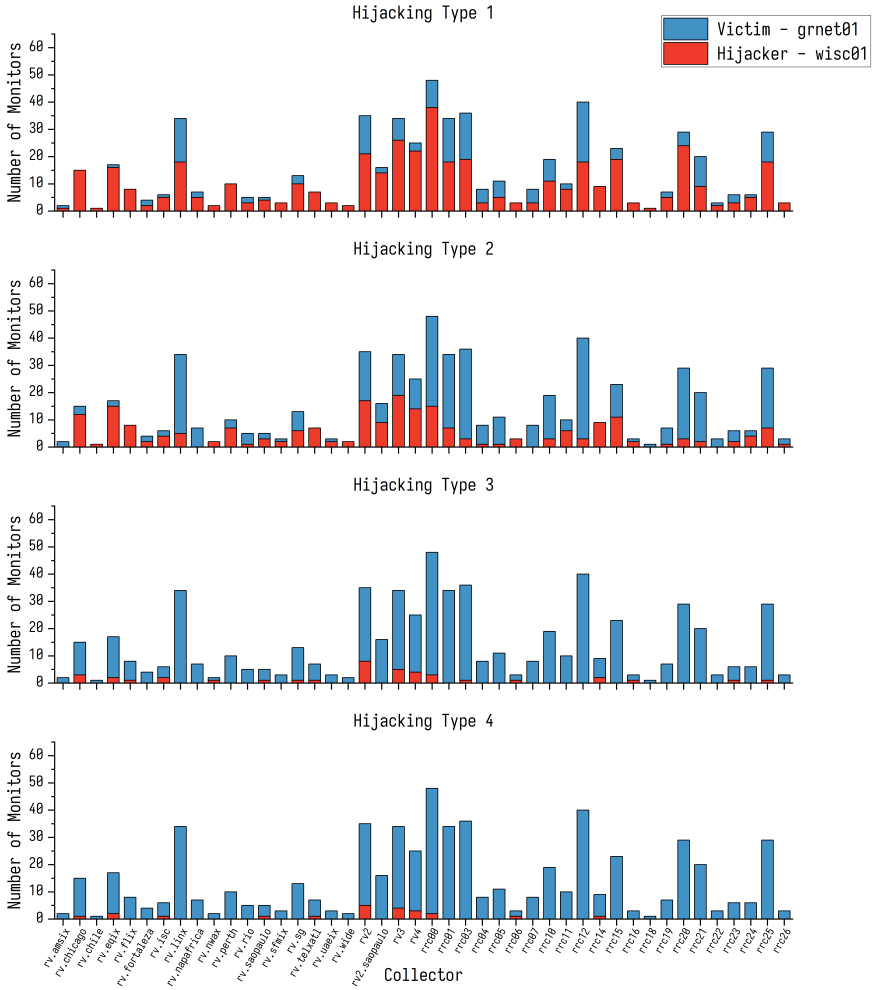


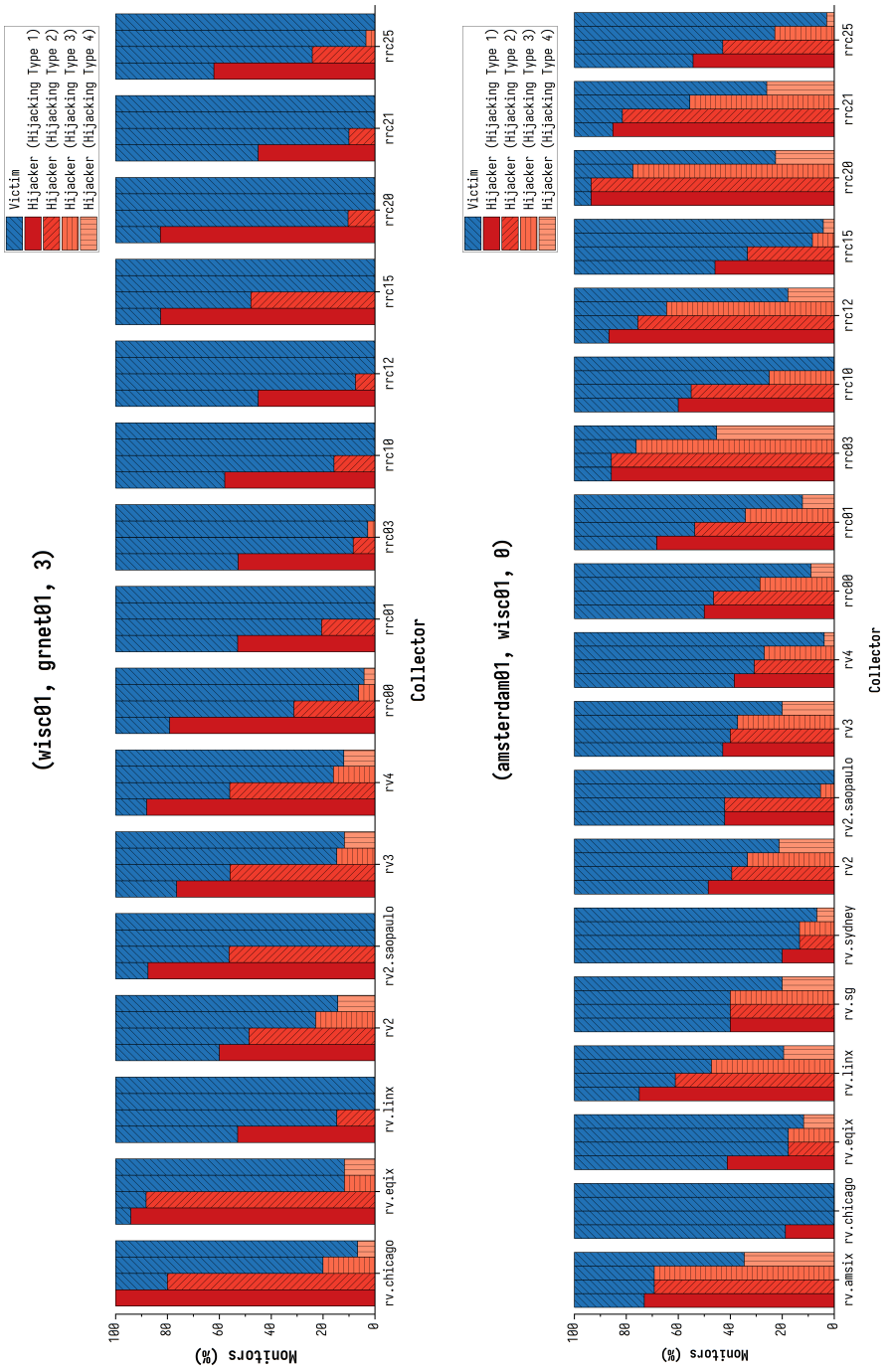
Figure 5.5: Number of monitors in experiment set *(wisc01, grnet01, 3)*

As it can be clearly observed from the four subplots, for each PRC as the number of hijacking types increases the size of the top blue bars increases and the size of bottom red bars decreases. This means that the proportion of hijacked monitors observed under each PRC is decreasing with the increasing hijacking type number.

In addition, Figure 5.6(a) shows a percentage stacked bar chart. This graph was created by transforming Figure 5.5 into percentages. To make the graph more intuitive, we have plotted the results of the four hijacker type numbers into one graph. And to reduce the length of the image, we have hidden some PRCs' result whose monitor number is trivial, *i.e.,* less than 10. In this graph, each PRC has four bars, and the top blue parts of each bar are the percentage of monitors with no hijackers observed, while the bottom red parts are the percentage of monitors with hijacking observed. The lighter red part at the bottom of each of the four bars from left to right represents the increasing number of hijacking types. Our use of lighter colors means that as the hijacking type number increases, the hijacking becomes less competitive with the victim and therefore less visible. The results are in line with our expectations, with a decreasing percentage of hijacked monitors observed for higher hijack type numbers at each PRC.

In another set of experiments, we change the locations of the hijacker and the victim and announce the victim's prefix at Wisconsin (wisc01) and the hijacker's prefix at Amsterdam (amsterdam01, with the experiment identified as (amsterdam01, wisc01, 0). Similar to the previous Hijacker-Victim location, two graphs are plotted shown in Figure 5.7 and Figure 5.6(b) (respectively).

An interesting point is that in Figure 5.5 and Figure 5.7 which show the number of monitors per route collector, the total number of monitors under some PRCs is not the same. This is an indication that the geographical locations of the victim and the hijacker matters. We discuss more about the geographic factor and how it affects stealthy hijacks in Section 5.4.



(a) (b)

Figure 5.6: Percent of monitors in experiment sets *(wisc01, grnet01, 3)* and *(amsterdam01, wisc01, 0)*

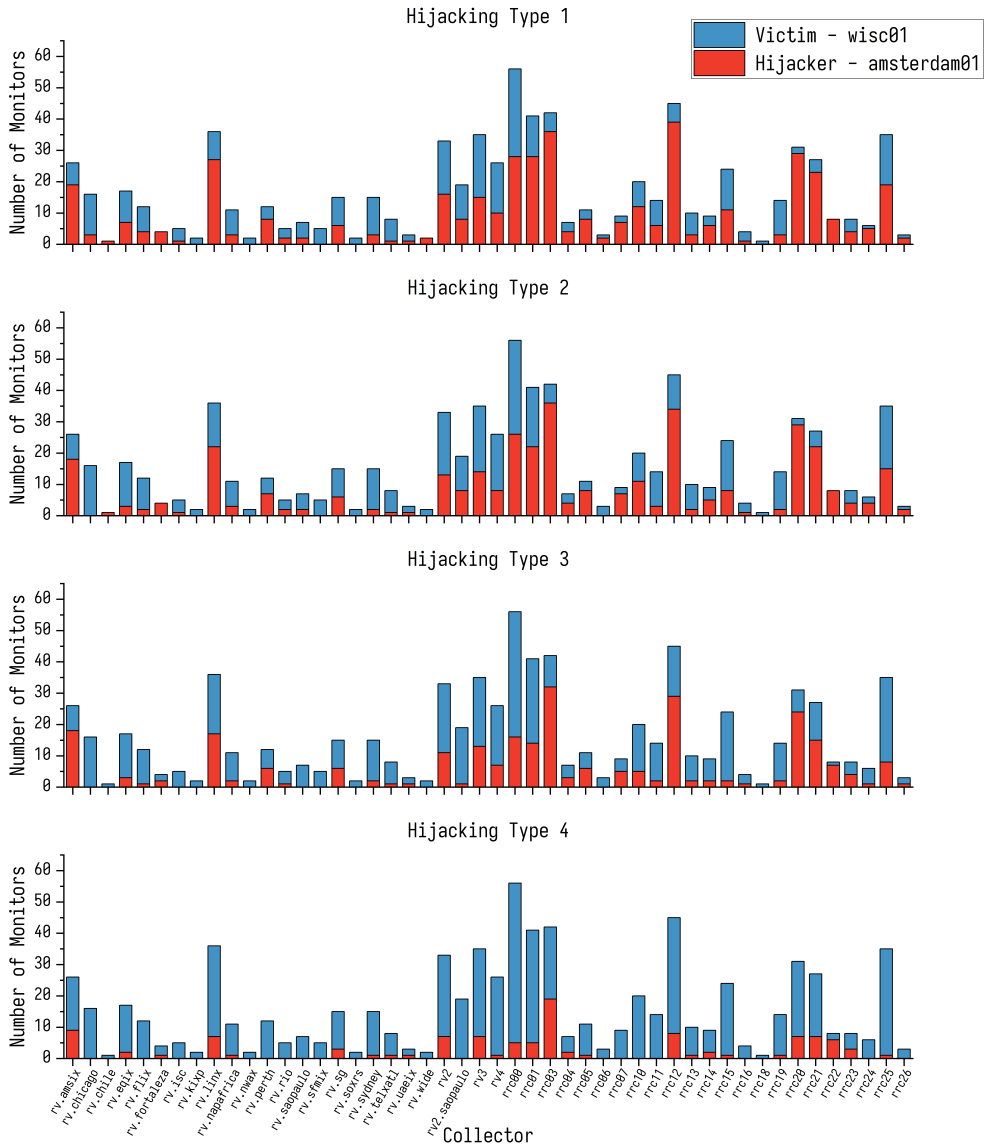


Figure 5.7: Number of monitors in experiment set *(amsterdam01, wisc01, 0)*

5.3 Prediction of Hijack Stealthiness by a Simple Algorithm

With the base experiment unit described in Section 5.2, we obtained the paths of the victim's prefix and the hijacker's prefix in each monitor. In Section 3.4.1 and Section 4.3.2 we presented our plan to use a simple algorithm to evaluate the hijacker's ability to infer the stealthiness of a possible hijack from the AS paths disclosed from route collectors , i.e., to determine whether a hijack would be observed by a monitor if it were announced. Here, we show the two plot results planned from Section 4.3.2: (i) The complete stealthy indicator, and (ii) the ROC space.

First, we focus on the complete stealthy indicator. As described in Section 4.3.2.1, we should fist discuss the effect of infinity. As a reminder, we defined the complete stealthy indicator for each PRC as the minimum number of hijacking types that would allow each monitor to remain stealthy after removing the infinity results from the algorithm. Hence, we make a plot of the results of the complete stealthy indicator after removing infinity for each PRC for every group of our experiments. Furthermore, we also produce plots without the removal of infinity.

Out of our experiments, we have chosen three data sets with the identifiers *(amsterdam01, wisc01, 3)*, *(wisc01, amsterdam01, 3)* and *(wisc01, grnet01, 3)*. Here these three sets of plots represent different experimental sets in which the hijacking is in majority, the hijacking is in minority, and hijacking is from majority to minority as the hijacking type number increases from 1 to 4, respectively. We say the hijacking is in *majority* (*minority*) if the majority of the monitors of report the hijacker (the victim).

The *(amsterdam01, wisc01, 3)* experiment, where the victim is in Wisconsin and the hijacker is in Amsterdam, represents an experiment set where the hijacking is always in the majority, with the number of stacked bars shown in Figure 5.8 and the result of the complete stealthy indicator shown in Figure 5.10. The reverse experiment, *(wisc01, amsterdam01, 3)*, where the hijacker is in Wisconsin and the victim is in Amsterdam, represents an experiment set where the hijacking is always in the minority, with the number of stacked bars shown in Figure 5.9 and the result of the complete stealthy indicator are shown in Figure 5.11. The *(wisc01, grnet01, 3)* experiment, where the victim is situated this time in the Greek Research and Technology Network (grnet), represents an experiment with a shift from majority to minority hijacking, with the number stacked bars shown in Figure 5.5, and

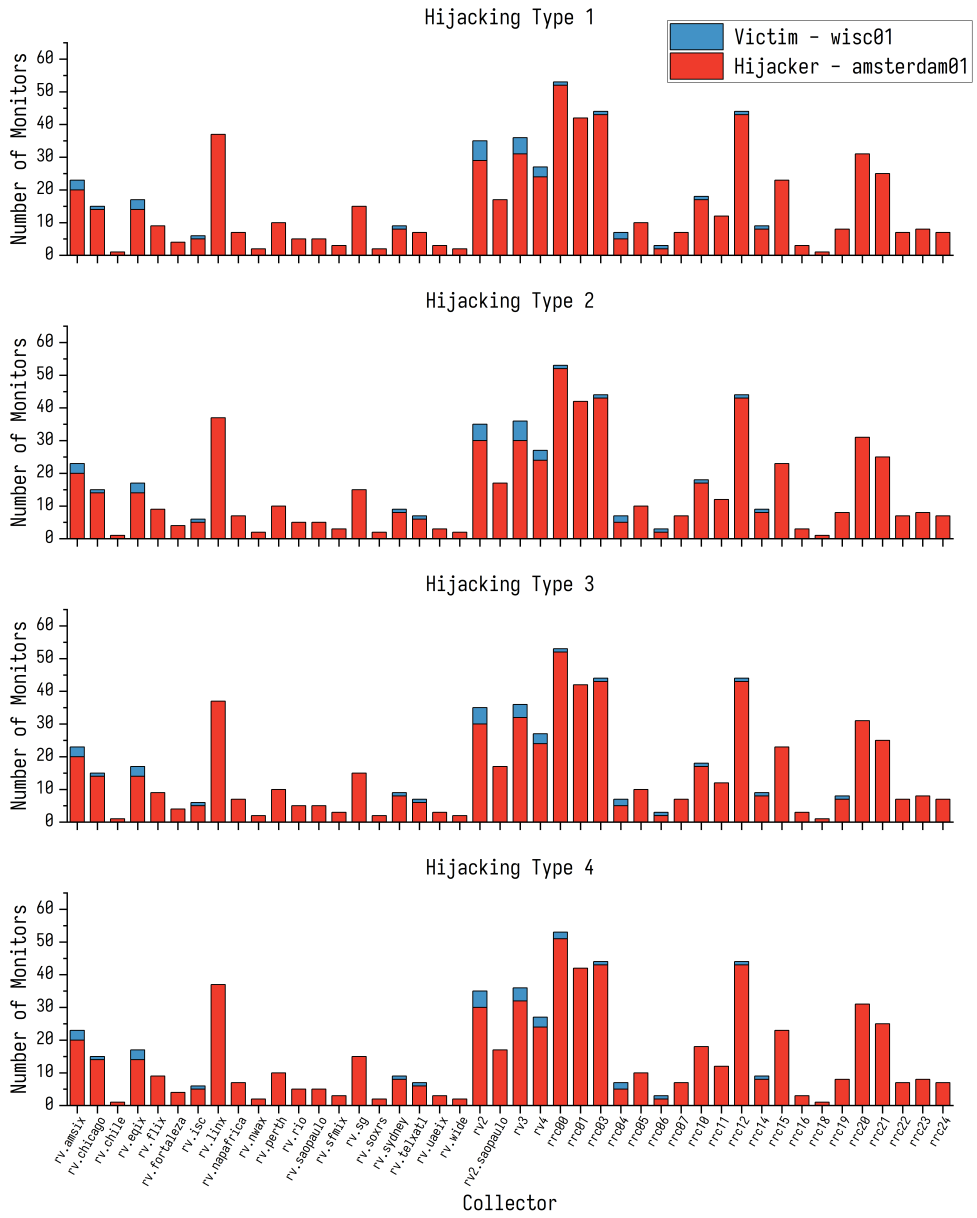


Figure 5.8: Number of monitors in experiment set *(amsterdam01, wisc01, 3)*.

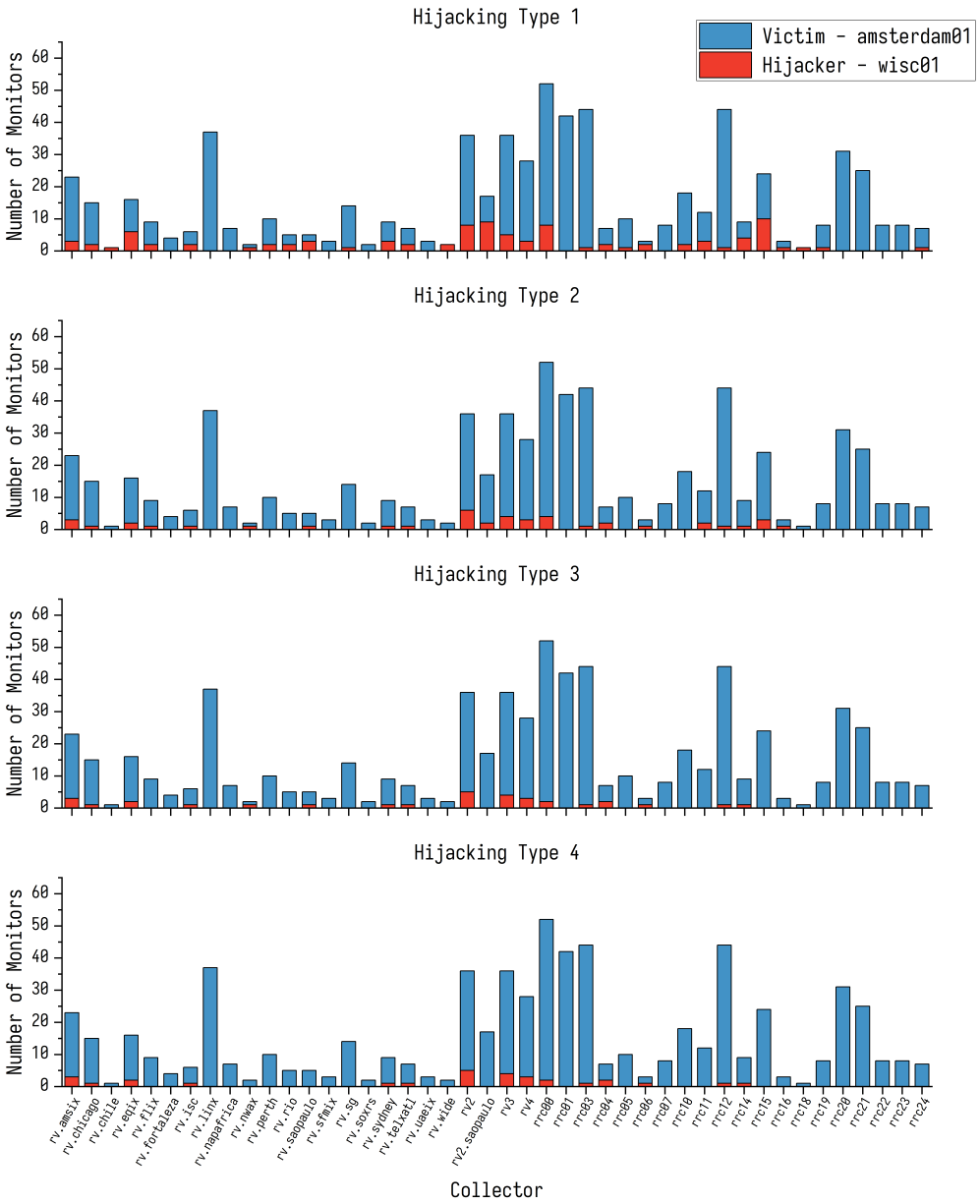


Figure 5.9: Number of monitors in experiment set *(wisc01, amsterdam01, 3)*.

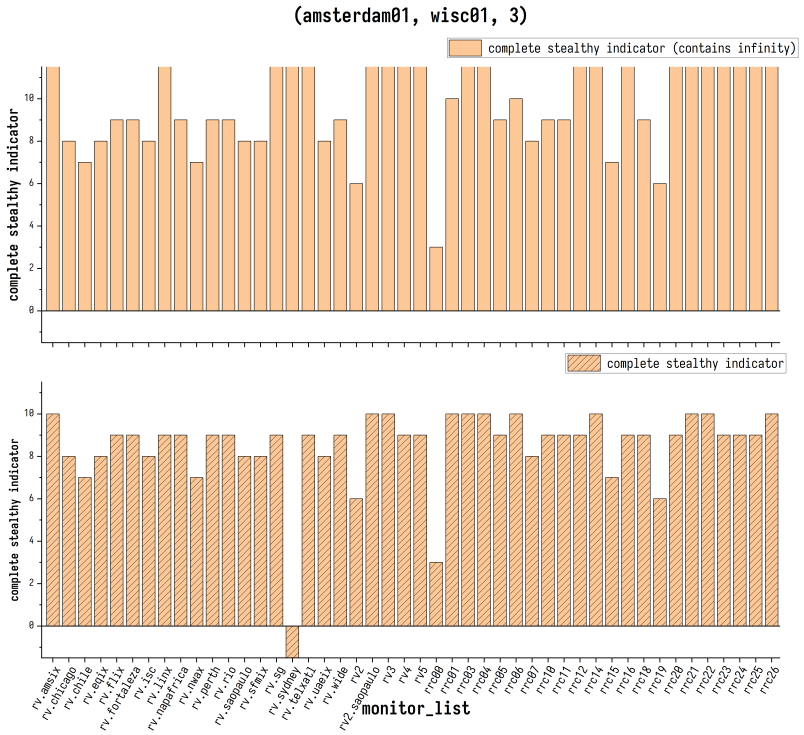


Figure 5.10: Complete stealthy indicator of experiment set *(amsterdam01, wisc01, 3)*. Each bar illustrates the minimum hijacking Type that would allow each monitor to remain stealthy for the corresponding PRC

the result of the complete stealthy indicator shown in Figure 5.12. These three plots of the complete stealthy indicator results have some out-of-range positive and negative value bars in the Y-axis (presented as the maximum Y-axis value 11.5 and the minimum Y-axis value -1.5) as some monitors were computed with an $+\infty$ and a $-\infty$ stealthy indicator value (respectively). An interesting result can be seen from the data of the*rv.sydney* PRC in Figure 5.10, where the result before removing the infinity is $+\infty$, while the result after removing the infinity becomes $-\infty$. This is actually caused due to the fact that the data of *rv.sydney* without the removal of infinity are all $+\infty$. When the infinity is removed, the monitors of *rv.sydney* are all excluded and therefore the visibility is set to the default minimum value $-\infty$.

Observing these figures, we note a very interesting finding. For the complete stealthy indicator plot with the $+\infty$ removed, we see that the maximum stealthy indicator value for both *(wisc01, amsterdam01, 3)* and

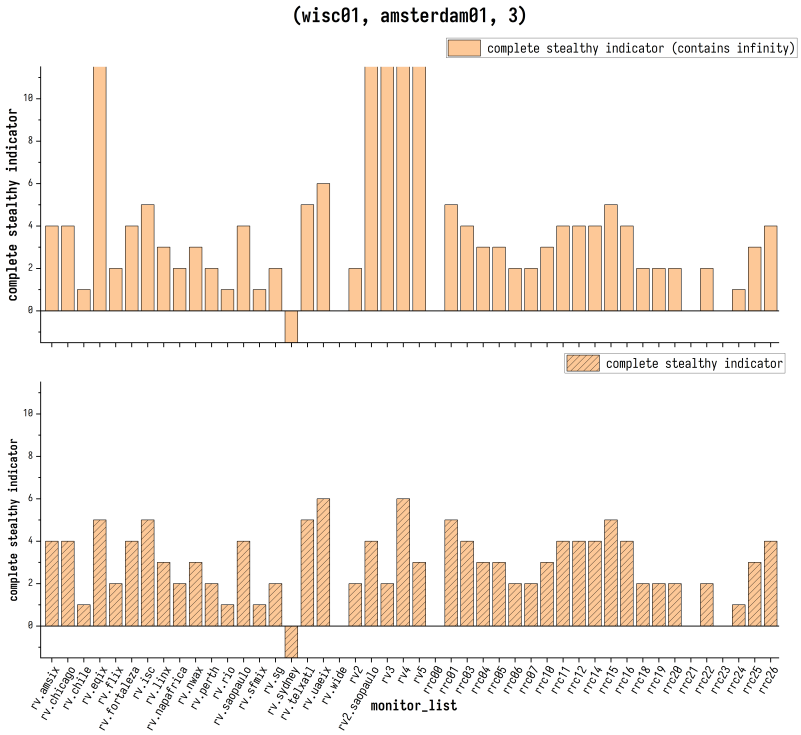


Figure 5.11: Complete stealthy indicator of experiment set *(wisc01, amsterdam01, 3)*. Each bar illustrates the minimum hijacking Type that would allow each monitor to remain stealthy for the corresponding PRC

(wisc01, grnet01, 3) (Figures 5.11 and 5.12 respectively) are the same, *i.e.*, a value of 6. Meanwhile, the maximum value for *(amsterdam01, wisc01, 3)* shown in Figure 5.10 is 10. These results correspond to the previous ones shown in Section 5.2. For Type-4 hijacks, we see that most monitors don't report the hijack in the *(wisc01, amsterdam01, 3)* and in the *(wisc01, grnet01, 3)* experiments (Figures 5.5 and 5.9 respectively). But for Type-4 hijacks in the *(amsterdam01, wisc01, 3)* experiment (Figure 5.8), most monitors observe the hijack. Based on this, we believe that the complete stealthy indicator is somewhat of a guide for hijackers. It attempts to express a worst-case scenario, where the hijacker needs to at least design these hijacking Type numbers to completely hide from the monitors of each route collector, according to our algorithm.

As the complete stealthy indicator corresponds only to the maximum required hijack Type to hide from each route collector, we also show Table 5.1

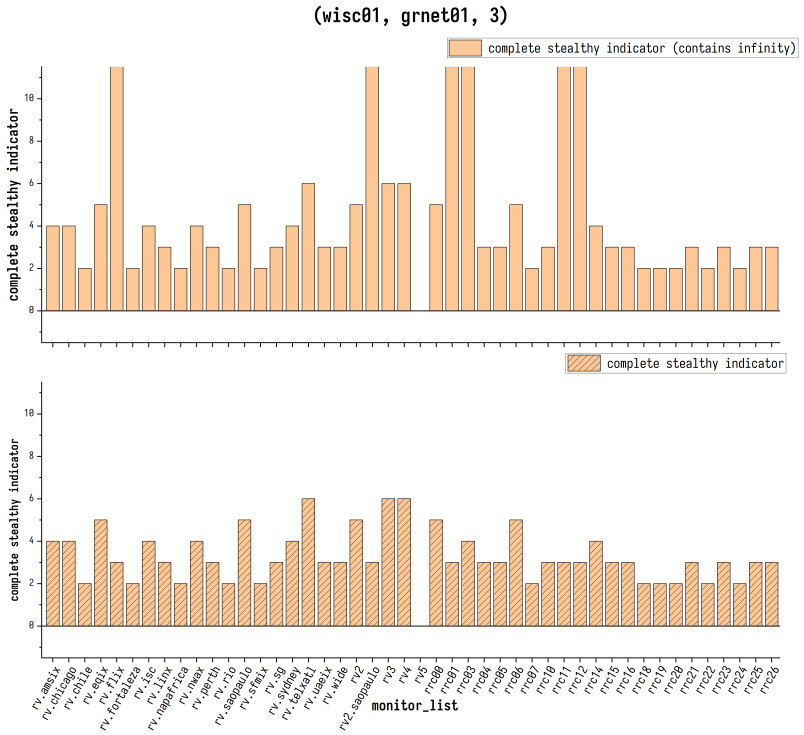


Figure 5.12: Complete stealthy indicator of experiment set *(wisc01, grnet01, 3)*. Each bar illustrates the minimum hijacking Type that would allow each monitor to remain stealthy for the corresponding PRC

which provides more statistics on how to potentially hide from the majority of monitors of some interesting collectors. In this table, statistics are shown (per route collector) for the three hijack experiments *(wisc01, grnet01, 3)*, *(amsterdam01, wisc01, 3)* and *(wisc01, amsterdam01, 3)*. The min (max) represents the hijack Type that needs to be designed to hide from a single (all) the monitors of that route collector. The median, 5%, and 95% percentiles represent the hijacking Type required to hide from 50%, 5%, and 95% of the monitors, respectively (all designed hijacking Types need to be greater than the values presented in the table). Negative Type values indicate the existence of monitors which naturally will not observe the hijack, *e.g.,* because such monitors are closer to the victim. Finally, the "$+\infty$ count" counts how many monitors' have a max (complete stealthy indicator) value of $+\infty$, meaning that the hijack is theoretically unconcealable for them, *e.g.,* because such monitors are closer to the hijacker.

Table 5.1: Some interesting PRC statistics for *(wisc01, grnet01, 3)*, *(amsterdam01, wisc01, 3)* and *(wisc01, amsterdam01, 3)*

collector	min	max	median	5%	95%	$+\infty$ count
(wisc01, grnet01, 3)						
rv.napafrica	2	2	2	2.0	2.0	0
rv.linx	5	7	6	5.6	7.0	0
rrc14	2	4	2	2.0	3.6	0
rrc20	6	7	6	6.0	6.6	0
...	...	...	...	...	...	...
total	-1	6	2	1.0	3.0	6
(amsterdam01, wisc01, 3)						
rv.napafrica	6	9	8	6.3	9.0	0
rv.linx	5	9	7	5.0	9.0	4
rrc14	3	7	7	4.2	7.0	0
rrc20	8	10	9	8.0	10.0	2
...	...	...	...	...	...	...
total	-3	6	7	4.0	9.0	60
(wisc01, amsterdam01, 3)						
rv.napafrica	-1	2	0	-1.0	1.7	0
rv.linx	-1	3	1	-1.0	3.0	0
rrc14	1	5	1	1.0	3.8	0
rrc20	-2	0	-1	-2.0	0.0	0
...	...	...	...	...	...	...
total	-5	6	1	-1.0	3.0	9

We note that for the statistics from all the collectors, maximum value and 95% is the same for *(wisc01, grnet01, 3)* and *(wisc01, amsterdam01, 3)*. This means if we want to hide the hijack from the monitors, we need similar hijacking type number for these two sets. The results in Figure 5.9 and Figure 5.5 show that in both experiments the hijack can be hid from most monitors when the hijacking type number is 4. However, as we notice, their value differs in the 5% percentile. This corresponding to a difference between the two experiments when the hijacking type number is 1 and 2. For *rv.napafrica*, the results are similar. When the Hijacking Type number is 1, *rv.napafrica* (Figure 5.5) shows that most of the monitors reported the hijack while Figure 5.9 does not. In the case of *rv.linx*, this is even more obvious. Larger difference on 5% and 95% values correspond to differences in hijacking Type 1 and Type 2 in Figure 5.5 and Figure 5.9. For *rrc14*, we can focus on *(wisc01, grnet01, 3)* in Figure 5.5 and *(amsterdam01, wisc01, 3)*

in Figure 5.8. When the hijacking type is either 1 or 2, we observe that most monitors report the hijack. This corresponds to the 5% value which is 2.0 and 4.2 (respectively).

We also note that for the *(wisc01, grnet01, 3)* experiment for the route collectors *rv.linx* and *rrc20*, the 5% and 95% percentile values are unusually high. Figure 5.5 shows that when the hijacking type number is greater than 2, most monitors will not report the hijack. However, the values calculated here are greater than 5 for even the 5% percentile, which is not consistent with other experiment results. The value calculated in this experiment by the algorithm is much larger than the other experiment results, which means that some monitors in the algorithm will choose the hijacker for a smaller hijacking type number, while the victim is actually chosen in fact. The algorithm appears to be more aggressive in inferring whether the hijacking will be detected or not. In fact, we notice drawbacks of this simple algorithm also through the performance in the ROC space.

Figure 5.13 shows the distribution of the three experiments in the ROC space. We treat our algorithm as a classifier, with whether a monitor can observe the hijack or not as an judging instance. Given a hijacking type number, the algorithm can give a judgment based on the data from the base experiment units (classifier input) as to whether this hijacking will be observed for a given monitor (classifier output). Meanwhile, we have the hijacking experiment units as true instances (ground truth). So we can check the number of False Positive with all negative instances. An interesting finding is that in each curve the FPR and TPR decreases as the hijacking type number increases from 1 to 4 (in most circumstances). Looking at the graph, the points with a hijacking type number of 1 are on the far right, whereas the points with a hijacking type number of 4 are on the far left, with the distribution of points going from right to left as the hijacking type number increases throughout the curve. This simple algorithm classifier has different results for different hijacking type numbers. It shows that both FPR and TPR decrease as the number of hijacking types increases. However, this may be related to the fact that we have not implemented a fully stealthy hijack. In terms of the complete stealthy indicator, none of our hijacking type numbers are of a size that would allow the hijack to theoretically hide from all the monitors. A possible assumption is that the ROC performance of the classifier may be related to the distance between the hijacking type number of the operation and the complete stealthy indicator. Overall, the classifier results are somewhat instructive, but not really practical.

Another notable finding is that the results for *(wisc01, amsterdam01, 3)* are

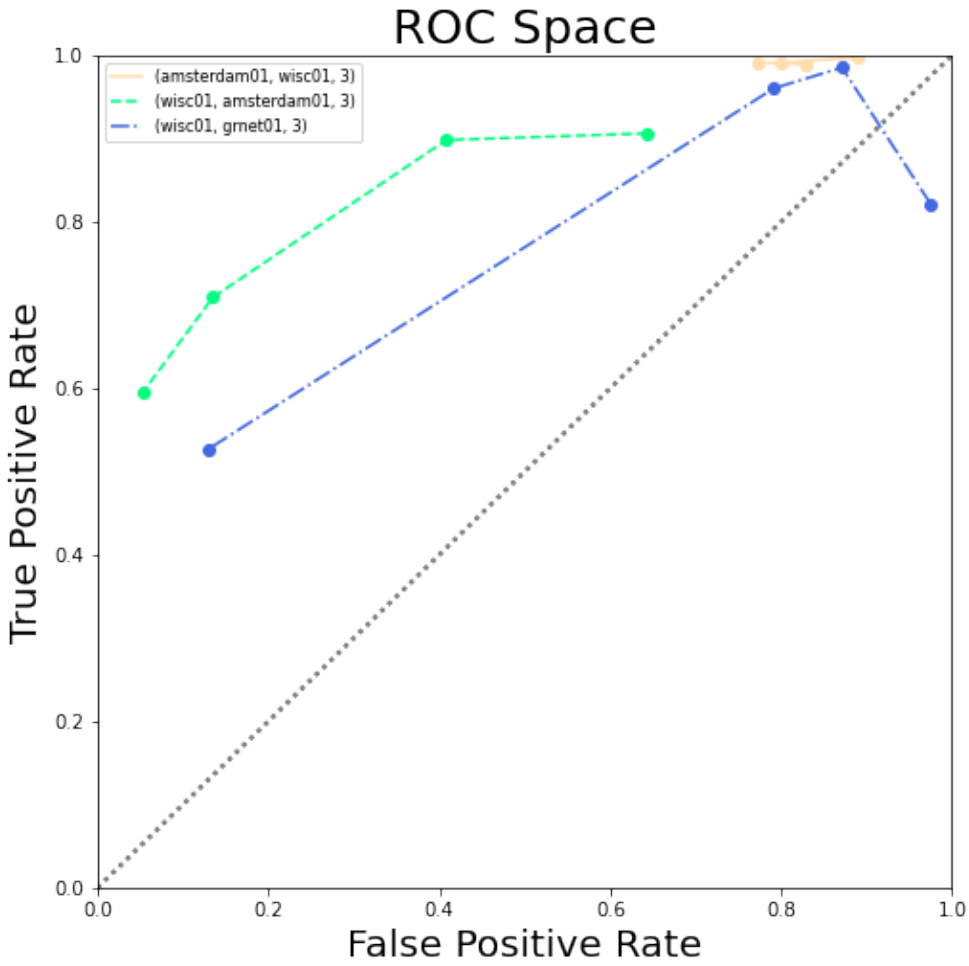


Figure 5.13: Simple algothrim accuracy in ROC space

basically on the left side, the results for *(amsterdam01, wisc01, 3)* is basically on the top right side, while the results for *(wisc01, grnet01, 3)* are in between as we increase the hijacking type number. This means that, in general, this algorithm performs *(wisc01, amsterdam01, 3)* better than *(wisc01, grnet01, 3)*. And it performs worst in *(amsterdam01, wisc01, 3)* among these three. One obvious problem is that we can see that the FPR is over-performing when the TPR is at an acceptably high level. This means that the algorithm is “overestimate" in our experiments. It is possible that for some monitors, the output of the classifier is that the monitor will report the hijack, but in the results the monitor does not actually report. This systematic difference could be the result of strategies within each ASes in real networks, or it could be

that an algorithm that infers visibility based on As-path prepending does not to accurately describe the state of connectivity in real networks very well. Again, from these results, we clearly observe that the closer the hijacking type number is to the complete stealthy indicator, the higher performance of this algorithm. Our hijacking type number is limited by the peering testbed to a maximum of 4, but if we can increase the hijacking type number to be closer to the complete stealthy indicator of *(amsterdam01, wisc01, 3)* and *(wisc01, grnet01, 3)*, the algorithm may perform better. However, this is simple a conjecture as we are unable to produce empirical evidence at this time.

5.4 The Impact of Meta Parameters in Hijacking

The first meta parameter that we analyze is the factor of geographical location for the possibility of hijacking. In fact, the typical cases we have chosen in Section 5.2 and Section 5.3 are also typical results of the geographical location factor. The *(wisc01, amsterdam01, 3)* represented in Figure 5.9 and the *(amsterdam01, wisc01, 3)* represented by Figure 5.8 have significantly different results in the histogram of the number of monitors and in the ROC curve. The results show that *amsterdam01* is performing more “aggressively" than *wisc01* for monitors connected to the PRC. The word “aggressively" here means that this location may be possibly more attractive for monitors. The monitors are also the PRC’s peers, which means that the monitors also have some certain characteristics in terms of geographical location. When a hijacker announced a hijack at *amsterdam01*, we observe it is more difficult for the hijacker to hide from PRC; conversely, it is much easier for the hijacker to hide from PRC if the victim announces its prefix from the *amsterdam01*. This may be due to the logical more central position of amsterdam01 in the Internet, or it may be that *amsterdam01* is in closer position to the monitors. On the other hand, we also see that *(wisc01, grnet01, 3)* represented by Figure 5.5 is based on moving the victim to a less “aggressively" position, which cause a very different results comparing to Figure 5.9. This means that the invisibility of the hijacker can vary to a greater extent with the hijacking type number.

The second meta parameter that we analyze is the impact caused to hijacking by the victim’s AS-path prepending. To compare the impact of AS-path prepending, we use the results of three experimental sets *(amsterdam01, wisc01, 2)*, *(amsterdam01, wisc01, 1)*, and *(amsterdam01, wisc01, 0)*. The numbers 2, 1, and 0 in each experiment set represent the number of prepended

ASes by the victim. A histogram of the plotted percentage stack is shown in Figure 5.14. To reduce the image size, we only present visibility results for PRCs where the total number of monitors is more than 15. Each group of bars consists of three bars from left to right, which show the visibility of the hijack for the victim's prepending (ASes prepended: two to zero from from left to right). It is obvious from the graph that the percentage of monitors that observe the hijacking decreases as the victim prepending number decreases. This is the case for the hijacking experiments with Type numbers 2, 3 and 4. Interestingly, we notice that the visibility changes only a little for hijacking experiments where the hijacking type number is 1. For example, under a hijacking Type number of 1 for the *rv.amsix* route collector, 88% of the monitors observe the hijack with a victim prepending of one AS which is a little better than the 85% with a victim prepending of two ASes. As can be seen the difference between the two is small, which makes it difficult to determine the cause of this phenomenon. We consider two possible causes for this increased visibility: an excessively long victim prepending number triggers some AS strategy that has some reverse effect, or that the data fluctuates simply because of network changes between experiment sets. There may also exist the possibility that there is a threshold for the effect of increasing the victim prepending number to reduce hijacking stealthiness. This is because in data with a hijacking type number of 1 and victim prepending numbers of 2 and 3, the hijacker is visible enough, but some PRCs still retain around 15% monitors from observing the hijacking (e.g. *rv.asix*). This means that victims who resort to increasing the victim prepending number in an attempt to expose the hijacker may not be effective for some specific monitors, and vice versa hijackers who prepend the As-path may not be effective in hiding the hijack from some specific monitors. Again, we suspect that this may be due to network topology and AS routing policies.

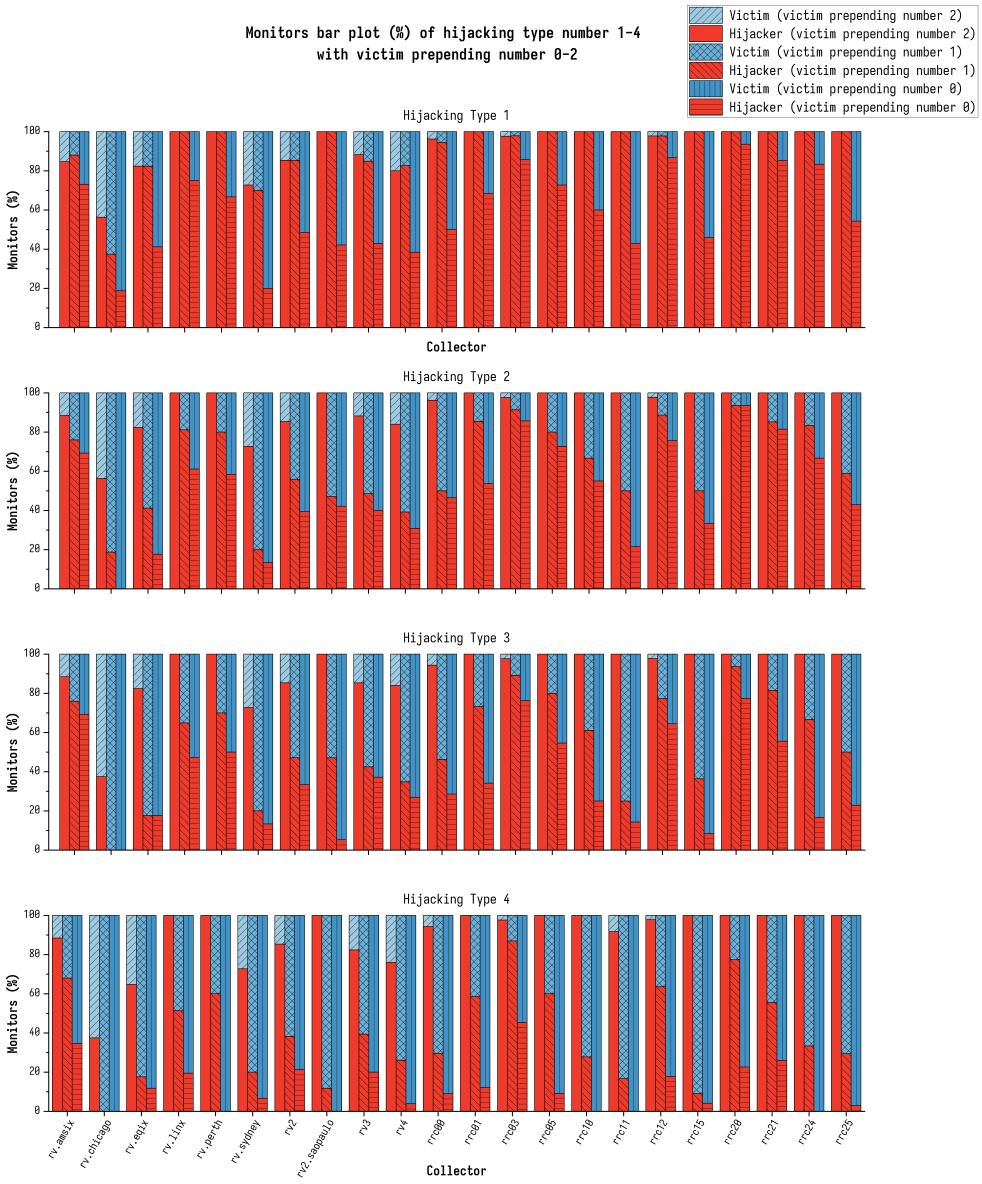


Figure 5.14: Hijack visibility results for the AS-path prepending done by the victim in the experiment sets *(amsterdam01, wisc01, 2)*, *(amsterdam01, wisc01, 1)*, and *(amsterdam01, wisc01, 0)*. The numbers 2, 1, and 0 represent the number of ASes prepended.

5.5 Structure of the Topology and the Presence of Key Nodes

As described in Section 3.4.3, when we focus on single experiment units, we try to visualize the AS paths reported. Here we build up two different types of topology: a network graph and a tree graph. Here we show an experiment unit with a hijacking type 2 in experiment set *(wisc01, grnet01, 3)*. As shown in Figure 5.5, for a hijacking type number is 2, there are a fair number of monitors that either observe or do not observe the hijacker, so this is a good topology.

To create the network graph, we simply connect the ASNs in the AS path. If the monitor reports a path containing the hijacker, we set every AS in the path as red to indicate the hijacked ASes. Otherwise, If the monitor reports instead the valid path announced by the victim, we set every AS in the path as blue to indicate unaffected ASes. We represent all ASes observed in the AS-paths as nodes in the graph. If the paths through this node are all blue, we set this node to blue. If the paths through this node are all red, we set this node to red. In particular, if a path through a node has both blue and red, indicating a partially hijacked AS, we set said nodes to green. Here we find an interesting fact: there are nodes that are involved in building both the AS path to the hijacker and the AS path to the victim. As shown in Figure 5.15, we can see that there exists some green nodes and we zoomed in on the node with ASN 174 for easier observation.

For the tree graph, we want to build up a tree graph just like the one we shown in Figure 2.7. We combined the same nodes appearing in different paths into one and created a tree graph based on the relationships before and after in the path. As we found in network graph, some nodes will appear in both paths to victim and hijacker, so one node may appear in two different branches. As Figure 5.16 shows, we clearly observe the tree structure and the zoomed node with ASN 174 in two branches. Here we also find another interesting fact: for both paths to the victim and to the hijacker, there actually exists some nodes near the root. If one hijacker could make such key nodes to not choose the hijacking path, this could potentially be another way of creating more stealthy hijacks.

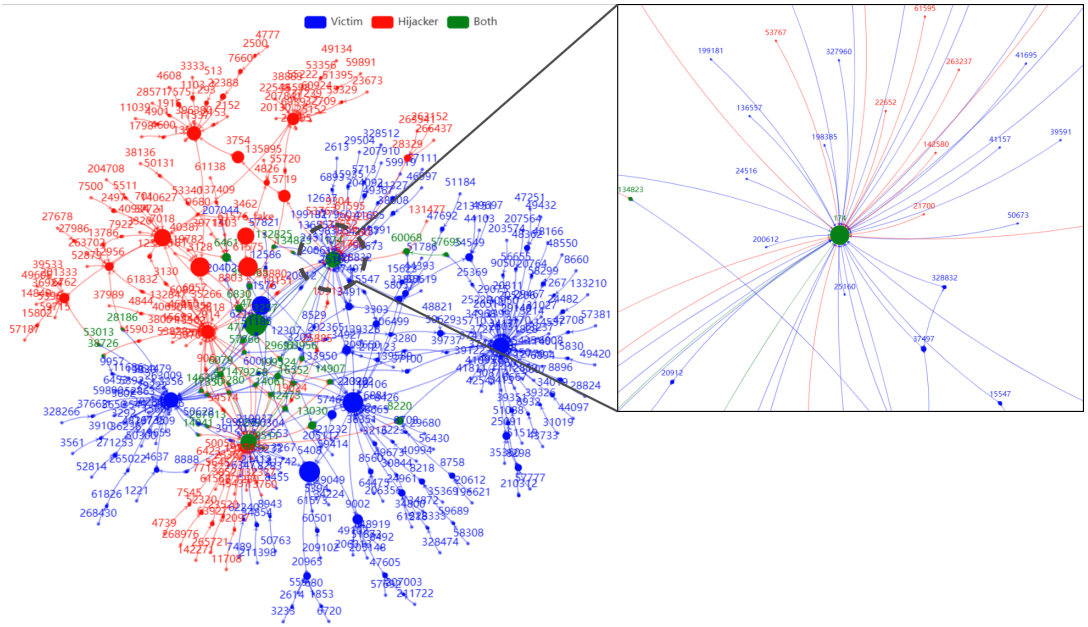


Figure 5.15: AS path topology net graph for experiment unit with hijacking type 2 in *(wisc01, grnet01, 3)*

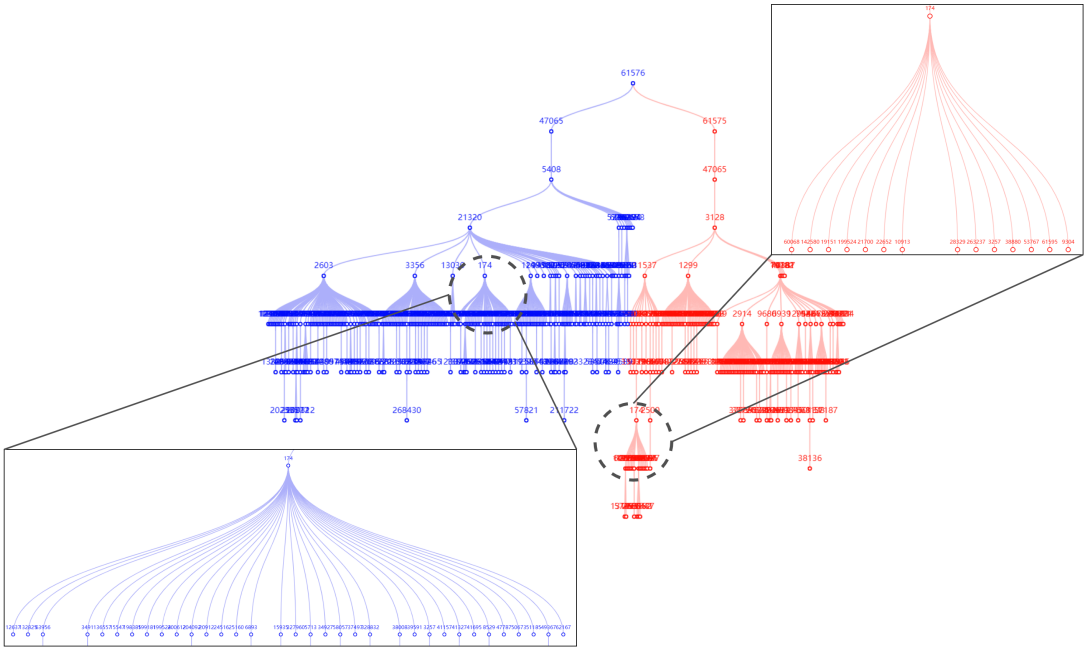


Figure 5.16: AS path topology tree graph for experiment unit with hijacking type 2 in *(wisc01, grnet01, 3)*

5.6 Reliability Analysis

Because of the time scale (each experiment set taking more than 8 hours), we were unable to consider the experiment sets from repeated experiments as the same data since the network will change with time passing by. However, we observed the same patterns and small fluctuations in the data in the experiment sets, so we believe that our experiment is reliable. The data collection was implemented using BGPStream, and the whole process is reproducible. The process of parsing and building our own database is also reproducible.

5.7 Validity Analysis

Peering Testbed was used to declare and withdraw BGP messages during our experiments. During the experiments, our BGP announcements and withdrawals were all confirmed by collecting observations from the grnet looking glass [37]. Data collection was implemented through the Python library provided by BGPStream. The data shown in the article are all fetched from a full run of the program. The relevant fields of our own database have also been manually tested to ensure that there are no errors.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The aim of this thesis is to investigate whether hijackers can use traffic engineering methods to reduce the visibility of hijacks to monitors, then finally try to hide from most PRC. The traffic engineering method we use in this thesis is ASPP. For this purpose, we have divided the project into three subgoals. By building experiments in real networks using platforms, such as Peering Testbed, we give answers to our experimental goals. For our subgoal 1, our results shown in Section 5.2 indicate that the use of ASPP is helpful in improving the stealthiness of hijacking. Then, for subgoal 2, while we observe that designing an algorithm for inferring the stealthiness of hijacking from the routes reported by PRC could be feasible since ASPP indeed improve the stealthiness for some monitors, our proposed simple algorithm is not very practical. Finally, we conclude that geographical location is important for the stealthiness of hijacking. The victim's traffic engineering through ASPP has also an important impact on stealthiness.

From the conclusions that we reached, we also offer conjectures about some phenomena. If victims actively use ASPP to expose stealthy hijacking, it may not be effective for some monitors potentially due to local device policies of ASes in real networks. Throughout the data we collected, we did not achieve a fully stealthy hijack due to the Peering Testbed which restricted the type number of hijacks to no more than 4. Despite the existence of experimental sets where only a few monitors originally observed the hijacking, at the end there were still PRCs that were able to observe the hijacking. We speculate that under our experimental conditions, the hijacking type number may have to reach a greater value (*i.e.,* ≥ 7) if a fully stealthy hijack is to be achieved.

6.2 Limitations

There are many limitations to our experiments, mainly due to limitations in the Peering Testbed platform. Firstly, the Peering Testbed platform limited our hijacking type number that we could announce to less than or equal to 4, which resulted in us not being able to create a fully stealthy hijack experiment. Secondly, our exploration of meta-parameter was also limited by the options available in the Peering Testbed. Finally, the Peering Testbed provides limited prefixes and ASNs, and we could only design experiments serially by constructing individual experiment sets rather than in parallel experiment sets.

6.3 Future work

Future work could be approached in two directions: breaking the limits from the platform and combining ASPP with other methods. The fact that we did not construct a fully stealthy hijack in our experiments may be due to the limitations of our experiments. If we can break the restriction on the hijacking type number, we may be able to achieve a fully stealthy hijacking in our experiments. We also note that the hijacking type 4 hijacking still does not result in a fully stealthy hijacking, which may mean that stealthy hijacking in general is not feasible through ASPP alone in the real Internet. More sophisticated stealthy hijackings may be able to be designed by combining other traffic engineering methods, such as AS path poisoning. Also, at the analytical level, we could probably go further. The algorithm we have implemented is simply a comparison of the length of AS paths. More advanced algorithms, such as algorithms using machine learning, could produce better results. Furthermore, by combining network topology and some other meta-information (e.g. AS rank), we may be able to control the stealthiness of hijacking from a higher dimension. As shown in Section 5.5, hijackers may be able to poison some specific AS to hide from large number of monitors.

6.4 Sustainability and Ethics

From the perspective of sustainable development, it is helpful to study BGP hijacking to promote cooperation between different Internet entities and infrastructure construction. BGP is a decentralized network protocol, and each network entity has its own privacy and strategy. The occurrence of hijacking

can mean a conflict between multiple network entities. It's hard to know whether conflicts are the result of accidental misconfiguration or intentional, but we can try to reduce and prevent potential conflicts. From this perspective, PRC are necessary. At the same time, PRC as the hijacking detection infrastructure, can become a supporting industry for Internet construction.

From an ethical point of view, our experiments use ASN and prefixes that we applied, so there is no risk of harm to the public. However, further research may have ethical implications. For example, poisoning existing AS on the Internet can have serious consequences if researcher wants to try BGP AS path poisoning.

6.5 Reflections

The aim of this project is to increase the stealthiness of hijacking, which ultimately leads to the hiding from PRCs. Researchers can build a more sophisticated hijacking or analysis algorithm on the basis of this research, or they can further explore how to better deploy more PRCs for such problems. Although the experiments are in real networks, the ASNs and prefixes of both the hijacker and the victim are requested for the experiments, and no other operations at the data layer (traffic black holes, etc.) are performed in the experiments. The data sources for the experiments were public data from public platforms, and the APIs used to build the project were also publicly provided by several projects. Therefore, this project does not raise public risks or ethical hijacking issues.

References

[1] P. Sermpezis, V. Kotronis, P. Gigis, X. Dimitropoulos, D. Cicalese, A. King, and A. Dainotti, "ARTEMIS: Neutralizing BGP Hijacking within a Minute," Jun. 2018, arXiv:1801.01085 [cs]. [Online]. Available: http://arxiv.org/abs/1801.01085 [Pages 1, 9, and 19.]

[2] "About PEERING | PEERING - The BGP Testbed." [Online]. Available: https://peering.ee.columbia.edu/ [Pages 2, 17, and 19.]

[3] A. Siddiqui, "Not just another BGP Hijack," Apr. 2020. [Online]. Available: https://www.manrs.org/2020/04/not-just-another-bgp-hijack/ [Page 2.]

[4] S. Kent, C. Lynn, and K. Seo, "Secure Border Gateway Protocol (S-BGP)," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4, pp. 582–592, Apr. 2000. doi: 10.1109/49.839934 Conference Name: IEEE Journal on Selected Areas in Communications. [Pages 2, 11, and 18.]

[5] M. Lepinski and S. Kent, "An Infrastructure to Support Secure Internet Routing," Internet Engineering Task Force, Request for Comments RFC 6480, 2012, num Pages: 24. [Online]. Available: https://datatracker.ietf.org/doc/rfc6480 [Pages 2 and 11.]

[6] R. Lychev, S. Goldberg, and M. Schapira, "Bgp security in partial deployment: Is the juice worth the squeeze?" *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, p. 171–182, aug 2013. doi: 10.1145/2534169.2486010. [Online]. Available: https://doi.org/10.1145/2534169.2486010 [Page 2.]

[7] "NIST RPKI Monitor." [Online]. Available: https://rpki-monitor.antd.nist.gov/ [Page 2.]

[8] E. Gregori, A. Improta, L. Lenzini, L. Rossi, and L. Sani, "On the incompleteness of the as-level graph: A novel methodology for bgp route collector placement," in *Proceedings of the 2012 Internet Measurement Conference*, ser. IMC '12. New York, NY, USA: Association for Computing Machinery, 2012. doi: 10.1145/2398776.2398803. ISBN 9781450317054 p. 253–264. [Online]. Available: https://doi.org/10.1145/2398776.2398803 [Page 3.]

[9] A. Milolidakis, "Understanding the capabilities of route collectors to observe stealthy hijacks: Does adding more monitors or reporting more paths help?" Ph.D. dissertation, KTH Royal Institute of Technology, 2022. [Page 4.]

[10] J. Scudder, R. Fernando, and S. Stuart, "BGP Monitoring Protocol (BMP)," RFC 7854, Jun. 2016. [Online]. Available: https://www.rfc-editor.org/info/rfc7854 [Pages 5, 12, and 18.]

[11] Y. Rekhter, S. Hares, and T. Li, "A Border Gateway Protocol 4 (BGP-4)," Internet Engineering Task Force, Request for Comments RFC 4271, 2006, num Pages: 104. [Online]. Available: https://datatracker.ietf.org/doc/rfc4271 [Pages 7 and 8.]

[12] L. Gao and J. Rexford, "Stable internet routing without global coordination," *IEEE/ACM Transactions on Networking*, vol. 9, no. 6, pp. 681–692, 2001. doi: 10.1109/90.974523 [Page 9.]

[13] J. Karlin, S. Forrest, and J. Rexford, "Pretty good bgp: Improving bgp by cautiously adopting routes," in *Proceedings of the 2006 IEEE International Conference on Network Protocols*, 2006. doi: 10.1109/ICNP.2006.320179 pp. 290–299. [Pages 11 and 19.]

[14] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (s-bgp)," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4, pp. 582–592, 2000. doi: 10.1109/49.839934 [Page 11.]

[15] P.-A. Vervier, O. Thonnard, and M. Dacier, "Mind Your Blocks: On the Stealthiness of Malicious BGP Hijacks." in *NDSS*, 2015. [Page 11.]

[16] C. Testart, P. Richter, A. King, A. Dainotti, and D. Clark, "Profiling bgp serial hijackers: Capturing persistent misbehavior in the global routing table," in *Proceedings of the Internet Measurement Conference*, ser. IMC '19. New York, NY, USA: Association for Computing Machinery, 2019.

doi: 10.1145/3355369.3355581. ISBN 9781450369480 p. 420–434. [Online]. Available: https://doi.org/10.1145/3355369.3355581 [Page 11.]

[17] "Quagga Software Routing Suite." [Online]. Available: https://www.nongnu.org/quagga/ [Page 12.]

[18] C. Orsini, A. King, D. Giordano, V. Giotsas, and A. Dainotti, "Bgpstream: A software framework for live and historical bgp data analysis," in *Proceedings of the 2016 Internet Measurement Conference*, ser. IMC '16. New York, NY, USA: Association for Computing Machinery, 2016. doi: 10.1145/2987443.2987482. ISBN 9781450345262 p. 429–444. [Online]. Available: https://doi.org/10.1145/2987443.2987482 [Pages 12 and 18.]

[19] P. Marcos, L. Prehn, L. Leal, A. Dainotti, A. Feldmann, and M. Barcellos, "AS-Path Prepending: There is no rose without a thorn," in *IMC '20*. ACM, 2020. doi: 10.1145/3419394.3423642. ISBN 978-1-4503-8138-3 pp. 506–520, accepted: 2021-03-17T00:14:14Z. [Online]. Available: https://researchcommons.waikato.ac.nz/handle/10289/14183 [Pages 14 and 19.]

[20] "BGPStream." [Online]. Available: https://bgpstream.caida.org/ [Pages 17 and 18.]

[21] B. Schlinker, T. Arnold, I. Cunha, and E. Katz-Bassett, "Peering: Virtualizing bgp at the edge for research," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, ser. CoNEXT '19. New York, NY, USA: Association for Computing Machinery, 2019. doi: 10.1145/3359989.3365414. ISBN 9781450369985 p. 51–67. [Online]. Available: https://doi.org/10.1145/3359989.3365414 [Page 17.]

[22] "Route Views – University of Oregon Route Views Project." [Online]. Available: https://www.routeviews.org/routeviews/ [Pages 18 and 19.]

[23] "Routing Information Service (RIS) Raw Dataset." [Online]. Available: https://labs.ripe.net/datarepository/data-sets/routing-information-service-ris-raw-dataset/ [Pages 18 and 19.]

[24] B. Kumar, "Integration of security in network routing protocols," *SIGSAC Rev.*, vol. 11, no. 2, p. 18–25, apr 1993. doi:

10.1145/153949.153953. [Online]. Available: https://doi.org/10.1145/153949.153953 [Page 18.]

[25] B. Smith, S. Murthy, and J. Garcia-Luna-Aceves, "Securing distance-vector routing protocols," in *Proceedings of SNDSS '97: Internet Society 1997 Symposium on Network and Distributed System Security*, 1997. doi: 10.1109/NDSS.1997.579225 pp. 85–92. [Page 18.]

[26] B. Smith and J. Garcia-Luna-Aceves, "Securing the border gateway routing protocol," in *Proceedings of GLOBECOM'96. 1996 IEEE Global Telecommunications Conference*, vol. MiniConfInternet, 1996. doi: 10.1109/GLOCOM.1996.586129 pp. 81–85. [Page 18.]

[27] OpenBMP, "OpenBMP · OpenBMP Documentation," Mar. 2021. [Online]. Available: https://www.openbmp.org/ [Page 18.]

[28] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. Wu, and L. Zhang, "Detection of invalid routing announcement in the internet," in *Proceedings International Conference on Dependable Systems and Networks*, 2002. doi: 10.1109/DSN.2002.1028887 pp. 59–68. [Page 19.]

[29] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur, "Topology-Based Detection of Anomalous BGP Messages," in *Recent Advances in Intrusion Detection*, G. Vigna, C. Kruegel, and E. Jonsson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. ISBN 978-3-540-45248-5 pp. 17–35. [Page 19.]

[30] "PEERING client controller," May 2022. [Online]. Available: https://github.com/PEERINGTestbed/client [Page 35.]

[31] "BGPStream Structure." [Online]. Available: https://bgpstream.caida.org/docs [Page 38.]

[32] "PyBGPStream Elems." [Online]. Available: https://bgpstream.caida.org/docs/api/pybgpstream/pybgpstream.html [Page 40.]

[33] "MongoDB: The Developer Data Platform." [Online]. Available: https://www.mongodb.com [Page 40.]

[34] "PyMongo — MongoDB Drivers." [Online]. Available: https://www.mongodb.com/docs/drivers/pymongo/ [Page 40.]

[35] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006. doi: https://doi.org/10.1016/j.patrec.2005.10.010 ROC Analysis in Pattern Recognition. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016786550500303X [Pages 44 and 45.]

[36] J. A. Swets, "Measuring the accuracy of diagnostic systems," *Science*, vol. 240, no. 4857, pp. 1285–1293, 1988. doi: 10.1126/science.3287615. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.3287615 [Page 44.]

[37] "GRNET Looking Glass." [Online]. Available: https://mon.grnet.gr/lg/ [Page 69.]

TRITA-EECS-EX- 2022:00

For DIVA

{
"Author1": { "Last name": "Wang",
"First name": "Kunyu",
"Local User Id": "u1n8y8ja",
"E-mail": "kunyuw@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
}
},
"Cycle": "2",
"Course code": "DA246X",
"Credits": "30.0",
"Degree1": {"Educational program": "Master's Programme, Communication Systems, 120 credits"
,"programcode": "TCOMM"
,"Degree": "Master degree"
,"subjectArea": "Information and Communication Technology"
},
"Title": {
"Main title": "Investigating the Effectiveness of Stealthy Hijacks against Public Route Collectors",
"Subtitle": "Is AS-Path Prepending Enough to Hide from Public Route Collectors?",
"Language": "eng" },
"Alternative title": {
"Main title": "Undersökning av effektiviteten hos smygande kapningar mot offentliga ruttinsamlare",
"Subtitle": "Är AS-Path Prepending tillräckligt för att dölja från offentliga ruttinsamlare?",
"Language": "swe"
},
"Supervisor1": { "Last name": "Milolidakis",
"First name": "Alexandros",
"Local User Id": "u1wjyoh1",
"E-mail": "miloli@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
"L2": "DIVISION OF SOFTWARE AND COMPUTER SYSTEMS" }
},
"Examiner1": { "Last name": "Chiesa",
"First name": "Marco",
"Local User Id": "u18vjbx4",
"E-mail": "mchiesa@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
"L2": "DIVISION OF SOFTWARE AND COMPUTER SYSTEMS" }
},
"National Subject Categories": "10201, 10206",
"Other information": {"Year": "2023", "Number of pages": "xvii,81"},
"Series": { "Title of series": "TRITA-EECS-EX" , "No. in series": "2022:00" },
"Opponents": { "Name": "Yulian Luo"},
"Presentation": { "Date": "2023-02-08 11:00"
,"Language":"eng"
,"Room": "via Zoom https://kth-se.zoom.us/j/64204203876"
,"City": "Stockholm" },
"Number of lang instances": "2",
"Abstract[eng]": €€€€

BGP hijacking is a threat to network organizations because traditional BGP protocols were not designed with security in mind. Currently, research to combat hijacking is being done by detecting hijacking in real time from Public Route Collectors. However, by using AS-Path Prepending, a well-known traffic engineering technique, hijackers could adjust the influence scope of hijacks to potentially avoid Public Route Collectors. This thesis investigates fist, whether AS-Path Prepending is sufficient to hide from Public Route Collector, and second whether the hijacker can predict its hijack's stealthiness by simply comparing the AS path length with the victim. Last, we investigate the non-hijacker-controlled parameters, which are the geographical locations and victim prepending times if the victim also enable AS-Path Prepending for traffic engineering in our study. Our results show that on one hand, AS-Path Prepending benefits stealthy hijacks to route collectors. While on the other hand, it is not sufficient to completely hide from route collectors only using it. By simply comparing the AS paths length, the hijacker's prediction is constructive but not practical. And non-hijacker-controlled parameters indeed can significantly affect the stealthiness of hijacking.

€€€€,
"Keywords[eng]": €€€€
BGP, BGP Hijack, Stealthy IP prefix hijacking, AS-Path Prepending, BGP monitoring €€€€,
"Abstract[swe]": €€€€

BGP-kapning är ett hot mot nätverksorganisationer eftersom traditionella BGP-protokoll inte har utformats med säkerheten i åtanke. För närvarande bedrivs forskning för att bekämpa kapning genom att upptäcka kapning i realtid från offentliga ruttinsamlare. Genom att använda AS-Path Prepending, en välkänd trafikteknik, kan kapare dock justera kapningarnas inflytande för att eventuellt undvika offentliga ruttinsamlare. I den här avhandlingen undersöks för det första om AS-Path Prepending är tillräckligt för att dölja sig för Public Route Collector och för det andra om kaparen kan förutsäga hur smygande kapningen är genom att helt enkelt jämföra AS Path-längden med offrets. Slutligen undersöker vi de parametrar som inte kontrolleras av kaparen, dvs. geografiska platser och offrets prependingtider om offret också aktiverar AS-Path Prepending för trafikteknik i vår studie. Våra resultat visar att AS-Path Prepending å ena sidan gynnar smygande kapningar av ruttinsamlare. Å andra sidan räcker det inte för att helt och hållet dölja sig för ruttinsamlare om man bara använder det. Genom att helt enkelt jämföra AS-vägarnas längd är kaparens förutsägelser konstruktiva men inte praktiska. Parametrar som inte kontrolleras av kaparen kan faktiskt påverka kapningens smygande på ett betydande sätt.

€€€€,
"Keywords[swe]": €€€€
BGP, BGP Hijack, Stealthy IP prefix hijacking, AS-Path Prepending, BGP-övervakning €€€€,
}