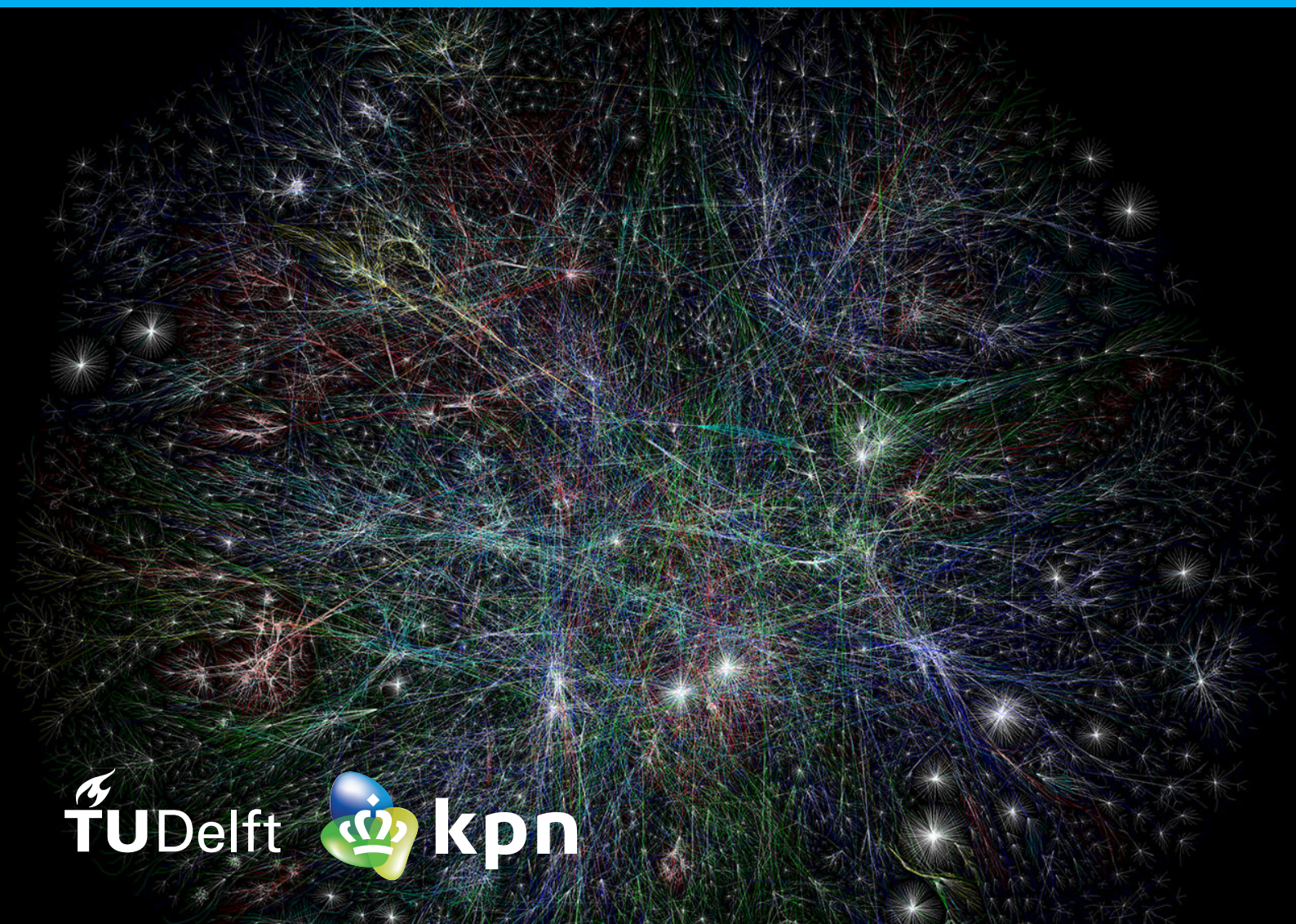
Inferring relationship types and simulating BGP traffic between Autonomous Systems using the valley-free constraint

F. P. Kastelein

November 2018

In cooperation with KPN



TU Delft kpn

Inferring relationship types and simulating BGP traffic between Autonomous Systems using the valley-free constraint

by

F. P. Kastelein

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday November 15, 2018 at 10:00 AM.

Student number:	4044533	
Project duration:	September 1, 2017 – November 1, 2018	
Thesis committee:	Dr. C. Doerr,	TU Delft, supervisor
	Dr. Ir. J. C. A. van der Lubbe,	TU Delft
	Dr. Ir. F. A. Kuipers,	TU Delft

This thesis is confidential and cannot be made public until November 1, 2019.

TU Delft kpn

Preface

In the late summer of 2017, I began my internship at KPN, a large Dutch supplier of ICT services. In the nine months that followed, the team of which I was a part wrote six reports to make a number of KPN's large clients aware of the dangers caused by the Border Gateway Protocol (BGP) and to demonstrate how crucially important it is to consider BGP as a major factor in risk analyses.

Initially, the research focused on predicting BGP-related anomalies. However, it soon became apparent that there was insufficient data available for realising this goal. The focus of the research then shifted to improving the available data by making use of trace route data and a BGP simulator that I developed myself, which is able to calculate the 'possible' impact of BGP hijacks.

I would like to extend my thanks to KPN, and especially to Anne-Sophie Teunissen and Koen van Rhee for providing me with the opportunity during the course of my internship of developing myself further, both in terms of social and work-related skills. I am very much indebted to KPN for letting me take part in a presentation workshop, allowing me to improve my presentation skills.

The final months of my research were spent writing this thesis and finalising both the simulator and the algorithm that not only increases the amount of available BGP data but also improves the quality of this data. Writing a thesis can be a lonely job, and I am grateful to Erik Wesselius, Willem Vaandrager and Olaf Maas for making this easier and giving me their highly valued support over countless cups of tea and coffee.

Toutes les bonnes choses, même l'écriture de ce mémoire, ont une fin: all things have to end sometime, even writing this thesis. I would like to thank my parents for their unfailing support and help and finally, I would like to express my gratitude to my Professor, Christian Doerr for his constructive feedback and support over the past year.

F. P. Kastelein
Delft, November 2018

Contents

1 Introduction

In today's world, the Internet is the backbone of our society: without the World Wide Web, money and electricity simply cannot flow. Although certain protocols like the Internet Protocol (IP), Transmission Control Protocol (TCP) and Domain Name Service (DNS) are widely known, the Border Gateway Protocol (BGP) is relatively unknown. However, being the Internet's default interdomain routing protocol, its functioning is crucial in ensuring worldwide interconnectivity, given the fact that it interconnects Internet Service Providers (ISPs) and allows for engineering of Internet traffic. What is also virtually unknown is the fact that BGP lacks security [21].

Cyber security is only as strong as its weakest link: a company can have the most secure network, but if all employees have access to the server room, security is compromised. When BGP is used with malicious intent, Internet traffic is rerouted in such a way that it gives malicious parties access to data that is not intended for them.

BGP in a nutshell All communicating electronic devices over the Internet need an IP address. The routing between IP addresses is carried out by **Internet Service Providers (ISPs)**. A customer pays an ISP to perform this routing. On the Internet, IP addresses are clustered in IP ranges. Each ISP has one or more **Autonomous Systems (ASes)** over which the IP ranges are divided. An AS is a collection of network devices that operate as a single network entity. ASes are interconnected and provide each other with routes to their own IP ranges. Consequently, it is possible to send data from one device to another device. BGP ensures that ISPs stay interconnected, and that data packets, being sent from and to devices in different ISPs, end up at the intended destinations.

1.1. Societal Relevance of BGP

While BGP is crucial in enabling world wide connectivity through the Internet, its vulnerability gives malicious parties an opportunity for abuse, such as rerouting of data, sending spam, or allowing Internet censorship.

BGP and rerouting data

The Internet is highly dynamic: the distribution of IP addresses by ISPs is constantly changing. BGP ensures that these changes propagate from AS to AS. Unfortunately, BGP has no built-in verification methods [39], and as a consequence, malicious changes are also propagated. This means that an AS can announce IP ranges it does not own, resulting in the unwanted rerouting of data over the Internet. An example of this unwanted rerouting was caused by the government-controlled ISP Rostelecom in 2017:

- Confidential networking traffic of MasterCard, Visa, and other large financial services was briefly rerouted through the Russian government-controlled ISP Rostelecom. Although unwanted rerouting of data caused by BGP-related misconfigurations can occur, the sudden rerouting of a dozen of IP ranges for important financial services is, to say the least, suspicious [1].

BGP comprises over 20 decision criteria that determine whether a new learned route, used to interconnect ASes and direct Internet traffic, will be accepted by an AS. Very few people understand BGP and are able to configure the hardware for it. Moreover, adding new IP ranges, for expanding connectivity, is a manual process which can easily cause misconfigurations leading to *unintentional* BGP hijacks. As a result of these unintentional BGP hijacks, the malicious actor performing the hijack can simply state that *it was a misconfiguration*. A few months after the Rostelecom hijack, another suspicious incident occurred where popular American IP ranges were hijacked. This was again picked up by the media in late 2017.

- Traffic sent to and from Google, Facebook, Apple, and Microsoft was briefly rerouted through a previously unknown Russian AS, again under circumstances researchers say were suspicious and intentional. Between 40 and 80 separate IP ranges were affected in two hijacks and the AS responsible for the hijack, which received terabytes of data, was only active during the hijack [2].

BGP and spam

BGP is also misused for sending spam. In 2018, Bitcanal, a Portugees company, hijacked IP ranges and used it to send spam. With the help of the Internet community and publicly available BGP data, the large ISP Hurricane Electric and Portugal's IPTelecom managed to disconnect Bitcanal from the Internet. Bitcanal hijacked a total of 130 IP ranges comprising over 240k IP addresses [3].

BGP and Internet censorship

The interdomain routing protocol is sometimes used for applying Internet censorship. Again, given the complexity of the protocol, and the way ISPs are interconnected, neighboring ASes can also experience unintentional negative effects. The way in which Pakistan knocked YouTube offline and how Iran imposed a ban on the messaging app Telegram serve as prime examples.

- In February 2008 Pakistan's state-owned ISP managed to cut YouTube off the Internet. The ISP, after receiving a censorship order from the telecommunications ministry, began announcing Youtube's IP ranges to black hole all traffic destined for Youtube servers from Pakistan citizens. While they were meant to stay within country borders, BGP announcements reached ISPs in neighboring countries resulting in a world-wide outage of the Youtube website [4].

- To tighten the state-imposed ban on the Telegram messaging app, an Iranian ISP temporarily rerouted Telegram app traffic in July 2018. For one hour, BGP traffic destined for Telegram was black holed rendering the app unusable. By altering the routing of Telegram traffic, the Iranian ISP also rendered the app useless for other users in neighbouring countries [5].

As the incidents above demonstrate, route leaks, the forwarding of BGP messages to ASes that are not supposed to receive them, are a great risk to the stability of the Internet [4] [5]. The largest route leak related incident occured in August 2017 where large parts of Japan could not reach the Internet for a couple of hours because of route leaks of over 135.000 IP ranges by Google. Consequently, the largest ISP of Japan used Google's AS to reach other parts of the Internet. However, because Google is not an ISP, the AS could not handle all the traffic and caused the outage [10] [11].

BGP and security

Back in the days when BGP was developed, not many ASes existed, and relationships between ASes were mostly built on trust. Consequently, BGP did not need authentication measures for announcing routing information. Nowadays, trust alone is not sufficient and the absence of proper security measures makes BGP more vulnerable to hijacking attacks. A security extension called BGPSec exists. BGPSec implements path validation and ensures that information about the path, taken by the BGP announcement, cannot be altered. Unfortunately, BGPSec is hardly implemented because new hardware is required. Also, the security extension only works if everyone in the path supports it.

The last 3 years there have been around 200-250 BGP-related incidents per month [12] [13]. There are two main reasons why these numbers do not drop:

- For ISPs, there is a lack of awareness: the tendency of not sharing BGP traffic does not help to solve the hijacking problem. Of the 70k plus ASes that exist, only about 500 ASes share their BGP traffic with forums that are open to all parties with an interest in increasing visibility and stopping hijacks. As a consequence, the impact of hijacks is only partly visible. Moreover, configuring BGP is a challenging task, making it a specialty.
- For an Internet user, BGP is not visible and the impact of hijacks is not always immediately clear. But, the consequences of a hijack can be severe as shown earlier: in the case of the Rostelecom hijacks in 2017, confidential data was compromised by a Russian ISP. When Internet users have no knowledge about the impact of hijacks, ISPs are less motivated to address this problem.

This lack of visibility concerning BGP-related information gives malicious actors free rein. A hijack on an IP range far from its source, most likely will not be mitigated if detected. To increase awareness about the importance of sharing BGP-related information, there should be more insight into how hijacks are affecting Internet routing. Given that the currently available BGP routing data does not cover all regions, simulating the impact of hijacks can provide insight about where coverage is lacking and about who is affected. To be able to perform accurate simulations, the simulation database should resemble the Internet topology as much as possible.

Finally, around 20% of the hijacks last less than ten minutes, and some can pollute 90% of the Internet in less than two minutes [40]. This means that the real-time detection of BGP anomalies is required to be able to act accordingly. But, just detecting them is not enough: operators need to still mitigate the attack.

1.2. State of the Art

The research presented in this thesis will mainly focus on correctly mapping the Internet: this is a prerequisite for running a better simulation and for improving the coverage. Relevant organisations, services and methods that aim to a) improve the classification of ASes, b) collect BGP routing data and c) detect hijacks, are listed below. Route collectors, that collect BGP data, and types of relationships between ASes, that are used for BGP routing, play an important role.

Route Collectors A BGP Route Collector (RC) is an AS that is configured to forward all the received BGP data from peers to a storage node. If a peered AS sends all its BGP data to the RC, the RC has full insight into which paths that peered AS is using to reach IP ranges. In the case of multiple peers, all the received routes enable the RC to create a local map of the Internet, and as a result, the RC can detect hijacks locally.

AS relations There are 4 types of relations between ASes as defined by L. Gao [21]: Peer-to-Peer (`p2p`) where both ASes do not differ much in size. Provider-to-Customer (`p2c`) and Customer-to-Provider (`c2p`) relations concern ASes that provide and receive transit while Sibling-to-Sibling (`s2s`) relations connect two ASes in an ISP. These relations are important because they determine how BGP messages are sent through the Internet.

RouteViews Project, RIPE, BGPMon, BGPStream.com are organisations, services, or databases that all either detect hijacks or provide BGP data. Although they all serve their purpose, each service or data source has its limitations which are set forth below.

RouteViews Project

The University of Colorado's RouteViews Project owns 18 RouteViews Route Collectors (RRCs) and also provides real-time access to BGP routing data. The 733 number of ASes that are peered with them are relatively small, resulting in less visibility. Of the 733 ASes, 95 provide all their BGP data to the route collectors of RouteViews.

RIPE

The French *Réseaux IP Européens* (RIPE), *European IP Networks* in English, is an organisation that aims to maintain and support the development of the Internet. RIPE's most prominent activity is to act as the Regional Internet Registry (RIR) providing assignment of IP ranges and AS numbers. Their most relevant resources for this research are their 21 RIPE Route Collectors (RRCs) in which they are peering with 296 large ASes of which only 124 ASes provide all their routes.

BGPMon and BGPStream.com

The service BGPMon monitors the routing of IP ranges and, in case of relevant path changes, notifies the user. The service is not free and it only shows alerts for the IP ranges the user has selected. In the background, around 200 ASes are peered with BGPMon and provide BGP data. These ASes are based in the United States and are all relatively small. This clustering of ASes will not provide world coverage, and due to their small size, less BGP data is captured compared to the ASes that are peered with the RIPE RRCs. Moreover, BGPMon gives little insight in how it detects anomalies. BGPStream.com, a website of BGPMon, presents a selection of hijacks that have been found with BGPMon.

Out of the over 70k ASes, only 1029 are peered with either RouteViews RCs or RIPE RCs. This is not enough for monitoring all paths between ASes. Moreover, only 219 ASes provide all their BGP data, further reducing the monitoring capabilities. RouteViews Project, RIPE, BGPMon, BGPStream.com all either store BGP related data or detect hijacks. However, without a correct and complete AS topology, detecting and mitigating hijacks is a challenging task. Therefore, two algorithms and a service that provide mappings of the Internet are next described, since correctly mapping the Internet is the main goal of this research.

Currently, methods to expand and verify the AS-relations data set assume that ASes that provide the core connectivity of the Internet are peered with more ASes than their smaller counterparts [26], [37], [14]. Although this might seem a good assumption and it is valid most of the time, the number of connections an AS has does not always say something about its location in the Internet graph. Some algorithms expect that BGP paths have a particular order of `c2p`, `p2c`, `p2p` and `s2s` links. These so-called *valley-free* paths will be further discussed in the next chapter.

Gao's and IMC' 13's algorithms, both described below, infer AS relations using data from route collectors. Next, Caida's AS Rank, an Internet service where AS relations can be retrieved, is briefly touched. The next chapter will explain the two algorithms and the service in more detail. Gao, besides being the first to develop an AS classification which is still used today, also created an algorithm that has been improved on over the years by others. The IMC' 13's algorithm is included because it is used by AS Rank, the largest up-to-date database of AS relations, and also uses elements from Gao's algorithm.

Gao's algorithm

In 2001, Lixin Gao introduced a method to extract the 4 types of relations between ASes using data from route collectors [26]. The algorithm assumes that all paths are *valley free* and she validated and found that 99.1% of the links were inferred correctly. `c2p` and `p2c` links are often wrongly inferred as `p2p` [7]. However, the biggest shortcoming is that Gao's algorithm only works with data from route collectors. Given the limited coverage of the route collectors, not all links between ASes are visible and will be inferred.

IMC' 13's algorithm

In 2013, Luckie *et al.* presented a new improved algorithm to infer `c2p`, `p2c` and `p2p` links [37]. The IMC' 13's algorithm filters misconfigurations, route leaks and paths that will affect the inferring of links in a negative way. The algorithm does not rely on *valley free* paths. The algorithm does not infer `s2s` links. IMC' 13's algorithm inferred `c2p` and `p2c` links with 99.6% and `p2p` links with 98.7% accuracy. IMC' 13's algorithm, just as Gao's algorithm, only works with data from route collectors.

Caida's AS Rank

Caida's goal is to provide insights into Internet infrastructure, behaviour, usage, and evolution as well as fostering a collaborative environment in which data can be acquired, analysed, and shared. One of their services, called AS Rank, derives the relationships between ASes using two different methods. Their first method infers relationships with the IMC' 13's algorithm utilising BGP routing data from route collectors [37]. Their second method enriches the AS relation data set created in the first method using a) BGP looking glasses, real-time sources of routing and BGP related information at IXPs and b) using trace routes, which contain information on how traffic flows through the Internet [27]. Of the 4 types of relationships, both methods are only able to derive `p2p`, `p2c` and `c2p` link relations. As of 2018, AS Rank has the most up-to-date database of AS relations in the world [7].

Because both Gao's algorithm and IMC' 13's algorithm only use data from route collectors, they have a limited view resulting in only a limited number of AS relations that is inferred. This results in a partial view of the Internet topology. Without a complete and correct view it is not possible to accurately analyse BGP-related anomalies. Also, AS Rank is not able to infer `s2s` links. Although the amount of `s2s` links is limited, this small amount of wrongly inferred links will affect the inferring of other links. Furthermore, the way in which Caida's AS Rank set using traces routes works, is not documented.

Trap's BGP simulator

In a study done by the TU Delft, C.H. Trap simulated the 2008 Pakistan YouTube incident. The involved data set consisted of 5624 ASes and 9 IP ranges and was capable of running distributed over multiple servers in real-time. However, Trap used a topology generated with 2018 routing data instead of a topology generated with 2008 routing data. Furthermore, many AS configurations were not known. As a result, analysis concerning the availability of YouTube.com during the incident differed from the analysis carried out by Dyn [6] [43].

1.3. Research Questions

Gao's algorithm and IMC' 13 algorithm only use data from route collectors, resulting in an incomplete view of the structure of the Internet. The lack of knowledge on how AS Rank uses trace routes does not allow for proper validation. Furthermore, the simulator created by the TU Delft is not capable of simulating a large amount of IP ranges. But even if this would be possible, the simulator would still require a more complete Internet topology to generate accurate simulation results.

This thesis therefore proposes a new algorithm that uses data from route collectors, AS Rank, and trace routes to generate a more complete and correct Internet topology. With this more complete Internet topology, a new developed BGP simulator will attempt to simulate more BGP traffic better and faster compared to the TU Delft's simulator. With a more accurate topology in combination with a better BGP simulator, BGP anomalies can be better detected and regions where route collectors provide no coverage can be pinpointed with more accuracy. Compared to the current topology generating algorithms, the newly proposed topology generating algorithm only expects that BGP paths are *valley-free*: these paths have a particular order of `c2p`, `p2c`, `p2p` and `s2s` links.

Research Question 1: Can the currently known AS Internet topology be improved with a topology generating algorithm that uses data from route collectors, AS Rank's *serial-2* data set, and trace route data in such a way that more relationship types are inferred and the percentage of correctly inferred relationship types is higher using only the condition that BGP paths need to be *valley-free*?

Research Question 2: Can BGP traffic be simulated in such a way that the simulated BGP routes match actual used BGP routes in an efficient way when a large number of ASes and IP ranges are involved?

1.4. Outline

Chapter 2 will give a more detailed description of *the state-of-the-art* methods and data sets presented in this chapter. Next, chapter 3 will presents Kastelein's algorithm: a new topology generating algorithm, a mock graph generator which is used to validate the algorithm, and the the way in which the BGP simulator functions. After the methodology has been presented, chapter 4 will validate Kastelein's algorithm and the new BGP simulator. Moreover, the new BPG simulator's scalability is going to be measured. Finally, chapter 5 will provide a conclusion and work to be done in the future.

2

Related Work

The preceding chapter has shown the importance of simulating BGP traffic and improving the underlying AS relations, as this enables establishing exactly where hijack coverage is lacking. This chapter first explains how BGP works and lists methods that have been developed for detecting BGP anomalies. Next, two algorithms are described that are capable of predicting and improving AS relations, and which will be used in the following chapter to introduce a new algorithm that uses trace routes. Following this, a description is given of AS Rank, which uses one of these two algorithms and has the largest database of AS relations in the world. Finally, this chapter will explain how the Trap's BGP simulator, developed by the TU Delft, works. This will serve as a reference for the third chapter of this thesis, where a new BGP simulation tool is presented.

In order to explain the workings of AS topology generating algorithms and Mininet, background knowledge of BGP is first presented. Based on the survey paper of Al-Musawi *et al.*, the working of BGP, the types of AS relations, the types of BGP anomalies, and different BGP anomaly detecting methods are explained in sections 2.1, 2.2, 2.3, and 2.4 [21].

2.1. Border Gateway Protocol (BGP)

The Internet consists of many interconnected networks, called Autonomous Systems (ASes). BGP is a protocol that interconnects ASes. An AS has its own network agenda that operates on behalf of a single administrative entity or domain. Each AS has a unique identifier called an AS number. Originally, an AS number was stored in a 2 byte field, resulting in only 65k unique ASes. But, with the growth of the Internet and number of ASes, 4 byte AS numbers were introduced to extend this limit. As of 2018, 70k ASes exist.

An AS provides access to one or more prefixes and exchanges Network Reachability Information (NRI), information on how to reach IP prefixes. Two BGP routers, also called BGP speakers, communicate through Internal BGP (IBGP) with each other if they are located in the same AS. Two routers from different ASes use External BGP (EBGP). In this thesis, BGP will refer to EBGP. IBGP does not lie in the scope of this research.

IP addresses and prefixes

Each AS takes care of traffic between groups of continuous IP addresses, so-called IP prefixes. Two types of IP addresses exist: IPv4 and IPv6 addresses. An IPv4 address is a 32 bit number, resulting in 2^{32} IPv4 addresses, and is denoted as X.X.X.X where X is an integer in the range of 0 to 256. An IPv6 address can be denoted as XXXX:XXXX:XXXX:XXXX:XXXX:XXXX:XXXX:XXXX where each X is a hexadecimal number. Every 2 hexadecimals represent a byte making an IPv6 address 128 bits long which results in 2^{128} IPv6 addresses.

IP addresses are bundled in IP prefixes. An IPv4 prefix takes the form X.X.X.X/Y where Y lies in the range between 0 and 32. The number of IPv4 addresses in an IPv4 prefix is $2^{32-Y} - 1$. The same /Y notation is used for IPv6 prefixes where Y lies between 0 and 128, with the number of IPv6 addresses contained by the IPv6 prefix equal to $2^{128-Y} - 1$. Y determines the size of the prefix and the smaller the Y, the larger the prefix. The example on the next page shows how the first and last IP address of an IP prefix are calculated.

Example: calculating the first and last IPv4 address of the IPv4 prefix 192.168.2.0/22 The method presented below can also be used for calculating the first and last IPv6 addresses of IPv6 prefixes. 192.168.2.0/22 first needs to be written in its binary representation:

192.168.2.0/22 = 11000000 10101000 00000010 00000000 / 22

To calculate the first IP address of a prefix, the binary presentation of the IP address is taken and the AND operation is performed with the subnet mask. The subnet mask consists of a sequence of 1s with the length equal to Y, in this case 22, followed by a sequence of 0s with the length $32-Y$ and is used to divide an IP prefix. If it concerns an IPv6 prefix, there are $128-Y$ 0s. The first IPv4 address in 192.168.2.0/22 is:

$$\begin{array}{rl} & \overbrace{11111111\ 11111111\ 111111}^{\text{length = 22}}\overbrace{00}^{\text{32-22 = 10}}\ 00000000 \\ \texttt{AND} & 11000000\ 10101000\ 000000\,10\ 00000001 \ = \ 192.168.2.0 \\ \hline & 11000000\ 10101000\ 000000\,00\ 00000000 \ = \ 192.168.0.0 \end{array}$$

To calculate the last IP address of a prefix, again the binary presentation of the address is taken and the OR operation is performed with a sequence of 0s with a length equal to Y, in this case 22, followed by a sequence of 1s with the length $32-Y$. If it concerns an IPv6 prefix, there are $128-Y$ 1s. The last IPv4 address in 192.168.2.0/22 is:

$$\begin{array}{rl} & \overbrace{00000000\ 00000000\ 000000}^{\text{length = 22}}\overbrace{11}^{\text{32-22 = 10}}\ 11111111 \\ \texttt{OR} & 11000000\ 10101000\ 000000\,10\ 00000001 \ = \ 192.168.2.0 \\ \hline & 11000000\ 10101000\ 000000\,11\ 11111111 \ = \ 192.168.3.255 \end{array}$$

BGP Messages

BGP is an incremental protocol. After a complete exchange of the Routing Information Base (RIB) - a data table stored in a BGP router that lists the routes to IP prefixes - only the changes to the RIB are communicated through new announcement messages, withdrawal messages or update messages concerning an existing route attribute. A RIB consists of **Adj-RIBs-In**, **Loc-RIB**, and **Adj-RIBs-Out** tables:

- The **Adj-RIBs-In** table contains routing information learned from neighbouring ASes, which has not yet been processed.
- The **Loc-RIB** table stores routes that will be used to perform routing based on the **Adj-RIBs-In** table content and local policies.
- The **Adj-RIBs-Out** table has routing information that is ready for advertisement to peered ASes.

In order to send and receive routing information, BGP uses Transmission Control Protocol (TCP) with port number 179 to exchange OPEN, UPDATE, NOTIFICATION, and KEEPALIVE messages. An OPEN message is the first message sent after establishing a TCP connection between two BGP peers. When the other side accepts this message, KEEPALIVEs are periodically transmitted to confirm that the two BGP peers are still interconnected. A NOTIFICATION message supplies information regarding a terminated session. However, the most important message is the UPDATE message which is used to announce a new route, withdraw a route that was advertised previously, or update an existing route with new parameters.

BGP Attributes

BGP attributes are a set of properties stored in a BGP UPDATE message and are used to determine the best route among many possible routes to an IP prefix. Attributes are divided in four types and are sent from BGP router to BGP router if the type allows for it:

- **Well-known mandatory**: attributes which **should** be recognised and included by BGP routers.
- **Well-known discretionary**: attributes which **should** be recognised but **may** be included by BGP routers.
- **Optional transitive**: attributes which **may** be recognised by BGP routers but **should** be included even if they are not r
- **Optional non-transitive**: attributes which **may** be recognised by BGP routers and **may** be included.

The most well-known and used attributes are: **ORIGIN**, **AS_PATH**, **LOCAL_PREF**, **AGGREGATOR**, and Multi Exit Discriminator (**MED**).

- **ORIGIN** is a **well-known mandatory** attribute created by the BGP router starting the announcement, also called the originator AS. The attribute indicates how an AS has learned a particular route and can have the values 0, 1, or 2. The value 0 is used for routes to prefixes where the originator AS provides the final routing. The value 1 indicates that the route is learned via the - now obsolete - Exterior Gateway Protocol. The value 2 tells that the origin of the route is unknown or learned using a different protocol [14].
- **AS_PATH** is a **well-known mandatory** attribute that stores all the ASes that the UPDATE message has traversed as a result, prevents routing loops. The **AS_PATH** attribute is updated while moving from AS to AS. For example, AS1 starts announcing a route and the route flows through AS2, AS3 and AS4. The resulting AS_PATH in AS4 is $[1,2,3,4]$. The next AS that processes and sends the announcement is appended at the end of the AS_PATH. The route that the actual data takes equals the inverted AS_PATH list.
- **LOCAL_PREF** is a **well-known discretionary** attribute and represents the degree of preference for routes coming from a peered AS where a high value of this attribute shows a strong preference. If the attribute is not provided in the BGP message, the BGP router will use its own locally stored **LOCAL_PREF** given to the AS from which it receives the route.
- **AGGREGATOR** is an **optional transitive attribute**. It contains information about the BGP speaker that aggregates the route. Although the aggregation helps to reduce the number of advertising routes, it can hide **AS_PATH** and other attributes of the aggregated prefixes.
- **Multi Exit Discriminator (MED)** is an **optional non-transitive attribute** and is used to suggest to peered ASes which route to the same prefix they should use. For example, AS1 is peered with AS2 via two BGP routers and prefers, for a particular route, that the route sent from the second BGP router is used. By giving this announcement a lower **MED** value than the announcement sent from the first BGP router, AS1 can give AS2 its preferred used path.

When announcements are received by a BGP router, it uses the following sequence of comparisons to find the best route to a prefix. More criteria exist, however, those presented below are the most relevant ones.

1. Highest **LOCAL_PREF** value
2. Lowest **AS_PATH** length
3. Lowest **ORIGIN** type
4. Lowest **MED** value

   ...
14. Lowest BGP router ID

BGP messages are sent to reflect changes in the topology and policy of Ases. When a BGP router receives a BGP message that changes its routing table it will propagate that message to all or a group of its neighbors based on its local policies.

The fact that a route is stored in the **Loc-RIB** table does not mean it that is going to be used. For redundancy reasons, multiple routes to the same prefix are stored in a BGP router. To determine the best path to an IPv4 or IPv6 address, the BGP router first determines the smallest prefix that will fit the address. If only one route remains, this route will be used. If there are multiple routes to the same prefix, the BGP router uses the sequence of comparisons presented above. When the first criteria results in a single route, this route is used. If this is not the case, the next criteria are tested one by one until only one route remains. When, finally, there are still multiple routes to choose from, the BGP router picks the route learned from the peered BGP router with the lowest ID.

2.2. AS relations

The propagation of a BGP message depends on the relation between ASes. There are 4 types of relations between ASes according to the definition by L. Gao [26]:

- `p2p` - *Peer-to-Peer*
- `p2c` - *Provider-to-Customer*
- `c2p` - *Customer-to-Provider*
- `s2s` - *Sibling-to-Sibling*

An AS must pay for data that is sent to parts of the Internet that are not in the customer cone of the AS. The customer cone of an AS is the set of IPv4 and IPv6 addresses that are either owned by the AS or can be reached by visiting its customers, and also include the customers of the customers. A `c2p` link connects a customer with a provider that will provide the customer access to the part of the Internet that is not in the customer's customer cone. A `c2p` link between ASX and ASY concerns the same link as the `p2c` link between ASY and ASX.

Two ASes that have agreed to exchange traffic data coming from each other's customer cones on a quid pro quo basis, use a `p2p` link. The two ASes can be seen as mutual upstream providers. The customer cones and data rates of the two ASes have to be *more or less* equal. This results in the fact that a `c2p` link can be followed by a `p2p` link.

However, when data originating from an upstream AS is sent over a `p2p` link, the upstream AS does not have to pay for the data transfer. Consequently, the smaller downstream AS has to bear the cost for data not originating from its customer cone. Therefore, a `p2c` link cannot be followed by a `p2p` link.

To summarise, BGP learned routes first consist of zero or more `c2p` links, then zero or one `p2p` link, followed by zero or more `p2c` links. Finally, an `s2s` link can appear anywhere in the path. This is called a *valley-free* path since a `c2p` link can never appear between two `c2p` links and vice versa. An `s2s` link interconnects two ASes in the same orginasation or ISP.

Giotsas *et al.* showed that as many as 13% of all the links between ASes have different relationships, also called hybrid relationships, for IPv4 and IPv6 prefixes. Also, 13% of all the IPv6 paths do not follow the *valley-free* rule. The authors claim that this is done to maintain IPv6 reachability. Giotsas *et al.* suggest that the inferring of AS relations should be separated for IPv4 and IPv6 prefixes [28]. The following section elaborates on the different types of anomalies when BGP is (unintentionally) misused.

2.3. Types of BGP Anomalies

Al-Musawi *et al.* [21] refer to anomalies as harmful changes in BGP behaviour [21]. They have constructed a taxonomy of BGP anomalies in four categories:

- **Direct intended**: Intentional BGP hijacks which can appear in different scenarios such as prefix hijacks and sub-prefix hijacks which both can also include AS hijacks.
- **Direct unintended**: Unwanted BGP traffic generated because of misconfigurations. For example, creating route leaks by forwarding BGP messages to ASes that are not supposed to receive them, or by announcing used or not-used prefixes. Note that announcing used IP prefixes is the same as a prefix hijack, however, in this case the AS performing the hijack has no harmful intentions.
- **Indirect**: Although BGP is a routing protocol for managing Internet reachability information between ASes, it can experience periods of instability caused by, for example, viruses, botnets or Distributed Denial of Service (DDoS) attacks.
- **Link failure**: Many ASes are interconnected via so-called Internet Exchange Points (IXPs). When, for example, there is an outage in an IXP, all the peering sessions can be terminated. This sudden drop of links will result in the rerouting of BGP traffic and in the overusing of still available links between ASes.

Direct intended anomalies

As above, there are different types of direct intended anomalies. Since these are the most relevant for this research, they will be elaborated on more in depth.

- **Prefix Hijack**: During a prefix hijack, an malicious party uses a BGP router to announce a prefix that is not owned by the AS. BGP allows any BGP router to announce any prefix.
- **Prefix and Its AS Hijack**: In this scenario, an adversary again announces a prefix that is not owned by the AS. In order to avoid a Multiple Origin AS (MOAS) conflict, the adversary also changes the AS_PATH attribute in such a way that it appears that the prefix in the announcement comes from the AS that has registered the prefix.
- **Sub-Prefix Hijack**: A sub-prefix hijack works in the same way as a prefix hijack. but the difference is that here the adversary uses a so-called *more-specific*, a sub-prefix that fits in the original prefix the adversary wants to hijack. For example, the IPv4 prefix 10.0.0.0/24 is a more specific of the IPv4 prefix 10.0.0.0/16: the IPv4 range 10.0.0.0 - 10.0.0.255 fit in 10.0.0.0 - 10.0.255.255 and contains less IPv4 addresses. If the sub-prefix is not registered, an MOAS conflict is avoided. As mentioned earlier, a BGP router uses the route with the smallest prefix for an IPv4 or IPv6 address it needs to forward. Thus, when an attacker announces new routes to a *more-specific* of the prefix he or she wants to hijack, these newly learned routes are going to be preferred by other ASes.
- **Sub-Prefix and Its AS Hijack**: The attacker combines the strength of the sub-prefix hijack and disguises himself as the origin AS. Consequently, there is no MOAS conflict.

2.4. Detecting BGP Anomalies

Although this research is not going to create a new BGP hijacking detection method, an overview of BGP anomaly detecting methods is provided to give insight on existing BGP hijacking detection methods and the resulting findings. To better compare the different methods, Al-Musawi *et al.* [21] has organised them in five categories: time series analysis, machine learning, statistical pattern recognition, validation of BGP updates based on historical BGP data, and reachability checks.

Time Series Analysis

By applying time series analysis to detect BGP anomalies, Bloomfield [33] and Al-Musawiet *et al.* [20] showed interesting BGP routing properties. Bloomfield applied the Fast Fourier Transform (FFT) to BGP routing update rates using data from large ISPs. The technique does not provide a way to identify the cause or source of routing instability, possibly caused by direct intended anomalies. However, Bloomfield and Al-Musawiet *et al.* showed that rapid changes in routing updates correlate with instability.

BGP updates sent from BGP routers have the characteristics of determinism, recurrence, and non-linearity based on Recurrence Quantification Analysis (RQA), an advanced non-linear analysis technique that uses BGP volume and average length of AS-PATH as BGP features extracted every second [20]. Both methods have not been tested for detecting direct intended anomalies.

Machine Learning

Li *et al.* presented an Internet Routing Forensic (IRF) framework to detect BGP anomalies based on using the machine learning algorithm C4.5 [44] [36]. The IRF framework, which has not been validated, is based on the control plane using the RouteViews and RIPE RRC data. An IRF-related framework, which can use different data mining algorithms, was used by Cazenave *et al.* and showed that the Support Vector Machine (SVM) gives better performance than decision tree and Naive Bayes, a vector-based classifier construction technique [23]. Furthermore, Cazenave *et al.* showed that the IRF-related framework is able to detect misconfiguration, blackout, and worm attacks.

A new machine learning mechanism was introduced by Al-Rousan and Trajkovic [22]. It consists of two main phases: an advance features extraction from BGP updates and a classifier for classifying BGP updates as normal or abnormal.

Lutu *et al.* presented a system to detect BGP anomalies at an early stage based on prefix visibility, the occurrence of a prefix in the global routing table at every sampling moment, at the AS level [38]. The proposal can only detect direct unintended anomalies.

None of these approaches address detecting direct intended anomalies or are able to identify the source cause of he anomaly. Just as the time series analysis techniques, all these machine learning approaches only use control plane data.

Statistical Pattern Recognition

A technique to detect BGP node, link, and peer failure using a form of Principal Component Analysis (PCA), a dimension-reduction tool that can be used to reduce a large set of variables to a small set without losing relevant information, was presented by Huang *et al.* [32]. The method detects and differentiates between the three failures. It requires information of router configuration and it takes between 10 and 100 minutes making it unsuitable for real-time detection.

Deshpandeet *et al.* [24] presented a BGP anomaly detection approach based on the Generalized Likelihood Ratio Test (GLRT), a standard statistical technique used in hypothesis testing. The authors showed that using AS-PATH and rare AS in AS-PATH features with message volume improved the false positive rate compared with using message volume alone.

A data-mining approach used to produce relevant information from BGP updates, called Higher-Order Path Analysis (HOPA) presented by Ganiz *et al.* [25], was used to detect BGP anomalies. It is able to differentiate between indirect BGP anomalies and link failure. HOPA has not been evaluated with direct anomalies.

Theodoridis *et al.* [42] introduced an unsupervised mechanism to detect BGP hijacking using control plane BGP raw data. This mechanism is based on observing the geographic changes of intermediate AS in the AS-PATH between the competing routes.

Techniques based on statistical pattern recognition can detect different types of BGP anomalies and identify the source causes.

Validation of BGP based on historical BGP data

This approach to BGP anomaly detection uses a history of RIB table and/or BGP updates to validate new BGP updates, assuming that the Internet topology does not frequently change. Pretty Good BGP (PGBGP) is a detection and mitigation system against BGP attacks [34]. PGBGP uses the history of both RIB and BGP updates downloaded from the RouteViews project to validate new updates. Moreover, old unused routes are automatically deleted and new suspicious routes, where the prefix and origin AS pair is not yet known, are propogated with a low LOCAL_PREF. As described earlier, the LOCAL_PREF attribute may be included in BGP UPDATE messages. All the approaches rely on prefix origin change in order to validate BGP updates.

Lad *et al.* developed the Prefix Hijack Alert System (PHAS). PHAS analyses BGP data in real-time and detects when a prefix hijacking event occurs [35]. The system requires users to register their prefixes including which ASes are allowed to announce them. There is no secure mechanism for differentiating between a legitimate owner and an attacker.

NetReview, presented by Haeberlen *et al.*, is prototype that detects BGP faults at the AS level by keeping logs containing BGP updates from and to ASes [30]. NetReview is capable of spotting link failures, misconfigurations, BGP hijacks and cases where ASes violate policies. The drawback of NetReview is that additional policy data from ASes as well as one year window of BGP updates are required. As a Consequence, there are scalability issues, especially for large ISPs.

Argus is a system for detecting prefix hijacking and identifying the attacker in real-time using BGP updates, the Internet Routing Registry (IRR) and trace routes. With over 2 months of historical BGP data, the system classifies new BGP updates as normal or suspicious. Afterwards, it checks the reachability using trace routes to verify the suspicious updates. Argus cannot detect sub-prefix hijacking, indirect anomalies and link failures [40], [45].

Reachability Checks

This type of approach uses the BGP data plane to check reachability to a certain prefix using different types of tools such as *hping*, a free packet generator and analyzer for the TCP/IP protocol, *Nmap*, a network mapper, and also trace routes. Multiple approaches exist [47], [31], [41], [46] of which [47], [31], and [41] are capable of detecting direct intended anomalies.

Zhang *et al.* iSPY's ability is limited to detecting regular prefix hijacking only: other types of hijacking such as sub-prefix hijacking cannot be detected by iSPY. The system uses the observation that connectivity to victim hosts is lost during hijacking attempts [46]. All approaches use data plane data with the exception of the approach by Hu and Mao [31] that also uses control plane data.

The assumption that the network location for a prefix remains unchanged over time and that a significant change in the hop count between source and destination AS can be a sign of a possible anomaly was made by Zheng *et al.* The hop count refers to the number of ASes through which BGP announcements must pass between source and destination AS. Zheng *et al.* proposed a light-weight distributed scheme for detecting IP prefix hijacks where a vantage point to a certain IP may be used as an indicator of hijacking [47]. Tahara *et al.*

[41] proposed a method to detect prefix hijacks by using a ping test. Assuming that a hijack will not affect all ASes, pings ending up at different BGP routers could be a sign of a hijack.

In [31], X. Hu *et al.* used a set of fingerprints such as host OS properties, IP identifier, TCP time stamp, and ICMP time stamp to identify the attackers. The system is difficult to deploy since it relies on complicated probing and requires installation of customised software at its BGP routers.

2.5. Topology Generating Algorithms

As presented in the introductory chapter, route leaks, the forwarding of BGP messages to ASes that are not supposed to receive them, can have severe consequences [4] [5] [10] [11]. Any system designed to detect route leaks needs to have an accurate and complete AS-relations database to correctly distinguish between customers and providers. In the case of BGP hijacks, knowledge of the way in which the AS topology changes over time helps to improve the analysis of BGP-related incidents.

Gao, besides being the first to develop an AS classification which is still used today, also created an algorithm that has been improved on over the years by others. This section will first describe the algorithm presented by Gao [26], followed by the algorithm of Luckie *et al.* [37]. The IMC' 13's algorithm is included because it is used by AS Rank, the largest up-to-date database of AS relations, and because it uses elements from Gao's algorithm. The differences between the two algorithms are also presented. Finally, the limitations and benefits of the two *state-of-the-art* algorithms are presented and substantiated. Both algorithms derive `p2p`, `p2c` and `c2p` links with Goa's algorithm also including `s2s` links. Both algorithms can only use BGP control plane data, thereby limiting the number of possible inferred relations.

2.5.1. Gao's Algortihm

In 2001, Lixin Gao distinguished four types of relations between ASes: `p2p`, `p2c`, `c2p` and `s2s`. She also introduced a method to extract these four types from BGP control plane data. Gao's algorithm is based on the assumption that a provider is larger than his customers and that the size of an AS is proportional to the AS' node degree, i.e., the number of links with other ASes. The algorithm can be split in six stages:

- **Stage 1**: *Extracting AS_PATH information from BGP control plane data and the initialisation of variables* The algorithm first takes all BGP control data and adds the AS_PATH information to a list. The first AS in every path should always be the AS announcing the prefix. Next, stage 1 removes sequences of recurring ASes in the paths: the algorithm does not support AS path prepending. Stage 1 then initialises the *transit* counter to zero and sets the *not_p2p* flag to `false` for each unique link. The *transit* counter counts the number of paths indicating that the link is possibly a `c2p` or `p2c` link while the *not_p2p* flag is used, as the name suggests, to indicate that a link cannot be `p2p`. Gao's algorithm next creates an empty list of neighbours for each AS which will be used in **stage 2** to compute the node degree. Finally, this stage initialises the state variable which is going to be used to store the link type between ASes in later stages.

Algorithm Goa's 1st stage

1: **create** an AS path list l with BGP control plane data
2: **for each** AS path p **as** $[AS_1, AS_2, \ldots, AS_{n-1}, AS_n]$ **in** l **do**
3: **for** $i = 1, \ldots, n-1$ **do** ▷ remove AS path prepending
4: **if** $AS_i = AS_{i+1}$ **then**
5: **remove** the i_{th} AS **from** p.
6: **for each** AS path $[AS_1, AS_2, \ldots, AS_{n-1}, AS_n]$ **in** l **do**
7: **for** $i = 1, \ldots, n-1$ **do**
8: state$[AS_i, AS_{i+1}] \leftarrow -1$
9: state$[AS_{i+1}, AS_i] \leftarrow -1$
10: transit$[AS_i, AS_{i+1}] \leftarrow 0$
11: transit$[AS_{i+1}, AS_i] \leftarrow 0$
12: not_p2p$[AS_i, AS_{i+1}] \leftarrow$ `false` ▷ link can or cannot be p2p
13: not_p2p$[AS_{i+1}, AS_i] \leftarrow$ `false`
14: neighbors$[AS_i] \leftarrow []$ ▷ empty list, used for the node degree

- **Stage 2**: *Computing the node degree for each AS*
  Using each path in the list constructed in **stage 1**, the second stage counts and stores the node degree, the number of unique links, for each AS. The algorithm is based on the assumption that a provider has a larger size compared with his customers and that the size of an AS is proportional to its degree.

Algorithm Goa's 2nd stage

```
1: for each AS path [AS_1, AS_2, ..., AS_{n-1}, AS_n] in l do
2:     for i = 1, ..., n-1 do
3:         if AS_{i+1} not in neighbors[AS_i] then
4:             add AS_{i+1} to neighbors[AS_i]
5:         if AS_i not in neighbors[AS_{i+1}] then
6:             add AS_i to neighbors[AS_{i+1}]
7: for each AS u in neighbors do
8:     degree[u] ← |neighbors[u]|          ▷ the unique number of neighbors
```

- **Stage 3**: *Counting the number of paths that could infer a `c2p` link for each link.*
  Assuming that each path is *valley-free* and that a provider has a larger node degree than its customer, the AS in the path with the highest degree is the top AS. All links leading up to the top AS can be `c2p` and all links starting from the top AS can be `p2c`. In other words, for each path in the list constructed in **stage 1**, the third stage first selects the AS with the highest degree, and in the case of multiple ASes, chooses the first AS with the highest degree of the path. Next, the third stage increments the *transit* counter for the links leading up to the highest degree AS and increments the *transit* counter for the reversed links starting at the highest degree AS.

Algorithm Goa's 3rd stage

```
1: for each AS path [AS_1, AS_2, ..., AS_{n-1}, AS_n] in l do
2:     find the smallest j such that degree[AS_j] = max_{1≤i≤n} degree[AS_i]
3:     for i = 1, ..., j-1 do
4:         transit[AS_i, AS_{i+1}]++
5:     for i = j, ..., n-1 do
6:         transit[AS_{i+1}, AS_i]++
```

- **Stage 4**: *Inferring of `s2s`, `p2c`, and `c2p` relationships*
  When two ASes provide transit to each other, the link can concern an `s2s` link. In the case that only one AS provides transit and not vice versa, this link may be inferred as `c2p` or `p2c`. In this stage, the link type cannot be detected if no more than L paths infer transit for a link in one direction and more than L in the other direction. Gao sets L to 1. The fourth stage first selects a number **L** larger than zero. Next, for each link in each path in the list constructed in **stage 1**, stage 4:
  - Marks a link `s2s` when the link's *transit* counter and the reversed link's *transit* counter are either a) both higher than **L** or b) both higher than **0**, and at most, **L**.
  - Marks a link `p2c` when the link's *transit* counter is higher than **L** and the reversed link's *transit* counter is **0**.
  - Marks a link `c2p` when the reversed link's *transit* counter is higher than **L** and the link's *transit* counter is **0**.

Algorithm Goa's 4th stage

choose integer L larger than zero
for each AS path $[AS_1, AS_2, \ldots, AS_{n-1}, AS_n]$ **in** l **do**
  for $i = 1, \ldots, n-1$ **do**
    if transit$[AS_i, AS_{i+1}] > L$ **and** transit$[AS_{i+1}, AS_i] > L$ **then**
      state$[AS_i, AS_{i+1}] \leftarrow$ `s2s`
      state$[AS_{i+1}, AS_i] \leftarrow$ `s2s`
    else if $0 <$ transit$[AS_i, AS_{i+1}] \leq L$ **and** $0 <$ transit$[AS_{i+1}, AS_i] \leq L$ **then**
      state$[AS_i, AS_{i+1}] \leftarrow$ `s2s`
      state$[AS_{i+1}, AS_i] \leftarrow$ `s2s`
    else if transit$[AS_{i+1}, AS_i] > L$ **and** transit$[AS_i, AS_{i+1}] = 0$ **then**
      state$[AS_i, AS_{i+1}] \leftarrow$ `p2c`
      state$[AS_{i+1}, AS_i] \leftarrow$ `c2p`
    else if transit$[AS_i, AS_{i+1}] > L$ **and** transit$[AS_{i+1}, AS_i] = 0$ **then**
      state$[AS_i, AS_{i+1}] \leftarrow$ `c2p`
      state$[AS_{i+1}, AS_i] \leftarrow$ `p2c`

- **Stage 5**: *Identification of possible `p2p` links*
  Given the *valley-free* constraint, links leading up to but not including and starting from and not including the top AS cannot be inferred as `p2p`. The algorithm uses the heuristic that the top AS is more likely to peer with an AS of the same node degree. Consequently, when two ASes are both linked with the top AS, the AS linked AS that has a higher degree cannot be inferred as `p2p`. This reasoning does not hold when either of the links to and from the top AS are already inferred as `s2s`. In other words, for each path in the list constructed in **stage 1**, the fifth stage:
  - Selects the AS with the highest degree. In the case of multiple ASes, the one listed first is chosen.
  - Sets the *not_p2p* flag to `true` for the links leading up to, but not containing, the highest degree AS.
  - Sets the *not_p2p* flag to `true` for the reversed links starting at, but not containing, the highest degree AS.
  - Sets the *not_p2p* flag to `true` for the link **starting** in the highest degree AS if both following conditions hold:
    - the links starting and ending in the highest degree AS are both not assigned as `s2s`
    - the degree of the AS **before** the highest degree AS is higher than the degree of the AS **after** the highest degree AS
  - Sets the *not_p2p* flag to `true` for the link **ending** in the highest degree AS if both following conditions hold:
    - the links starting and ending in the highest degree AS are both not assigned as `s2s`
    - the degree of the AS **after** the highest degree AS is higher than the degree of the AS **before** the highest degree AS

Algorithm Goa's 5th stage

for each AS path $[AS_1, AS_2, \ldots, AS_{n-1}, AS_n]$ **in** l **do**
  find the smallest j such that degree$[AS_j] = \max_{1 \leq i \leq n}$ degree$[AS_i]$
  for $i = 1, \ldots, j-2$ **do**
    not_p2p$[AS_i, AS_{i+1}] \leftarrow$ `true`
  for $i = j+1, \ldots, n-1$ **do**
    not_p2p$[AS_i, AS_{i+1}] \leftarrow$ `true`
  if state$[AS_{j-1}, AS_j] \neq$ `s2s` **and** state$[AS_j, AS_{j+1}] \neq$ `s2s` **then**
    if degree$[AS_{j-1}] >$ degree$[AS_{j+1}]$ **then**
      not_p2p$[AS_j, AS_{j+1}] \leftarrow$ `true`
    else
      not_p2p$[AS_{j-1}, AS_j] \leftarrow$ `true`

- **Stage 6**: *Inferring of p2p links*
  Stage 5 determined links that cannot be inferred as p2p. The sixth stage infers links that can be inferred as p2p given that the node degree between the ASes in the link does not differ more than **R** times: ASes interconnected via p2p links usually do no differ much in size. In other words, this stage selects a number **R** between 0 and 1 and assigns a p2p relationship to each link if the two following conditions hold:
  - The link and the reversed link are not both identified as non-p2p links in **stage 5**. The reversed link of $\{AS_x, AS_y\}$ is defined as $\{AS_y, AS_x\}$.
  - The ratio between the degree of the two ASes in the link is not more than **R**.

Algorithm Goa's 6th stage

choose the real number R between 0 and 1
for each AS path $[AS_1,\ AS_2,\ \ldots,\ AS_{n-1},\ AS_n]$ **in** l **do**
  for $i = 1,\ \ldots,\ n-1$ **do**
    if not not_p2p$[AS_i,\ AS_{i+1}]$ **and not** not_p2p$[AS_{i+1},\ AS_i]$ **then**
      if degree$[AS_i]$ / degree$[AS_{i+1}] < R$ **and** degree$[AS_{i+1}]$ / degree$[AS_i] < \frac{1}{R}$ **then**
        state$[AS_i,\ AS_{i+1}] \leftarrow$ p2p
        state$[AS_{i+1},\ AS_i] \leftarrow$ p2p

2.5.2. IMC' 13's algorithm

In 2013, Luckie *et al.* presented a new improved algorithm to infer c2p, p2c and p2p links [37]. The IMC' 13's algorithm can, compared to Gao's algorithm, filter out poisoned paths. Poisoned paths are paths that prevent the sending of data destined for IP prefixes. Also, the IMC' 13's algorithm does not rely on *valley-free* paths. The authors state that, because it is difficult to distinguish between route leaks and s2s links, s2s links are not inferred by the algorithm.

The algorithm uses RIBs from RouteViews RRCs and RIPE RRCs and double listings of ASes in paths are removed. Paths that contain unassigned ASes are removed. The authors also inferred p2c links using the RIPE WHOIS database where routing policies are stored using the Routing Policy Specification Language (RPSL). Given the scope of this thesis, the inferring of p2c links using the RIPE's WHOIS database will not be further discussed. The validation AS relations data set, used to validate of the algorithm, consists of RSPL, BGP communities, and email exchange data combined with two older data sets generated by older - not specified - algorithms.

The node degree, the number of unique neighbours of an AS, and transit degree, the number of unique paths traversing through an AS, are the two metrics the algorithm uses. The transit degree is first used to sort the ASes where next the node degree serves as a tie braker. Inferred tier-1 ASes are always placed at the top. As mentioned earlier, an tier-1 ASes are the largest ASes located at the core of the internet with many p2c links to lower-tier customers. The final presented algorithm consists of **14** stages.

- **Stage 1**: *Extracting of AS_PATH information from BGP control plane data and the initialisation of variables*
  The algorithm first takes all BGP control data and adds the AS_PATH information to a list. The first AS in every path should always be the AS announcing the prefix. Next, stage 1 removes sequences of recurring ASes in the paths: the algorithm does not support AS path prepending. Finally, stage 1 creates an empty list of neighbours for each AS which is going to compute the node degree in **stage 2**.

Algorithm IMC' 13's 1st stage

create an AS path list $l_{complete}$ with BGP control plane data
for each AS path p **as** $[AS_1,\ AS_2,\ \ldots,\ AS_{n-1},\ AS_n]$ **in** $l_{complete}$ **do**
  for $i = 1,\ \ldots,\ n-1$ **do** ▷ remove AS path prepending
    if $AS_i = AS_{i+1}$ **then**
      remove the i_{th} AS **from** p.
  for $i = 1,\ \ldots,\ n-1$ **do**
    neighbors$[AS_i] \leftarrow$ [] ▷ empty list, used for node degree

- **Stage 2**: *Discard paths with artefacts*
  The second stage removes a path in the list computed in **stage 1** if an AS in the path has an artefact: it is listed twice or more and is separated by one or more other AS(es). These so-called poisoned paths will lead to ASes preferring each other to reach a prefix. If an AS in the path is unassigned [**DEF**], stage 2 also removes the path from the list computed in **stage 1**.

Algorithm IMC' 13's 2nd stage

```
1: for each AS path p as [AS_1, AS_2, ..., AS_{n-1}, AS_n] in l_complete do
2:     for i = 1, ..., n do                          ▷ remove duplicate ASes in a path
3:         for j = i+1, ..., n-1 do
4:             if AS_i = AS_j then
5:                 remove p from l_complete
6:     for i = 1, ..., n do
7:         if AS_i not assigned then
8:             remove p from l_complete
```

- **Stage 3**: *Computing the node and transit degree and, next, sorting ASes in decreasing order of computed transit degree, then node degree*
  For all filtered AS paths in the list computed in **stage 2**, the third stage stores the unique neighbours for each AS and increases the transit degree for each AS that is not listed first or last in the path. The transit degree of an AS is the number of AS paths where this AS is used for transferring data from other ASes. Finally, this stage initialises the state variable which is going to be used to store the link type between ASes in later stages.

Algorithm IMC' 13's 3rd stage

```
1: sorted ← []
2: for each AS path p as [AS_1, AS_2, ..., AS_{n-1}, AS_n] in l_complete do
3:     state[AS_i, AS_{i+1}] ← -1
4:     state[AS_{i+1}, AS_i] ← -1
5:     for j = 1, ..., n-1 do
6:         if AS_{j+1} not in neighbors[AS_j] then
7:             add AS_{j+1} to neighbors[AS_j]
8:             add AS_{j+1} to sorted
9:         if AS_j not in neighbors[AS_{j+1}] then
10:            add AS_j to neighbors[AS_{j+1}]
11:            add AS_j to sorted
12:    for j = 2, ..., n-1 do
13:        transit[AS_j]++                            ▷ transit degree
14: for each AS u in neighbors do
15:    degree[u] ← |neighbors[u]|                     ▷ node degree
16: sort the ASes in [sorted] in decreasing order of transit degree, then node degree
```

- **Stage 4**: *Inferring the clique at top of the AS topology*
  The fourth stage first finds the maximum clique, the largest set of ASes that are all interconnected with each other, C_1 for the group of ten ASes that have the largest transit degree. Next, **stage 4** determines for every other AS, in decreasing order of transit degree and then node degree, if it is peered with all ASes in C_1. If so, stage 4 adds the AS to C_1. When the AS is peered with all but one ASes in C_1, it is added to C_2. The final inferred clique is the maximum clique of the ASes in C_1 and C_2.

Algorithm IMC' 13's 4th stage

```
1: C_1, C_2 ← []
2: for i = 1, ..., 10 do                          ▷ the ten ASes with the largest transit degree, then node degree
3:     add sorted[i] to C_1
4: C_1 ← maximum_clique(C_1)
5: for i = 11, ..., len(sorted) do
6:     counter ← len(C_1))
7:     for j = 1, ..., 10 do
8:         if C_1[j] not in neighbor[i] then
9:             counter ← counter − 1
10:    if counter = len(C_1) then                  ▷ AS is peered with all ASes in the clique C_1
11:        add sorted[i] to C_1
12:    else if counter = len(C_1) − 1 then         ▷ AS is peered with all but one ASes in the clique C_1
13:        add sorted[i] to C_2
14: C_final ← maximum_clique(C_1 ∪ C_2)
15: for i = 1, ..., len(C_final) do                ▷ links between ASes in the clique are inferred as p2p
16:    for j = i + 1, ..., len(C_final) do
17:        state[AS_i, AS_j] ← p2p
18:        state[AS_j, AS_i] ← p2p
```

- **Stage 5**: *Removal of poisoned paths*
  A clique AS in C_{final} is by definition *transit-free*: it has no c2p links since it is located at the core of the Internet. As a result, any path indicating that a clique AS is not *transit-free* is poisoned. Thus, AS paths where two ASes in the clique are separated by an AS not in the clique are poisoned and are removed in this stage.

Algorithm IMC' 13's 5th stage

```
1: for each AS path p as [AS_1, AS_2, ..., AS_{n−1}, AS_n] in l_complete do
2:     for i = 1, ..., n − 2 do
3:         if AS_i in C_final and AS_{i+1} not in C_final and AS_{i+2} in C_final then
4:             remove p from l_complete
```

- **Stage 6**: *Generating a list of all stub ASes*
  Stub ASes are ASes that **do not** appear in the middle of any path. This classification is used in later stages.

Algorithm IMC' 13's 6th stage

```
1: l_possible stubs ← []
2: l_non-stubs ← []
3: for each AS path p as [AS_1, AS_2, ..., AS_{n−1}, AS_n] in l_complete do
4:     if AS_1 not in l_non-stubs then
5:         add AS_1 to l_possible stubs
6:     if AS_n not in l_non-stubs then
7:         add AS_n to l_possible stubs
8:     for i = 2, ..., n − 1 do
9:         if AS_i not in l_non-stubs then
10:            add AS_i to l_non-stubs
11:            if AS_i in l_possible stubs then
12:                remove AS_i from l_possible stubs
13: l_stubs ← l_possible stubs
```

- **Stage 7**: *Generating a list of only AS path triplets*
  The authors use AS path triplets, a path containing only three ASes. These triplets provide the necessary information, shown in the next stages, for inferring c2p and p2c links.

Algorithm IMC' 13's 7th stage

1: $l_{triplets} \leftarrow []$
2: **for each** AS path p **as** $[AS_1,\ AS_2,\ \ldots,\ AS_{n-1},\ AS_n]$ **in** $l_{complete}$ **do**
3: **if** $n \geq 3$ **then**
4: **for** $i = 1,\ \ldots,\ n-2$ **do**
5: **add** $[AS_i, AS_{i+1}, AS_{i+2}]$ **to** $l_{triplets}$

- **Stage 8**: *Inferring of c2p links top-down using the above ranking.*
  Stage 8 visits all ASes in decreasing order of transit degree, then node degree. For each visited AS, the AS path triplets which refers to the visited AS are selected and processed. Given the AS path triplet $[AS_1, AS_2, AS_3]$ where AS_3 is the selected AS, the link between AS_2 and AS_3 is inferred as p2c if the link between AS_1 and AS_2 is p2c or p2p.

Algorithm IMC' 13's 8th stage

1: **for** $i = 1,\ \ldots,$ *len*(sorted) **do**
2: **for each** AS path p **as** $[AS_1,\ AS_2,\ AS_3]$ **in** $l_{triplets}$ **do**
3: **if** sorted$[i] = AS_3$ **then**
4: **if** state$[AS_1,\ AS_2]$ = p2p **or** state$[AS_1,\ AS_2]$ = p2c **then**
5: state$[AS_2,\ AS_3] \leftarrow$ p2c
6: state$[AS_3,\ AS_2] \leftarrow$ c2p

- **Stage 9**: *Inferring of c2p links from VPs inferred not to be announcing provider routes.*
  The algorithm assumes that *partial vantage points (VPs)*, ASes that are peered with route collectors which provide provide routes to fewer than 2.5% of all ASes, either only export customer routes or have their connections to the route collector configured as p2c and have a default route to their providers.

  In other words, the path $[AS_1, AS_2, AS_3]$ where AS_1 is a VP assumes that the link between AS_1 and AS_2 is either p2c or p2p. The link cannot be c2p because the VP does not export its BGP traffic to its provider or has a default route to its provider. Consequently, the link between AS_2 and AS_3 has to be p2c given the fact that p2c and p2p links can only be followed by p2c links.

Algorithm IMC' 13's 9th stage

1: **for each** AS path p **as** $[AS_1,\ AS_2,\ AS_3]$ **in** $l_{triplets}$ **do**
2: **if** AS_1 is a *partial vantage point* **then**
3: state$[AS_2,\ AS_3] \leftarrow$ p2c
4: state$[AS_3,\ AS_2] \leftarrow$ c2p

- **Stage 10**: *Inferring of c2p links for ASes where the provider has a smaller transit degree than the customer.*
  The algorithm assumes that c2p relations where the transit degree, the number of AS paths where this AS is used for transferring data from other ASes, of the customer is larger than the transit degree of the provider are rare, but exist. Given a path $[AS_1, AS_2, AS_3]$ where a) the link between AS_1 and AS_2 is p2c, and b) the transit degree of AS_3 is larger than the transit degree of AS_2, the link between AS_2 and AS_3 has to be p2c when the path is not poisoned.

  In order to filter out possible poisoned paths, the AS path triplet has to be the terminated triplet. For example, of the two possible triplets in $[AS_1, AS_2, AS_3, AS_4]$, only $[AS_2, AS_3, AS_4]$ can be used. Segments of poisoned paths do not announce prefixes, and therefore only appear in the middle of the path.

Algorithm IMC' 13's 10th stage

1: **for each** AS path p **as** $[AS_1, AS_2, \ldots, AS_{n-1}, AS_n]$ **in** $l_{complete}$ **do**
2: **if** state$[AS_{n-2}, AS_{n-1}]$ = p2p **and** transit$[AS_n]$ > transit$[AS_{n-1}]$ **then**
3: state$[AS_{n-1}, AS_1] \leftarrow$ p2c
4: state$[AS_n, AS_{n-1}] \leftarrow$ c2p

- **Stage 11**: *Inferring p2c links for ASes with no c2p links*
  When an AS has no provider(s), or its provider(s) are not visible for route collectors, it can only be seen via p2p links. To find these provider-less ASes, a path needs to be found where an unknown link follows after a sequence of c2p links or the unknown link is the first link in the path. As described earlier, a *valley-free* path first consists of zero or more c2p links, followed by zero or one p2p link, and finishes with zero or more p2c links. The unknown link cannot be the last link of the path. Luckie *et al.* do not give a explanation why this should hold. The links after the first unknown link can all be inferred as p2c given the *valley-free* constraint. This stage is possible since stages 8 and 10 already inferred c2p and p2c links.

Algorithm IMC' 13's 11th stage

1: **for each** AS path p **as** $[AS_1, AS_2, \ldots, AS_{n-1}, AS_n]$ **in** $l_{complete}$ **do**
2: index $\leftarrow 1$
3: stopped $\leftarrow$ false
4: **for** $i = 1, \ldots, n-1$ **do**
5: **if not** stopped **and** state$[AS_i, AS_{i+1}]$ = c2p **then**
6: index $\leftarrow$ index $+ 1$
7: **else**
8: stopped $\leftarrow$ true
9: **if** state$[AS_{\text{index}}, AS_{\text{index}+1}] = -1$ **then**
10: state$[AS_{\text{index}}, AS_{\text{index}+1}] \leftarrow$ p2p
11: state$[AS_{\text{index}+1}, AS_{\text{index}}] \leftarrow$ p2p
12: **for** $i =$ index $+ 1, \ldots, n-1$ **do**
13: state$[AS_{\text{index}}, AS_{\text{index}+1}] \leftarrow$ p2c
14: state$[AS_{\text{index}+1}, AS_{\text{index}}] \leftarrow$ c2p

- **Stage 12**: *Inferring of c2p links between stubs and clique ASes*
  Stage 8 required AS path triplets. Consequently, the links between a stub AS - that does not appear in the middle of any path - and clique AS could not be inferred with a path of length 2. The authors of the algorithm state that stub ASes are very unlikely to share a p2p link with clique ASes. Therefore, all paths with length 2, where a) the first AS in the path is a clique AS and the second a stub AS, and b) the link between the two ASes has not yet been inferred, are inferred as p2c.

Algorithm IMC' 13's 12th stage

1: **for each** AS path p **as** $[AS_1, AS_2, \ldots, AS_{n-1}, AS_n]$ **in** $l_{complete}$ **do**
2: **if** len$(p) = 2$ **then**
3: **if** AS_1 **in** C_{final} **and** AS_2 **in** l_{stub} **and** state$[AS_1, AS_2] = -1$ **then**
4: state$[AS_1, AS_2] \leftarrow$ p2c
5: state$[AS_2, AS_1] \leftarrow$ c2p

- **Stage 13**: *Inferring of c2p links where adjacent links have no relationship inferred*
  Just as **stage 8**, this stage also visits ASes in decreasing order of transit degree, then node degree. However, in this stage the requirement is dropped that the first link of the AS path triplet already has to be inferred. This is done to prevent sequences of p2p when remaining links are inferred as p2p in **stage 14**. For each visited AS, AS path triplets $[AS_a, AS_b, AS_c]$ where AS_b is the visited AS are located where all the links have not yet been inferred. These links would otherwise both be inferred as p2p in **stage 14**. Next, for each triplet $[AS_a, AS_b, AS_c]$, if at least one triplets $[AS_a, AS_b, AS_x]$ is located where AS_x is a provider of AS_b, it is possible to infer AS_b as a provider of AS_a.

Algorithm IMC' 13's 13th stage

```
1: for i = 1, ..., len(sorted) do
2:     AS_x ← sorted[i]
3:     for each AS path p as [AS_1, AS_2, AS_3] in l_triplets do
4:         if AS_x = AS_2 and state[AS_1, AS_2] = −1 and state[AS_2, AS_3] = −1 then
5:             for each AS path p as [AS_4, AS_5, AS_6] in l_triplets do
6:                 if AS_x = AS_2 and AS_1 = AS_4 and state[AS_5, AS_6] = c2p then
7:                     state[AS_1, AS_2] ← c2p
8:                     state[AS_2, AS_1] ← p2c
```

- **Stage 14**: *Inferring of remaining links as p2p*
  All links between ASes that have not been inferred in previous stages are inferred as p2p.

Algorithm IMC' 13's 14th stage

```
1: for i = 1, ..., len(sorted) do
2:     for j = 1, ..., len(sorted) do
3:         AS_x ← sorted[i]
4:         AS_y ← sorted[j]
5:         if i ≠ j and state[AS_x, AS_y] = −1 then
6:             state[AS_x, AS_y] ← p2p
```

2.5.3. Validation, strong and weak points of Gao's and IMC' 13's algorithms

In 2001, Gao validated her algorithm with data from AT&T, a large ISP, and WHOIS data and found that 99.1% of the links were inferred correctly. In 2016, Luckie *et al.* validated Gao's algorithm and found that it inferred links as c2p and p2c with 84.7% and links as p2p with 99.5% accuracy. The makers of the IMC' 13's algorithm, showed that their algorithm inferred c2p and p2c links with 99.6% and p2p links with 98.7% accuracy. Although the accuracy of both algorithms is high, this still will result in many wrongly inferred links when considering thousand of links [7].

While both algorithms are not difficult to implement given the clearly described stages, the number of relations they can infer is limited because of the limited coverage the route collectors provide. It is not possible to add additional BGP routes in the form of trace routes. In the end, all BGP paths, either retrieved from RIBS or using trace routes, should be *valley-free*. There are special hybrid cases where an AS can have multiple type of relationships with another AS, but these cases are rare [29].

The assumption made by Gao and Luckie *et al.* that the customer cone of a customer is smaller than its provider is not always valid, resulting in possibly wrong inferred links. Also, IMC' 13's algorithm assumes that stub ASes, ASes that do not appear in the middle of any path, are very unlikely to share a p2p link. This may be the case, however, given the lack of coverage of the route collectors, it is not sure if these ASes are really stub. Finally, IMC' 13's algorithm does not infer s2s links. A possible s2s link, which most of the time is inferred as a p2p, can appear anywhere in a BGP path. This free placing changes the inferring of other links.

The final stage of both algorithms is to infer all as yet unknown links as p2p. If previous stages have not inferred all c2p and p2c links, some links will be marked p2p while they are not.

To conclude, aside from these assumptions, the largest drawback is the fact that both algorithms only work with BGP routes from RIBs. For the limited amount of links they infer both algorithms perform well, but at the end of day, a complete map of the Internet is needed.

2.6. Caida's AS Rank

Caida's mission with AS rank is to create a map of the Internet that is as accurate as possible. AS Rank contains two types of AS data sets: *serial-1* and *serial-2* [17] [18] [27]. In a presentation in May 2016, Matthew Luckie explains how the *serial-1* data set is created with the IMC' 13's algorithm using data from RIPE and RouteViews route collectors [7]. Currently, there are 296 ASes peering with these route collectors of which 124 provide their full routing table. The *serial-2* data set contains:

- AS relationships in the *serial-1* data set.
- Inferred relationships using BGP looking glasses, real-time sources of routing and BGP related information at IXPs, and BGP control plane data.
- Inferred relationships using Ark trace route data

Inferring link types using BGP looking glasses and control plane data Many Internet Exchange Points (IXPs) provide public Looking Glass (LG) interfaces that provide reading access to their BGP routing table data. In order infer relationships using a LG of an IXP, the algorithm first obtains a list of all ASes connected to that IXP including the IP addresses the ASes have with the IXP. Next, the algorithm saves which prefix is announced by which AS that is connected to the IXP. Using a subset of prefixes, the algorithm then determines, based on community values stored provided by the LG, which ASes are peered with each other and infers `p2p` links accordingly. BGP control plane data is also used to infer relationships. In a nutshell, every BGP path is analysed and when two or more ASes appear that are connected to the same IXP, relationships are inferred based on community values [27].

How the trace route data is used to infer relationships for the *serial-2* data set is not documented. Of the 4 types of relationships, the *serial-1* and *serial-2* data sets only contain `p2p`, `p2c` and `c2p` relationships. As of 2018, AS Rank has the most up-to-date database of AS relations in the world [7].

2.7. Trap's BGP simulator

In a study done by the TU Delft, C.H. Trap simulated the 2008 Pakistan YouTube incident [43]. Initially Mininet and Quagga were used for this. Mininet, which interconnected the involved ASes, is a program that can create a realistic virtual network, running real kernel, switch and application code. It can be run on a single machine or be distributed over multiple servers if necessary. Quagga is a routing software suite that can simulate ASes. The involved data set counted 5624 ASes. After a custom optimised version of Mininet was written and the computation was distributed over multiple servers, the memory and CPU bottleneck was solved. The simulator was capable of running the 5624 ASes in real-time with 9 involved prefix announcements. The simulator used 32.5 GB of memory of which half was used to store all the BGP routes.

Although AS Rank has archived the *serial-1* data set every month from 1998 to the present day, the authors did not use the 2008 *serial-1* data set: instead a topology generated in 2018 was chosen. Since many ASes use configurations that are not known. As a result, the simulated availability of YouTube.com during the incident differed from the analysis done by Dyn [6]. Actual BGP routes generated by the simulator were not compared with BGP routes captures by the RIPE route collectors.

3

Methodology

The previous chapter described how BGP works, explained the different kinds of hijacks and what has been done so far to detect them. It continued with listing two AS inferring algorithms, of which one is used in AS Rank, and also showed that both algorithms did not generate a complete AS data set, since they only use BGP control plane data. Trap's BGP simulator, which lacks scalability, was also explained. This chapter will first present an improved method, using trace routes, for inferring additional AS relations by making use of existing data sets. Generated results are then used to simulate traffic using the new, improved BGP simulator which is further described in more detail. To validate the new AS topology generating algorithm and the BGP simulator in the next chapter, an AS graph generator that mimics the structure of the real Internet, also called a mock graph generator, is used and presented in this chapter. When relationship types are removed from or changed in a mock graph, the new algorithm should infer all the removed or changed relationship types.

3.1. Kastelein's algorithm

The previous chapter described Gao's algorithm and IMC' 13's algorithm. Both algorithms only use BGP control plane data, resulting in a limited number of inferred relationships. AS Rank's serial-2 data set contains more relationships and uses trace routes, but, how this is achieved is not documented.

To also infer relationships that are not visible for route collectors, trace routes can be used. A trace route from router A to router B should follow the inverted BGP path from the announced prefix from the AS in which router B is located to the AS in which router A is located. Consequently, a single device that performs trace route measurements can find the BGP paths used to reach all available IP prefixes from its AS. Although a BGP router peered with a route collector can also provide all its routes, route aggregation limits the number of visible paths seen by the AS. Trace routes, originating in ASes that are not peered with route collectors, can enlarge the AS relations data set. In order to use trace routes, they need to be converted to BGP paths because they follow the inverted BGP path and do not yet contain AS numbers.

Converting trace routes to BGP paths A trace route operation is performed resulting in the path [**10.0.0.55, 20.0.4.22, 30.8.0.1**]. Carrying out a reversed look-up shows that the IP addresses are part of the IP prefixes 10.0.0.0/8, 20.0.4.0/24, and 30.8.0.0/16 respectively. The prefixes are registered at ASes AS1, AS2, and AS3 respectively. As a result, it is possible to infer that AS path [3, 2, 1] is used.

Kastelein's algorithm improves on an already existing AS relations data set by making use of trace routes which contain additional information compared to the available BGP control plane data. Examples of existing AS relations data sets are Caida's AS Rank relations and relations inferred by using Gao's or the IMC' 13's algorithm using BGP control plane data. The first assumption the algorithm makes, it that **most** found BGP paths are *valley-free*. Based on this assumption, missing AS relations can be inferred. When multiple trace routes infer multiple relationships for the same link, the relationship with the highest occurrence is chosen.

Kastelein's algorithm can be split in 6 stages. In the first stage, trace routes and AS relations are loaded, the trace routes are converted to BGP paths, and AS path prepending is removed. The second stage is used to add known correct `s2s` links since the algorithm is not inferring `s2s` links. Because not all ASes in an organisation are interconnected by default, `s2s` links are only added when they are found in either the trace routes or the existing AS relations. The third stage finds links using the converted trace routes that do not follow the *valley-free* constraint or are not yet inferred. These wrong and missing links are grouped in the fourth stage to be inferred in fifth stage. To counter poisoned paths, the fifth stage stores all the possible inferred types per link. The sixth and final stage selects the most inferred type for each link. In the case that not all the links have been inferred, the fifth and sixth stages need to be repeated until no further changes are detected.

- **Stage 1**: *Initialising data and variables*
  Trace routes first need to be translated to BGP paths, after which AS prepending is removed because the algorithm does not support it. The not_p2p flag is initialised and will be used in later stages to determine whether a link can be inferred as `p2p`.

Algorithm Kastelein's 1st stage

1: **create** a trace route list $l_{trace\ routes}$
2: **create** an AS relations list $l_{AS\ relations}$
3: $l_{BGP\ paths} \leftarrow []$ ▷ contains converted trace routes
4: **for each** trace route $[IP_1, IP_2, \ldots, IP_{n-1}, IP_n]$ **in** $l_{trace\ routes}$ **do** ▷ list of IP addresses
5: $p \leftarrow []$
6: **for** $i = n, \ldots, 1$ **do**
7: **find** the smallest IP prefix x that contains IP_i
8: **find** the AS AS_x that holds IP prefix x
9: **add** AS_x **to** p
10: **add** path p **to** $l_{BGP\ paths}$
11: **for each** AS path p **as** $[AS_1, AS_2, \ldots, AS_{n-1}, AS_n]$ **in** in $l_{BGP\ paths}$ **do**
12: **for** $i = 1, \ldots, n-1$ **do**
13: not_p2p$[AS_i, AS_j] \leftarrow$ `false`
14: not_p2p$[AS_j, AS_i] \leftarrow$ `false`
15: **if** $AS_i = AS_{i+1}$ **then** ▷ remove AS path prepending
16: **remove** the i_{th} AS **from** p.

- **Stage 2**: *Finding wrong and missing links*
  For each converted trace route, first determine if the links are present in the AS relation database. If not, mark them missing. For all the links, verify if the inferred types are in line with the *valley-free* approach using the converted trace routes. BGP learned routes first consist of zero or more `c2p` links, then zero or one `p2p` link, followed by zero or more `p2c` links. Finally, an `s2s` can appear anywhere in the path. When a path does not follow this *valley-free* constraint, one or more links have been wrongly inferred according to that path. In other words, a `c2p` link after a `p2c` or `p2p` link is, according to that path, not inferred correctly. The same holds for `p2c` links before `c2p` and `p2p` links. Finally, when there are multiple `p2p` links in a path, all the `p2p` links are marked as wrong.

Algorithm Kastelein's 2nd stage

```
1: for each AS path p as [AS_1, AS_2, ..., AS_{n-1}, AS_n] in in l_{BGP paths} do
2:     p2p_counter ← 0
3:     for i = 1, ..., n − 1 do                                  ▷ finding multiple p2p links in a path
4:         if state[AS_i, AS_{i+1}] = p2p then
5:             p2p_counter ← p2p_counter + 1
6:         else if state[AS_i, AS_{i+1}] = −1 then                ▷ marking missing links in a path
7:             state[AS_i, AS_{i+1}] ← MISSING
8:             state[AS_{i+1}, AS_i] ← MISSING
9:     if p2p_counter > 1 then                                    ▷ marking multiple p2p links in a path
10:        for i = 1, ..., n − 1 do
11:            if state[AS_i, AS_{i+1}] = p2p then
12:                state[AS_i, AS_{i+1}] ← WRONG
13:                state[AS_{i+1}, AS_i] ← WRONG
14:    found ← false
15:    for i = 1, ..., n − 1 do          ▷ finding and marking c2p links after a p2c or p2p link in a path
16:        if state[AS_i, AS_{i+1}] = c2p or state[AS_i, AS_{i+1}] = p2p then
17:            found ← true
18:        else if state[AS_i, AS_{i+1}] = p2c and found then
19:            state[AS_i, AS_{i+1}] ← WRONG
20:            state[AS_{i+1}, AS_i] ← WRONG
21:    index ← 0
22:    for i = 1, ..., n − 1 do                                  ▷ finding first c2p or p2p link in a path
23:        if index = 0 then
24:            if state[AS_i, AS_{i+1}] = c2p or state[AS_i, AS_{i+1}] = p2p then
25:                index ← i
26:    if index ≠ −1 then                          ▷ marking p2c links before a c2p or p2p link in a path
27:        for i = 1, ..., index do
28:            if state[AS_i, AS_{i+1}] = p2c then
29:                state[AS_i, AS_{i+1}] ← WRONG
30:                state[AS_{i+1}, AS_i] ← WRONG
```

- **Stage 3**: *Grouping wrong and missing links*
  All the wrong and missing links found in **stage 2** are grouped to allow **stage 4** to infer possible types.

Algorithm Kastelein's 3rd stage

```
1: l_{missing links} ← []
2: l_{wrong links} ← []
3: for each AS path [AS_1, AS_2, ..., AS_{n-1}, AS_n] in l_{BGP paths} do
4:     for i = 1, ..., n − 1 do
5:         link x ← {AS_i, AS_{i+1}}
6:         if state[AS_i, AS_{i+1}] = MISSING and x not in l_{missing links} then
7:             add x to l_{missing links}
8:         else if state[AS_i, AS_{i+1}] = WRONG and x not in l_{wrong links} then
9:             add x to l_{wrong links}
```

- **Stage 4**: *Inferring possible solutions for wrong and missing links*
  For each wrong and missing link, all the paths are selected in which that wrong of missing link is present. Given the fact that these filtered paths contain enough routing information, the type of the link is inferred. When the order of the path is in the inverted direction, the inferring as c2p becomes p2c and vice versa. The algorithm uses five rules presented below to infer c2p, p2c, and p2p links. The not_p2p flag is used to determine whether a link can be inferred as p2p.

  1. $\boxed{\ldots \rightarrow ? \rightarrow \texttt{c2p} \rightarrow \ldots} \Longrightarrow \texttt{c2p}$
     An unknown link followed by a c2p link is inferred as c2p
     The **not_p2p** flag for this link is set to `true`
  2. $\boxed{\ldots \rightarrow ? \rightarrow \texttt{p2p} \rightarrow \ldots} \Longrightarrow \texttt{c2p}$
     An unknown link followed by a p2p link is inferred as c2p
     The **not_p2p** flag for this link is set to `true`
  3. $\boxed{\ldots \rightarrow \texttt{c2p} \rightarrow ? \rightarrow \texttt{p2p} \rightarrow \ldots} \Longrightarrow$ p2p, c2p, or p2c
     An unknown link in between a c2p link and a p2c link is inferred as
     - p2p when not_p2p is `false`
     - c2p ($p = 0.5$) or p2c ($p = 0.5$) when not_p2p is `true`
  4. $\boxed{\ldots \rightarrow ? \rightarrow \texttt{p2c} \rightarrow \ldots} \Longrightarrow$ c2p p2p, c2p, or p2c
     An unknown link followed by a p2c link is inferred as
     - p2p when **not_p2p** is `false`
     - c2p ($p = 0.5$) or p2c ($p = 0.5$) when **not_p2p** is `true`
  5. $\boxed{\ldots \rightarrow \texttt{p2p} \rightarrow ? \rightarrow \ldots} \Longrightarrow \texttt{p2c}$
     An unknown link after a p2p link is inferred as p2c
     The **not_p2p** flag for this link is set to `true`
  6. $\boxed{\ldots \rightarrow \texttt{p2c} \rightarrow ? \rightarrow \ldots} \Longrightarrow \texttt{p2c}$
     An unknown link after a p2c link is inferred as p2c
     The **not_p2p** flag for this link is set to `true`

  The 6 basic inferring rules above do not take s2s links into account. For example, $\boxed{? \rightarrow \texttt{s2s} \rightarrow \texttt{c2p}}$ should also infer a c2p link. The algorithm will therefore use the following refined rules to infer c2p, p2c, and p2p links:

1. $\boxed{\ldots \rightarrow \texttt{?} \rightarrow \overset{x\text{ times}}{\texttt{s2s}} \rightarrow \texttt{c2p} \rightarrow \ldots} \Longrightarrow \texttt{c2p}$
   An unknown link followed by a possible sequence of s2s links and next a c2p link is inferred as c2p
   The **not_p2p** flag for this link is set to true
2. $\boxed{\ldots \rightarrow \texttt{?} \rightarrow \overset{x\text{ times}}{\texttt{s2s}} \rightarrow \texttt{p2p} \rightarrow \ldots} \Longrightarrow \texttt{c2p}$
   An unknown link followed by a possible sequence of s2s links and next a p2p link is inferred as c2p
   The **not_p2p** flag for this link is set to true
3. $\boxed{\ldots \rightarrow \texttt{c2p} \rightarrow \overset{x\text{ times}}{\texttt{s2s}} \rightarrow \texttt{?} \rightarrow \overset{x\text{ times}}{\texttt{s2s}} \rightarrow \texttt{p2c} \rightarrow \ldots} \Longrightarrow \texttt{p2p}, \texttt{c2p}, \text{ or } \texttt{p2c}$
   An unknown link in between a) a c2p link followed by a possible sequences of s2s links and b) a possible sequences of s2s links followed by a p2c link is inferred as:
   - p2p when **not_p2p** is false
   - c2p ($p = 0.5$) or p2c ($p = 0.5$) when **not_p2p** is true
4. $\boxed{\ldots \rightarrow \texttt{?} \rightarrow \overset{x\text{ times}}{\texttt{s2s}} \rightarrow \texttt{p2c} \rightarrow \ldots} \Longrightarrow \texttt{p2p}, \texttt{c2p}, \text{ or } \texttt{p2c}$
   An unknown link followed by a possible sequence of s2s links and next a p2c link is inferred as:
   - p2p when **not_p2p** is false
   - c2p ($p = 0.5$) or p2c ($p = 0.5$) when **not_p2p** is true
5. $\boxed{\ldots \rightarrow \texttt{p2c} \rightarrow \overset{x\text{ times}}{\texttt{s2s}} \rightarrow \texttt{?} \rightarrow \ldots} \Longrightarrow \texttt{p2c}$
   An unknown link after a possible sequence of s2s links followed by a p2c link is inferred as p2c
   The **not_p2p** flag for this link is set to true
6. $\boxed{\ldots \rightarrow \texttt{p2p} \rightarrow \overset{x\text{ times}}{\texttt{s2s}} \rightarrow \texttt{?} \rightarrow \ldots} \Longrightarrow \texttt{p2c}$
   An unknown link after a possible sequence of s2s links followed by a p2p link is inferred as p2c
   The **not_p2p** flag for this link is set to true

When looking at the first rule in either the basic or refined version, it would be possible to already infer all the unknown links before the $\boxed{\texttt{?}}$ as c2p. This is not done since the algorithm infers missing and wrong links one by one. The same holds for the third rule.

Algorithm Kastelein's 4th stage - basic version

```
1: for each AS link {AS_x, AS_y} in l_missing links and l_wrong links do
2:     c2p[AS_x, AS_y] ← 0                          ▷ number of times link is inferred as c2p
3:     p2c[AS_x, AS_y] ← 0                          ▷ number of times link is inferred as p2c
4:     p2p[AS_x, AS_y] ← 0                          ▷ number of times link is inferred as p2p
5:     for each AS path [AS_1, AS_2, ..., AS_{n-1}, AS_n] in l_BGP paths do
6:         use ← false
7:         index ← −1
8:         reversed ← false
9:         for i = 1, ..., n − 1 do
10:            if AS_x = AS_i and AS_y = AS_{i+1} then
11:                use ← true
12:                index ← i
13:            else if AS_x = AS_{i+1} and AS_y = AS_i then
14:                use ← true
15:                index ← i
16:                reversed ← true
17:        if use then
18:            found_p2c ← false
19:            found_c2p ← false
20:            found_p2p ← false
21:            if state[AS_{index+1}, AS_{index+2}] = c2p or state[AS_{index+1}, AS_{index+2}] = p2p then
22:                found_c2p ← true
23:                not_p2p[AS_i, AS_{i+1}] ← false
24:                not_p2p[AS_{i+1}, AS_i] ← false
25:            else if state[AS_{index-1}, AS_{index}] = c2p and state[AS_{index+1}, AS_{index+2}] = p2c then
26:                if not_p2p[AS_{index}, AS_{index+1}] = false then
27:                    found_p2p ← true
28:                else
29:                    if random(0, 1) < 0.5 then
30:                        found_c2p ← true
31:                    else
32:                        found_p2c ← true
33:            else if state[AS_{index+1}, AS_{index+2}] = p2c then
34:                if not_p2p[AS_{index}, AS_{index+1}] = false then
35:                    found_p2p ← true
36:                else
37:                    if random(0, 1) < 0.5 then
38:                        found_c2p ← true
39:                    else
40:                        found_p2c ← true
41:            else if state[AS_{index-1}, AS_{index}] = p2c or state[AS_{index-1}, AS_{index}] = p2p then
42:                found_c2p ← true
43:                not_p2p[AS_i, AS_{i+1}] ← false
44:                not_p2p[AS_{i+1}, AS_i] ← false
45:            if ( found_c2p and not reversed ) or ( found_p2c and not reversed ) then
46:                p2c[AS_x, AS_y]++
47:                c2p[AS_y, AS_x]++
48:            if ( found_c2p and not reversed ) or ( found_p2c and reversed ) then
49:                c2p[AS_x, AS_y]++
50:                p2c[AS_y, AS_x]++
51:            if found_p2p then
52:                p2p[AS_x, AS_y]++
53:                p2p[AS_y, AS_x]++
```

Algorithm Kastelein's 4th stage - refined version

```
1: for each AS link {AS_x, AS_y} in l_missing links and l_wrong links do
2:     initialise c2p[AS_x, AS_y], p2c[AS_x, AS_y], and p2p[AS_x, AS_y]          ▷ same as in the basic version
3:     for each AS path [AS_1, AS_2, ..., AS_{n-1}, AS_n] in l_BGP paths do
4:         if use then
5:             initialise found_p2c, found_c2p, and found_p2p                    ▷ same as in the basic version
6:             stop ← false
7:             for i = index, ..., n − 1 do                                      ▷ infer rules 1 and 2
8:                 if state[AS_{i+1}, AS_{i+2}] = p2c then
9:                     stop ← true
10:                else if ( state[AS_{i-1}, AS_i] = c2p or state[AS_{i-1}, AS_i] = p2p ) and ( stop = false ) then
11:                    found_c2p ← true
12:                    stop ← true
13:                    not_p2p[AS_index, AS_{index+1}] ← false
14:                    not_p2p[AS_{index+1}, AS_index] ← false
15:            found_1 ← false
16:            stop ← false
17:            for i = index, ..., n − 1 do                                      ▷ infer rule 2
18:                if state[AS_{i+1}, AS_{i+2}] = c2p or state[AS_{i+1}, AS_{i+2}] = p2p then
19:                    stop ← true
20:                else if state[AS_{i+1}, AS_{i+2}] = p2c and stop = false then
21:                    found_1 ← true
22:            found_2 ← false
23:            stop ← false
24:            for i = index, ..., 0 do                                          ▷ infer rule 2
25:                if state[AS_{i-1}, AS_i] = p2c or state[AS_{i-1}, AS_i] = p2p then
26:                    stop ← true
27:                else if state[AS_{i-1}, AS_i] = c2p and stop = false then
28:                    found_2 ← true
29:            found_3 ← false
30:            stop ← false
31:            for i = index, ..., n − 1 do                                      ▷ infer rules 4
32:                if state[AS_{i+1}, AS_{i+2}] = c2p or state[AS_{i+1}, AS_{i+2}] = p2p then
33:                    stop ← true
34:                else if state[AS_{i+1}, AS_{i+2}] = c2p and stop = false then
35:                    found_3 ← true
36:            if ( found_1 = true and found_2 = true ) or ( found_3 = true ) then
37:                if not_p2p[AS_index, AS_{index+1}] = false then
38:                    found_p2p ← true
39:                else
40:                    if random(0, 1) < 0.5 then
41:                        found_c2p ← true
42:                    else
43:                        found_p2c ← true
44:            stop ← false
45:            for i = index, ..., 0 do                                          ▷ infer rules 5 and 6
46:                if state[AS_{i-1}, AS_i] = c2p then
47:                    stop ← true
48:                else if ( state[AS_{i-1}, AS_i] = p2c or state[AS_{i-1}, AS_i] = p2p ) and ( stop = false ) then
49:                    stop ← true
50:                    found_p2c ← true
51:                    not_p2p[AS_index, AS_{index+1}] ← false
52:                    not_p2p[AS_{index+1}, AS_index] ← false
53:            increment c2p[AS_x, AS_y], p2c[AS_x, AS_y], and p2p[AS_x, AS_y]   ▷ same as in the basic version
```

- **Stage 5**: *Select the most inferred type for each wrong and missing link*
  For each inferred wrong and missing link, select the type that is inferred the most and remove the link from the list of wrong and missing links. When **stage 5** is not able to infer all wrong and missing links, **stage 2**, **stage 3**, **stage 4** and **stage 5** are repeated until no further changes are detected.

Algorithm Kastelein's 5th stage

1: **for each** AS link x **as** $\{AS_x, AS_y\}$ **in** $l_{missing\ links}$ **and** $l_{wrong\ links}$ **do**
2: found ← `false`
3: **if** c2p$[AS_x, AS_y] > 0$ **and** c2p$[AS_x, AS_y] \geq \max($ p2c$[AS_x, AS_y],$ p2p$[AS_x, AS_y]\ )$ **then**
4: state$[AS_x, AS_y]$ ← `c2p`
5: state$[AS_y, AS_x]$ ← `p2c`
6: found ← `true`
7: **if** p2c$[AS_x, AS_y] > 0$ **and** p2c$[AS_x, AS_y] \geq \max($ c2p$[AS_x, AS_y],$ p2p$[AS_x, AS_y]\ )$ **then**
8: state$[AS_x, AS_y]$ ← `p2c`
9: state$[AS_y, AS_x]$ ← `c2p`
10: found ← `true`
11: **if** p2p$[AS_x, AS_y] > 0$ **and** p2p$[AS_x, AS_y] \geq \max($ c2p$[AS_x, AS_y],$ p2c$[AS_x, AS_y]\ )$ **then**
12: state$[AS_x, AS_y]$ ← `p2p`
13: state$[AS_y, AS_x]$ ← `p2p`
14: found ← `true`
15: **if** found = `true` **then**
16: **if** x in $l_{missing\ links}$ **then**
17: remove x from $l_{missing\ links}$
18: **else if** x in $l_{wrong\ links}$ **then**
19: remove x from $l_{wrong\ links}$

3.2. AS mock graph generator

The AS mock graph generator creates a topology that mimics the AS topology of the Internet. An AS Mock graph will be used to validate Kastelein's algorithm. When a number of relationship types is removed from or changed in a mock graph, the algorithm, BGP control plane data, and trace routes should be able to completely restore the graph.

Looking at the Internet topology, ASes can be grouped in tiers where ASes in a higher tier, represented by a lower number, have more `p2c` links than ASes in lower tiers. ASes in a tier can be interconnected via `p2p` links. The AS mock graph generator does not produce `s2s` links since their absence should not interfere with the validation of Kastelein's algorithm. The generator has the following parameters:

- The number of tier-1 ASes S_1
- The AS multiplier per tier S_x
- The number of tiers N
- The probability that a n_{th} tier AS has two `c2p` links $p_n^{\text{second c2p}}$
- The probability that two n_{th} tier ASes share a `p2p` link p_n^{p2p}

The first tier consists of S_1 ASes, and a next tier has S_x times more ASes than the previous tier. In order to have a connected graph, and make sure that all BGP traffic can flow through the mesh of tier-1 ASes, all tier-1 ASes need to be interconnected using `p2p` links. If this is not the case, a BGP announcement has to traverse two `p2p` links in the tier-1 mesh which is not allowed. ASes in lower tiers do not necessarily need to be interconnected because announcements can always make use of `c2p` links to higher tier ASes. However, each AS that is not a tier-1 AS has to have at least one `c2p` link to be able to reach other parts of the AS graph.

The algorithm is split up in three stages. In the first stage the variables are initialised and each tier receives its amount of ASes. The second stage interconnects ASes in each tier with `p2p` links with the probability p_n^{p2p}. To ensure that all tier-1 ASes are interconnected, p_1^{p2p} equals 1. The third and final stage adds 1 or 2 `c2p` and `p2c` links per AS between tiers given the probability $p_n^{\text{second c2p}}$.

- **Stage 1**: *Initialising variables and adding ASes to tiers*

Algorithm MGG 1st stage

choose S_1, S_x, and N ▷ number of tier-1 ASes, AS multiplier per tier, and number of tiers
counter $\leftarrow 1$
for $n = 1, \ldots, N$ **do**
  $T_i \leftarrow []$ ▷ ASes in the i_{th} tier
  choose $p_n^{\texttt{p2p}}$ ▷ probability that two n_{th} tier ASes share a p2p link
  if $n = 1$ **then**
    size $\leftarrow S_1$
  else
    size $\leftarrow$ size $\times S_x$
    choose $p_n^{\text{second c2p}}$ ▷ probability that a n_{th} tier AS has two c2p links
  for $i =$ counter, $\ldots$, counter + size **do**
    add AS_i **to** T_n

- **Stage 2**: *Interconnecting all the ASes in the separate tiers with p2p links*
  Given the probability $p_n^{\texttt{p2p}}$, two n_{th} tier ASes will share a p2p link.

Algorithm MGG 2nd stage

for $n = 1, \ldots, N$ **do**
  for $i = 1, \ldots, len(T_n)$ **do**
    for $j = 1, \ldots, len(T_n)$ **do**
      $AS_x \leftarrow T_n[i]$
      $AS_y \leftarrow T_n[j]$
      if $AS_x \neq AS_y$ **and** $random(0,1) < p_n^{\texttt{p2p}}$ **then**
        state$[AS_x, AS_y] \leftarrow$ p2p
        state$[AS_y, AS_x] \leftarrow$ p2p

- **Stage 3**: *Interconnecting a lower tier AS with a lower tier AS using c2p and p2c links*
  Given the probability $p_n^{\text{second c2p}}$, A n_{th} tier ASes will have an extra c2p link.

Algorithm MGG 3rd stage

for $n = 2, \ldots, N$ **do**
  for $i = 1, \ldots, len(T_n)$ **do**
    choose a *random* AS_x from the $n-1_{th}$ tier
    state$[AS_x, AS_i] \leftarrow$ p2c
    state$[AS_i, AS_x] \leftarrow$ c2p
    if $random(0,1) < p_n^{\text{second c2p}}$ **then**
      choose a *random* $AS_y \neq AS_x$ from the $n-1_{th}$ tier
      $[AS_y, AS_i] \leftarrow$ p2c
      $[AS_i, AS_y] \leftarrow$ c2p

3.3. BGP Simulator

The BGP simulator simulates BGP traffic, and as a result, allows the user to see where data for a prefix would flow. The user first loads the AS topology and sets forwarding rules for the ASes. Next, the user inserts announcements and the simulator starts simulating BGP traffic. The simulator can visualise AS relations and routing states for smaller topologies. The working of the simulation tool is divided into three stages: the AS Graph (*ASG*), Routing Table (*RT*) and Routing State (*RS*) stages.

The *ASG* stage stores the relationships and parameters of ASes and is used for simulating the actual BGP traffic. The *RT* stage contains information about which RIB entry an AS will use to reach a certain prefix. The *RS* stage contains information about which AS ends up at which source AS depending on a prefix and also provides a connectivity summary. The working of the *ASG*, *RT* and *RS* stages are described in detail in sections 3.3.1, 3.3.2 and 3.3.3. After the working is described, section 3.3.4 presents simulator results using sample AS topologies.

3.3.1. AS Graph (*ASG*) stage

The *ASG* stage has four main usages. First, the *ASG* stage stores the relations between ASes. For each neighbour of an AS, the LOCAL_PREFERENCE and the type of link (`p2c`, `c2p`, `p2p` or `s2s`) are stored. Second, in the *ASG* stage the forwarding rules of an AS can be set. For example, an AS can or cannot forward an announcement coming from a `p2p` link to a `p2p` link. With the possibility to change the forwarding rules per AS, route leaks can be simulated. A cause of route leaks is when ASes forward BGP announcements to ASes that are not supposed to receive them.

Next, the *ASG* stage has an IN, RIB and OUT table for each AS to simulate a BGP router. All the announcements received by an AS are stored in the IN table before being processed. During processing the announcements may be added to the Routing-Information-Base (RIB) table and the OUT table. This depends on the BGP rules. Announcements in the OUT tables are sent to other ASes' IN tables if forwarding rules allow for this.

Listing 3.1 shows the data structure of an announcement used in the *ASG* and *RT* stages. The variable *LOCAL_PREFERENCE* is the LOCAL_PREFERENCE which the AS has of the neighbouring AS the announcement was received from. In this case, the LOCAL_PREFERENCE of AS4 at AS5 is 100. Next, the variable *extra_path_length* is used to simulate AS path prepending, the process of adding a sequence of the same AS to a path in order to make the route less attractive. Finaly, the variable *good*, either true of false, indicates if the announcement originated from an AS that is allowed to announce the prefix.

Listing 3.1: Data structure of an announcement

```
{
  "LOCAL_PREFERENCE": 100,
  "extra_path_length": 0,
  "good": true,
  "path": ["1","2","3","4","5"],
  "prefix": "192.168.0.0/24",
  "prefix_length": "24",
  "source_AS": "1",
}
```

IN, RIB and OUT table

The flowing of announcement from and to IN, RIB and OUT tables can be described in three steps. The first step initialises the announcement after which step two and three are repeated until no more changes are recorded in the *ASG* stage. The list below contains a description of the three steps:

- **Step 1**: An announcement - either marked *good* or *malicious* - is added at the source AS. During this step the announcement is stored as a route in the RIB table of the source AS, and is added to the OUT table of the source AS.
- **Step 2**: For all ASes all announcements that are currently stored in the OUT tables are copied to the IN tables of neighbouring ASes if forwarding rules allow this. At this time, the neighbouring AS, where the announcement is copied to, is added to the path in the announcement. When the route stored in OUT table was created in **step 1**, i.e., the length of the route is 1 and therefore not received but created, all forwarding rules are ignored: the route is copied to all IN tables of neighbouring ASes. This is normal behaviour when an AS starts announcing a prefix to ensure that all the ASes can have a route to that prefix.

- **Step 3**: All announcements now stored in the IN tables are copied to the RIB and/or OUT table of that same AS if BGP rules allow this. The simulator uses the following decision criteria, in the given order, to accept a route. If a criteria is not met, the simulator moves the next criterion. If a criterion is met, the simulator performs its included action(s) and the rest of criteria are ignored:

    1. The given prefix is not yet present in the RIB table
    Add the announcement to *RIB* and *OUT* tables
    2. The LOCAL_PREFERENCE value for the prefix is the highest compared to all other LOCAL_PREFERENCE values of the same prefix in the RIB table.
    Add the announcement to *RIB* and *OUT* tables
    3. The path length for the prefix is the shortest compared to all other path lengths of the same prefix in the RIB table.
    Add the announcement to *RIB* and *OUT* tables
    4. The path length for the prefix equals the largest path lengths of the same prefix in the RIB table.
    Add the announcement to *RIB* table only
    5. The path length for the prefix is smaller than the largest path length of the same prefix in the RIB table.
    No action
    6. **Add the announcement to *RIB* and *OUT* tables**

At this point, all the announcements in the *ASG* stage have been sent and received by the ASes. Steps one, two and three can be repeated when a new announcement is added. Multiple announcements can be added at the same time before the iterating process starts. This means that a different order of adding and iterating will lead to different results. BGP routes can also be directly added to the *ASG* stage therefore bypassing a) the IN and OUT tables, and b) the BGP rules in place to determine how announcements flow through the AS graph. In addition to simulating BGP traffic, learned routes need to be added to verify that the simulator is working properly. These learned routes are obtained from RIPE and RouteViews RRCs.

3.3.2. Routing Table (*RT*) stage

The *RT* stage is created using the *ASG* stage. For each AS found in the *ASG* stage, the actual used route to reach a prefix - found in its RIB table - is retrieved and stored. In other words, if an AS wants to send data to an IP-address belonging to a prefix, it has to choose which route to use from the *ASG* stage' RIB table. As a result of the way BGP works, an AS can have multiple routes to a prefix. Whether a route is used depends on the LOCAL_PREFERENCE, AS_PATH and PREFIX_LENGTH attributes.

3.3.3. Routing Stage (*RS*) stage

The *RS* stage uses the data from the *RT* stage to compute where the data of an AS destined for a prefix will eventually end up. An AS can be labeled GOOD, BAD, CONTESTED, or NO_ROUTING. For a GOOD AS its data destined for that prefix is received by a source AS that is allowed to announce that prefix. For a BAD AS its data destined for the prefix is received by a source AS that is not allowed to announce the prefix. An AS labelled GOOD and BAD is labelled CONTESTED. The label NO_ROUTING is given to an AS which cannot reach that prefix.

3.3.4. Simulator Output

Following the description of the working of the simulation tool, the simulation results are presented and explained. To start, AS relations in the *ASG* stage and the routing state can be visualised. In order to understand the figures, it is necessary to know how to read the graphs. This is explained below. Next, AS relations are shown, and finally, a fictional hijack is simulated and visualised.

Reading the graphs

Figure 3.1 shows that in all relations the arrow determines the direction. In other words, there is a `p2p` from AS1 to AS2 and a `p2p` relation from AS2 to AS1. A `p2p` relation is depicted as a normal solid line. An `s2s` relation is represented by a dotted line. A `c2p` relation is visualised using a bold line. Finally, a dashed line depicts a `p2c` relation. See figure 3.2, an AS can be any of the following:

- GOOD
  The data for a given prefix is routed to a *good* source AS.
  Displayed as an oval.
- BAD
  The data for a given prefix is routed to a *malicious* source AS.
  Displayed as a hexagon.
- CONTESTED
  The data for a given prefix can be routed to a *good* and a *malicious* source AS.
  Displayed as a diamond.
- NO_ROUTING
  The data for a given prefix cannot be routed to any source AS.
  Displayed as a rectangle.

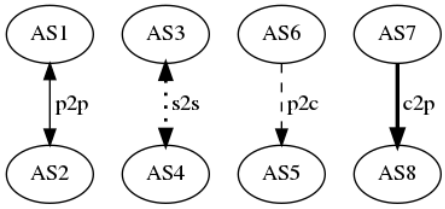


Figure 3.1: Types of AS relations

Figure 3.2: Types of ASes

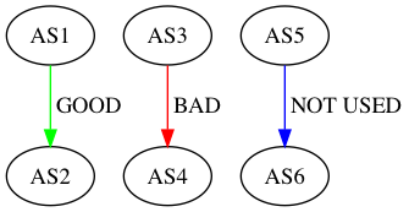


Figure 3.3: Types of routes

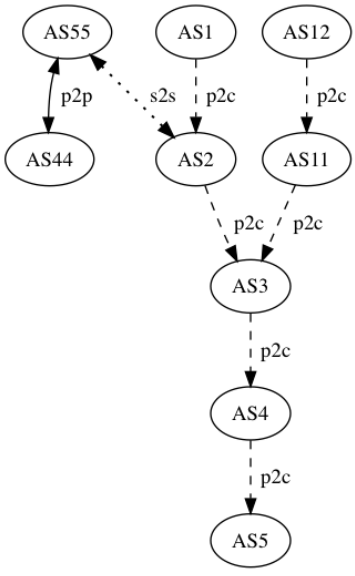


Figure 3.4: Example of AS relations

As shown in figure 3.3, a route can be either GOOD, BAD or `NOT_USED`. In the case that a route is not used, it is not possible to see if it is either coming from a *good* or a malicious source AS and is colored blue. A route originating from a *good* source AS, is green. If the route originates from a *malicious* source AS, it is colored red. The properties of a route are added as a label to the link between ASes. They consist first of the AS number, followed by the LOCAL_PREFERENCE, PATH_LENGTH and PREFIX_SIZE.

Visualising AS relations

Figure 3.4 shows the AS relations. AS4, AS5 and AS11 all only have a single provider. AS3 has two providers. There is a `p2p` relation between AS44 and AS55 while AS2 and AS55 are part of the same organisation and therefore have an `s2s` link between them. Only `p2c` links are drawn here: a `p2c` link also implies a `c2p` link.

Visualising BGP routes

This section simulates and visualises a fictional hijack. The same AS graph has been used as in figure 3.4. However, the link between AS2 and AS55 is removed. This results in the configuration shown in figure 3.5. First, AS1 starts announcing the prefix *192.168.0.0/24*. Figure 3.6 shows how the announcements have flown through the AS graph. AS44 and AS55 cannot receive any announcements from AS1 since they are not connected to AS1 in any way. However, AS11 and AS12, which can reach AS1, do not have a route to *192.168.0.0/24*. According to the forwarding rules, a route learned from a `p2c` link cannot be sent over the `c2p` link from AS3 to AS11. Data sent from green oval shaped ASes to the prefix *192.168.0.0/24* will end up at a *good* source AS. Blue boxed shaped ASes cannot reach the prefix.

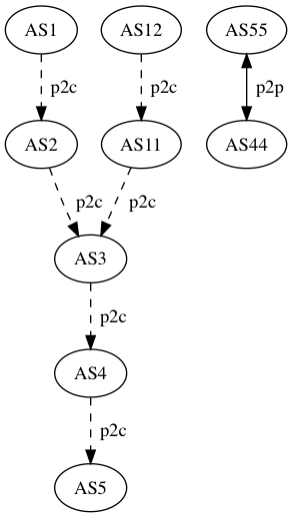


Figure 3.5: AS topology

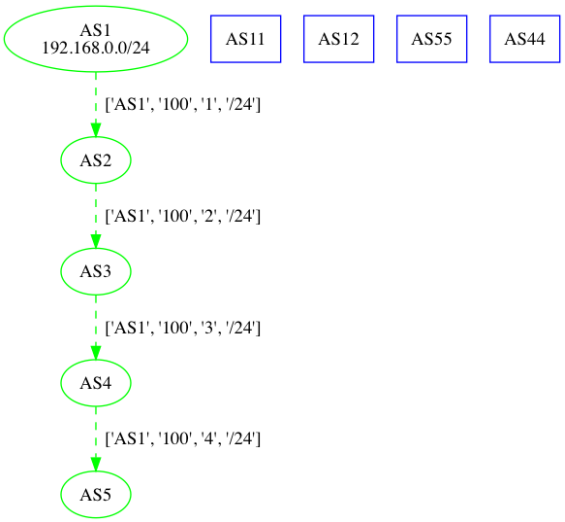


Figure 3.6: AS1 announcing *good* prefix *192.168.0.0/24*.

Next, *malicious* AS12 also starts announcing prefix *192.168.0.0/24*. This is visualised in figure 3.7. The *malicious* announcement coming from AS11 and received by AS3 is not passed on to AS2 due to the forwarding rules in place. However, it is sent down to AS4 and AS5. AS3 cannot decide which route to choose: both the LOCAL_PREFERENCE and the PATH_LENGTH are the same for both routes. As a result, AS3 is marked CONTESTED meaning it cannot decide to which AS it has to send data destined for prefix *192.168.0.0/24*. Because AS4 and AS5 have to pass AS3, they are also marked CONTESTED. AS44 and AS55 cannot receive any announcements from AS1 and AS12 since they are not connected to them in any way.

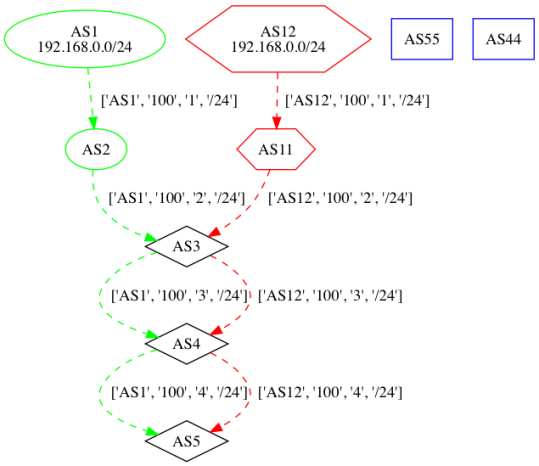


Figure 3.7: *good* AS1 announcing prefix *192.168.0.0/24* and *malicious* AS12 also announcing *192.168.0.0/24*.

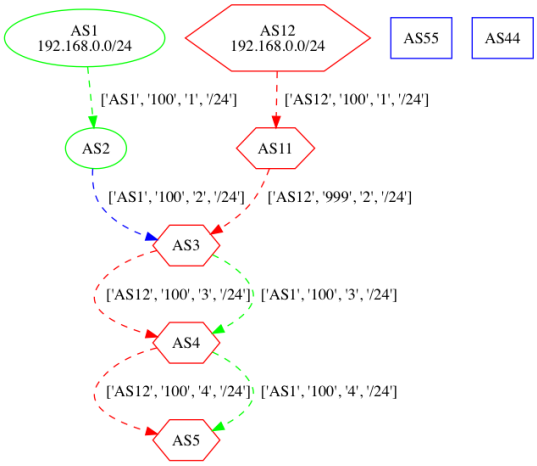


Figure 3.8: *good* AS1 announcing prefix *192.168.0.0/24* and *malicious* AS12 also announcing *192.168.0.0/24*. Note that AS3 has a higher LOCAL_PREFERENCE of 999 of AS11.

Finally, in figure 3.8, AS3 increases the LOCAL_PREFERENCE to 999 for announcements coming from AS11. AS3 has to decide which path to take: to AS1 via AS2 or to AS12 via AS11. Since the LOCAL_PREFERENCE is higher for AS12 this path is chosen. As a result, data coming from the cone of AS3 ends up at AS12 also resulting in the BAD AS4 and AS5. The announcement from AS2 to AS3 is not used, therefore colored blue. Note that the *good* announcement coming from AS1 is still visible at AS5. This is because the announcement of AS1 is first iterated through the AS graph before the *malicious* announcement from AS12 is added. Therefore, AS5 now has two routes to choose from. But, even it is chooses the *good* route, data headed for *192.168.0.0/24* still ends up at the AS that is hijacking the prefix.

Routing Table stage statistics

The amount of data in the RT stage, where the actual used routes to reach prefixes are retrieved and stored, can quickly grow when multiple ASes and announcements are added. To give an insight, and make it easier to perform debugging, the simulator generates an overview that groups per prefix which source AS(es) an AS uses to reach that prefix.

See the example in listing 3.2. In *data*, to reach prefix 192.168.0.0/24, AS1, AS2, and AS3 use routes that lead to AS1. AS1 uses its own announcement. AS4 either uses the route originated in AS1 or in AS4. The *NO_ROUTING* list contains ASes that cannot reach the prefix 192.168.0.0/24.

In *summary*, instead of a list, the amount of ASes is provided.

Listing 3.2: Example of RT stage statistics

```
{
  "data": {
    "192.168.0.0/24": {
      "AS1": ["AS1","AS2","AS5"],
      "AS1 + AS12": ["AS4"],
      "AS12": ["AS11","AS12","AS3"],
      "NO_ROUTING": ["AS55","AS44"]
    }
  },
  "summary": {
    "192.168.0.0/24": {
      "AS1": 3,
      "AS1 + AS12": 1,
      "AS12"      : 3,
      "NO_ROUTING": 2
    }
  }
}
```

Routing State stage statistics

The *RS* stage, which holds where the data of an AS destined for a prefix will eventually end up, also provides an overview. GOOD, BAD, CONTESTED, and NO_ROUTING ASes are grouped as well as good and bad source ASes. In *summary*, the amount and percentage of GOOD, BAD, CONTESTED, and NO_ROUTING ASes are presented.

Listing 3.3: Example of RS stage statistics

```
{
  "data": {
    "bad_ASes": ["AS11","AS12","AS3","AS5","AS4"],
    "bad_source_ASes": ["AS12"],
    "contested_ASes": [],
    "good_ASes": ["AS1","AS2"],
    "good_source_ASes": ["AS1"],
    "no_routing_ASes": ["AS55","AS44"]
  },
  "summary": {
    "keys" :
      ["good","bad","contested","no_routing"],
    "amount" :
      [ 0, 7, 0, 2 ],
    "percentage" :
      [ 0.0, 77.78, 0.0, 22.22 ]
  }
}
```

4

Validation & Results

The previous chapter explained how Kastelein's algorithm functions, how the AS mock graph generator can be tuned, and how BGP traffic is simulated using the BGP simulator. This chapter validates Kastelein's algorithm with BGP control plane data, trace routes, and a mock graph from which specific links are changed and removed and need to be restored. The algorithm's performance is further determined and compared with other AS relation data sets and topology generating algorithms: to be able to compare these data sets and algorithms, the topologies generated by the algorithms are used. Finally, the BGP simulator's scalability is determined after which the simulator is validated using BGP control plane data.

4.1. Validating Kastelein's algorithm

Kastelein's algorithm is validated with the use of a generated mock graph which mimics the AS topology of the Internet. When relationship types of links are removed from a mock graph or changed in a mock graph, the algorithm should be able to completely restore the mock graph using the trace route data based on the original mock graph. Therefore, a method to generate this data using a mock graph is also needed.

In order to generate trace routes using a mock graph, the BGP simulator is used. The mock graph is first loaded into the simulator. Then, a prefix announcement is added to every AS. The size of the IP prefix is not relevant: it is only important to know how BGP traffic flows between ASes with separate resources. Therefore, IP prefixes must not overlap. After the IP prefix announcements are loaded, BGP traffic is simulated using the *AS Graph* stage of the BGP simulator. Next, the actual used BGP routes for reaching prefixes are generated in the BGP simulator's *Routing Table* stage. These BGP routes, after having been inverted, can be used as trace routes: where trace route paths flow towards the AS announcing the prefix, BGP paths start in the announcing AS. To mimic the fact that not all ASes provide their routing data, BGP routes originating from certain ASes can be removed from the set of generated trace routes.

Five different cases that involve the removing of the relationship types of links from a mock graph and changing of the relationship types of links in a mock graph are presented here. The removal of a link's relationship type only renders the type unknown: the connection itself remains intact. The generated scatter plots presented in the next chapter will show on the x-axis, depending on the case, the average percentage of relationship types of links removed from the mock graph and/or the percentage of links changed in the mock graph over multiple runs. The y-axis shows a) the percentage of links where the relationship type differs from the relationship type in the original mock graph or b) the percentage of links that are not inferred in the mock graph produced by the algorithm but are present in the original mock graph. To be able to compare the 5 cases, the same mock graph is used.

Of the 70k ASes that exist, around 219 (≈ 0.31%) provide routing data to route collectors. Therefore, to stay close to this percentage, only 0.5%, 1%, 2%, and 3% of all ASes in the mock graph will be used to generate trace routes for Kastelein's algorithm. To capture how the algorithm performs when more ASes are used, the percentages 25%, 50%, 75% and 100% will also be included. All measurements in all 5 cases are run 10 times. The average results are depicted in scatter plots.

- **Case 1**: *Removing relationship types of randomly chosen links*
  This case demonstrates the algorithm's ability to restore the mock graph when the relationship types of links are not inferred. Knowing how many relationship types of links from a mock graph can be removed while the algorithm is still able to restore the original mock graph, is useful since it can be used to indicate how well the algorithm performs in improving *real* incomplete and partially wrong AS relation data sets. In steps of 5%, the relationship types of 5% to 100% of randomly chosen links in the mock graph are removed. The same method is repeated for 0% to 20% in steps of 1%.
- **Case 2**: *Changing relationship types of randomly chosen links*
  This case demonstrates the algorithm's ability to restore the mock graph where part of the links are wrongly inferred. As in **case 1**, the same reasoning holds: how many relationship types of links can be altered in the mock graph before the algorithm can no longer restore it? In steps of 5%, the relationship types of 0% to 100% of the randomly chosen links in the mock graph are changed. For the reason given in **case 1**, the same method is repeated for 0% to 20% in steps of 1%. A p2p relationship type is changed in either a c2p or p2c relationship type with a probability of 50% respectively. A c2p or p2c relationship type is changed to a p2p relationship type.
- **Case 3**: *Removing and changing relationship types of randomly chosen links*
  In the current *real-world* known Internet topology, some links are wrongly inferred or not inferred at all: this case closer represents the *real-world* use case by combining **case 1** and **case 2**. In steps of 5%, the relationship types of 0% to 100% of the randomly chosen links in the mock graph are first altered. Then the same percentage of relationship types of randomly chosen links are removed. For the reason given in **case 1**, the same method is repeated for 0% to 20% in steps of 1%. Relationship types of links are changed in the same manner as described in **case 2**.
- **Case 4**: *Removing randomly chosen c2p and p2c relationship types involving customer-less ASes*
  This case focuses on the algorithm's ability to infer c2p and p2c links. In steps of 5%, the relationship types of 0% to 100% of randomly chosen c2p and p2c links are removed involving ASes that do not contain p2c links. For the reason given in **case 1**, the same method is repeated for 0% to 20% in steps of 1%. Kastelein's algorithm contains more inferring rules, see section 3.1 on page 27, for c2p and p2c relationship types than for p2p relationship types. Furthermore, a link that can be inferred as p2p can also be inferred as c2p or p2c and not break the *valley-free* constraint. Therefore it would be interesting to see how the Kastelein's algorithm handles both c2p and p2c links, and p2p links. This case only involves customer-less ASes to validate that Kastelein's algorithm is capable of correctly inferring all c2p and p2c links with inferring rules 1, 2, and 5 in section 3.1 on page 27.
- **Case 5**: *Removing relationship types of randomly chosen p2p links*
  This case, compared to **case 1**, only tests the algorithm's ability to infer p2p links. In steps of 5%, 0% to 100% of the relationship types of randomly chosen p2p links are removed from the mock graph. For the same given in **case 1**, the same method is repeated for 0% to 20% in steps of 1%. The same reason to include this case holds as the reason given in **case 4**.

Mock graph design

The mock graph used in these 5 cases should mimic the *real-world* AS topology. However, the number of generated BGP paths using a mock graph with the size of the entire Internet will be too large and the currently known AS Internet topology contains wrong relationship types. Therefore, a smaller mock graph that still mimics the structure of the Internet is desired. This mock graph should not contain any wrong relationship types to be able to compare results from all 5 cases. The used parameters for the mock graph, introduced in section 3.2 on page 30, are substantiated next:

- *The number of tiers N*:
  There are 3 types of ASes: tier-1, tier-2, and tier-3. Therefore $\boxed{N = 3}$.

- *The number of tier-1 ASes S_1 and the AS multiplier per tier S_x*:
  Tier-1 ASes are *transit-free*: they can reach the entire Internet via their customer cone or via the customer cones of their peers. The customer cone is the set of IPv4 and IPv6 addresses that is either owned by the AS or can be reached by visiting its customers, and also includes the customers of the customers. The number of tier-1 ASes varies from 7 to 25 according to different sources [8] [9]: determining whether an AS is tier-1 is difficult since no clear definition exists.

  Given that tier-1 consists of S_1 ASes, and a next tier has S_x more ASes than the previous tier, the total number of ASes N_{total} in a mock graph can be calculated using the formula $\boxed{N_{\text{total}} = S_1 + S_1 \cdot S_x + S_1 \cdot {S_x}^2}$. As a consequence of hardware limitations, the simulator can only handle around 5000 ASes. This limitation has no negative effect on the validation results as long as the tier-1, tier-2, and tier-3 structure is kept. To accomplish this structure, the AS multiplier per tier S_x is first calculated for 70k ASes and 16 tier-1 ASes: the average of 7 and 25 tier-1 ASes [8] [9]. $70000 = 16 + 16 \cdot S_x + 16 \cdot {S_x}^2$ renders $S_x \approx 65$. To keep this tier-1, tier-2, and tier-3 structure, S_x is fixed which leaves the variable S_1 to solve: $5000 = S_1 + S_1 \cdot 65 + S_1 \cdot 65^2$ renders $S_1 \approx 1.16$. S_1 needs to be an integer and needs to have at least one p2p link in tier-1, therefore $\boxed{S_1 = 2}$. Finally, with 5000 ASes, $5000 = 2 + 2 \cdot S_x + 2 \cdot {S_x}^2$ renders $\boxed{S_x \approx 49}$.

- *The probability that there is a p2p link between two tier-1 ASes p_1^{p2p}*:
  To ensure that the entire graph is connected, all tier-1 ASes have to be connected via p2p links. Therefore $\boxed{p_1^{\text{p2p}} = 1}$

- *The probability that there is a p2p link between two tier-2 ASes p_2^{p2p}*:
  Tier-2 ASes **may** share a p2p. Around 25 large and important tier-2 ASes exist [19]: based on AS Rank's *serial-1* data set [17], the average number of p2p links between them is $\boxed{7.15}$. Using these averages, the probability that two tier-2 ASes share a p2p link is $\boxed{p_2^{\text{p2p}} = \frac{7.15}{25 \cdot 2} = 0.143}$.

- *The probability that there is a p2p link between two tier-3 ASes p_2^{p2p}*:
  Since tier-3 ASes almost solely purchase transit, i.e., use c2p links to reach other parts of the Internet, the probability that two tier-3 ASes are peered is assumed to be 0. Therefore $\boxed{p_3^{\text{p2p}} = 0}$

- *The probability that a tier-2 AS has two c2p links $p_2^{\text{second c2p}}$*:
  To determine the probability that a tier-2 AS has a second c2p link to a tier-1 AS, the average number of c2p links between tier-2 and tier-1 ASes is calculated. Using AS Rank's *serial-1* data set, a tier-2 AS has either 2.3 or 3.3 links to a tier-1 AS when 7 or 25 ASes are considered tier-1 respectively [8] [9]: to be more redundant in the case of link failures and provide more efficient routing for tier-3 ASes, tier-2 ASes have multiple c2p links to tier-1 ASes. Therefore $\boxed{p_2^{\text{second c2p}} = 1}$

- *The probability that a tier-3 AS has two c2p links $p_3^{\text{second c2p}}$*:
  Since tier-3 ASes almost solely purchase transit, i.e., tier-3 ASes use c2p links to reach other parts of the Internet, the probability that two tier-3 ASes are connected is assumed to be 0. Therefore $\boxed{p_3^{\text{second c2p}} = 0}$

Because the Internet topology is highly complex, a simplified model as the one presented above can never precisely represent the AS topology structure of the Internet. This model neglects the presence of Internet Exchange Points (IXPs) where clusters of ASes are interconnected with p2p links. Furthermore, ASes with more than two c2p links exist, and the current randomised p2p link placement method does not always represent the AS topology where there are clusters of ASes interconnected via p2p links. However, Kastelein's algorithm is designed in such a way that knowledge of the exact topology is not needed: the only requirement is that the provided BGP paths or trace routes follow the *valley-free* constraint.

Case 1: Removing relationship types of randomly chosen links

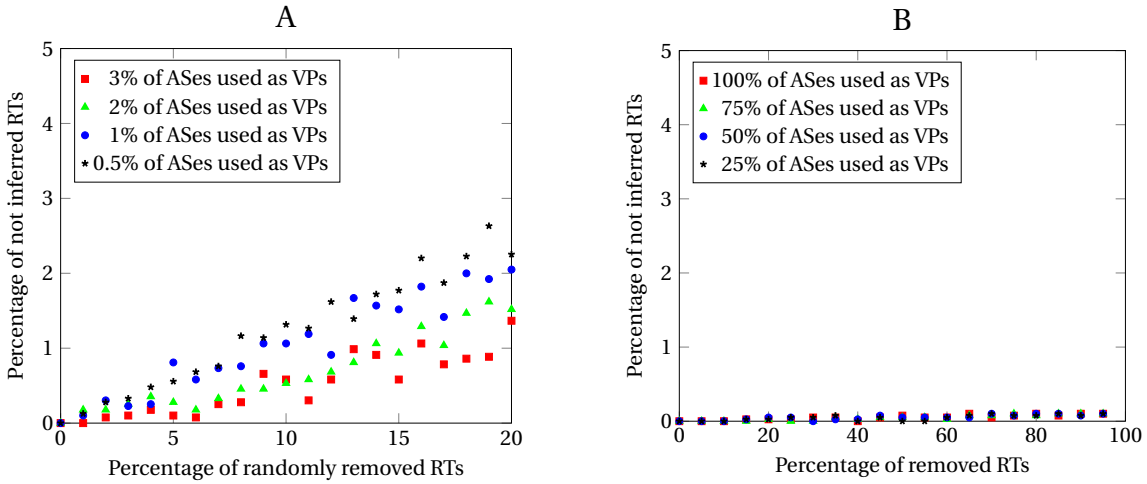


Figure 4.1: The ability of Kastelein's algorithm to infer removed relationship types of randomly chosen links from a mock graph.
The **x-axis** shows the percentage of randomly removed relationship types of links from the mock graph used by the algorithm.
The **y-axis** shows the percentage of relationship types of links not inferred by the algorithm compared to the total links in the mock graph from which no relationship types are removed.
RT = relationship type. Vantage Points (**VPs**) are ASes providing routing data.

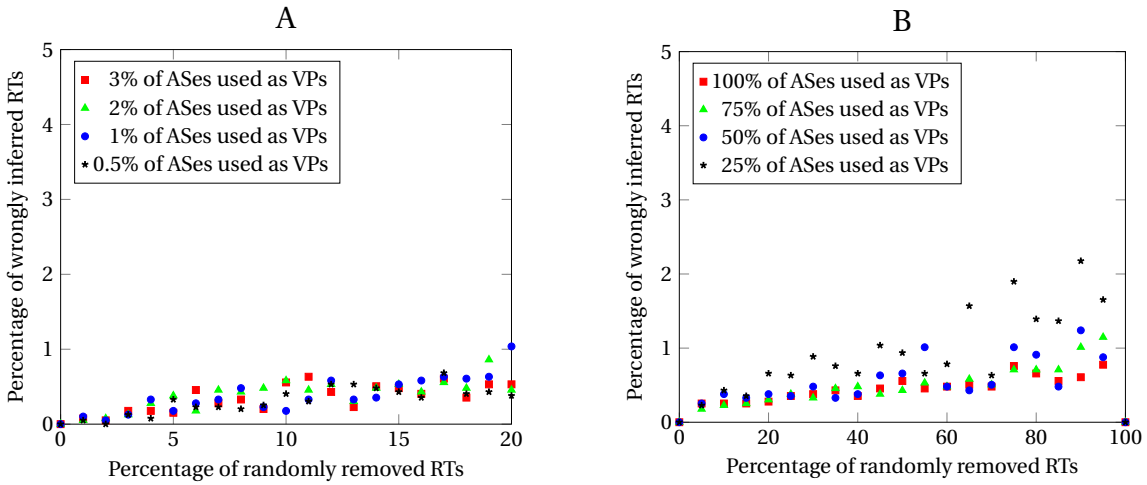


Figure 4.2: The ability of Kastelein's algorithm to infer removed relationship types of randomly chosen links from a mock graph.
The **x-axis** shows the percentage of randomly removed relationship types of links from the mock graph used by the algorithm.
The **y-axis** shows the percentage of relationship types of links wrongly inferred by the algorithm compared to the links in the mock graph from which no relationship types are removed.
RT = relationship type. Vantage Points (**VPs**) are ASes providing routing data.

Analysis of case 1

The higher the percentage of VPs, the more not inferred relationship types of links that can be inferred. The more relationship types of randomly chosen links that are removed, the higher the number of wrongly inferred relationship types. When at least 25% of the ASes are used as VPs, Kastelein's algorithm is capable of re-inferring all the relationship types of links with a maximum of 4% of the relationship types wrongly inferred.

The results involving 0.5%, 1%, 2%, and 3% of the ASes used as VPs most resemble the *real-world*. For these percentages of VPs hold that for up to 20% every X% of removed relationship types only a maximum of $\frac{X}{4}$% cannot be inferred and $\frac{X}{20}$% will be wrongly inferred. The results presented in figures 4.1 and 4.2 show that Kastelein's algorithm is capable of restoring not inferred relationship types when the mock graph itself does not contain any wrongly inferred relationship types.

Case 2: Changing relationship types of randomly chosen links

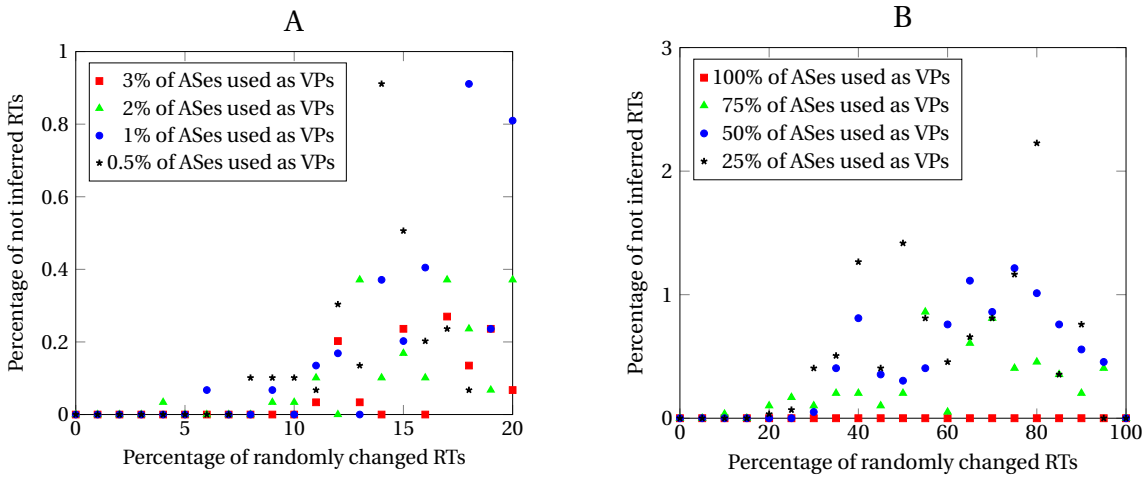


Figure 4.3: The ability of Kastelein's algorithm to infer changed relationship types of randomly chosen links in a mock graph.
The **x-axis** shows the percentage of changed relationship types of randomly chosen links in the mock graph used by the algorithm.
The **y-axis** shows the percentage of relationship types of links not inferred by the algorithm compared to the total links in the mock graph in which no relationship types are changed.
RT = relationship type. Vantage Points (**VPs**) are ASes providing routing data.

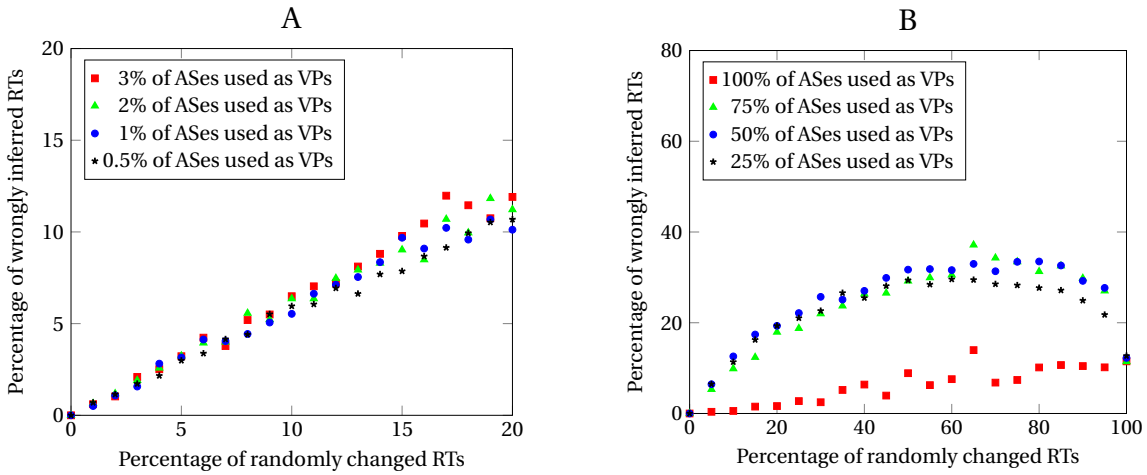


Figure 4.4: The ability of Kastelein's algorithm to infer changed relationship types of randomly chosen links in a mock graph.
The **x-axis** shows the percentage of changed relationship types of randomly chosen links in the mock graph used by the algorithm.
The **y-axis** shows the percentage of links wrongly inferred relationship types by the algorithm compared to the links in the mock graph in which no relationship types are changed.
RT = relationship type. Vantage Points (**VPs**) are ASes providing routing data.

Analysis of case 2

Contrary to **case 1**, the percentage of not inferring links does not increase linearly when more relationship types of links are changed. Kastelein's algorithm **stage 3** - where wrong and not inferred relationship types are found and removed - can remove more relationship types than initially were changed in the mock graph. This amount of removed relationship types highly depends on the other relationship types in the mock graph.

The algorithm is more accurate in re-inferring not inferred relationship types than in re-inferring wrongly inferred relationship types. Graph A in figure 4.4 shows that the algorithm correctly infers about half of the changed relationship types therefore improving the AS relationship set. Graph B in figure 4.4 shows that when 25%, 50%, and 75% of the ASes are used as VPs and more than 50% of the relationship types in the mock graph are changed, the percentage of wrongly inferred links stabilises. This is caused by the way in which the relationship types are changed. `c2p` and `p2c` links are changed to `p2p` links which can be easily detected by the algorithm. `p2p` links are changed in either `c2p` or `p2c` links. The algorithm has a higher error rate in

inferring these relationship types as any p2p link can also be a c2p or p2c in a mock graph that only contained *valley-free* paths.

When 100% of the relationship types have been changed in graph B in figure 4.4, all the relationship types that were changed into p2p relationship types are detected and removed. The relationship types that are changed into either c2p or p2c are a) not spotted by the algorithm and b) do not influence the inferring of other links and are therefore the only links that are wrongly inferred. In graph B in figure 4.4, the percentage of wrongly inferred relationship types when 100% of the relationship types are changed is equal to the percentage of c2p and p2c links in the original mock graph.

Case 3: Removing and changing relationship types of randomly chosen links

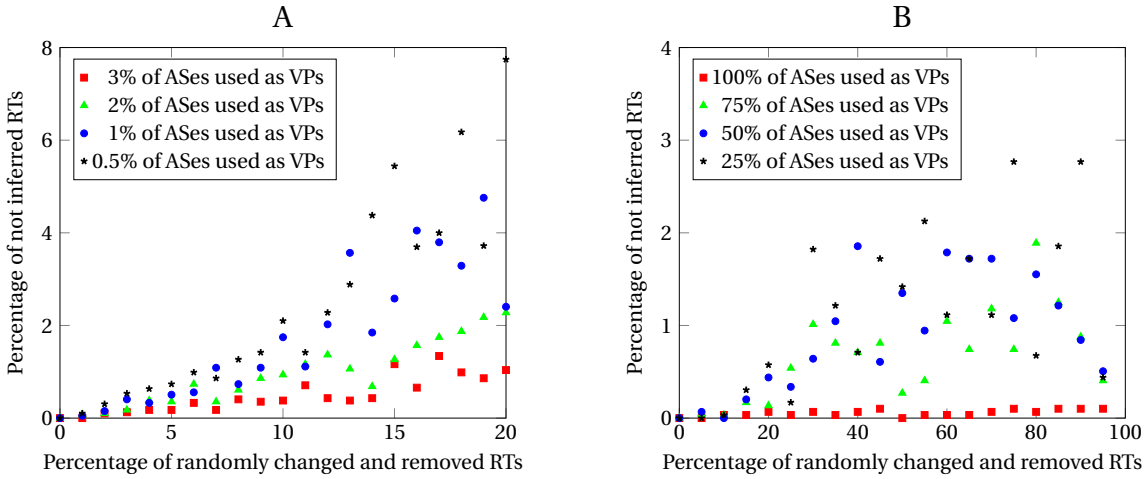


Figure 4.5: The ability of Kastelein's algorithm to infer a) changed relationship types of randomly chosen links in a mock graph and b) removed relationship types of randomly chosen links from a mock graph.
The **x-axis** shows the percentage of randomly relationship types of links first changed in the mock graph after which the same percentage of relationship types of links are randomly removed from the mock graph. This altered mock graph is used by the algorithm.
The **y-axis** shows the percentage of relationship types of links not inferred by the algorithm compared to the total links in the mock graph from which no relationship types are removed and in which no relationship types are changed.
RT = relationship type. Vantage Points (**VPs**) are ASes providing routing data.

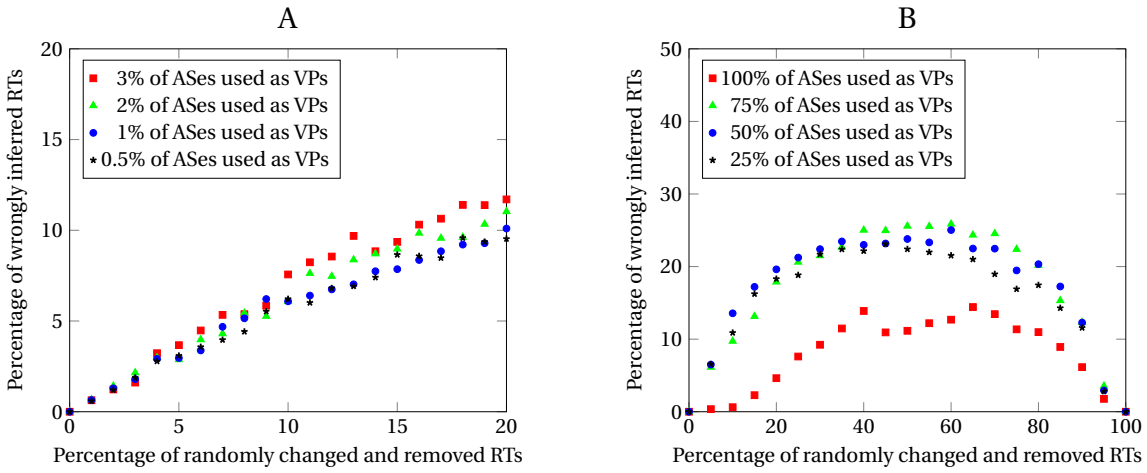


Figure 4.6: The ability of Kastelein's algorithm to infer a) changed relationship types of randomly chosen links in a mock graph and b) removed relationship types of randomly chosen links from a mock graph.
The **x-axis** shows the percentage of randomly links first changed in the mock graph after which the same percentage of relationship types of links are randomly removed from the mock graph. This altered mock graph is used by the algorithm.
The **y-axis** shows the percentage of relationship types of links wrongly inferred by the algorithm compared to the links in the mock graph from which no relationship types are removed and in which no relationship types are changed.
RT = relationship type. Vantage Points (**VPs**) are ASes providing routing data.

Analysis of case 3

Compared to **case 2** where only the relationship types were changed, **case 3** shows around 5 times more not inferred relationship types. In graph B in figure 4.5, the percentage of not inferred links stays close to 0% until 10% of the relationship types are first randomly altered and next randomly removed. This percentage drops after 90%: at this stage there are so few links left that the algorithm is performing in a similar way as in graph B in figure 4.1 where only the relationship types were removed.

When in graph A in figure 4.6 3% of the ASes are used as VPs, the algorithm produces more wrongly inferred links than when only 0.5% of the ASes are used. This may sound odd at first, however, the number of not inferred links in graph A in figure 4.5 shows that when 3% of the ASes are used as VPs the percentage of not inferred links is lower compared to when 0.5%, 1%, and 2% of the ASes are used. A lower percentage of not inferred links therefore leads to a higher percentage of wrongly inferred links. This trend continues in graph B in figure 4.6: the percentage of wrongly inferred links is higher when 75% of the ASes are used than when 25% or 25% of the ASes are used as VPs. The only exception in graph B in figure 4.6 is when all the ASes are used as VPs: there is some threshold between 75% and 100% where Kastelein's algorithm starts performing better.

Case 4: Removing randomly chosen `c2p` and `p2c` relationship types involving customer-less ASes

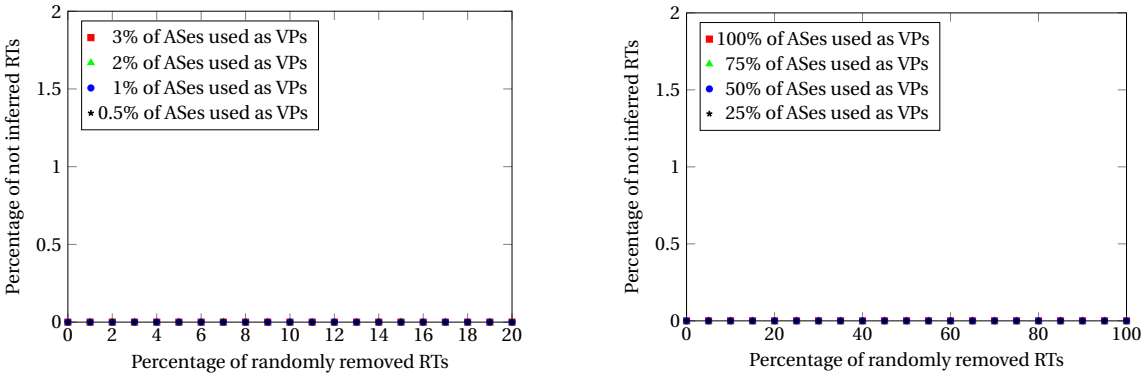


Figure 4.7: The ability of Kastelein's algorithm to infer removed `c2p` and `p2c` relationship randomly chosen types of links involving customer-less ASes from a mock graph.
The **x-axis** shows the percentage of randomly removed `c2p` and `p2c` relationship types from the mock graph used by the algorithm.
The **y-axis** shows the percentage of relationship types not inferred by the algorithm compared to the total links in the mock graph from which no relationship types are removed.
RT = relationship type. Vantage Points (**VPs**) are ASes providing routing data.

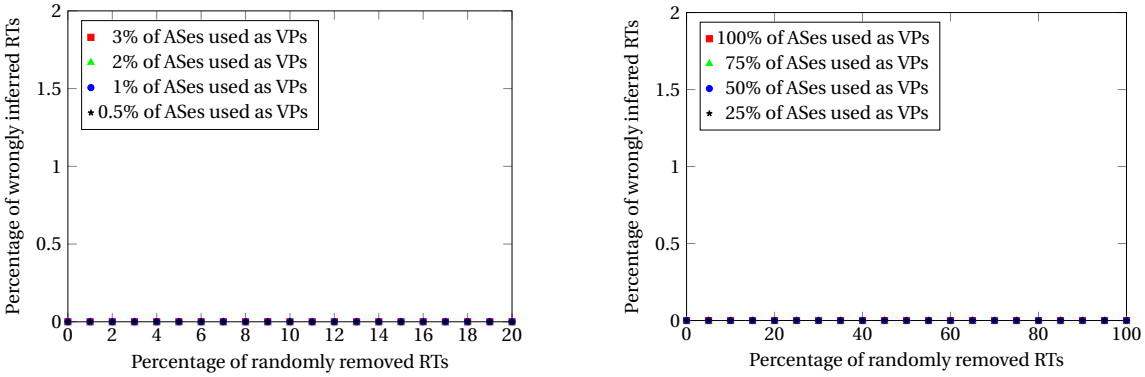


Figure 4.8: The ability of Kastelein's algorithm to infer removed `c2p` and `p2c` relationship types of randomly chosen links involving customer-less ASes from a mock graph.
The **x-axis** shows the percentage of randomly removed `c2p` and `p2c` relationship types from the mock graph used by the algorithm.
The **y-axis** shows the percentage of relationship types wrongly inferred by the algorithm compared to the links in the mock graph from which no relationship types are removed.
RT = relationship type. Vantage Points (**VPs**) are ASes providing routing data.

Analysis of case 4

Kastelein's algorithm is capable of re-inferring all removed c2p and p2c relationship types for any combination of ASes used as VPs and any percentage of relationship types removed from the mock graph. These relationship types are easily inferred because:

(a) They are seen by every AS which is used as a VP

(b) The probability that a tier-3 AS has two c2p links is $p_3^{\text{second c2p}} = 0$

(c) The probability that there is a p2p link between two tier-3 ASes $p_2^{\text{p2p}} = 0$

AS a result, all tier-3 ASes only have a single link to reach other ASes and therefore always appear at the beginning or the end of a BGP path resulting in a perfect inferring score.

Case 5: Removing relationship types of randomly chosen p2p links

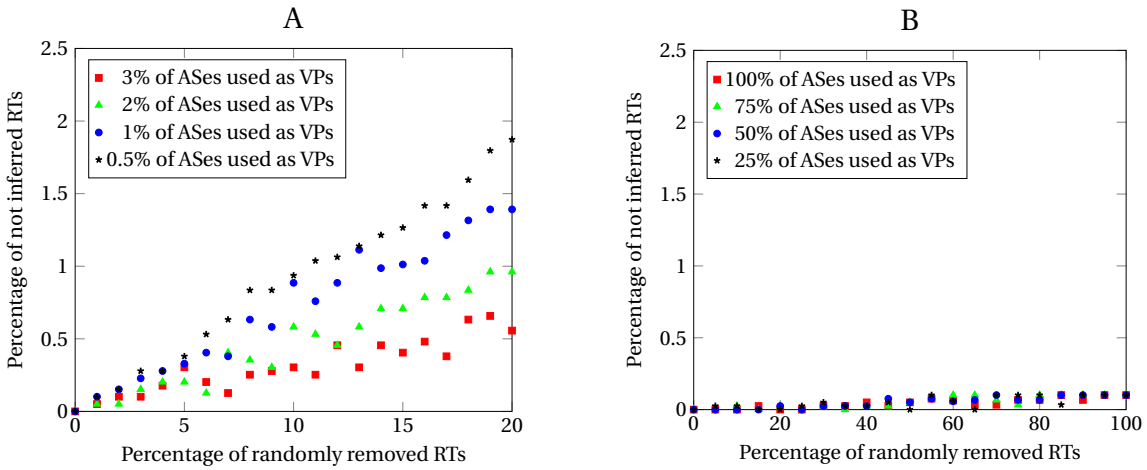


Figure 4.9: The ability of Kastelein's algorithm to infer removed randomly chosen p2p links from a graph.
The **x-axis** shows the percentage of randomly removed p2p relationship types from the mock graph used by the algorithm.
The **y-axis** shows the percentage of relationship types not inferred by the algorithm compared to the total links in the mock graph from which no relationship types are removed.
RT = relationship type. Vantage Points (**VPs**) are ASes providing routing data.

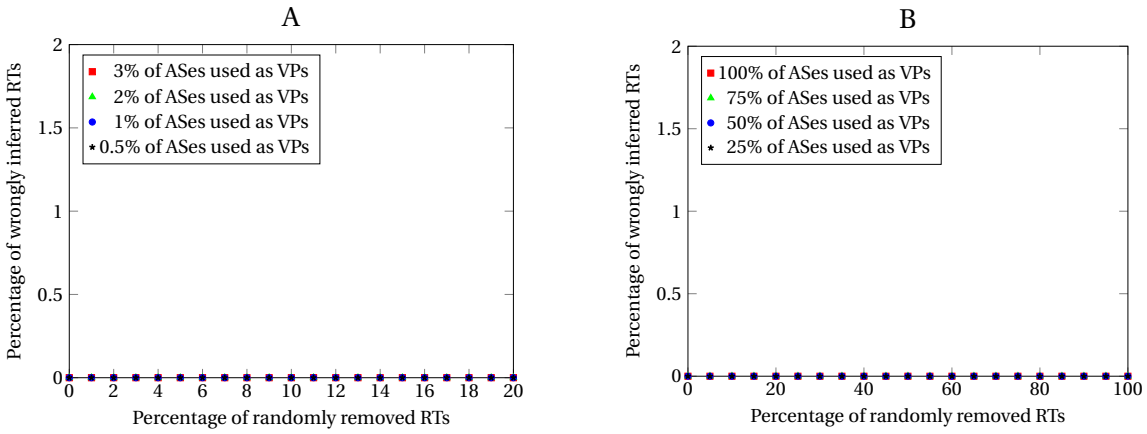


Figure 4.10: The ability of Kastelein's algorithm to infer randomly removed p2p relationship types of links from a graph.
The **x-axis** shows the percentage of randomly removed p2p relationship types from the mock graph used by the algorithm.
The **y-axis** shows the percentage of relationship types wrongly inferred by the algorithm compared to the links in the mock graph from which no relationship types are removed.
RT = relationship type. Vantage Points (**VPs**) are ASes providing routing data.

Analysis of case 5

Compared to **case 4** where only the relationship types of c2p and p2c links were removed, **case 5** sees a small percentages of not inferred links. This is caused by the fact that a p2p link in a mock graph can be interchanged with a c2p or p2c link and therefore cannot always be inferred: this includes p2p links at the beginning of a path.

The ratios presented on the y-axis in graph A and B in figure 4.9 are based on all the links in the mock graph. However, only p2p relationship types are removed in **case 5**. When only including p2p relationship types, the ratios need to be multiplied by 5 since around 20% of all the links are p2p. This means that for all percentages of ASes that are VPs Kastelein's algorithm is able to infer a higher percentage p2p relationship types than the percentage of p2p relationship types that are being removed from the mock graph.

4.2. Measuring the performance of Kastelein's algorithm

Because the entire Internet topology is not known, Kastelein's algorithm has first been validated with a mock graph that mimicked the 3-tiered structure of the Internet in section 4.1. When relationship types were changed or removed Kastelein's algorithm was capable of better restoring the mock graph than the validation method was polluting it. This showed that the algorithm can improve *fictional* AS relation data sets.

The next step is to see whether Kastelein's algorithm can also improve *real-world* data sets and outperform other *state-of-the-art* topology generating algorithms. The following AS relation data sets and topology generating algorithms will be validated and compared. To be able to compare the algorithms and data sets, the generated AS topologies by the algorithms are used for the comparison.

- **Set 1**: *AS Rank's serial-1 data set*
  AS Rank's *serial-1* data set is created using the IMC' 13 algorithm and BGP control plane data RouteViews [16]. This set is included because, currently, AS Rank has the most up-to-date AS relation data sets [7] [17].
- **Set 2**: *AS Rank's serial-2 data set*
  AS Rank's *serial-2* data set contains all the AS relations found in the *serial-1* data set the plus relationships inferred using trace routes using the Caida's Archipelago (ARK) Measurement Infrastructure. This set is included for the same reason for which **set 1** is included [7] [17] [18].
- **Algorithm 1**: *The AS relationship set generated with Gao's algorithm using BGP control plane data*
  Both Kastelein's algorithm and IMC' 13's algorithm share elements from Gao's algorithm. Kastelein's algorithm marks relationship types between ASes which cannot be inferred as p2p and the IMC' 13 algorithm makes use of the the number of unique neighbours of an AS. Moreover, the author of the Gao's algorithm also introduced the 4 relationship types s2s, p2p, c2p, and p2c.
- **Algorithm 2**: *The AS relationship set generated by IMC' 13's algorithm using BGP control plane data*
  The IMC' 13's algorithm is used to construct AS Rank's *serial-2* data set and Kastelein's algorithm uses the same method as in the IMC' 13 algorithm to remove poisoned paths and AS path prepending. Although **set 2** is partially generated using the IMC' 13 algorithm, it also makes sense to include the AS relationship set generated by IMC' 13's algorithm alone.
- **Algorithm 3**: *The AS relationship set generated with Kastelein's algorithm using the serial-2 data set, BGP control plane data, and trace routes*
  As explained in section 3.1, Kastelein's algorithm is also capable of using trace routes where Gao's algorithm and IMC'13's algorithm are limited to the use of BGP control plane data. Trace routes are generated by the Technical University Delft and by Caida's Archipelago (ARK) Measurement Infrastructure [15]. Kastelein's algorithm improves existing data sets: AS Rank's *serial-2* data set is used since it contains the highest number of links between ASes.

In an ideal world, an AS relationship data set contains **all** the links between ASes and **all** the relationship types of those links are also correctly inferred. The validation consists of comparing the **coverage ratios** that determine how many relationship types a set contains and **correctness scores** that indicate how many relationship types of a set are correctly inferred.

Coverage ratio

The most complete list of unique links between ASes contains all the unique links between ASes found in the available BGP control plane data and all the available trace routes. When an algorithm is capable of inferring all these unique links found in the complete AS relation data set, its coverage ratio is said to be 100%. Every link equally contributes to the **coverage ratio**: when there are N unique links in the complete AS relation data set and X links in the set for which the coverage ratio is calculated, the coverage ratio of this set is $\frac{X}{N} \cdot 100\%$. The *correctness* of these links is **not** captured with the coverage ratio, but with the correctness score.

Correctness score

The **correctness score** indicates how many relationship types of links are correctly inferred. A generated set where both the available BGP control plane data and trace routes do not contain any non-*valley-free* paths, receives the maximum **correctness score** of 100%.

Every link equally contributes to the **correctness score**. When a link is found in C paths but only correctly inferred according to the *valley-free* constraint in B paths, it generates a score of $\frac{B}{C}$. The overall **correctness score** is the average correctness score of the individual links.

***Valley-free* path** A BGP path that first consists of zero or more `c2p` links, then zero or one `p2p` link, followed by zero or more `p2c` links where a `s2s` can appear anywhere in the path is called *valley-free*.

- **Set 1**: AS Rank's *serial-1* data set
- **Set 2**: AS Rank's *serial-2* data set
- **Algorithm 1**: data set generated with Gao's algorithm using BGP control plane data
- **Algorithm 2**: data set generated with IMC' 13's algorithm using BGP control plane data
- **Algorithm 3**: data set generated with Kastelein's algorithm using AS Rank's *serial-2*, BGP control plane data, and trace routes.
  Wrongly inferred types **are not** removed.
- **Algorithm 4**: data set generated with Kastelein's algorithm using AS Rank's *serial-2*, BGP control plane data, and trace routes.
  Wrongly inferred types **are** removed.

	Coverage ratio	**Correctness score**
Set 1	60.00%	97.57%
Set 2	79.23%	97.48%
Alg 1	34.47%	99.89%
Alg 2	62.42%	69.03%
Alg 3	**86.56%**	99.53%
Alg 4	84.46%	**100.00%**

Table 4.1: Coverage ratio and correctness score comparison of the data sets of AS Rank and data sets generated by Goa's algorithm, IMC'13's algorithm and Kastelein's algorithm.

While AS Rank's *serial-1* **set 1** contains less ASes, its correctness score is slightly higher than the AS Rank's *serial-2* **set 2** containing more ASes: the fact that the *serial-2* set also includes trace routes is clearly visible with a coverage ratio that is almost 20% higher than its *serial-1* counterpart. Both AS Rank sets have an expected high correctness score: around 3% of the link types are not inferred.

While 62.42% of the links are found in BGP control plane data, Gao's algorithm presented in **algorithm 1** is only capable of inferring 34.47% of the total links. However, the relationship types that Gao's algorithm does infer have the second highest correctness score. **Algorithm 2** containing the AS relationship set generated with the IMC' 13 algorithm has a much higher coverage ratio and the lowest correctness score. The low correctness score is caused by stage 14 of the algorithm that infers all the remaining links as `p2p`: **X%** of the total links inferred by the algorithm are inferred in that stage. Other stages that should infer `c2p` and `p2c` links infer too few links. This low `c2p` and `p2c` infer rate is probably caused by an implementation or interpretation error: the paper describing the algorithm did not include any pseudocode and the authors of the IMC' 13 algorithm could have given a more detailed description how the algorithm needs to be implemented.

A correctness score of **100.00%** for **algorithm 4** is the maximum score an AS relations data set can have. Although this might sound to good to be true, when Kastelein's algorithm repeats stage 3, stage 4, stage 5, and stage 6 to generate an AS relation data set with more and more relationship types that do not contain non-valley-free paths, all the relationship types of links that violate this constraint are removed. As a result, the correctness score will always be 100. However, the correctness score itself is a good method to measure the quality of an AS relation data set: **set 1**, **set 2**, and **algorithm 1** have similar high correctness scores. The motivation behinds Gao's AS classification using the types `c2p`, `p2c`, `p2p`, and `s2s` rests on the constraint that BGP routes between ASes are *valley-free*. It would therefore make sense to only use the *valley-free* constraint to determine the quality of these sets.

Finally, **algorithm 4**, where the wrongly inferred links are not removed, has a higher coverage ratio and correctness score than **set 1**, **set 2**, and **algorithm 2**. Only **algorithm 1** generated using Goa's algorithm has a higher correctness score. However, this set only has a coverage ratio of 34.47% When using the *valley-free* constraint as a metric, the set generated by Kastelein's algorithm outperforms all other sets included in the measurements.

4.3. Measuring the scalability of the BGP simulator

The scalability of the BGP simulator is measured to see how it will perform for different use cases. For example, the BGP simulator can be used to show how traffic flows to reach a certain prefix or to analyse large BGP-related incidents involving many ASes and prefixes. As explained in section 3.3, the BGP simulator can simulate BGP traffic using a network of ASes, BGP attributes and prefix announcements. Therefore, three methods that measure the simulator's scalability are presented below that involve different combinations of numbers of ASes and prefix announcements. Each method involves simulations done by the BGP simulator using mock graphs of different sizes and various amounts of added *fictional* prefix announcements. All three methods will create a scatter plot where the **y-axis** shows the computation time of the BGP simulator. The data shown on the **x-axis** is presented below. Besides measuring the computation time, the memory usage is also captured and compared with the memory usage of Trap's simulator.

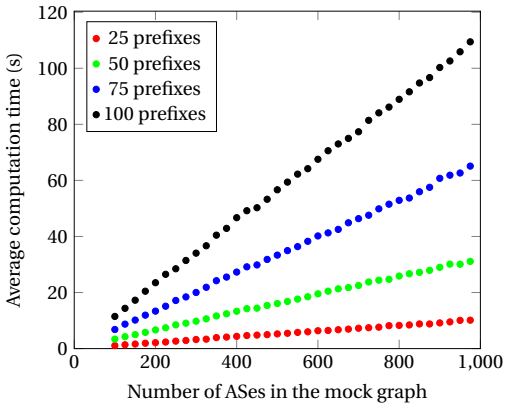
- **Method 1**: *Varying the number of ASes in the mock graph while a fixed number of ASes is announcing one prefix*
  This method represents the use case where the BGP simulator is used to show how traffic flows to reach certain prefixes. This results in the computation time of the simulator being measured for an increasing mock graph size, starting where either 25, 50, 75 or 100 ASes announce a singe prefix to determine with which rate the computation time increases when more and more ASes are added. The placing of the prefixes in the mock graph is randomised and the announced prefixes do not overlap to allow for a fair comparison between methods and the number of prefix announcements.
- **Method 2**: *Varying the number of ASes that announce one prefix with a mock graph containing a fixed number of ASes*
  This method does not have a direct use case. However, knowing how the BGP simulator's computation time scales when only the amount of prefix announcements is changing provides insight on how the BGP simulator works internally. The computation time of the simulator is measured for an increasing number of ASes that announce a unique and non-overlapping prefix where the mock graph consists of either 250, 500, 750 or 1000 ASes. The average value is taken where the placing of the prefixes in the mock graph is randomised and the announced prefixes do not overlap to allow for a fair comparison between methods and the number of prefix announcements.
- **Method 3**: *Varying the number of ASes that all announce one prefix*
  This method represents the use case where the entire AS Internet topology is simulated: each AS is also announcing a prefix. The computation time of the BGP simulator is measured for a mock graph with an increasing number of ASes which each announce a unique and non-overlapping prefix. The announced prefixes do not overlap to allow for a fair comparison between methods and the number of prefix announcements

In order to spot trends, the computation time of each pair of number of ASes and number of announcing prefixes is measured 10 times and the average value is calculated. The parameters for the mock graph generator, described in section 3.2 on page 30, also play an important part. A mock graph is used because existing AS relationship data sets are difficult to scale down given the alternating number of ASes involved in each measurement.

To mimic the 3-tiered structure of the Internet with in total 5000 ASes, the same number of tier-1 ASes S_1, multiplier per tier S_x, and number of tiers N will be used as explained in section 4.1: $\boxed{S_1 = 2}$, $\boxed{S_x = 49}$, and $\boxed{N = 3}$. This 5000 limit is a consequence of hardware limitations: when too many prefix announcements and ASes are added, the amount of memory needed will grow too large. To compare measurements involving different sized mock graphs, starting with tier-1, each tier is filled before the next tier is filled. For example, when a measurement involves 100 ASes, the first 2 will be tier-1 ASes and the other 98 ASes will be in the tier-2 as a result of the S_1 and the S_x parameters.

To ensure that all the ASes in mock graph can reach each other, the probability that two tier-1 ASes share a p2p link is $p_1^{\text{p2p}} = 1$. With an average of 7.15 p2p links between 25 large tier-2 ASes [19], the probability that two tier-1 ASes share a p2p link is $p_2^{\text{p2p}} = \frac{7.15}{25 \cdot 2} = 0.143$. To mimic that tier-3 ASes are not interconnected with each other using a p2p link, the probability that two tier-3 ASes share a p2p link is $p_3^{\text{p2p}} = 0$. To enable that measurements involving different mock graphs share a similar topology, the probability that a tier-2 AS has two c2p links is $p_2^{\text{second c2p}} = 1$ and the probability that a tier-3 AS has two c2p links is $p_3^{\text{second c2p}} = 0$.

Method 1: *Varying the number of ASes in the mock graph while a fixed number of ASes is announcing one prefix*

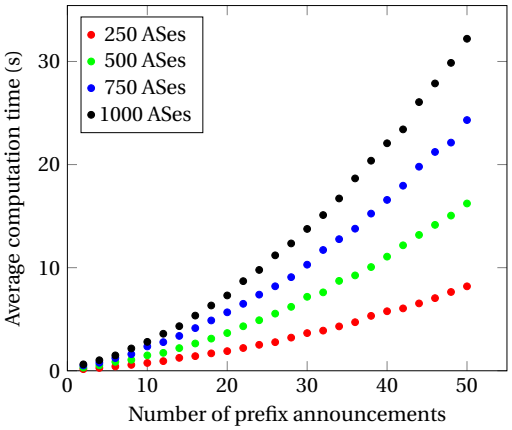


The computation time is approximately equal to $t(s) = a \cdot \#\text{ASes}$ where a is **0.010**, **0.031**, **0.066**, or **0.11** for **25**, **50**, **75** and **100** ASes that announce a prefix respectively. In other words, adding twice as many ASes that do not announce a prefix to the BGP simulator results in a computation time twice as high. Doubling the number of ASes that announce a prefix does not result in a slope twice as steep: $\frac{0.031}{0.010} \approx 3.10$, $\frac{0.066}{0.031} \approx 2.13$, $\frac{0.11}{0.066} \approx 1.67$. The adding of additional non-announcing ASes will result in more BGP traffic in the AS graph: BGP-learned paths cannot contain any loops and are only passed through when they improve the current routing status.

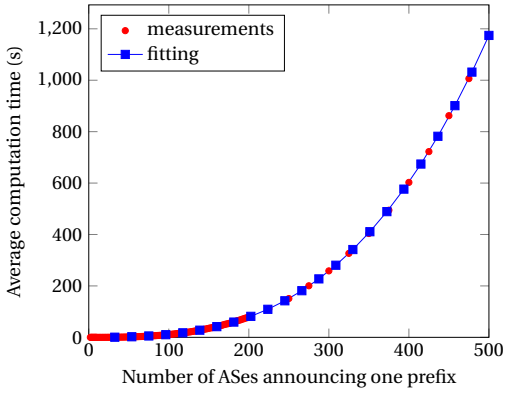
The placement and number of links an AS has with other ASes will effect the computation time differently: the higher the degree and the closer to the core of the AS graph the AS is, the higher the additional computation time will be.

Method 2: *Varying the number of ASes that announce one prefix with a mock graph containing a fixed number of ASes*

There is a non-linear relation between the computation time and number of ASes announcing a prefix for a mock graph where the number of ASes not announcing a prefix is fixed. This non-linear behaviour is a consequence of the need to compare all prefixes in an AS in the routing table (*RT*) stage. The *RT* stage of the simulator contains information about which route an AS takes to reach a certain prefix. When a *more-specific*, in other words, a prefix that fits in another prefix, is found, the route associated with this *more-specific* is preferred over the route belonging to the larger prefix.



Method 3: *Varying the number of ASes that all announce one prefix*



Using polynomial regression, the computation time is approximately $\boxed{t(s) = 9.35 \cdot 10^{-6} \cdot x^3 - 4.03 \cdot 10^{-5} \cdot x^2 + 0.0318 \cdot x^1 - 0.782}$. Under the assumption that the BGP simulator has enough memory, the computation time of the BGP simulator with 10k, 25k and 70k ASes will take around **100 days**, **5 years**, and **13 years** respectively. These high computation times, mainly caused by the prefix comparisons in the the *RT* stage, render the BGP simulator unusable in situations that require an up-to-date set of used BGP routes. For example, when a prefix is hijacked, an up-to-date set of used BGP routes is desired immediately possible to notify other ASes or ISPs.

Performance comparison with Trap's simulator

As discussed in section 2.7, Trap's simulator used 32.5 Gb of memory when announcing 9 prefixes with an AS graph consisting of 5056 ASes in real-time. Therefore, to be able to compare the BGP simulator presented in this thesis with Trap's simulator, the memory usage and computation time of the BGP simulator was measured with a mock graph consisting of 5056 ASes of which 5, 10, 15, or 20 ASes also announce a prefix.

Number of prefixes	5	10	15	20
Memory usage	≈ 225Mb	≈ 329Mb	≈ 411Mb	≈ 469Mb
Computation time	≈ 11s	≈ 16s	≈ 24s	≈ 35s

Table 4.2: Memory usage and computation time of the BGP simulator for a mock graph with 5056 ASes where 5, 10, 15, or 20 randomly chosen ASes announce a prefix.

The BGP simulator is not designed to run in real-time. However, it does manage to simulate 9 prefixes in less than 16 seconds. This should be sufficient for most use cases. Compared to Trap's simulator the BGP simulator is using around 10 times less memory.

4.4. Validating the BGP simulator

Because the BGP simulator only takes the LOCAL_PREFERENCE, AS_PATH, and prefix length BGP attributes into account, simulation results will never fully match the currently used BGP routes between ASes. And although the simulator is capable of simulating BGP traffic with directly added routes with AS path prepending, it cannot predict whether ASes will apply AS path prepending. Moreover, the AS Internet topology is not completely known, and with an incomplete and partially wrong AS topology, simulation results will differ from the real-world BGP routes.

In order to demonstrate that the BGP simulator - with only the basic attributes - is working correctly, section 3.3 already provided several examples which included hijacked and non-hijacked prefixes and showed the simulator's ability to correctly use the LOCAL_PREFERENCE, AS_PATH, and prefix length attributes. However, these examples all used AS graphs with only a few ASes. Therefore, the BGP simulator will be further validated using the larger improved AS relations data set generated by Kastelein's algorithm. This algorithm uses AS Rank's *serial-2* data set as a base in combination with a) available BGP control plane data and b) trace routes to correctly infer wrongly inferred and not inferred links in the *serial-2* data set.

The validation consists of the computation of the **path match percentage** by matching BGP control plane data routes with BGP simulator routes: the BGP simulator should simulate the same routes as the BGP routes found in the BGP control plane data. BGP control plane data contains BGP routes received by ASes and captured by route collectors.

Path match percentage

The **path match percentage** is a metric that indicates how similar a BGP control plane data route is to the *best matching* BGP simulator route. Its computation requires the following steps:

- **Step 1**: *Selecting random prefixes in the BGP control plane data*
  With the way in which the BGP simulator is currently implemented, it is not possible to simulate all the prefixes.
- **Step 2**: *Finding the corresponding ASes that announce these prefixes in the BGP control plane data*
  The ASes that are announcing the prefixes, also called source ASes, selected in **step 1** are needed to simulate BGP traffic.
- **Step 3**: *Finding the associated BGP routes of the selected random prefixes in the BGP control plane data*
- **Step 4**: *Loading the improved AS relations data set into the BGP simulator*
- **Step 5**: *Adding the random prefixes selected with the associated source ASes to the BGP simulator and simulating BGP routes*
- **Step 6**: *Grouping BGP control plane data routes per announcing prefix and receiving AS*
  Only BGP routes that include the same prefix, source AS and destination AS are compared with each other. The destination AS is the last AS in the path of the BGP route that **could** use this BGP route to reach the prefix announced by the source AS.
- **Step 7**: *Grouping BGP simulator routes per announcing prefix and receiving AS*
  The same reasoning holds as given in **step 6**.
- **Step 8**: *Matching BGP routes from **step 7** with BGP routes from **step 6** that are in the same group*
  At this point, every route is assigned to a pair containing the announcing prefix and destination AS of that BGP route. For each pair, all the BGP control plane data routes are matched with the *best matching* BGP simulator route from the same pair: the BGP simulator route that shares the most links with the BGP control plane data route is chosen. The **path match percentage** is equal to the number of links that are present in the BGP control plane data route and are **not** present in the BGP simulator route divided by the total number of links in the BGP control plane data route.

When a BGP control plane data route and a BGP simulator route are identical, the BGP control plane data route's **path match percentage** is said to be 100%. Given the BGP control plane data route $[1,2,3,4]$ and the BGP simulator route $[1,2,8,4]$, the **path match percentage** is 33.3% since only the link $[1,2]$, and not the links $[2,3]$ and $[3,4]$, is present in the BGP control plane data route and in the BGP simulator route.

For either 1000 or 4000 random prefixes found in the BGP control plane data, BGP simulator routes are compared with the BGP routes found in the BGP control plane data. By computing and grouping **path match percentages**. 1000 or 4000 random prefixes are included to detect trends.

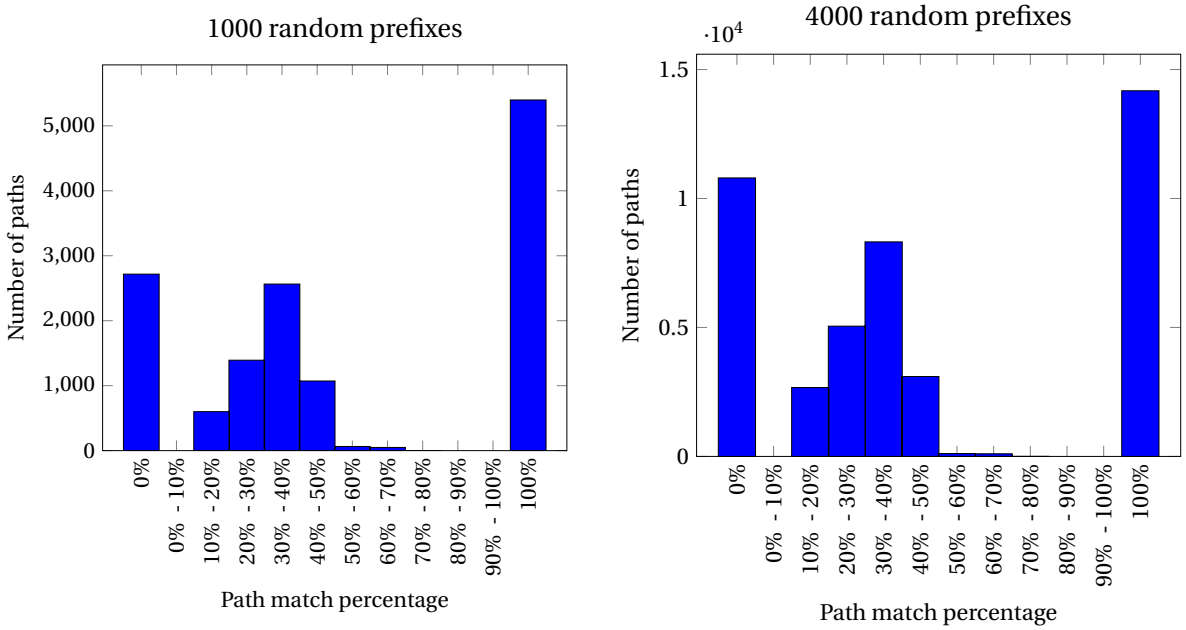


Figure 4.11: Both graphs contain the **path match percentage** for BGP control plane data routes from either 1000 or 4000 randomly selected prefixes. The **x-axis** groups and shows the BGP control plane data routes in slices of 10% and filters out the BGP control plane data routes with a **path match percentage** of either 0% or 100%. The **y-axis** displays the number of BGP control plane data routes.

Table 4.3 shows that $\frac{5398}{13858} \approx 38\%$ and $\frac{14176}{44322} \approx 32\%$ of the BGP control plane data paths have a direct match with a BGP simulator path for 1000 and 4000 prefixes respectively. It should be noted these BGP control plane data paths with a 100% **path match percentage** are not the only BGP paths simulated by the BGP simulator: the BGP simulator has an average of 2.72 paths for every bundle of BGP routes with the same prefix, source AS and destination AS. Since only the basic BGP attributes are known, the BGP simulator cannot decide which BGP route to use, therefore keeping all the possible BGP routes.

	0%	0% - 10%	10% - 20%	20% - 30%	30% - 40%	40% - 50%	50% - 60%	60% - 70%	70% - 80%	80% - 90%	90% - 100%	100%	all
1000 prefixes	**2717**	0	600	1392	2564	1072	64	49	2	0	0	**5398**	**13858**
4000 prefixes	**10797**	0	2672	5050	8317	3097	111	102	1	0	0	**14176**	**44322**

Table 4.3: The **path match percentage** for BGP control plane data routes from either 1000 or 4000 randomly selected prefixes. The number of BGP control plane data routes are grouped in slices of 10% and BGP control plane data routes with a **path match percentage** of either 0% or 100% are shown.

Because a BGP control plane data route either has a perfect match with a BGP simulator route or has at least two links in its path that are different to the BGP simulator path, the 60% to, but not including, 100% **path match percentage** range contains virtually no paths. With an average path length of 4.68, the expected BGP control plane data routes with two non-matching links start at a **path match percentage** of $1 - \frac{2}{4.68} \approx 57\%$.

$\frac{2717}{13858} \approx 20\%$ and $\frac{10797}{44322} \approx 24\%$ of the BGP control plane data paths do not have a single link that is also present in the BGP simulator routes for either 1000 or 4000 prefixes. These 24% and 23%, and the other BGP control plane data routes in figure 4.4 with a low **path match percentage** again show that the BGP simulator only simulates four BGP attributes and lacks information regarding LOCAL_PREFERENCE and AS path prepending. Both graphs in figure 4.4 show similar trends:

- 38% and 32% of the BGP control plane data routes have a **path match percentage** of 100% for 1000 and 4000 random prefixes respectively
- 20% and 24% of the BGP control plane data routes have a **path match percentage** of 0% for 1000 and 4000 random prefixes respectively
- The distribution of the BGP control plane data routes with a **path match percentage** between 20% and 70% is similar for both 1000 and 4000 random prefixes.

1000 prefixes

	Match	**Mismatch**
Start	9295 (≈ 67.1%)	4563 (≈ 32.9%)
Middle	3168 (≈ 19.8%)	12872 (≈ 80.2%)
End	7379 (≈ 54.7%)	6114 (≈ 45.3%)

4000 prefixes

	Match	**Mismatch**
Start	25660 (≈ 57.9%)	18663 (≈ 42.1%)
Middle	8174 (≈ 14.3%)	49092 (≈ 85.7%)
End	22528 (≈ 52.2%)	20669 (≈ 47.8%)

Table 4.4: Number of matched and mismatched links in the BGP control plane data routes for 1000 and 4000 randomly selected prefixes. The links in the routes are grouped. All the links that are found at the beginning of a path are placed in the group **start**. All the links that are found at the end of a path are placed in the group **end**. All the remaining links are added to the group **middle**.

The distribution of BGP control plane data paths with a **path match percentage** between 20% and 70% is caused by the limited number of possible **path match percentages**. Table 4.5 shows that the path lengths of 2, 3, and 4 are the most common. The possible associated percentages for these paths are:

- **BGP paths with a path length of 2**: *0%, 50%, and 100%*
  A single wrong AS in a path results in two wrong links. As a result, the **path match percentage** of 50% can never occur in a BGP path with length 2.
- **BGP paths with a path length of 3**: *0%, 33%, 66%, and 100%*
  The **path match percentage** of 66% in a BGP path with length 3 can never occur since a single wrong AS in a path results in two wrong links.
- **BGP paths with a path length of 4**: *0%, 25%, 50%, 75%, and 100%*
  Using the same reasoning as in BGP paths with a path length of 2 and 3, the **path match percentage** of 75% in a BGP path with length 3 can never occur.

This distribution and the fact that a single wrong AS results in two wrong links means that BGP paths with a path length of 2, 3, and 4 can only have **path match percentages** of 0%, 25%, 33%, 50%, and 100%. The bars presented in figure 4.4 show **path match percentages** of BGP paths in slices of 10%. The **path match percentages** of 25%, 33%, and 50% end up in the groups 20%-30%, 30%-40%, and 40%-50%.

Table 4.4 shows that approximately 67.1% and 57.9% of the first links found in the collector paths are matched with with first link found in the best matching BGP simulator route for 1000 and 4000 involved prefixes respectively. For the last links in the collector paths approximately 54.7% and 52.2% are present in the best matching BGP simulator route. Links found in the middle of the BGP control plane data paths are mismatched approximately 65.70% and 69.98% of the time for 1000 and 4000 involved prefixes respectively. These percentages are twice as high compared to the mismatch percentages of the **start** and **end** groups.

Path length	1	2	3	4	5	6	7	8	**All**
Match	≈ 83.6%	≈ 85.1%	≈ 50.9%	≈ 27.2%	≈ 25.5%	≈ 17.7%	≈ 16.5%	≈ 15.6%	≈ 45.7%
Mismatch	≈ 16.4%	≈ 14.9%	≈ 49.1%	≈ 72.8%	≈ 74.5%	≈ 82.3%	≈ 83.5%	≈ 84.4%	≈ 54.3%
#links	365	7430	16083	11636	6055	1566	224	32	43391
#paths	365	3715	5361	2909	1211	261	32	4	13858

Table 4.5: Percentage of matched and mismatched links in BGP control plane data routes for 1000 randomly selected prefixes grouped per path length. For example, approximately 83.6% of the links found in the routes with path length of 1 can also be matched with the links found in the best matching BGP simulator routes with the same path length of 1.

A single mismatched AS in a path results in two mismatched links. As a result, all the links not matched in the **start** and **end** groups can also be present in the **middle** group. See table 4.5, $\frac{365+7430}{43391} \approx 18\%$ of the BGP control plane data paths have a path length larger than 3 when 4000 prefixes are involved. When neglecting these 18% of paths, at least 4912 and 17066 links or at most 10033 and 33766 appear in a) the **start** and **middle** or b) the **end** and **middle** group for 1000 and 4000 involved prefixes respectively.

The higher the path length, the lower the percentage of matched links: every extra link that a BGP announcement has to traverse introduces more uncertainties. Even with the improved AS relationship data set generated with Kastelein's algorithm only 50.9% of the links for the routes with a path length of 3 can be matched. This sudden drop compared to the 85.1% of routes with a path length of 2 can be caused by only a few different LOCAL_PREFERENCE values, resulting in totally different preferred paths. However, it should be noted that a more detailed analysis is needed to verify this claim.

Path length	1st link	2nd link	3rd link	4th link	5th link	6th link	7th link	8th link
1	≈ 83.6%	-	-	-	-	-	-	-
2	≈ 88.7%	≈ 81.5%	-	-	-	-	-	-
3	≈ 60.7%	≈ 37.5%	≈ 54.6%	-	-	-	-	-
4	≈ 54.1%	≈ 14.5%	≈ 5.5%	≈ 34.6%	-	-	-	-
5	≈ 58.6%	≈ 26.9%	≈ 7.0%	≈ 4.1%	≈ 30.6%	-	-	-
6	≈ 51.0%	≈ 26.1%	≈ 10.0%	≈ 1.5%	≈ 0.8%	≈ 16.9%	-	-
7	≈ 62.5%	≈ 21.9%	≈ 12.5%	0%	0%	≈ 3.1%	≈ 15.6%	-
8	≈ 50.0%	≈ 25.0%	0%	0%	0%	0%	≈ 25.0%	≈ 25.0%

Table 4.6: Percentage of matched BGP control plane data routes with the best matched BGP simulator routes grouped per path length and position of the link in the route for 1000 randomly selected prefixes.

The percentage of matched links for all the routes with a path length of 1 should be 100% since all BGP control plane data routes and BGP simulator routes that are compared have the same source and destination AS pair, therefore always resulting in a match. However, because the AS relationship data set generated with Kastelein's algorithm does not contain all the links also found in the BGP control plane data routes, BGP traffic is not always forwarded to all the ASes.

The first column in table 4.6 shows that for all routes it holds that the first link has the highest match percentage compared to links appearing further in the route. The last links appearing in the routes generally hold the second highest match percentage. For both sets of links it holds that at most a single wrong AS in the path can result in a mismatch. This is not the case for links appearing in the middle of paths where two mismatched ASes can cause the link to be mismatched. Consider the path $[AS1, AS2, AS3, AS4]$, the link between AS1 and AS2 can only be marked wrong when AS2 is not found in the best matching BGP simulator route since AS1 is used to group BGP control plane data routes and BGP simulator routes. The link between AS2 and AS3 can be a mismatch when AS2 and/or AS3 do not appear in the BGP simulator route.

A clear pattern is showing: the further the links appear in a path, the lower the match percentage and the longer the path, the lower the percentage. See table 4.5, compared to the total number of routes there are only $224 + 32 = 256$ routes with a path length of at least 7. These routes are furthermore very poorly matched: the BGP simulator prefers routes with smaller path lengths.

The general conclusion that can be drawn is that there the BGP simulator incorporates too few BGP attributes and, because of the lack of insight on how ASes operate, critical information such as the LOCAL_PREFERENCE values are missing. This results in a BGP simulator that cannot accurately simulate BGP traffic.

5

Conclusion & Future Work

The first chapter of this thesis stated two main goals: to design a topology generating algorithm that outperforms the current *state-of-the-art* topology generating algorithms and AS relation data sets while only using the *valley-free* constraint, and to design a BGP simulator that scales properly when large numbers of ASes and prefix announcements are included.

Although BGP is crucial in enabling world wide connectivity, at the same time it allows for rerouting of data, sending of spam, and application of Internet censorship with detrimental effects [4] [5]: as seen in the introductory chapter, this misuse of BGP has severe consequences such as disconnecting entire countries from the Internet and theft of confidential networking traffic. In order to better spot these anomalies an improved AS relationship data set is needed in combination with a BGP simulator that can simulate large BGP-related events and anomalies.

5.1. Kastelein's algorithm

Both Gao's algorithm, described in section 2.5.1, and IMC' 13's algorithm, described in section 2.5.2, are not difficult to implement given their clearly described stages. Both algorithms are limited in their use since they use BGP control plane data only. Adding trace routes - besides the BGP routes learned via ASes providing their routes to route collectors - will result in relationship types of links being wrongly inferred.

The algorithms proposed by Gao *et al.* and Luckie *et al.* both assume that the customer cone of a customer is smaller than the customer cone of the higher tiered AS. The customer cone of an AS is the set of IPv4 and IPv6 addresses that are either owned by the AS or can be reached by visiting its customers, and also include the customers of the customers. Furthermore, Luckie *et al.* also assume that ASes that do not appear in the middle of any path are very unlikely to be interconnected with another AS via a `p2p` link. This claim only holds when these so called stub ASes are actually stub. But, since the coverage of the route collectors is limited, this cannot be said with certainty. Stub ASes are ASes that do not appear in the middle of any BGP path.

Gao's algorithm uses the parameters R and L in stage 4 and stage 6. The parameter L in combination with the number of ASes using an AS to reach parts of the Internet is used to infer `s2s`, `c2p`, and `p2c` relationship types. The parameter R is used as a threshold to determine when two ASes are interconnected via a `p2p` link. Different parameter values will lead to a different set of AS relations. This means that only with adequate tweaking Gao's algorithm could be capable of correctly generating the AS topology of the Internet.

The algorithm presented by Luckie *et al.* is described in 14 stages and infers `c2p`, `p2c`, and `p2p` relationship types. Other than Gao's algorithm, the IMC' 13 algorithm first filters out poisoned paths. Poisoned paths are paths where the same AS is listed twice after AS path prepending has been filtered out.

Both algorithms have both strong and weak points. In order to not re-invent the wheel, Kastelein's algorithm takes advantage of these strong points such as removing AS path prepending and removing poisoned paths, and the fact that AS Rank's *serial-1* and *serial-2* AS relationship data sets already exist. Kastelein's algorithm does not replace the current *state-of-the-art* topology. Instead it uses this topology to generate a more complete and more correct AS relationship data sets.
Kastelein's algorithm excels in its simplicity: instead of using many assumptions, it only uses the fact that nearly all BGP routes on the Internet follow the *valley-free* constraint.

Research Question 1: Can the currently known AS Internet topology be improved with a topology generating algorithm that uses data from route collectors, AS Rank's *serial-2* data set, and trace route data in such a way that more relationship types are inferred and the percentage of correctly inferred relationship types is higher using only the condition that BGP paths need to be *valley-free*?

Kastelein's algorithm, described in section 3.1 on page 23, was validated a) using a mock graph that shared the structure of the Internet where relationship types were changed and/or removed and b) by comparing the output of the algorithm with the AS relation sets created with Gao's algorithm, IMC' 13's algorithm, as well as AS Rank's *serial-1* data set and *serial-2* data set.

The validation method involving a mock graph in figures 4.1 to 4.10 on page 40 to page 44 show that Kastelein's algorithm is capable of correctly inferring more relationships types than the number of relationships types that are removed from or changed in the mock graph. However, as seen in figure 4.10 on page 40, when 0.5% of the ASes provide routing data in the mock graph, and relationship types are changed in the mock graph as well as removed from the mock graph, Kastelein's algorithm is inferring around 50% wrong relationship types less than relationship types changed in and removed from the mock graph. This ratio needs to be decreased in further research.

Changing and removing relationship types with a low percentage of ASes provide routing data in the mock graph resembles the *real-word* where the current *state-of-the-art* AS relation data sets both contain wrongly inferred links and missing links. The fact that Kastelein's algorithm is capable of better restoring the mock graph than the validation method is polluting it, shows that the algorithm can be used to effectively improve *real-world* AS relation data sets.

As seen in table 4.1 on page 46, Kastelein's algorithm outperforms every other algorithm and its generated relationship types have both a higher coverage ratio and correctness score compared to existing AS relation data sets. Even AS Rank's *serial-2* data set that is created with the use of ARK trace routes and the IMC' 13's algorithm has a lower coverage ratio of 79.22% compared to the 86.56% of Kastelein's algorithm coverage ratio and correctness score of 97.48 compared to Kastelein's 99.53. The AS relationship topology generated by IMC' 13's algorithm has a very low correctness score of 69.03. The low `c2p` and `p2c` infer rate in stages 4, 8, 9, 11 and 13 resulted in all the links being inferred as `p2p` in stage 14. This is probably caused by an implementation or interpretation error: the paper describing the algorithm did not include any pseudocode and the authors of the IMC' 13 algorithm could have given a more detailed description on the workings of the algorithm.

Although the coverage ratio and correctness score of Kastelein's algorithm are high, this does not automatically make the algorithm the best performing one: more validation methods than only using the *valley-free* constraint exist and the fact that most of the BGP paths indicate the inferred links are *valley-free* does not necessarily means that the links themselves are correctly inferred. As a result, Kastelein's algorithm needs to be further validated using other BGP-related sources such as WHOIS data.

5.2. BGP simulator

Trap's BGP simulator used Mininet and Quagga to simulate 5624 ASes and 9 prefix announcements in real-time with. The simulator was distributed over multiple servers and consumed 32.5 GB of memory. However, Trap used a topology generated with 2018 routing data instead of a topology generated with 2008 routing data. Furthermore, many AS configurations were not known. As a result, analysis concerning the availability of YouTube.com during the incident differed from the analysis carried out by Dyn [6]. Trap states that when all AS configurations were known and an up-to-date AS relationship data set was used the simulator would have produced similar results. A new BGP simulator was designed that is:

(a) easy to set-up

(b) light-weight and scalable

(c) properly validated with BGP control plane data

Research Question 2: Can BGP traffic be simulated in such a way that the simulated BGP routes match actual used BGP routes in an efficient way when a large number of ASes and IP ranges are involved?

The validation process of the BGP simulator, described in sections 4.3 and 4.4, consisted of two parts: the scalability was determined by making use of a mock graph after which simulator-generated BGP routes were compared with BGP routes from route collectors. The computation time of the BGP simulator increases linearly as more ASes that do not announce a prefix are added. However, when more prefix announcements are added, the computation time no longer increases in a linear fashion. Assuming that the BGP simulator has sufficient memory, the computation time of the BGP simulator with 10k, 25k and 70k ASes that all announce a prefix will take 100 days, 5 years, and 11 years respectively. These high running times are mainly caused by the prefix comparisons in the the routing table stage. In order to make the BGP simulator usable in situations that require an up-to-date set of used BGP routes, the computation time needs to be reduced.

For the same configuration as the one used for Trap's simulator, the BGP simulator is using 10 times less memory and only needs around 16 seconds to complete while Trap's simulator is capable of simulating these 9 prefixes and 5056 ASes in *real-time*. The computation time of the BGP simulator should be short enough for most use cases.

Besides having a computation time that is short enough to be usable, the BGP simulator output needs to match *real-world* BGP routes. When 1000 and 4000 random prefixes were used, *real-world* BGP routes directly matched 56% and 57% of the BGP simulator generated routes respectively. It should be noted that for every BGP control plane data route an average of 2.72 BGP simulator routes exist. This renders the simulator far from perfect. This imperfection is caused by the fact that the BGP simulator can only simulate the BGP attributes LOCAL_PREFERENCE and AS_PATH and supports, but does not predict, AS path prepending.

The AS relationship data set generated with Kastelein's algorithm that was used for the validation did not contain all the links found in the BGP control plane data. This resulted in lower percentages of matched links than expected. The further the link appears in a path or the longer the path, the lower the match percentages of the involved links are. The BGP simulator was not capable of simulating the same routes with path lengths of at least 6. This is caused by the simulator's preference of using routes with smaller path lengths.

The BGP simulator needs to be improved further before it can be used to accurately simulate BGP-related incidents. All the BGP attributes need to be included and more information regarding ASes such as their LOCAL_PREFERENCE values need to be added.

Bibliography

[1] Russian-controlled telecom hijacks financial services' Internet traffic. https://arstechnica.com/information-technology/2017/04/russian-controlled-telecom-hijacks-financial-services-internet-traffic/, . Accessed: 2018-16-08.

[2] "Suspicious" event routes traffic for big-name sites through Russia. https://arstechnica.com/information-technology/2017/12/suspicious-event-routes-traffic-for-big-name-sites-through-russia/, . Accessed: 2018-16-08.

[3] BGP hijacker booted off the Internet's backbone. https://www.theregister.co.uk/2018/07/11/bgp_hijacker_booted_off_the_internets_backbone/, . Accessed: 2018-16-08.

[4] How Pakistan knocked YouTube offline (and how to make sure it never happens again). https://www.cnet.com/news/how-pakistan-knocked-youtube-offline-and-how-to-make-sure-it-never-happens-again/, . Accessed: 2018-16-08.

[5] Iran's telecommunications company illegally rerouted Telegram app traffic. https://globalvoices.org/2018/08/06/irans-telecommunications-company-illegally-rerouted-telegram-app-traffic/, . Accessed: 2018-16-08.

[6] Pakistan hijacks YouTube. https://dyn.com/blog/pakistan-hijacks-youtube-1/, 2008-02. Accessed: 2018-10-10.

[7] CAIDA's AS-rank: Measuring the Influence of ASes on Internet Routing. https://www.youtube.com/watch?v=ONFpiYRg9yo, 2016-05. Accessed: 2018-28-08.

[8] The seven companies that really own the Internet. http://icaruswept.com/2016/06/28/who-owns-the-internet/, 2016-06. Accessed: 2018-04-09.

[9] Tier 1 network. https://en.wikipedia.org/wiki/Tier_1_network, 2016-06. Accessed: 2018-29-09.

[10] Google routing blunder sent Japan's Internet dark on Friday. https://www.theregister.co.uk/2017/08/27/google_routing_blunder_sent_japans_internet_dark/, 2017-08. Accessed: 2018-04-09.

[11] BGP leak causing Internet outages in Japan and beyond. https://bgpmon.net/bgp-leak-causing-internet-outages-in-japan-and-beyond/, 2017-08. Accessed: 2018-04-09.

[12] BGP Internet Routing: What Are the Threats? https://securityintelligence.com/bgp-internet-routing-what-are-the-threats/, 2017-12. Accessed: 2018-16-08.

[13] BGP Hijacking overview. Routing incidents prevention and defense mechanisms. (Updated). https://www.noction.com/blog/bgp-hijacking, 2018-08. Accessed: 2018-16-08.

[14] Cisco Wiki - Origin Attribute. http://docwiki.cisco.com/wiki/Border_Gateway_Protocol#Origin_Attribute, 2018-08. Accessed: 2018-28-08.

[15] Archipelago (Ark) Measurement Infrastructure. http://www.caida.org/projects/ark/, 2018-09. Accessed: 2018-10-09.

[16] AS relations. http://www.caida.org/data/as-relationships/, 2018-09. Accessed: 2018-18-09.

[17] AS Rank's *serial-1* AS relationships archive. http://data.caida.org/datasets/as-relationships/serial-1/, 2018-09. Accessed: 2018-04-09.

[18] AS Rank's *serial-2* AS relationships archive. http://data.caida.org/datasets/as-relationships/serial-2/, 2018-09. Accessed: 2018-04-09.

[19] Tier 2 network. https://en.wikipedia.org/wiki/Tier_2_network, 2018-09. Accessed: 2018-18-09.

[20] B. Al-Musawi, P. Branch, and G. Armitage. Detecting bgp instability using recurrence quantification analysis (rqa). In *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8, Dec 2015. doi: 10.1109/PCCC.2015.7410340.

[21] B. Al-Musawi, P. Branch, and G. Armitage. BGP Anomaly Detection Techniques: A Survey. *IEEE Communications Surveys Tutorials*, 19(1):377–396, Firstquarter 2017. ISSN 1553-877X. doi: 10.1109/COMST.2016.2622240.

[22] N. M. Al-Rousan and L. Trajković. Machine learning models for classification of bgp anomalies. In *2012 IEEE 13th International Conference on High Performance Switching and Routing*, pages 103–108, June 2012. doi: 10.1109/HPSR.2012.6260835.

[23] I. O. de Urbina Cazenave, E. Köşlük, and M. C. Ganiz. An anomaly detection framework for bgp. In *2011 International Symposium on Innovations in Intelligent Systems and Applications*, pages 107–111, June 2011. doi: 10.1109/INISTA.2011.5946083.

[24] S. Deshpande, M. Thottan, T. K. Ho, and B. Sikdar. An online mechanism for bgp instability detection and analysis. *IEEE Transactions on Computers*, 58(11):1470–1484, Nov 2009. ISSN 0018-9340. doi: 10.1109/TC.2009.91.

[25] M. C. Ganiz, S. Kanitkar, M. C. Chuah, and W. M. Pottenger. Detection of interdomain routing anomalies based on higher-order path analysis. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 874–879, Dec 2006. doi: 10.1109/ICDM.2006.52.

[26] Lixin Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, Dec 2001. ISSN 1063-6692. doi: 10.1109/90.974527.

[27] V. Giotsas, S. Zhou, M. Luckie, and K. Claffy. Inferring Multilateral Peering. In *ACM SIGCOMM Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pages 247–258, Dec 2013.

[28] Vasileios Giotsas and Shi Zhou. Detecting and assessing the hybrid ipv4/ipv6 as relationships. *SIGCOMM Comput. Commun. Rev.*, 41(4):424–425, August 2011. ISSN 0146-4833. doi: 10.1145/2043164.2018501. URL http://doi.acm.org/10.1145/2043164.2018501.

[29] Vasileios Giotsas and Shi Zhou. Inferring as relationships from bgp attributes. abs/1106.2417, 01 2011.

[30] Andreas Haeberlen, Ioannis Avramopoulos, Jennifer Rexford, and Peter Druschel. Netreview: Detecting when interdomain routing goes wrong. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'09, pages 437–452, Berkeley, CA, USA, 2009. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1558977.1559007.

[31] X. Hu and Z. M. Mao. Accurate real-time identification of ip prefix hijacking. In *2007 IEEE Symposium on Security and Privacy (SP '07)*, pages 3–17, May 2007. doi: 10.1109/SP.2007.7.

[32] Yiyi Huang, Nick Feamster, Anukool Lakhina, and Jim (Jun) Xu. Diagnosing network disruptions with network-wide analysis. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '07, pages 61–72, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-639-4. doi: 10.1145/1254882.1254890. URL http://doi.acm.org/10.1145/1254882.1254890.

[33] E. Kosluk I. O. de Urbina Cazenave and M. C. Ganiz. "an anomaly detection framework for bgp". *IEEE/ACM Transactions on Networking*, pages 107–111, Jun 2011.

[34] 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*, pages 290–299, Nov 2006. doi: 10.1109/ICNP.2006.320179.

[35] Mohit Lad, Dan Massey, Dan Pei, Yiguo Wu, Beichuan Zhang, and Lixia Zhang. Phas: A prefix hijack alert system. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1267336.1267347.

[36] Jun Li, Dejing Dou, Zhen Wu, Shiwoong Kim, and Vikash Agarwal. An internet routing forensics framework for discovering rules of abnormal bgp events. *SIGCOMM Comput. Commun. Rev.*, 35(5):55–66, October 2005. ISSN 0146-4833. doi: 10.1145/1096536.1096542. URL http://doi.acm.org/10.1145/1096536.1096542.

[37] Matthew Luckie, Bradley Huffaker, Amogh Dhamdhere, Vasileios Giotsas, and K. Claffy. As relationships, customer cones, and validation. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, pages 243–256, 10 2013.

[38] A. Lutu, M. Bagnulo, J. Cid-Sueiro, and O. Maennel. Separating wheat from chaff: Winnowing unintended prefixes using machine learning. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 943–951, April 2014. doi: 10.1109/INFOCOM.2014.6848023.

[39] Yakov Rekhter, Susan Hares, and Tony Li. A Border Gateway Protocol 4 (BGP-4). RFC 4271, January 2006. URL https://rfc-editor.org/rfc/rfc4271.txt.

[40] Xingang Shi, Yang Xiang, Zhiliang Wang, Xia Yin, and Jianping Wu. Detecting prefix hijackings in the internet with argus. In *Proceedings of the 2012 Internet Measurement Conference*, IMC '12, pages 15–28, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1705-4. doi: 10.1145/2398776.2398779. URL http://doi.acm.org/10.1145/2398776.2398779.

[41] Mitsuho Tahara, Naoki Tateishi, Toshio Oimatsu, and Souhei Majima. A method to detect prefix hijacking by using ping tests. In Yan Ma, Deokjai Choi, and Shingo Ata, editors, *Challenges for Next Generation Network Operations and Service Management*, pages 390–398, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-88623-5.

[42] Georgios Theodoridis, Orestis Tsigkas, and Dimitrios Tzovaras. A novel unsupervised method for securing bgp against routing hijacks. In Erol Gelenbe and Ricardo Lent, editors, *Computer and Information Sciences III*, pages 21–29, London, 2013. Springer London. ISBN 978-1-4471-4594-3.

[43] C.H. Trap. Real time distributed simulation of bgp on the entire current internet topology and internet-like topologies. 2018.

[44] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011. ISBN 0123748569, 9780123748560.

[45] Y. Xiang, Z. Wang, X. Yin, and J. Wu. Argus: An accurate and agile system to detecting ip prefix hijacking. In *2011 19th IEEE International Conference on Network Protocols*, pages 43–48, Oct 2011. doi: 10.1109/ICNP.2011.6089080.

[46] Zheng Zhang, Ying Zhang, Y. Charlie Hu, Z. Morley Mao, and Randy Bush. ispy: Detecting ip prefix hijacking on my own. *IEEE/ACM Trans. Netw.*, 18(6):1815–1828, December 2010. ISSN 1063-6692. doi: 10.1109/TNET.2010.2066284. URL http://dx.doi.org/10.1109/TNET.2010.2066284.

[47] Changxi Zheng, Lusheng Ji, Dan Pei, Jia Wang, and Paul Francis. A light-weight distributed scheme for detecting ip prefix hijacks in real-time. *SIGCOMM Comput. Commun. Rev.*, 37(4):277–288, August 2007. ISSN 0146-4833. doi: 10.1145/1282427.1282412. URL http://doi.acm.org/10.1145/1282427.1282412.