Otto-von-Guericke-University Magdeburg

Faculty of Computer Science

Networks and Distributed Systems Lab

SpeedCam: Efficient Detection of Multi-path Resource Overuse in SCIONLab

Master Thesis

Author:

Kilian Gärtner

Examiner and Supervisor:

Prof. Dr. David Hausheer

2nd Examiner:

Dr. Jonghoon Kwon

ETH Zurich

Magdeburg, 22.04.2018

Contents

Acknowledgements

I want to thank Prof. Dr. David Hausheer and Dr. Jonghoon Kwon for their intensive and very good support while supervising my thesis. The discussions with them and their suggestions were very helpful and improved the quality of the work. It was a productive and very enjoyable cooperation. Thanks to both of you!

Secondly, I want to say thank you to the SCIONLab team for their open minds and quick responses to questions. They also helped me with technical issues and solved them in cooperation with me. I hope that my work will help the SCIONLab team to accelerate the adoption of SCION in the world!

I've spent multiple month on this thesis, but never walked alone this road. I want to also say thank you to my friends in Magdeburg and Germany. They supported me in every possible way and without them I would not have the strength to do this report. They also proofread my thesis and provided me enormous feedback. Thanks to all of you!

Magdeburg, 23.04.2018

Kilian Gärtner

Abstract

This thesis presents a heuristic based mechanism to efficiently monitor traffic inside a network especially for multi-path communication in SCION. The goal is to detect user overusing their resources such as bandwidth and cause congestion in the network.

The requirements for the solution were a high detection rate with a low impact on the performance. Based on this, it should also be able to scale with hundreds of nodes. This was achieved by using a heuristic for the selection of probes and only using a partition of the network, 10% or fewer.

A concept for networks in general was proposed and implemented in SCION. The evaluation for it was done in SCIONLab over the course of 48 hours. It was shown that the solution reached the goals.

1 Introduction

1.1 Introduction

The current Internet has exceeded any expectations in terms of reach or various applications. The engineers creating it in the 1960s couldn't even imagine the impact on the society and various use of their creation. Only few users who know each other participated in the early Internet without considering security and protection against attacks as a key aspect.

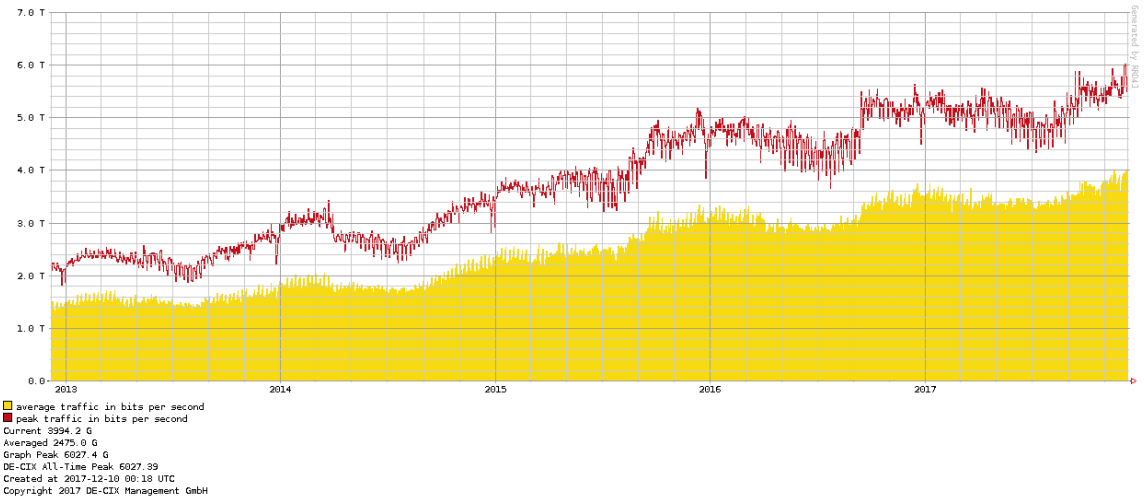
Today 50 years later there are billions of users and devices connected to each other [Min17], using different application from e-mail, streaming of video and audio or chatting which each other. Unfortunately, there are also malicious users trying to disturb the network by overusing resources or manipulating data to deceive the members. More and more services are created each year running on the Internet and trying to provide a service to users. Each company has to know the newest security risks and attack types to be competitive and trustworthy.

This growing interest and their security is only covered by multiple patches and fixes for the Internet Protocols. The problem of such an approach is that it must be first accepted by the majority of the users and then supported by network device manufacturers and administrators. For example, the acceptance of IPv6, a protocol released in 1996, is low in 2017 but the Internet ran out of IPv4 addresses in 2011 [ICA11].

Instead of fixing the current Internet and trying to patch the structural problems from past designs, there are also people creating a new solution. They want to solve the problems without limitations of the current protocols and their implicit fundamental problems. One of the solutions is **SCION**[PSRC17, BCP+17], that stands for **S**calability, **C**ontrol and **I**solation **O**n next-generation **N**etworks, which will be explained in detail in Chapter 2.

1.2 Motivation

One of the many issues is the strong increase of traffic on the Internet. The DE-CIX in Frankfurt, Germany, a data carrier, accounts each year a higher amount of data as shown in Figure 1.1. In 2014 the peak was around three TB/s. Three years later in 2017 the peak was at 6 TB/s, i.e. twice the amount.



https://www.de-cix.net/en/locations/germany/frankfurt/statistics (10.12.2017)

Figure 1.1: 5-year graph of the average exchanged traffic at DE-CIX in Frankfurt, Germany

In the advent of video streaming in form of Video on Demand (VOD) via services like YouTube[1] and Netflix[2] and live video streaming like Twitch[3] the data usage of the users increased rapidly. There is also an increasing demand for higher resolutions, which results in bigger video sizes. The current standard of Full HD for a video will be succeeded by new standards like the 4K. The user also demands for a higher frame rate. Currently, the videos are sent with 30fps, but there is a benefit using 60fps or even 144fps for really fast movements.

But videos are not the only reason for a higher bandwidth usage. Big Data companies processing petabytes of data each day rely on a stable and high bandwidth connection. There is also a trend with Internet-of-Things (IoT) to install small devices which collect and send data through the Internet.

The poor stability of the current Internet is also a motivation to improve the structure of it. Attacks like **D**istributed **D**enial-**o**f-**S**ervice (DDos) using the resources to disrupt the operation of companies or governments are increasing in size and frequency [Goo13a]. But also technical problems like a power outage [DC18] or an environmental hazard (earthquakes, floods) can disturb the stability of the Internet when a central node fails.

One solution to solve these problems is to scale-up and install more and more bandwidth capacities like fiber cables or newer devices. This is inefficient and consumes probably more energy and thus increases the running costs. There are also physical limitations for improving the performance so that a small benefit results in high costs.

Another solution is to better utilize the current resources as it is done with the multi-path feature in SCION. It allows the users to split their data and send the packets via multiple connections. This leads to a better usage of the available bandwidth in the network. In

[1] https://www.youtube.com, 20.04.2018
[2] https://www.netflix.com, 20.04.2018
[3] https://www.twitch.tv/, 20.04.2018

the current Internet this is not possible, because the routers are deciding for themselves which route to take to the destination.

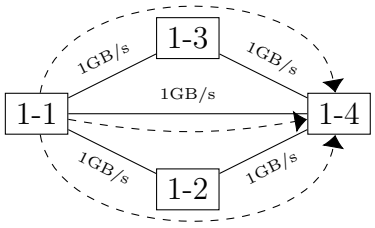


Figure 1.2: Example topology for multi-path communication.

This new method shall encourage users to better utilize the network, but it can also be misused. In Figure 1.2 an example is visualized. Each node represents an **A**utonomous **S**ystem (*AS*). All ASes in this figure but not 1-2 to 1-3 are connected with each other via a 1 GBit/s connection. AS 1-1 can send AS 1-4 data using single path with a 1 GBit/s directly. It could also use the paths 1-1→1-2→1-4 and 1-1→1-3→1-4 to send its data and increase its bandwidth to ideally 3 gigabit/s, if there is zero delay on the hops. This possibility rises the questions how to monitor and enforce the bandwidth usage of one particular AS. When using a single path connection the measurement point is at the destination. If this AS exceeds its limit the destination router can drop packets. This solution is not possible in a multi-path environment, because a single destination router is missing the information about the cumulative bandwidth of the source node. It is important to find an answer for this question to avoid misuse.

1.3 Goals

The work was planned and executed with the following described goals in mind. They will be evaluated in Chapter 4 and tested if they have been achieved by the proposed approach in this work.

1. The thesis' main goal is to propose a mechanism to identify network users who overuse their resources in a multi-path environment. These are considered *greedy*.
2. It is a goal that the solution must achieve a trade-off between the accuracy and the mechanism's efficiency. A detection rate of 100% is not necessary when the cost of computation time and memory is too high to be realizable on a network component.
3. There must be parameters to configure the detection mechanism. A network administrator can increase the detection rate when the additional performance impact is acceptable or the other way around. This results in a better acceptance of the solution.
4. The solution must be general enough to be adapted for other network types supporting multi-path communication. The solution can define requirements outside the multi-path capability to be working in such a network type.

5. Scalability is important for networks because of their growing size. The proposed answer must be able to scale with the size of the network in terms of detection rate and performance impact.
6. Based on the previous goals, the work will contain an implementation for SCION which can be used in its testbed SCIONLab. It is not a goal to provide a production ready application but a working prototype which scales and detect malicious users.

1.4 Main Contribution

The proposed contribution will provide a solution to identify greedy users inside a network with a small impact on the networks' performance. It will use a heuristic based selection algorithm to only monitor interesting nodes. This leads to a small amount of necessary nodes to receive a high enough coverage of the networks traffic to identify nodes which overuse their resources. The approach will be created for networks in general and will have a proof of work implementation for the SCIONLab network structure.

1.5 Thesis Outline

This thesis is divided into five chapters and an appendix. The first chapter Chapter 1 introduced the problem and provided a motivation for solving this problem. It defined goals to get achieved and outlined the main contribution to the topic.

The next chapter Chapter 2 explains the necessary requirements to be able to understand and follow this work and related work to this topic. It explains how the SCION network functions and what the used tool for this work is. The related work section will contain already existing works similar to the topic, such as SIPRA for SCION itself and results of other researchers.

The third chapter, Chapter 3 describes the main contribution of this work, the SpeedCam approach based on the inspection game. It is first described in a concept for networks in general and then specialized for the SCION structure. This chapter will also formulate theses and assumptions of the influence of featured variables.

These will be evaluated in Chapter 4, which contains the experiment to show the results. Firstly it will describe the setup for the evaluation and will then show and discuss the results of it.

This work will be concluded in the last chapter Chapter 5. This will summarize the work, show the implications of the results and give an overview of new questions and problems to solve in further works.

The appendix will contain configurations and listings to reproduce the results.

2 Background and Related Work

This chapter lists and describes the necessary topics to follow this thesis. It explains SCION's structure, functionality and details about the component as for example the border router. SCIONLab itself and its work is also outlined. At last, a short description about Prometheus and its used features will be given to understand how the experiment was done. The second part of the chapter will list related works which tried to solve the problem of multi path communication abuse by regulating or monitoring the behavior. The work on SIPRA and a VirtualCredit system will be presented, which are designed for SCION. Additional work for other types of networks will also be discussed.

2.1 Background

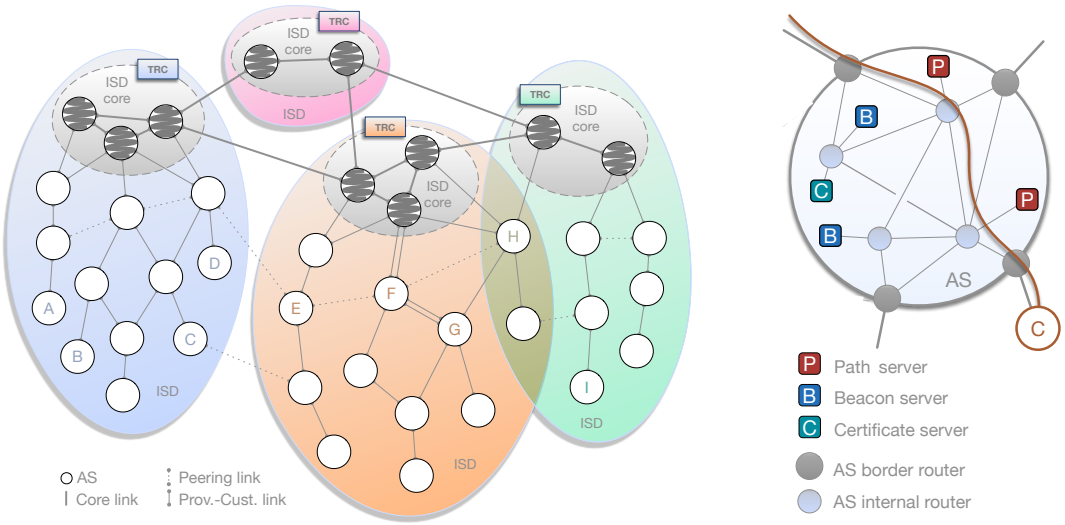
2.1.1 SCION

[BCP+17, PSRC17]

SCION, an acronym for **S**calability, **C**ontrol and **I**solation **O**n **N**ext-generation networks, provides a green-field solution for problems of the current Internet, which is named in this content as the *legacy Internet*. The main difference between them is *path-transparency*. In the current existing IP routing protocol the end-hosts cannot choose a path, but must rely on the router between them to find a good path. This leads to the problem, that the receiver cannot verify, if the packets were modified nor if the taken path was trustworthy. SCION allows the end hosts to choose between communication paths and also to see the path of a packet on the end-host. The fundamental change needs a different organization of the network nodes, which will be described in this part.

Figure 2.1a shows an example SCION network from [BCP+17]. This topology consists of **Is**olation **D**omains (ISD) and each node of it is considered an **A**utonomous **S**ystem (AS). They represent exactly one physical end-host, a whole companies network or an **I**nternet **S**ervice **P**rovider (ISP)'s network. Each ISD is administered and initially created by a small group of ASes, which are referred to *Core* ASes and form the *Core* of an ISD. They are responsible to define a policy which applies to all members of the ISD and is called the **T**rust **R**oot **C**onfiguration (TRC). Concluding this each ISD has its own rules to create connection and also defines the name the ISD. The Core ASes are also connected with other ISDs and link their ISDs to them. Additionally an AS can belong to multiple ISDs at one time, which is the case for the AS **H**.

Inside an AS (see Figure 2.1b) there exists multiple routers to route the traffic from one link to another. Each link is administered by one *border router*, while the *internal router* forwards the packets inside the AS itself.



(a) Four SCION ISDs with ISD Cores and ASes [BCP+17, Figure 1]

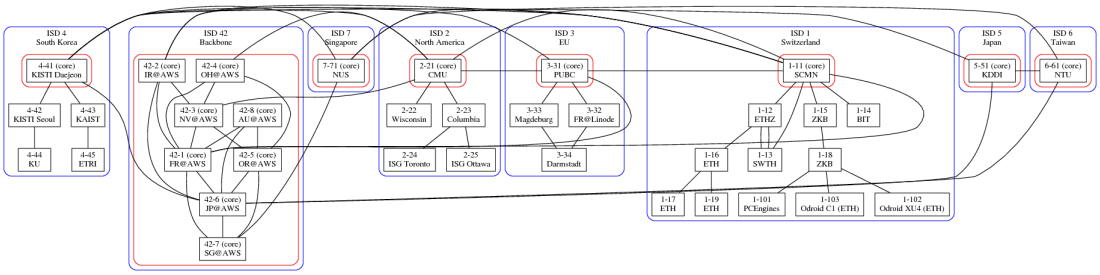
(b) Magnified view of an AS with its routers and servers [BCP+17, Figure 2b]

Figure 2.1: Structure of SCION with an example topology.

An AS needs to know the path to another AS to communicate to them. This routing problem is solved with a process called *beaconing*. A core AS announces such a beaconing inside its ISDs and learns the different paths from a core AS to a non-core AS. The ASes respond to this beaconing and over time the topology is created. On the other side, a non-core AS explores by beaconing the paths to the core-AS. The same procedure is done for inter-ISD communication by the core-ASes. The up-segment from a non-core AS to the Core and the down-segment from the beaconing of a core-AS are merged together and sent to the *path-server*. This component handles routing requests from one AS to another AS and responds with a set of possible paths to take. The elements of this set depend on the TRC of the ISD. For example, one ISD does not want to enable multiple path communication and only responds with the first available path.

One advantage of this structured routing is that a communication does not leave an ISD when it is not necessary. This enables to create ISDs for different security or efficiency scenarios. Packets are only routed in a trusted zone when they need to be. The legacy Internet does not guarantee this behavior. A packet from OvGU-Magdeburg to ETH-Zurich could be routed over different location in the world without control over it. If both of them were inside one ISD containing only research institutes, then the packet would be guaranteed to not leave this ISD because it is not necessary.

Another advantage is the possibility to choose an alternative way. This is interesting in cases a path segment has a failure and should be avoided for different reasons like malicious attacks or technical failures. The legacy Internet relies on the Border-Gateway-Protocol (BGP) [RLH05, Hal97], when was inspected by many researcher for its fault-tolerance and stability. It was also shown in [SKM06] that the reaction time (convergence) for realistic topologies takes seconds to minutes for the protocol to adjust to failures. A similar result was given by [LABJ01] of a delay of 3 minutes for 30% the cases and even higher as 15 minutes in total. In SCION this slow reaction problem is minimized by letting the user choose a different path. If one path is not responding the node can request for another path or use one of the provided alternatives.



https://www.scionlab.org/public/img/topology_no_labels.gv.png (18.04.2018)

Figure 2.2: SCIONLab topology. Shows the major ASes without the user VM. The core ASes are bordered in red

The multiple paths to the target can also be used for a better bandwidth utilization. Instead of sending the data over only one link the stream can be split into multiple sub streams and transfered over the available paths to the target. This leads to a better utilized network capacity because otherwise unused links are used and not idle. Bandwidth cannot be saved or stored, it can only be used in a moment or it would be wasted. But this can also lead to an unwanted flood of the network and can create multiple congestions, unintentionally. This would negate the advantage and should only be used with caution. Because of the potential misuse of such a multipath communication several studies have been conducted which are explained in Section 2.2

The legacy Internet cannot be replaced at once but can only be updated or replaced incrementally. This process can take many years as seen for the deployment of IPv6. The update for a necessary address space expansion was firstly published in 1998[DH98] and its adoption rate is in 2018 only at around 22% [Goo18] to 24% [Rip18a], even the IPv4 address space ran out on 03.02.2011[ICA11]. Similar to the transition from IPv4 to IPv6 there is a need for a transition from the legacy Internet to SCION, which is realized with the **S**CION-**I**P **G**ateway. It encapsulates the IP packets in SCION packets to interact with the legacy Internet.

This description is only an overview about the complexity and mechanism of SCION, but this will be sufficient to understand the nature of this work. For further reading it is recommended to start with the paper of Adrian et. al[BCP+17] and go into details with the published book of Adrian et. al[PSRC17].

2.1.2 SCIONLab

SCIONLab[1] is the test environment to test out SCION and its functionality in a running network. Its current topology can be seen in Figure 2.2. Any interested user is able to create an AS as a VM which is automatically configured and ready to deploy. They can choose different attachment points their AS is creating a connection to. This system enables the user to simply test out SCION in a sandbox without additional impact.

[1] https://www.scionlab.org, 18.04.2018

2.1.3 Prometheus

"Prometheus is an open-source systems monitoring and alerting toolkit [...]" [2]. This framework consisting of a *Prometheus server* and an API to create and gather data about programs, which is implemented by the *Prometheus clients*. Each client defines a data model of different *metrics* to monitor. These metrics are collected by the Prometheus server in a given scrap interval and stored on the server side. A simple Web UI enables the user to write queries and analyze the data, for example extracting the maximum bandwidth at a given time.

The client API defines four different metric types. The *counter* represents a data which can only go up while the *gauge* can also go down over time. An example for a counter is the total bandwidth in bytes since the border router started whereas a gauge can be the current memory usage. The two other types are the *histogram*, which samples observations in different buckets, and the *summary* which does the same as a counter but also calculates quantiles over a sliding time window. Each metric consists of the value itself as a double-precision floating point number and a UTC time stamp. The client defines the type of the metric and the name and continually updates the value. The metrics are exported over an HTTP based API. The endpoint is typically `http://[HOST]:[PORT]/metrics` which responds with a list of all metrics and their values at the time of request. The Prometheus client API was already implemented in the border router, the path, the beaconing and the certificate server. An example of this is seen in Section C.

2.2 Related Work

This section will list and describe work by other people with similar problems or thoughts. It will show their difference to this work and why this work exists.

2.2.1 SIBRA

The **S**calable **I**nternet **B**andwidth **R**eservation **A**rchitecture (SIBRA) is an approach against resource overuse by Basescu et. al [BRS+16]. The architecture enables nodes to reserve a certain bandwidth for a time slot. This bandwidth is guaranteed for the flow and is not shared with other users on the path. The main reason to establish such a system were the increasing DDoS attacks and the flaws of the defenses against it. Over the last years the amount and size of DDoS attacks are increasing as shown in the life visualization [Goo13a]. The largest recorded DDoS occurred on the 28th of February in 2018 on github.com with a peak of 1.35Tbps [Kot18]. SIBRA invented a mechanism that two nodes can communicate with each other even in the presence of malicious nodes disturbing the network.

This was achieved by introducing three layers of contracts for a guaranteed bandwidth. Each of them differ in time and dimension: The **core** are *long-term* contracts and are established between the core ASes of different ISDs. Within an ISD the ASes conclude contracts intermediate-term with each other which are called **steady**. These are used for

[2] https://prometheus.io/, 18.04.2018

a steady, but low bandwidth transition. A *short-term* contract, called **ephemeral**, is conducted between two end-hosts. These are reserved for high bandwidth traffic.

A link's capacity is divided into three partition: 80% is reserved for ephemeral path, 5% for steady paths, and 15% are for best-effort. An ephemeral path is created by sending a request through a steady path and contains the amount of bandwidth to reserve and a time slot to use. The default life time of such a path is 16 seconds and the time slot granularity is 4 seconds. A packet using a SIBRA reservation contains a flow ID, an expiration time, a bandwidth class, a path direction type and a reservation index. The SIBRA routers maintain a *bandwidth table* to store the reserved bandwidths of the router's neighbor, an accounting table to keep track of flow ids and their attributes, and a *pending table* for pending reservation. A SIBRA packet with an invalid header will use the best-effort bandwidth.

This enables SIBRA to efficiently control malicious flows. A botnet AS can be excluded from reserving bandwidth and the best-effort band is never disturbed. That mechanic solves the problems of this work and was already established and tested in SCION. The reason for this work is that SIBRA is very complex to implement and not suitable for SCIONLab's test environment. A simpler approach was needed to identify malicious user or prevent them.

2.2.2 Virtual Credit

[MG] Another approach to reserve bandwidth and limit the abilities of malicious user was introduced by Meyer et. al for SCION. The base idea was that an AS can only use resources when it also provides them to the network. This was realized by introducing a *virtual credit* an AS can use to buy capacity. It would only gain such a credit if it provides capacity. The intention was to motivate the user not only to consume the networks resource but also to provide them by establishing new connections or building new infrastructure elements like fiber cables or backbones.

It was based on the monitoring system RIPE Atlas [Rip18b] which enables the user to monitor traffic and send probe requests. To do so they need credits which they gain when they provide a usable probe.

One virtual credit is worth 10 MBits/s of capacity to reserve without an expiration. When one AS requests such a connection to another AS, the second AS would get the credit the requester invested. Each AS starts with 100 credits which is worth of 1 GBit/s. This simple design prevents AS to use more bandwidth than they would provide. A malicious user would need other ASes to establish connection to it to gain credit and use it for a malicious attack.

This alternative approach to prevent an overuse was not considered because it was easily exploitable as shown in the paper itself. A malicious AS can spawn multiple virtual ASes which connect to the malicious AS. This one now has the credits of the virtual ASes but they do not improve network. The malicious AS can use their credits to flood the network. Because of these exploits in such a basic concept it was not considered for this work.

2.2.3 Delay-Tolerant Networks

Zhu et. al [ZDG+14] created a system called *iTrust* to detect nodes in a delay-tolerant network which does not honestly forward messages. The forwarding of messages is important in such network as they suffer from frequent disconnection. Examples of them are sensor networks with a temporary connection to each other or mesh networks on smartphones. A malicious node could drop messages instead of forwarding them as intended. Zhu et. al use the inspection game to calculate the probability to inspect a single node if it acts suspicious. They showed that using this model can reduce the detection overhead. The idea of using Game Theory to model such a system was used in this work.

2.2.4 Inspection Game in other contexts

The *Inspection Game*, part of the *Game Theory*, is used in this work as a theoretical model for the monitoring. While described later in detail, the inspiration for this was from other applications. They are examples that the Inspection Game is a general construct with many application, even outside the topic of networking. Nozenzo et. al [NOSv14] examined the effectiveness of this model in regards to bonus for comply and a fine for not. They showed that the existence of such a fine improves the behavior of the players but that a bonus has a low impact.

Avenhaus et. al [ACM+96] showed the application of this game model for the nuclear arms control from the years 1961 to the present 1996. They described the cost of an inspection and that a constant inspection is not realizable because of limited resources. Also a real world application for it was done by Kirstein [Rol09] to show the effects for doping in sport.

3 Design and Implementation

This chapter will describe the following three parts. At first it will outline the general idea of the SpeedCam approach for any network type. After this it will describe the specific adaptation for the SCION network. At last there will be an explanation what has to be changed for SCIONLab and the reason for that.

3.1 Overview

SpeedCam is a heuristic approach to monitor network traffic with regards to limited resources. It analyses the network structure and decides based on the heuristic where are good places to monitor local traffic. The approach also tries to reduce the necessary amount of probes to minimize the resource impact on the machines while achieving a high detection accuracy.

3.2 Inspection game

The idea behind SpeedCam is based on the *Inspection Game* in Game Theory [Dut99, Owe13]. Inside the game there is a *task* to be done under defined *rules*. The execution of the task can be simpler, more efficient or more effective with a violation of these rules. The task is performed by one of the two parties, the *inspectee* and the compliance of the rules costs resources and is monitored by the second party, the *inspector*. The inspectee has a general interest to break the rules to perform better, while the inspector wants to reduce his amount of inspections to use as few as possible resources. The possible combinations are shown in Table 3.1.

		Inspector	
		Inspect	Ignore
Inspectee	Comply	A	C
	Violate	B	D

Table 3.1: Inspection game matrix.

The cases B and C are good investments of the inspectors' limited resources. In the first one he didn't inspect while the task was performed without violation and in the second one the inspector found the culprit. For case A the resources of the inspector were wasted because there was no reason to inspect. The inspector failed in its task in case D, because he missed a violation of the rules by not investing the resources.

Because of the shown cases and its results, it is obvious that a *pure strategy* doesn't make sense in this game. When the inspector always inspects he will waste resources and when the inspector always ignores he fails his task. Instead of a pure strategy there must exist a *mixed strategy* defined by probabilities for each action to be chosen to achieve a Nash equilibrium. At that point the inspector nor its inspectee would not gain any benefit from changing their strategy.[N+50]

3.3 Generalized concept

The previously described form of the Inspection game is adopted for networks for this work. The *nodes* of a network are the *inspectees*, their *task* is to *transfer* much data in a short time. The *rule* for all nodes is, that a transaction of data *shouldn't exceed the link limits* or disturb other nodes' transfer. These limits have been agreed between both partner and can be either a fraction of the physical limit or the complete possible bandwidth depending on how many users share the medium. A *detected violation* of these limits or rules can result in throttling the bandwidth or even exclude the node from the network, temporary or permanently. A node can try to exceed the limits and so breaking the established rules with the interest to use the additional bandwidth to transfer data faster.

The *inspector* in such a game has the possibility to monitor a nodes' network traffic. Monitoring a node costs computation resources such as memory or CPU cycles. Both can be critical for a switch to send packets. Therefore an inspection, necessary or not, will have an impact on the network throughput. The challenge of the inspector is to only monitor when a node is exceeding its limits. This will be presented in this work as the SpeedCam approach. It will give the inspector a heuristic with the goal to increase the amount of cases for B and C and to reduce it for A (not wasting resources) and D (fairness for network member).

The general concept of SpeedCam applies to types of networks which meet the following requirements. The network must consist of *nodes*, ASes, and *links*, which connect only two ASes. The nodes and links must form an undirected graph. For SpeedCam it is not necessary that the graph must be known in the first place nor to be static. This is important because real networks tend to be dynamic and decentralized with only partial knowledge about its structure. The inspector must have access to this network in such a form, that he can gain knowledge about the network structure and monitor a nodes' traffic.

When the requirements are met the inspector can be run as a component. It executes the SpeedCam approach and stores the necessary information about the nodes. It is described in this work as a centralized component, but it can be split into multiple inspectors, each responsible for a subnetwork.

One run of the SpeedCam approach is called an *episode* and is broken down in four phases, as shown in Figure 3.1.

The exploration phase is the 0-th phase because it can be skipped if the network is static and fully explored. This phase can also be running in the background in parallel to the

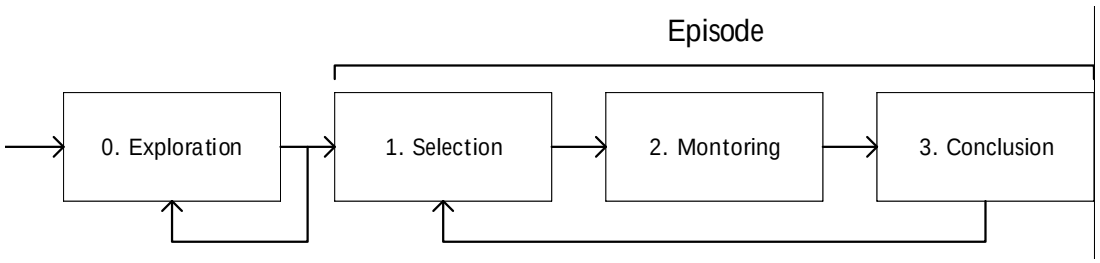


Figure 3.1: SpeedCam's phases

other phases. Phase 1 to 3 must be run in sequential order, because they depend on each others results. The next sections will explain each phase in detail.

3.3.1 Exploration

This phase has the goal to create an undirected graph of the network to monitor. The structure of the network is necessary for the following phases.

For the general concept of the SpeedCam approach there exists multiple possibilities to explore the underlying network. Because the only assumption about the network is that nodes are connected via links, these possibilities are also general and are lacking of details how to retrieve them. In later discussed specialized forms of the SpeedCam approach these are given.

The first possibility is to manually create the graph. A network, virtual or physically present, must be created by an entity, be it a humanoid administrator or a program. Every change of the network can be logged and used to create the graph. This is obviously a tedious task and not error prone.

Another possibility is to automatically explore the network from a known starting point, for example a central node. For each node it must be possible to retrieve the connections of the node and their targets. With these properties it is possible to explore the graph with any traversal algorithm like Breadth-First, Depth-First, Djikstra or Floyd-Warshall. [SW11] Using these algorithms is an established way to explore the network, but requires access to each node or a protocol to retrieve the edges of a node.

The exploration phase itself needs to be constantly repeated to react to changes inside the network. Inactive or disconnected nodes don't need to be monitored and can be removed from the graph. This reduces the graph which improves the performance impact by using fewer memory and fewer CPU cycles to iterate over.

3.3.2 Selection

Based on the results of the previous exploration phase, the selection phase uses the network information and returns a set of nodes to monitor. This phase uses the heuristic. It has to estimate in advance if the node will transfer any traffic.

The selection process has two goals to achieve. It has to minimize the amount of used probes to save resources. But also the probes have to form the maximum amount of traffic

cover to achieve a high accuracy of the monitoring. These two goals are contradictory. The minimum of probes is zero, which is equals to no inspection and results in no traffic monitored. On the other hand, the complete traffic can be monitored by using all nodes as probes, but this is not resource efficient.

Because of this, the selection heuristic must achieve a trade-off between as few probes as possible with a high enough precision of the monitored traffic. The heuristic will use information about the graph structure itself and information from past monitoring phases to increase the quality of a selection.

The quality of a selection will be defined as the following:

$$q(S, N) = \frac{\text{traffic}(S)}{\text{traffic}(N)} \tag{3.1}$$

where:

S = Set of probe nodes
N = Set of all nodes

Another criterion of this phase is the randomness of the selection process. The inspector shall not be predictable for the inspectees, otherwise they will have an advantage and may avoid an inspection. When a set of probes is predictable for the inspectees, they can decide to violate the rules outset of this set and chose other paths or times to transfer their data.

For example, Figure 1.2 shows a simple network. Lets assume, that the inspector is watching the node **1-2** and all the nodes inside the network are knowing this fact. With that knowledge, they will choose the path 1-1→1-3→1-4 instead and avoid the inspection. This problem is even more present for a real network, because they tend to be complex and provide even more alternative routes to avoid an inspection.

One solution to counter the predictability is to keep the algorithm and its configuration secret from the inspectees. This approach has two issues: The first one is, that *security-by-obscurity* doesn't work on the long term. The algorithm has to work even when the inspectees know how it is working. This can happen for multiple reasons: An employee with sufficient access can publish the configuration of the network for money, the result of dissatisfaction or by accident. Also a security issue in the system can be used by an attacker to retrieve the configuration and distribute it. Because there are multiple ways to break the secret, the approach needs to work even in the presence of knowledge about the configuration. The second issue it, that the inspectees can use the inspection's statistic distribution to reverse-engineer the configuration. They need to record when an inspection is happening to collect a history. This enables the inspectees to predict the next inspection based on the history and to circumvent it.

A better solution is to introduce an entropy to the selection process. Instead of using the set with the highest quality, the algorithm will randomly choose a set. This conflicts with the selection process itself. There is no need to select candidates based on information when every candidate can be chosen randomly.

The SpeedCam approach uses a candidate score $cs(n)$ to determine the probability of each node to be chosen. While not guaranteed to be chosen, the chance to be a probe is higher

for a good candidate than for a bad candidate. This process will result in a high quality while also not being fully deterministic. The following will describe what this candidate score is composed of.

Criteria

The following presented criteria are considered **static** and can be calculated without a history of past measurements. They require the topology of the network.

Degree [$deg(n)$]: The sum of incoming and outgoing links of node. This criterion represents the potential for a good transfer hub and can be used by other nodes as a central transfer hub. The higher the degree of a node, the higher the possibility to monitor a great range of traffic sources. A node on the edge of the network, a sink, has a low degree, mostly 1. These are uninteresting because they can only monitor the traffic of this node. A good real example for a node with a high degree are the nodes of submarine cables, as shown here[1]. They have the ability to connect two continents with each other and thus provide a good point to measure traffic.

Total capacity [$cap(n)$]: The sum of the connection capacity of a node. The capacity is the physical or between partners agreed limit to transfer bytes per second. This is similar to the **Degree**, but it differs in one aspect: The degree is only an indicator about the possible connectivity, but not for the possible throughput of traffic. A node with a high capacity can be preferred over a node with a high degree and lower capacity to transfer data. Because of this difference the two criteria are separate.

Average activity [$\overline{act_{t_1,t_2}}(n)$]: The relation between possible and used capacity of a node in a time interval between t_1 and t_2. The time span can be of any resolution, but should be consistent.

$$\overline{act_{t_1,t_2}}(n) = \frac{\text{traffic}_{t_1,t_2}(n)}{cap(n)} \tag{3.2}$$

Storing the timespan for an episode gives the possibility to create a profile for a node to be used by the selection process. The selection itself can now be time based. A node with a high activity in the morning can receive a higher candidate score than a node with a high activity in the evening. This can increase the quality of the set.

The meaningfulness of this criterion depends of the length on the history. A short history maybe represents an inaccurate result of the node, while a long history can hold outdated information about the behavior of that node.

[1] https://www.submarinecablemap.com, 02.04.2018

Candidate score

The score for each node is calculated as the following:

$$cs(n) = w_{deg} \cdot deg(n) + w_{cap} \cdot cap(n) + w_{act} \cdot \overline{act_{t_1,t_2}}(n) \tag{3.3}$$

where:

n = A node
w_{deg} = Weight for the degree, $\in \mathbb{R}_{\geq 0}$
w_{cap} = Weight for the capacity, $\in \mathbb{R}_{\geq 0}$
w_{act} = Weight for the activity, $\in \mathbb{R}_{\geq 0}$

The weights are a possibility to configure the algorithm for the needs of a network. The evaluation part will show the results of using different values.

The **average activity** will be zero for the first episode, because the inspector does not have information about the traffic of such a node.

Probability

The probability to be selected as a SpeedCam depends on the previously calculated candidate score, $cs(n)$ Equation 3.3. It is calculated for all nodes inside the network and assigned to the node itself for that episode. With that information, the probability can be calculated and assigned to the node with:

$$P_{sc}(n) = \frac{cs(n)}{\max(cs(N))} \tag{3.4}$$

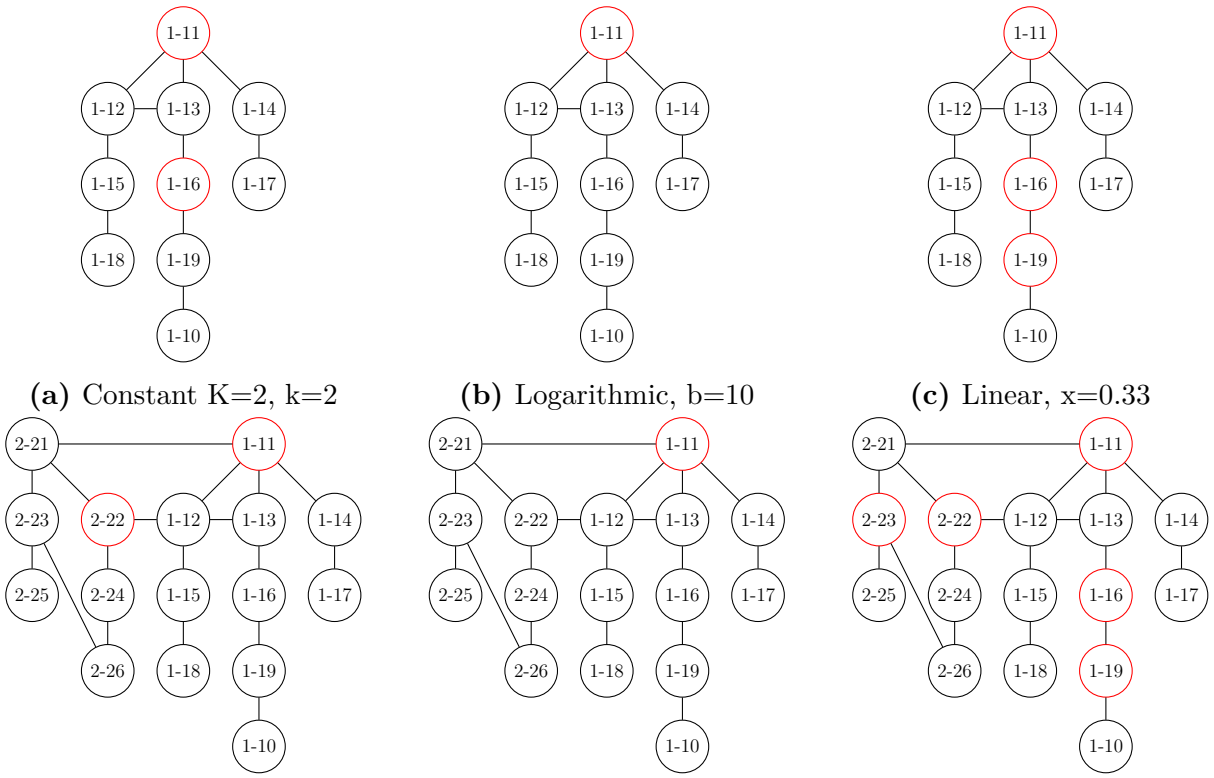
where:

N = All nodes inside the network

SpeedCam amount

Now exists a list of candidates with an assigned probability to be selected based on their candidate score. The inspector must now select k *SpeedCams* from this set. The size of k should scale with the network size $|N|$, but as discussed earlier it should be as small as possible. The question of this phase is how many nodes does the inspector need to deploy to find a greedy user with a high accuracy? There are a few possible scales, which were discussed in this work:

Constant [K]: A constant amount of nodes will be selected, ignoring the size of the network. The advantage is the predictability of the computation time and is suitable for static networks. But it is assumed that one node in a dynamic network increasing over time is not sufficient to achieve a high enough precision.



(a) Constant K=2, k=2 **(b)** Logarithmic, b=10 **(c)** Linear, x=0.33

(d) Constant K=2, k=2 **(e)** Logarithmic, b=10 **(f)** Linear, x=0.33

Figure 3.2: Scaling strategies

Logarithmic [$log(|N|)$]: A logarithmic amount of SpeedCams will scale nicely with any network because it needs very few nodes in large networks.

Linear [$x \cdot |N|, 0 \leq x \ll 1$]: A linear amount of SpeedCams will need more resources as the logarithmic approach, but will also raise the accuracy to sufficient levels. The factor x must be much-less than 1, otherwise the inspector will utilize half the network and this contradicts the goal to use as few resources as possible. The evaluation will use $x = 0.1$, $x = 0.2$ and $x = 0.33$.

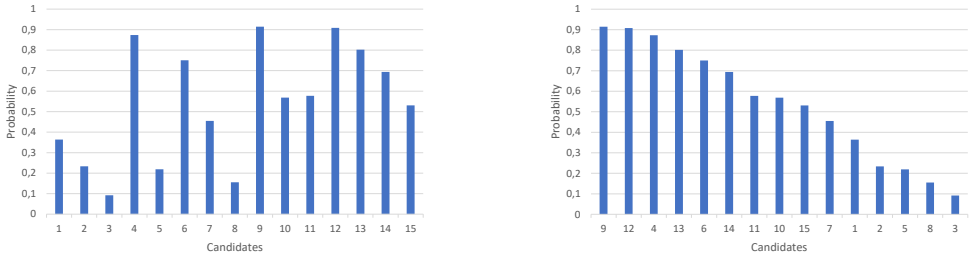
The different scale effects are visualized in Figure 3.2 when the network is expanding over time. The upper graph shows only one ISD, while the bottom graph shows the explored graph with ISD 1 and now discovered ISD 2 . A red circle indicates, that this node is selected. The top row from a) to c) shows a network of the nodes, while the bottom row from d) to f) expands this network to 16 nodes.

Select SpeedCams

The selection of SpeedCam from the candidates itself is a straight forward process. Iterate over all nodes, generate a random number[2] from 0 to 1 (exclusive) p and if $P_{sc}(n) \geq p$, then add the node n to the selected set. The iteration is over, when enough speed cams were selected.

[2]It is sufficient for this work to use a weak pseudo random generator

The order of the candidates matters, because k limits the size of the target set. This problem is shown in Figure 3.3. In Figure 3.3b they are sorted by their probability. The loop starts at the candidate with the highest probability. The chance, that p is high enough that n is selected, is higher for later n. Concluding from this, the chance for $n = 3$ to become a SpeedCam is lower in case Figure 3.3b than in the case Figure 3.3a. Because of time constrains for this work, this change is not evaluated.



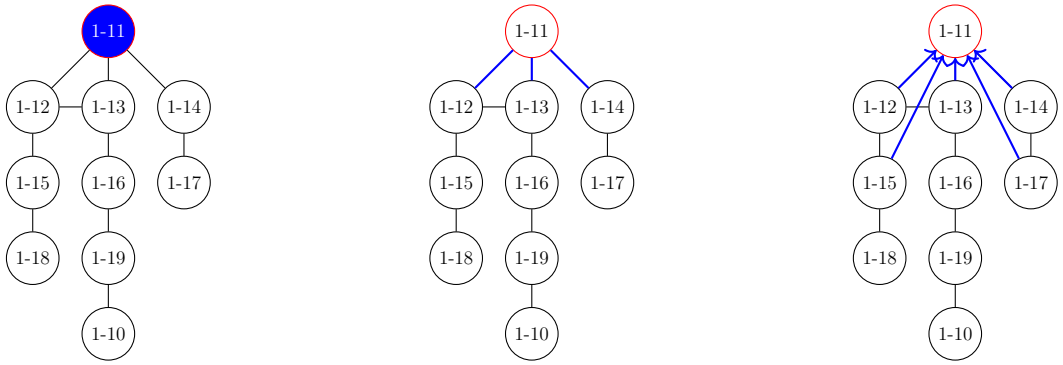
(a) Unsorted **(b)** Sorted by probability

Figure 3.3: Example of detection scheme

The selection phase ends with a set of k nodes, which are functioning for the next phases as **SpeedCams**.

3.3.3 Monitoring

The monitoring uses the set of nodes from the selection phase, which are now called SpeedCams. They have the task to log the transferred traffic over a timespan from t_1 to t_2. At the end of that timespan, they need to report the results to the inspector. There are different resolutions for the logged traffic, which are visualized conceptually in Figure 3.4. A red outline represents a probe and a blue color represents the possibility to measure traffic.



(a) Node **(b)** Link **(c)** Flow

Figure 3.4: Monitoring resolution

The lowest resolution is to summarize the in- and outgoing traffic of the node (Figure 3.4a). This resolution is simple to implement and produces the smallest amount of footprint, because it needs only one counter per node. But it lacks details and limits the possibility of the SpeedCam.

The next resolution is higher and logs the traffic per link separately (Figure 3.4b). The advantage over the previous resolution is that it is not only possible to log the activity of the node itself by aggregating all link traffics. It gives the possibility to also log a fraction of traffic from the neighbor node. The disadvantage is that each link has to be monitored and there is a counter needed for each link. This results in a higher footprint.

A higher resolution of the monitoring is achievable by also looking at a packets metadata. The previous resolutions only aggregate the packet sizes, while a packet inspection of the header enables the SpeedCam to construct flows (Figure 3.4c). With that information it is possible to not only reconstruct the neighbours traffic, but also the sender and receiver traffic. This approach has a higher footprint, because it needs to count the traffic for each node pair.

The choice of resolution is based on the amount of available computation power and the provided information of the network. The current Internet is currently structured according to the OSI-Lay model [Par88], which defines responsibilities and data models for each layer. Each layer has only a limited access to the data of a packet to provide a comparability with each other. That limits the possibilities of the inspector's options based on where he is placed.

These results are sent to the inspector and processed in the conclusion phase.

3.3.4 Conclusion

The conclusion uses the information from the monitoring phase about the logged traffic from the SpeedCams. The goal of that phase is to aggregate the traffic and reconstruct the flow inside the network to identify a misuse of network resources. It also feeds the results back to the graph and ends the episode.

The inspector receives the measured traffic from the SpeedCams and aggregates them in such a way, that the inspector can add them to the corresponding node. After this assignment, the inspector decides, which node violated the rules. A violation can be punished to prevent further ones. The punishment can be a temporary or permanent reduction of the available bandwidth or a banishment from the network. The work does not go into details about the punishment because it concentrates on the detection. Additional information about this topic can be found in Section 5.3.

The classification of the a node depends on two factors: The first one is if the inspectee violated the rule and overused its limits. The second one is if there was a *congestion* inside the network, so that one node was constantly at its bandwidth limit and unable to fulfil its task. A congestion occurs when the bandwidth of a link is exact or nearly the same as the links capacity and there are packets dropped because of a buffer overflow. The resulting cases are shown in Table 3.2.

	no congestion	congestion
stays in limit	I	II
overuses limit	III	IV

Table 3.2: Different classifications of a node

Case I - Fair

This node uses fewer bandwidth than it has subscribed to and doesn't violate the rule. There is also no congestion inside the network. In this case, there is no need to punish the user. They could use even more bandwidth, because the network is currently underutilized and possible bandwidth is wasted.

Case II - Global greedy

In this case there is a congestion inside the network. The currently used bandwidth is higher than the available. But there is not any node which violated the rule. This results in a global greedy behavior of all nodes, because their cumulative task needed more resources than were available. An example is seen in Figure 3.5.

It is theoretically possible to solve this case and by that to increase the efficiency of the network. These global greedy users can be identified using clustering-algorithms, such as *Jenks Natural Breaks* [Fre67]. These will partition the users by their bandwidth usage and the top user can be plead to reduce their consumption. Because they often have paid for this service, they cannot be forced to do so. In summary, it is possible to identify them, but without a defined protocol all users have agreed to it is not possible to solve this case.

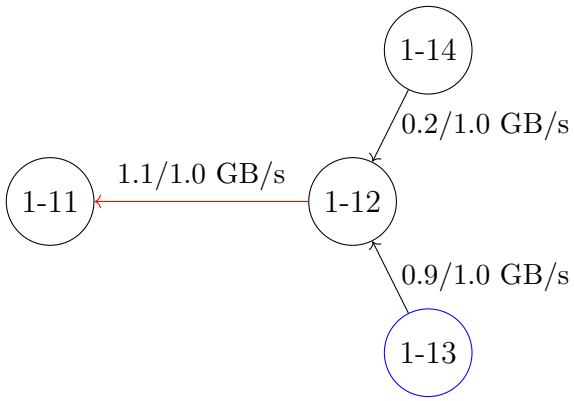


Figure 3.5: Example for Case II user (blue). It does stay in its limits, but uses much more bandwidth than AS 1-14.

Case III - Overusing with no congestion

This case occurs when there is no congestion inside the network, but a node uses more bandwidth than allowed. For example, a node has only a bandwidth limit of 1 GBit/s but used 1.2 GBit/s at the time of the inspection. The inspectee was violating the rules but without disturbing the task of other nodes because it didn't cause a congestion. In other words it utilized the available bandwidth.

This case isn't simple to solve. On the one hand the node clearly violated the established limits and tried to cheat. That behavior can lead quickly to congestion when every node is doing so and should be prevent by punishment or regulation.

On the other hand this utilized the available resources in a better way. This is not fair but more effective without a direct harm. Even the fact that there was bandwidth available to use implicates that the limits were not optimal and can be improved.

The handling could be based on a tolerance range for overusing for example an overuse of 10% is tolerated. As long as the user stays in this range and does not cause a congestion no punishment is necessary. To increase the fairness for other users it is possible to implement a reputation based punishment system. A greedy node's reputation would decrease of time and the lower the reputation the higher the punishment when the overuse lead to a congestion.

When the user overuses their resources even higher than the tolerance allows to it should be handled as a Case IV and be punished.

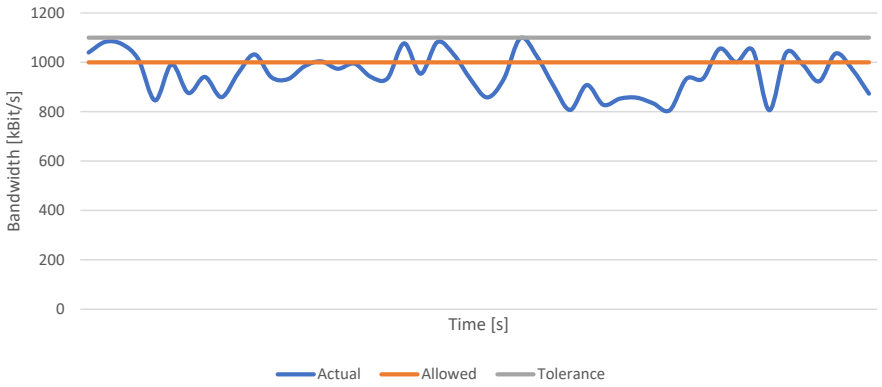


Figure 3.6: Example for a Case III user (blue). It exceeds its limit (orange) but stays in the tolerance (gray) range.

Case IV - Overuse resulted in a congestion

The inspectee violated the limits and maybe caused a congestion with that behaviour a congestion. This node should be punished to prevent this in future episodes.

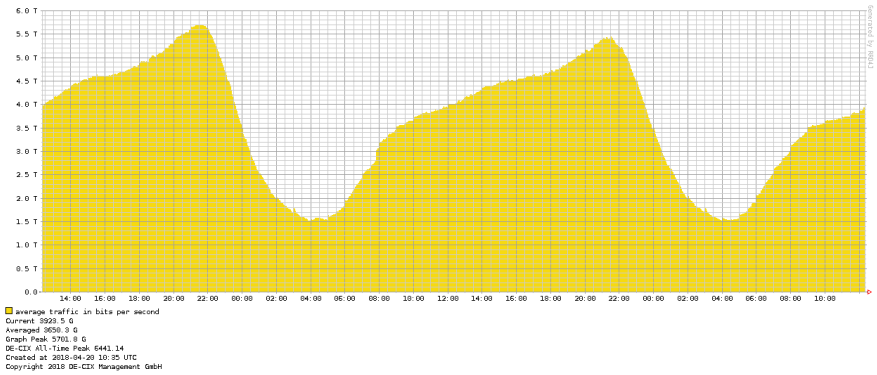
3.3.5 Repeat

The previously described phases need to be repeated in a certain interval. As with the other phases, there are several different strategies possible.

A *fixed interval* with an endless repeat mode is the first one. The inspector starts the inspection every t time unit and gets a constant stream of information about the network traffics. The precision of the inspection depends on the size of the interval, the smaller the interval is, the higher is the precision. But also increases/decreases the resource impact. This strategy gives also the possibility for the inspectees to recognize this pattern and adapt.

A *randomly chosen* time schedule prevents this problem. The inspector starts an inspection non-deterministically a few times over a given timespan, like a day or week. This decreases the precision of the measurement, because the interval changes randomly. The inspector loses the possibility to improve its selection quality by using the time profiles. This strategy can also waste the inspector resources, when the inspection is randomly chosen at a point of time with nothing happening of interest.

Another strategy is to utilize the knowledge of past episodes and the course of the traffic over a day. The inspector can check, if there are timestamps when the traffic is usually high, for example in the evening for consumers as seen in Figure 3.7. This strategy is *experiences based* and has the same advantages and issues as the **activity** criteria from Section 3.3.2. A longer history enables more precise decision, but a too long history can hold outdated information. Also is a purely experienced based strategy vulnerable to get stuck in a local optima. This convergence can be avoided by using a chance to pick a non optimal timestamp to learn something new.



https://www.de-cix.net/en/locations/germany/frankfurt/statistics (20.04.2017)

Figure 3.7: 2-day graph of the average exchanged traffic at DE-CIX in Frankfurt, Germany

3.3.6 Summary

This section described the SpeedCam approach for general network and the different strategies. It is designed to work on any network, but quite so it lacks of necessary answers for questions like "Which components enable the inspector to measure traffic?" or

"What protocols are necessary to exchange the monitored information?". The following section will show the example implementation in SCION and the section after especially for SCIONLab.

3.4 Implementation in SCION

This section will outline the implementation of the general concept as discussed in Section 3.3. SCION is under heavy development and hasn't reached a long-term-support (LTS) version yet. So it is necessary to mention, that the implementation was done with the version 24d2e97909599bf35dcbf6308aa2302610cf9689 from 2018-02-07T17:03:35Z.

3.4.1 Differences

There are for this work interesting differences from the general networks to SCIONs structure. They will be shown in this part and used later for changes inside the phases. The nodes or inspectees will be now called ASes, because they are named so in SCION.

The first difference is, that there are core ASes inside one ISD. They are forming the core of an ISD and provide information about its structure. An AS which wants to find a path for its data sends a path-request to a core AS. It is currently possible to log the path-server's requests. This information about active paths inside an ISD is useful for the exploration phase (Subsection 3.3.1), because it enables the inspector to reconstruct the graph by adding all paths. There is no need to implement a graph exploring algorithm.

Based on that difference, the Core is a good place to insert the inspector. It needs to have access to the ISD and its structure and this is one of the tasks of the core. It would also be the place to enforce the policy, that the member of that ISD can be monitored and eventually be punished.

Another useful difference is that an AS creates a new border router instance for each new link. Each instance collects the traffic metrics with the library Prometheus and provides them with a simple HTTP based protocol. The URL for the call can be bound to an open and externally accessible network interface. This fact simplifies the necessary implementation for the monitoring phase (Subsection 3.3.3). It also enables the resolution of link-based traffic, because each border router is responsible for one link. The work of Foster[For17] would count the bandwidth efficiently with a very small footprint, but until the used version his changes were not merged.

The structure of SCION with ISD gives the possibility to split the responsibility for the inspection into multiple regions, for example one inspector for each ISD. This will improve the scalability of the Inspector, but implementing it as a decentralized component would go beyond the scope of this work.

The next section will give an overview about the implementation using SCION as its network.

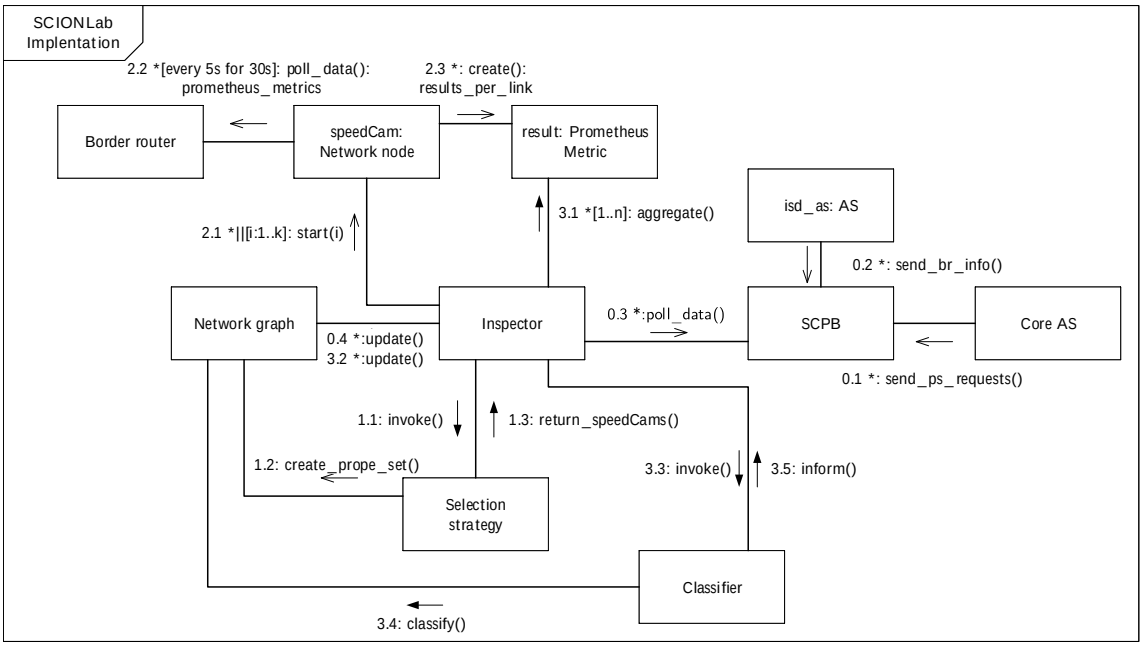


Figure 3.8: SCIONLab implementation - communication diagram

3.5 Implementation in SCIONLab

The implementation[3] was mostly written in Go[4] 1.9. SCION itself and most of its components are written in Go and because of that it made sense to also write the implementation in Go. The amount of components very specific to Go is limited and was written in an object-oriented way to be portable to other programming languages. It also uses as few SCION dependencies as possible to be lightweight and flexible to use. The implementation is sufficient enough for this work and should not be used in its current state in a production environment, but it can be used as a base for a production ready application. The documentation for this program can be found in the README.md of the project as also in Section B.

An overview about the communication between the implemented components and their corresponding phases is given as an UML communication diagram in Figure 3.8. Each phase and its specific communication is numbered, starting with 0 for the exploration phase. This phase is completely executed in the background. The other phases, methods starting with 1, 2 and 3 are run in sequence. In the following of this part, each method will be explained in detail and any difference from the concept, if existing, is mentioned.

The Prometheus metrics are published via a REST resource. Therefore, the inspector needs to know the URL of the border router. This information is stored on the AS itself and not externally available for security reasons. The solution for this problem was to create two components. The first one is a script written by Kwon, Jonghoon to parse the border router configuration file and extracts the Prometheus URL. The script itself is deployed to each AS. This information is send to the second component, the

[3] https://github.com/Meldanor/SCIONLab_SpeedCam, 05.04.2018
[4] https://golang.org, 05.04.2018

Speed**C**am**P**rometheus**B**ridge (SCPB) server program[5]. This in Java written application is a RESTful service to receive and serve information about Prometheus clients. It was originally written to add monitoring targets to the local running Prometheus server. In the progress of this thesis it was extended to not only handle Prometheus client information, but also temporary store Path server requests and make them available to the inspector. This behaviour is shown in Figure 3.8. (0.1) The script is sending the border router information, (0.2) another script[6] parses the path server requests on a core AS and both send this data to the REST resources of SCPB. (0.3) The inspector polls this data in an interval to (0.4) update his network graph. The exploration phase is done asynchronously and repeats every minute.

The selection phase was implemented as described in Subsection 3.3.2. The inspectors starts an episode and (1.) invokes the given selection strategy. Depending on the strategy it (2.) creates a set of probes by selecting the best candidates. The candidate score is missing the *capacity* information about a link. There is an attribute assigned by SCION, but in SCIONLab it was set inconsistently and therefore was unreliable. Currently, SCION itself does not make use of this attribute. In conclusion the capacity is for each node 1 and the *activity* is simply the average bandwidth of the link. The order of the candidates, as described in Section 3.3.2 does have an impact on the chance to be selected. This implementation uses the random order instead of the sorted, because there is no standard implementation of Go maps, which are sorted by there keys [Goo13b].

With the set of ASes to probe, (2.1) the inspector will construct SpeedCams and pass them the information for their border router to fetch data from. This starts the monitoring phase (Subsection 3.3.3). (2.2) The SpeedCams run simultaneously in separate Goroutines for 30s each and requests data every 5s seconds. The URL of the request is `http://[BORDER_ROUTER_IP]:[PROMETHEUS_PORT]/metrics`. The respond of the resource is a text formatted file forming a key-pair list. For the border router, the keys *border_input_bytes_total* for the ingress traffic and *border_output_bytes_total* for the egress traffic of this border router. Each value is a cumulative value continuously increasing over time and stored as a 64 bit floating point value. The bandwidth, bytes per second, is differentiated by using calculating the difference quotient (2.3). The arithmetic error can be reduced by narrowing down the scrap interval, but this would increase the necessary computation power and, due to more network called, also the used bandwidth.

For the conclusion phase (Subsection 3.3.4) the inspector needs to aggregate the data (3.1). The recorded bandwidth per link is averaged and added to the activity of both nodes of the link (3.2). The classifier is invoked (3.3) by the inspector with the updated graph. It starts to classify the nodes of the network based on the new data (3.4) and inform the inspector about it. For the evaluation of this work, the results of this episode are written to a JSON file. It contains the graph in its current state, the configuration of the program and the list monitored results per link. This ends the conclusion phase and the episode. The next episode is started by the inspector, depending on the selected strategy as described in Subsection 3.3.5.

[5] https://github.com/Meldanor/SCPB, 05.04.2018
[6] https://github.com/Meldanor/SCIONLab_SpeedCam/blob/master/ps_request_parser/ps_request_parser.go, 05.04.2018

3.6 Summary

This chapter introduced the SpeedCam approach based on the inspection game. At first a general concept for an efficient, decentralized monitoring was given, which can be applied to any kind of network. It was split into five phases to construct the network graph, to select probes, to monitor the traffic on the probes and to classify the nodes. Each of the phases can be executed by different strategies, which were also discussed. The general concept was adapted to SCION and implemented in SCIONLab for the evaluation of the discussed strategies. This will be shown in the next chapter.

4 Evaluation

This chapter contains the results of the experiments from the different strategies and methods as discussed in Chapter 3. The experiments shall show if the postulated SpeedCam approach achieves the goals (Section 1.3). At first, the test environment is described so that the experiment are repeatable and the results are verifiable. Then it list the different experiment configurations and lastly their results, which will be interpreted and discussed.

4.1 Test environment

The test environment was the SCIONLab network after its major update on 03.04.2018 and was monitored for 48 hours. The network is changing over time, because users can create a SCIONLab VM and automatically attach themselves to an existing end point such as AS 1-17. This enables the user to try out SCION in a virtual environment. A snapshot of the network can be seen in Figure 4.1, which is the topology after a few exploration episodes. It does not represent the complete topology, only the current observed one.

The inspector program is running on the same machine as the SCPB, while the scripts for information about border router interfaces and path server requests are running on different machines. The inspectors ran on a virtual machine on an Intel i7@3.75 GHz with one assigned core, 1GB of RAM and using a 52GB SSD. On this VM ran also the Prometheus server in version 2.2.1 and a Grafana[1] in version 5.0.1 for visualizing the network traffic. One problem with that test environment was that there weren't border router information for all nodes in the network. This results in fewer usable nodes than there were ASes as seen in Figure 4.1, which results in a lower precision. This has to be considered in the interpretation of the results.

4.2 Experiment

The experiment was ran on the SCIONLab network over 48 hours and ran multiple configurations of the SpeedCam program simultaneously. The script and the start parameters can be found in Section A. and differ from the standard configuration in its each instance. The standard weights all criteria equally with 1, start an inspection every 10 seconds and uses 20% of the network nodes. They were set intuitively while developing the software and serve as a baseline. This configuration used a history of six episodes for each node.

[1] https://grafana.com/, 18.04.2018

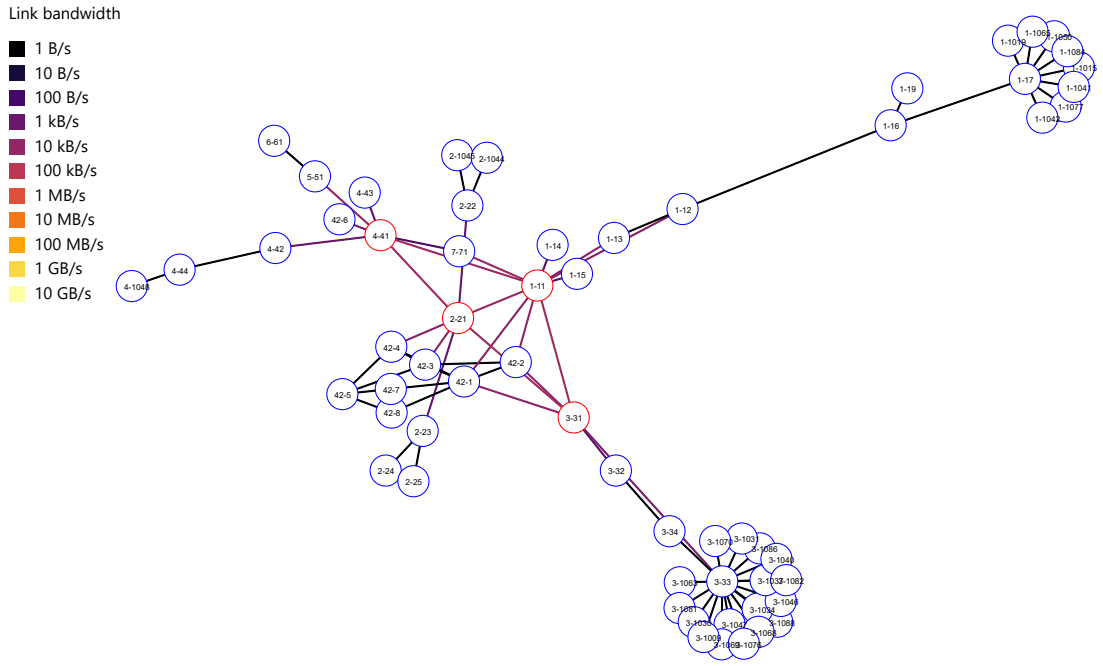


Figure 4.1: Selection result of one episode of *Linear02*. The red circles are SpeedCams.

The other configurations are:

- *Const* uses only one node as a SpeedCam and does not scale with the network. This shall show the inefficiency of such strategy for dynamic, unexplored networks like SCIONLab.
- *Linear01* uses 10%, *Linear033* 33% of the network nodes and will be used to evaluate the different scale methods. *Linear02* uses the standard values with 20% of the ASes. The naming convention is based on the factor, Linear01 conforms to $0.1 \cdot n$ and the same for the other configuration.
- *Log* uses a logarithmic scale with the base of 10 to calculate the amount of SpeedCams. As already discussed it is expected to result in a lower precision than the linear scaling methods.
- *Random* waits a random time between an inspection to start a new one. The random interval is between 10 seconds (default) to one hour. In contrast to this *fixex* waits always 60 seconds before starting a new inspection.
- *Experience* uses the strategy of using past inspections to predict the next one. *Experience12* uses twice as much episodes (total of 12) and *Experience03* uses half as much with a total of only 3 episodes. The experience strategy starts with the random strategy until there are 5 data points.

Each instance writes the result of an inspection to a JSON-file containing the complete network graph with past episodes and also the result of the inspection run.

A Prometheus server is additionally running to provide a baseline of the measured traffic inside the network. This instance fetch data from all border routers in an interval of 15 seconds. The difference to the inspector is that the Prometheus server fetches always the metrics. The data will be used to calculate the precision of the monitored data with Equation 3.1, where traffic(N) is equals to the Prometheus server data.

Because of the missing or irregular set capacity information a congestion could not be detected, but can only be guessed. This results in unverifiable cases of Table 3.2. Instead the hit rate of finding maximums is used to calculate the quality of a configuration. The result list of monitored ASes is sorted by their recorded bandwidth for both the inspector and the Prometheus server. When the inspector and the Prometheus Server have the same maximum, the inspection is considered a hit. To soften the possible error of differentiating the bandwidth at different points of time, the maximum is spanned until the third position. If the Prometheus maximum AS is at the 2nd or even the 3rd position of the inspectors, the inspection could also be a hit, but a weaker one. This is considered by using a difference of 0, 1 and 2 from the top position and for each difference the hit rate is calculated. Beside that, when the top AS is too far away, the inspection is considered an a miss.

The performance influence of the inspector is measured with the UNIX tool *top*[2] and creates a snapshot of the memory and CPU usage of the programs every 5 seconds. It was build with Go 1.9.5 linux/amd64 without any additional flags and all configurations ran as parallel processes without influencing each other. The IO influence was not captured because the inspector writes only a JSON-file with mostly fewer than 100 KB after an inspection. It is assumed that this has no impact on the overall performance and can even be disabled in a real application.

4.3 Result

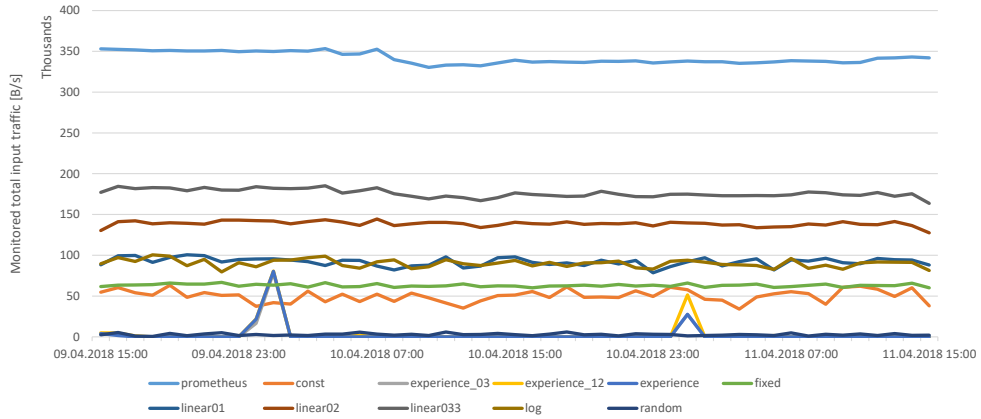
This part will list the results of the previously described configuration. Figure 4.2 displays the traffic in the time of the experiment, Figure 4.2a for the input of the interfaces and Figure 4.2b for the output of the interface. The series *prometheus* is the data from the Prometheus server and serves as the comparison. The difference in the behavior of this for the input and the output resides, that the monitored network is only a part of the SCIONLab and routed traffic flows inside the monitored than outside.

The figure even gives a good impression of the results and the influence of the configuration. The best configuration, *Linear033*, monitored only about 50% of the total traffic, while random based strategies as the *experience* ones has the worst coverage. This will be discussed in detail in Subsection 4.3.2.

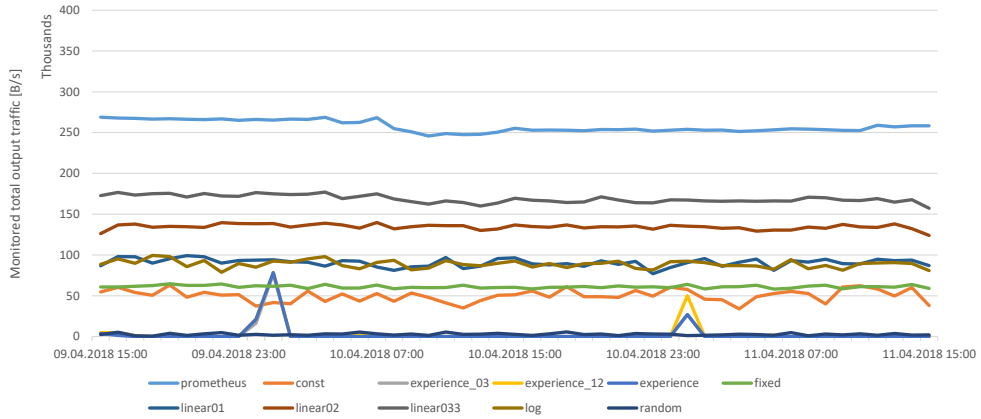
One problem with the evaluation was to have only a partition of ASes with an accessible Prometheus client interface. The ratio is shown in Figure 4.3. It displays the average amount of ASes the configuration has discovered, the candidates which has such an open client interface and the used SpeedCams per episode. *Random* has the lowest with 54 ASes, while *Fixed* has nearly twice that much of 102 ASes discovered. The reason for this is that *Random* has very few inspections and has a small chance to react to changes of the network. Additionally, new ASes were not only discovered when the path requests contained them, but also when a SpeedCam returns a border router result with an unknown AS. This happened a lot for the attachment point of the virtual machines, AS 1-17. While this did not have an influence on the candidates it will have one with an improved

[2] https://linux.die.net/man/1/top, 22.04.2018

possibility to monitor traffic. One example would be a possibility without knowing the border router's URL. Details will be listed in Chapter 5.



(a) Input



(b) Output

Figure 4.2: Monitored bandwidth in bytes per second over the experiment time. Down sampled from 15 seconds to 1 hour.

4.3.1 Selection heuristic

The result of a single selection can be seen in Figure 4.1 and its zoomed Figure 4.4. The red circles are the SpeedCams and it is shown in the figures that the inspector prefers the Core ASes. They are having a high degree but also a high bandwidth usage, because every inter-ISD traffic is routed by them.

4.3.2 Precision

The precision is the proportion of the real traffic by Prometheus to the monitored by the inspector. The higher the precision is, the better is the inspection. It is split into the monitored input and output traffic and shown in separate graphs. The results are in percentage in Figure 4.5 where higher is better, but also in total difference in Figure 4.6 where lower is better.

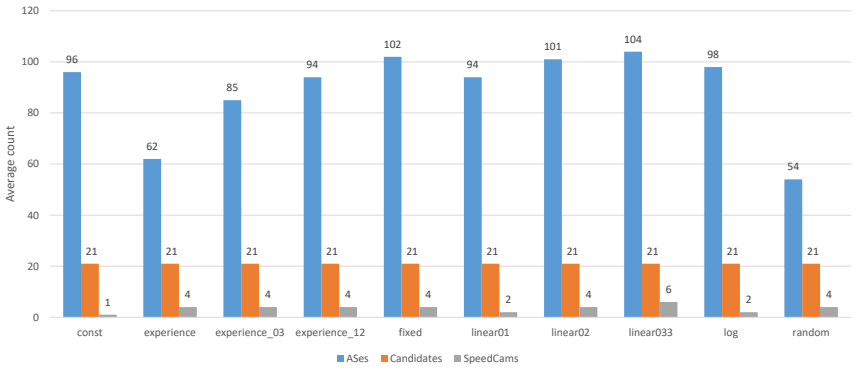


Figure 4.3: Average amount of discovered ASes, usable candidates and the used SpeedCams per configuration

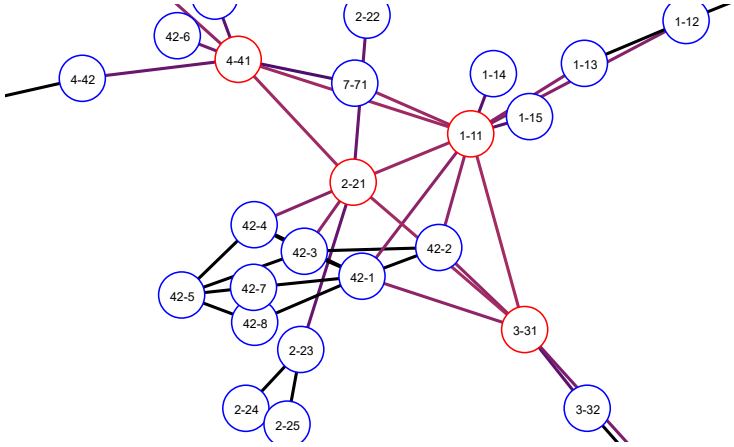


Figure 4.4: Zoomed version of Figure 4.1

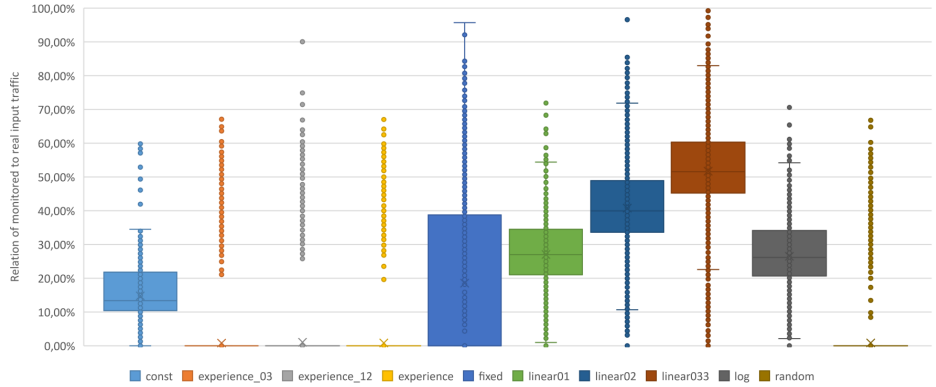
Surprisingly the precision for the experience based strategies are zero in average, what means that the experience strategy do not cover any traffic in the network. The same result applies to *random* and they are linked to each other. The experience strategy needs to gather a base of 5 data points and uses the random strategy for it and inherits the same result. Two possible solutions for this would be to use another training strategy or use a longer experiment duration instead of only two days. Alternatives are discussed in Chapter 5. The left interval based strategy *fixed* has a better result, but to far away from an acceptable distribution. It covers only 20% in average and the very high variance are indicators, that a fixed inspection interval of one minutes can lead to many false-true classification because of low monitored traffic.

The other strategies are using an interval of 10 seconds between an inspection and vary in their amount of utilized SpeedCams. *Const* brings the worst results with a coverage of 10-20%, but this is impressive for using only one node. It selected the core ASes *1-14* 1564 times, *3-31* 1019 times, *4-41* 541 times and *2-21* 381 times. Each of them is connected with the other ISDs and have high traffic, which is why there are selected so often. The *Const* strategy uses only 1 of 96 nodes to measure the traffic with a precision of 20%.

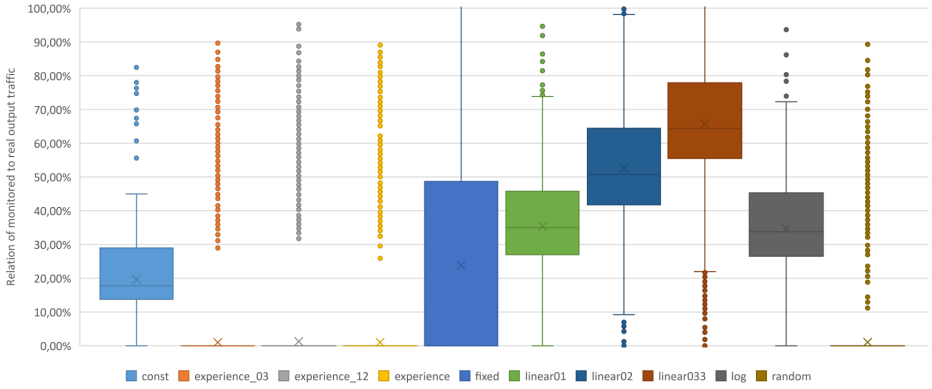
The *Log* uses only one to two nodes more with an increase of 10% points. A very similar picture is given by the *Linear01* scaling that happens because of the previously described problem with only 21 usable probes in the network. For that small number the scales

differ only in on or two nodes. The other linear scales achieve a higher precision by using more nodes. Interesting is *Fixed* and the linear configuration were able to monitor sometimes about 90 to 100% of the network with a very small amount of nodes. The reason for that phenomena is that the selection consisted of very good probe points as shown in Subsection 4.3.1.

The same results can be viewed with the total difference in bytes per second in Figure 4.6.



(a) Input



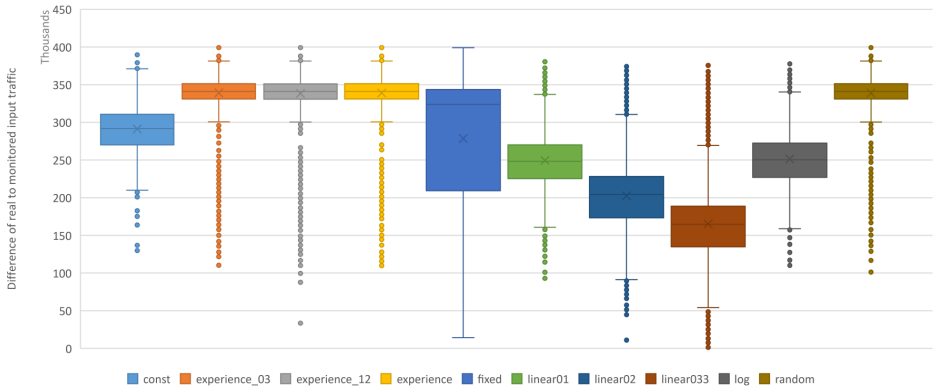
(b) Output

Figure 4.5: Distribution of percentage difference of real to monitored traffic

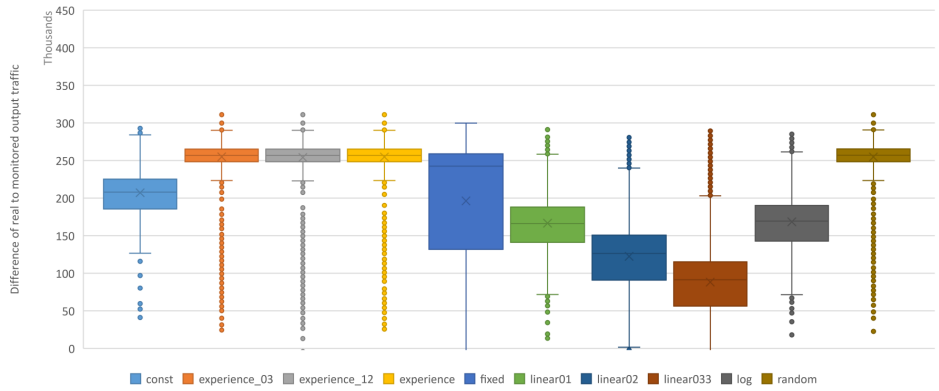
4.3.3 Hit and miss rate

The hit and miss rate is an important criteria to rate a strategy. A high precision but a low hit rate indicates a bad decision of monitoring ASes. As previously discussed the hit rate will be examined with three different distances. The higher the distance the higher the error of a miss classified AS. The results are displayed in Figure 4.7. Each diagram is split in the hit rate for the in- and the output because of the gap in the values as shown in Figure 4.2.

The first observation of Figure 4.7 is that the hit rate is different for the input and the output estimation of the inspector. The exact reason for this phenomena is unclear and needs



(a) Input



(b) Output

Figure 4.6: Distribution of total difference of real to monitored traffic

further investigations. An educated guess would be that because of the smaller gab between the real traffic and the monitored on the output interfaces (see Figure 4.2b) results in a smaller error in identifying the maximum bandwidth ASes.

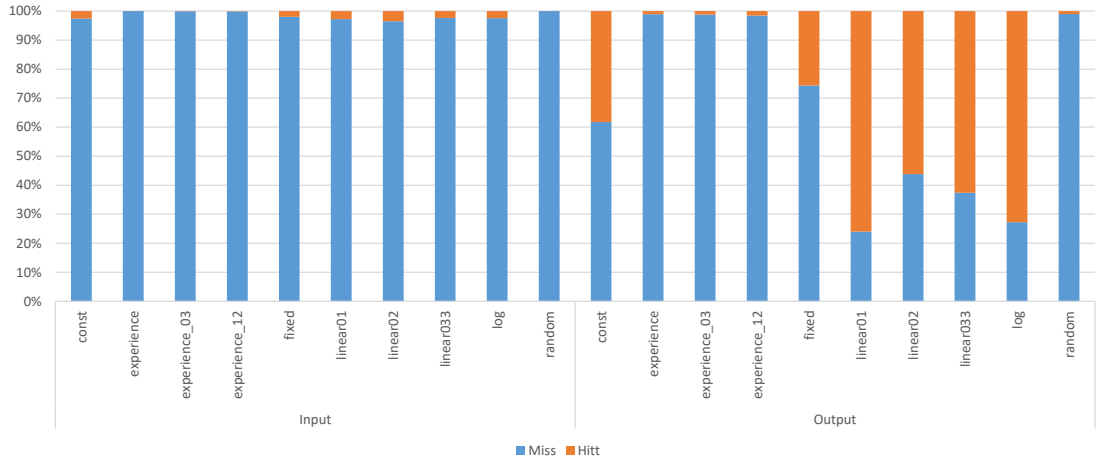
The hit rate of an exact hit is for the input so low that every configuration identified the wrong AS as the one with the most bandwidth usage. The result for the output is much better, so that the fixed interval based configuration, the *linear* ones, *const* and *fixed* itself, had an exact hit at 25% to 78% of the time. The difference between *fixed* and the one is that it uses a much longer interval between inspection (60s instead of 10s). This results in much more misses over the time because it doesn't run an inspection when Prometheus captures the traffic every 15 seconds. As seen in the different results of *const* with only one SpeedCam and the other scaling strategies the amount of probes does have a positive impact on the hit rate.

The hit rate increased when extending the acceptance range. Instead of an exact hit, a second or even third place is considered a hit. As seen for the linear configuration for the input, the increase is nearly 80% points, *const* increases by 50% points and *fixed* by 40% points. An increase, but not so high, is seen for the output guess. An increase of the hit distance results in a correct guess for 50% off the time for all but one not random based strategy. These are increasing, but only by a small amount.

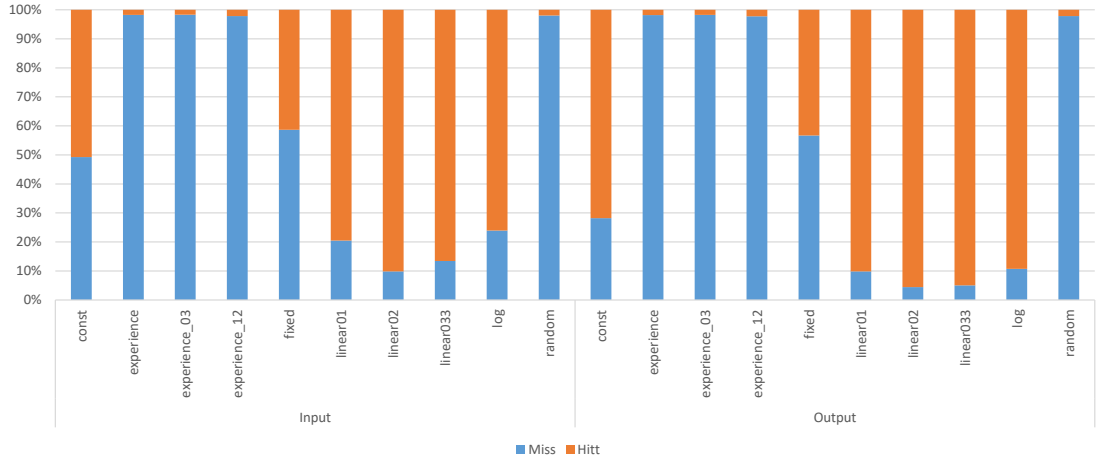
Finally, the hit rate increases even further by extending the range by another 1. The linear configurations are now guessing the correct AS in 95% of the time. This means, that the

inspector with an interval of 10s and an amount of 2-6 probes in a network of about 100 ASes had identified the correct top three bandwidth user. Even using only one probe as seen in *const* results in a high hit rate with that wide range.

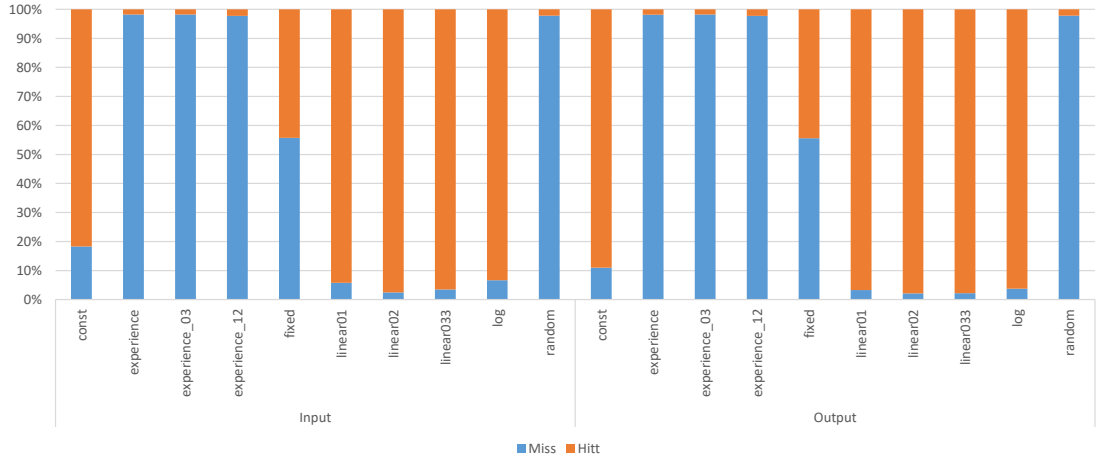
All in all, it can be assumed that the hit rate depends primarily on a small inspection interval and secondarily on the amount of used probes.



(a) Distance 0 - Exact hit



(b) Distance 1 - 1st or 2nd



(c) Distance 2 - 1st, 2nd or 3rd

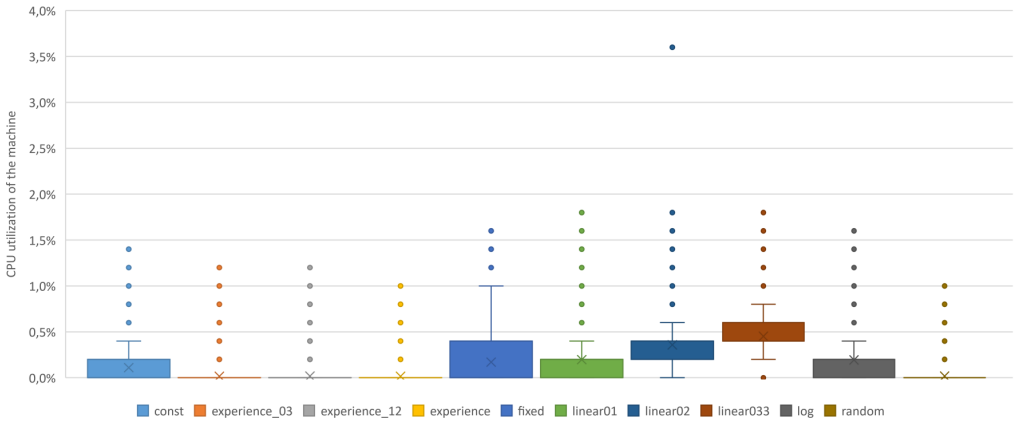
Figure 4.7: Hit and miss rate of the inspection result over time. A hit occurs when the inspector top bandwidth consuming AS is the same as the Prometheus one.

4.3.4 Performance

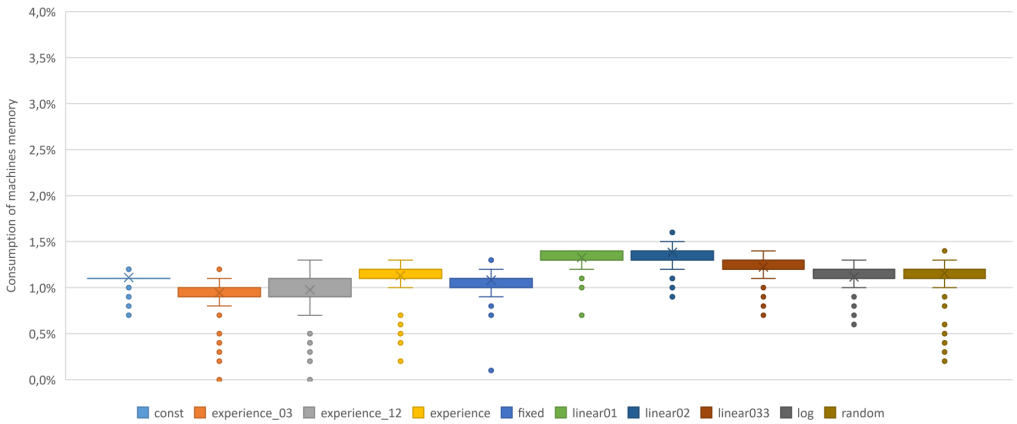
The performance impact of SpeedCam on the system is minimal as seen in Figure 4.8. Without the outliers, no configuration uses more than 1% of the CPUs computation time and uses in average fewer than 0.5%. The main impact of the CPU is the amount of utilized SpeedCams as seen in the linear configuration. The more SpeedCams existing, the more candidate scores needs to be calculated and the more monitoring results needs to get parsed. The random and the experience based configuration uses the fewest CPU in average, because most of the time they are sleeping and do not utilize the CPU.

The memory impact has a similar image as seen in Figure 4.8b. No configuration needed more than 1.5% of the memory. The amount of episodes per node has little to zero influence of the memory as seen for the experience configuration. A higher influence is the amount of used SpeedCams as seen for the *Const* configuration with always 1 SpeedCam, which uses nearly constant 1.1% of the memory.

All in all has no configuration a significant impact on the systems performance for the SCIONLab network, which was a goal of this thesis.



(a) CPU usage in %



(b) Memory usage in %

Figure 4.8: Performance impact per configuration over 48h with a snapshot every 5s

5 Conclusion

5.1 Summary

This work created an heuristic based approach to efficiently monitor traffic inside a network. The goal of the monitoring was to identify users inside it, who overuse their resources with the indent to punish them and to balance the network capacity for all user. This was especially done for the case of multi-path communication inside the SCION network structure, which allows to send data over multiple paths at the same time and to better utilize the available transfer capacity of the network.

At first the SCION network structure was described, which stands for a new structure of the Internet itself. Two already existing approaches were considered to load balance the bandwidth usage, SIPRA and the VirtualCredit. The project itself should be applicable for the test bed, SCIONLab, and for this, the SIPRA approach was considered as to complicate to implement. The latter approach was shown, that it is also to complicate to correctly realize without dead ends or possibilities to abuse the system.

A third way was chosen: To probe the networks traffic and identify the user with the most bandwidth. A requirement was to be unpredictable, otherwise the users could avoid the probes. The proposed solution should also be scalable and so uses as few probes as possible. It should also be adoptable for the networks circumstances. This approach was named SpeedCam and was in this work discussed and evaluated.

The problem of monitoring a unit, which has an interest in cheating, is not new but can be based on the Inspection Game - part of Game Theory. The general concept of SpeedCam is based on this with the inspector being the monitoring component and the users are the inspectees. Based on this game a general concept was developed and discussed, which can be applied to any type of network. The concept consists of four stages with a repeat strategy: The exploration, the selection, the monitoring and the conclusion. A sequence of them is called an episode and over time the inspector tries to learn the best positions to probe the traffic. Each stage can be done with multiple strategies which are discussed and evaluated in this work.

The general concept was adapted for the SCION structure and implemented in its test bed, SCIONLab. The monitoring was done with an established technology, the Prometheus framework. This should show the precision, the success rate of identifying a greedy user and the performance impact of the approach. The evaluation was done over 48h monitoring the traffic, where a Prometheus server captured the complete network and provided the real traffic inside the network. There were ten different configurations of the SpeedCam implementation running in parallel to show the impact of these different

parameters. A problem with this approach was the missing maximum bandwidth information and because of that the hit rate was slightly adapted to identify not a greedy user, but a maximum bandwidth usage. Another problem was that not all nodes exposed their interface to gather the bandwidth information, so that only 20% of the nodes were usable as SpeedCams.

5.2 Evaluation

The results showed that the different configurations have an influence of the hit rate, the precision and slightly on the performance itself. It was shown that the precision and the hit rate is primarily based on the inspection interval and secondarily on the amount of probes. The standard interval was 10s between an inspection and multiple configurations used this. The scaling configurations, which used only different amount of nodes but the same interval, had the highest precision and hit rat, but also the highest impact on the performance. On the contrary, the random based strategies had the worst results. The experience based one uses a random time to explore good points for monitoring, but this decision lead to a low precision and hit rate. Using a timespan of a day and a fixed interval to learn the networks behavior and then to use this experience to decide the time spots, could improve the results.

The proposed heuristic based on the networks structure and the previously recorded bandwidth is working in general, as seen with the *const* configuration with always only one probe. Its results were surprisingly and it was expected to have worse results. The hit rate for the two highest users was about 50% and it captured 15% of the networks total traffic in average. This will not scale with larger networks and traffic which resides only in parts of them. In SCIONLab, most of the traffic origins from the ISD 1 and is linked to it. But the inspector do not have to monitor the complete network as described in the general concept. Multiple inspectors can be deployed to different parts of the network, for example one per ISD, and only monitor this sub-network. This would counter the problem of the scalability.

The impact on the performance was acceptable and always under 5% of the machines resources in sight of CPU and memory consumption. This scales with the amount of utilized SpeedCams and the size of the monitored network. The performance could be even improved by optimizing the prototype, for example using a sparse graph matrix instead of a mirrored one.

Based on the evaluation, the solution achieves the **1**st goal (Section 1.3). SpeedCam is a mechanism to efficiently monitor nodes and to classify them, even in a multi-path environment. It uses a heuristic to reduce the necessary amount of nodes while achieving a high enough precision. This fulfills the **2**nd goal. The heuristic can be configured using weights and the solution contains different strategies for waiting and scalability. The prototype can be configured using them as shown in Section A. Goal **3** requires these and is therefore reached.

A general solution was firstly introduced, then specialized for the SCION environment and finally implemented and evaluated in SCIONLab. These are required for achieving the **4**th and **6**th goal. The solution can also scale with the size of the network and their impact on the performance and the detection rate was shown previously. The used protocol to

receive information about the traffic usage can improved for a better scalability. Currently every border router provides 360 lines of text information, but only five were necessary. This can affect the scalability as defined in the **5**th goal for larger networks. Because of that, this goal was only partially achieved.

All in all this work showed the use of the SpeedCam approach to efficiently monitor traffic inside a network with the capability of multi-path communications.

5.3 Future Work

There are multiple improvements of the proposed work and also next steps what to do. These ideas will be described in this section.

The most important idea would be to measure the hit rate of the inspection as proposed in the general concept. This needs the information about the capacity of a connection. It can be retrieved in SCION from the coordinator[1], but at this point of time the information was not reliable enough. A solution would be to create a protocol which is implemented by the inspector unit and all ASes via their border routers. This protocols could exchange the information about the capacity and reduce the amount of transfered data. It could also be used to issue a punishment by the inspector. It could also solve the problem with the missing exposed interface to monitor the data. This approach was considered in this work, but was discarded because it required changes on clients side which were too complicate to realize.

The punishment of the user was firstly considered to be discussed and implemented in this work, but was discarded to keep a focus on the detection mechanism of greedy users. The basic idea was to scale the punishment based on the size of the overuse. The concept is similar to speeding on the highway with a car. A slight exceeding of the speed limit only results in a fine, but a major one results in withdrawal of the license. A small overuse of the bandwidth resources would not lead to a banishment from the ISD, but only to a temporary throttling. The user also receives a long term punishment in form of points or marks. The punishment of a violation differs in the amount of marks the user has already. This is similar to the traffic regulation system in Germany[2]. The problem - and the reason for not handling it in this work - was that the realization of such a punishment was questionable. The connection and its capacity is often defined in contracts between the parties and would have to be changed to allow such a throttling or even banishment. The results of such active punishment can result in a better utilization of the networks resources while maintaining fairness among the users.

The results should be verified over a longer time, for example over a week or a month. This could show the development of the hit rate and the measure. The prototype implementation was stable for two days and could be used for longer inspection times. The used machine should have enough disk space, because the raw data of two days were 2,63 GB in size plus the Prometheus server data, which were around 8GB in size over a

[1] https://github.com/netsec-ethz/scion-coord, 15.04.2018
[2] https://www.kba.de/EN/Fahreignungs_Bewertungssystem_en/fahreignungs_bewertungssystem_node_en.html, 15.04.2018

timespan of ten days. A longer timespan will result in a higher data volume and should be considered.

As discusses in the monitoring phase the results could be improved by using a flow-based monitoring strategy. This could also cover not directly visible nodes and reduce the necessary amount of probes, but can also lead to a larger performance impact. This could be realized by using packet inspection on a low level and would require to implement a component sitting on top of the border router. It could also lead to a data privacy issue.

One issue of the current SCIONLab implementation is that it relies on the honest measurements of the nodes. A malicious member has the change to manipulate the information about the used bandwidth because the metrics are done by the client itself. Link and node based monitoring types are afflicted but that. The flow could be measured by trustworthy instances, for example a core AS. There is a need to find a solution for the other two to provide an honest and neutral measurement.

In summary, the SpeedCam approach is an expandable, but also simple to implement mechanism for networks. The results of such a monitoring can be used for further work like automatically regulate the network.

Appendices

A. Configuration

This listing contains the script to ran the different configuration in parallel.

```
#!/bin/sh
# Necessary arguments
STD_ARGS="-psUrl=http://localhost:8082/pathServerRequests -brUrl=http://localhost:8082/prometheusClient"

# Always use 1 node
CONST="-resultDir=$RESULT_DIR/const -scaleType=const -scaleParam=1"

# Linear with 0.1 as scale factor
LINEAR01="-resultDir=$RESULT_DIR/linear01 -scaleType=linear -scaleParam=0.1"

# Linear with 0.2 as scale factor
LINEAR02="-resultDir=$RESULT_DIR/linear02 -scaleType=linear -scaleParam=0.2"

# Linear with 0.33 as scale factor
LINEAR033="-resultDir=$RESULT_DIR/linear033 -scaleType=linear -scaleParam=0.33"

# Linear with 0.33 as scale factor
LOG="-resultDir=$RESULT_DIR/log -scaleType=log -scaleParam=10"

# Random wait time
RANDOM="-resultDir=$RESULT_DIR/random -intervalStrat=random"

# Fixed wait time of 60 seconds
FIXED="-resultDir=$RESULT_DIR/fixed -intervalStrat=fixed -intervalMin=60"

# Experience
EXPERIENCE="-resultDir=$RESULT_DIR/experience -intervalStrat=experience"

# Experience with 12 episodes (more)
EXPERIENCE12="-resultDir=$RESULT_DIR/experience_12 -intervalStrat=experience -cEpisodes=12"

# Experience with 3 episodes (fewer)
EXPERIENCE03="-resultDir=$RESULT_DIR/experience_03 -intervalStrat=experience -cEpisodes=3"
```

B. Usage

This section describes how the use the prototype to run the SpeedCam approach. Go 1.9 and Govendor[3] are required to ran the program.

Install

Install the dependencies via govendor (needs to be installed)

```
govendor sync
```

Run

The `core.go` contains the SpeedCam approach and can be run using a real SCION network.

```
go run core.go -psUrl=[URL] -brUrl=[URL]
```

or build with `go build core.go` and run with `./core -psUrl=[URL] -brUrl=[URL]`

Necessary parameter

This parameter are necessary for the program to run.

- `-psUrl=[URL]`, where `[URL]` points to an HTTP resource providing path server requests. Example: `-psUrl=http://localhost:8080/pathServerRequests`
- `-brUrl=[URL]`, where `[URL]` points to an HTTP resource providing border router information. Example: `-brUrl=http://localhost:8080/prometheusClient`

You can also run with the parameter `-h` or `--help` to print the help to console.

Optional parameter

This are optional and will use a default value if not provided. The default value is given in for each parameter in {}.

- `-cEpisodes=[INT]{6}` - Set the amount of stored episodes(inspection cycles) before overriding old ones.
- `-cWDegree=[FLOAT]{1.0}` - Set the weight of a nodes degree for its candidate score.
- `-cWCapacity=[FLOAT]{1.0}` - Set the weight of a nodes capacity for its candidate score. Currently not supported (missing capacity info for link)
- `-cWSuccess=[FLOAT]{1.0}` - Set the weight of a nodes success to identify congestion for its candidate score. Currently not supported.

[3] https://github.com/kardianos/govendor, 16.04.2018

- `-cWActivity=[FLOAT]{1.0}` - Set the weight of a nodes activity for its candidate score. Currently simplified because of missing capacity information.
- `-verbose=[BOOLEAN]{true}` - Enables/disables additional debug information.
- `-resultDir=[String]{""}` - If existing directory, the inspector will write the results to this directory as .JSON files. No result is written per default.
- `-scaleType=[String]{"linear"}` - Scaling of how many SpeedCams should be selected. Supported: **const**, **log** and **linear**. See scaleParam for more control.
- `-scaleParam=[FLAT]{0.2}` - The parameter for the scale func. Base for **log**, factor for **linear** and the const for **const**. See scaleType for more information.
- `-cSpeedCamDiff=[INT]{0}` - Additional(positive) or fewer(negative) SpeedCam to be selected. Will be added to result of scalType.
- `-intervalStratFlag=[String]{"fixed"}` - Strategy for waiting. Supported: **fixed**, **random** and **experience**. The last one uses the random configuration if there are too few time points in history.
- `-intervalMin=[INT]{10}` - Seconds to wait at minimum till next inspection.
- `-intervalMax=[INT]{3600}` - Seconds to wait at maximum till next inspection.

C. Prometheus metric

This is an excerpt of text file containing the metrics from the border router. The original one is 367 lines in size and the excerpt is reduced to the necessary lines for this work. The lines with a beginning hash are comments and indicates the meaning of the value and the type.

```
# HELP border_base_labels Border base labels.
# TYPE border_base_labels gauge
border_base_labels{elem="br1-10-1"} 1
# HELP border_input_bytes_total Total number of input bytes received.
# TYPE border_input_bytes_total counter
border_input_bytes_total{elem="br1-10-1",sock="intf:16"} 763646
border_input_bytes_total{elem="br1-10-1",sock="loc:0"} 652487
# HELP border_output_bytes_total Total number of output bytes sent.
# TYPE border_output_bytes_total counter
border_output_bytes_total{elem="br1-10-1",sock="intf:16"} 855351
border_output_bytes_total{elem="br1-10-1",sock="loc:0"} 786221
```

Bibliography

[ACM+96] Rudolf Avenhaus, Morton Canty, D. Marc Kilgour, Bernhard von Stengel, and Shmuel Zamir. Inspection games in arms control. *European Journal of Operational Research*, 90(3):383–394, 1996. doi:10.1016/0377-2217(95)00261-8.

[BCP+17] David Barrera, Laurent Chuat, Adrian Perrig, Raphael M. Reischuk, and Pawel Szalachowski. The scion internet architecture. *Commun. ACM*, 60(6):56–65, May 2017. URL: http://doi.acm.org/10.1145/3085591, doi:10.1145/3085591.

[BRS+16] Cristina Basescu, Raphael M. Reischuk, Pawel Szalachowski, Adrian Perrig, Yao Zhang, Hsu-Chun Hsiao, Ayumu Kubota, and Jumpei Urakawa. Sibra: Scalable internet bandwidth reservation architecture. In Srdjan Capkun, editor, *Proceedings 2016 Network and Distributed System Security Symposium*, Reston, VA, 2016. Internet Society. doi:10.14722/ndss.2016.23132.

[DC18] DE-CIX. Power outage, 10.04.2018. URL: https://twitter.com/DECIX/status/983464965075013634.

[DH98] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC Editor, 1998. doi:10.17487/RFC2460.

[Dut99] Prajit K. Dutta. *Strategies and games: Theory and practice / Prajit K. Dutta*. MIT Press, Cambridge, Mass. and London, 1999.

[For17] David Forster. *Improving the Security and Resource Control of the SCION Test Infrastructure*. Master, ETHZurich, Zurich, September 2017.

[Fre67] K. Frenzel. *International Yearbook of Cartography: 1967*. George Philip, 1967. URL: https://books.google.de/books?id=TixkQwAACAAJ.

[Goo13a] Google Inc. Digital Attack Map: Top daily DDoS attacks worldwide, 2013. URL: http://www.digitalattackmap.com.

[Goo13b] Google Inc. Go maps in action: Iteration, 2013. URL: https://blog.golang.org/go-maps-in-action.

[Goo18] Google Inc. IPv6 Adaption, 18.04.2018. URL: https://www.google.com/intl/en/ipv6/statistics.html.

[Hal97] Bassam Halabi. *Internet routing architectures*. New Riders Pub, Indianapolis, Ind., 1997.

[ICA11] ICANN. Available Pool of Unallocated IPv4 Internet Addresses Now Completely Emptied: The Future Rests with IPv6. *ICANN*, 03.02.2011. URL: https://www.icann.org/en/system/files/press-materials/release-03feb11-en.pdf.

[Kot18] Sam Kottler. February 28th DDoS Incident Report, 01.03.2018. URL: https://githubengineering.com/ddos-incident-report/.

[LABJ01] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet routing convergence. *IEEE/ACM Transactions on Networking*, 9(3):293–306, 2001. doi:10.1109/90.929852.

[MG] Dennis Meyer and Kilian Gärtner. A Virtual Credit based Resource Allocation Mechanism for SCION.

[Min17] Miniwatts Marketing Group. INTERNET USAGE STATISTICS: The Internet Big Picture, 31.12.2017. URL: https://www.internetworldstats.com/stats.htm.

[N+50] John F Nash et al. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

[NOSv14] D. Nosenzo, T. Offerman, M. Sefton, and A. van der Veen. Encouraging Compliance: Bonuses Versus Fines in Inspection Games. *Journal of Law, Economics, and Organization*, 30(3):623–648, 2014. doi:10.1093/jleo/ewt001.

[Owe13] Guillermo Owen. *Game theory*. Emerald, United Kingdom, fourth edition edition, 2013.

[Par88] Craig Partridge. *Innovations in Internetworking*. Artech House, Inc., 1988.

[PSRC17] Adrian Perrig, Pawel Szalachowski, Raphael M. Reischuk, and Laurent Chuat. *SCION: A secure Internet architecture*. Information security and cryptography. Springer, Cham, Switzerland, 2017.

[Rip18a] Ripe network coordination centre. IPv6 Enabled Networks, 18.04.2018. URL: http://v6asns.ripe.net/v/6?s=_ALL.

[Rip18b] Ripe network coordination centre. RIPE Atlas, 19.04.2018. URL: https://atlas.ripe.net/.

[RLH05] Yakov Rekhter, Tony Li, and Susan Hares. A border gateway protocol 4 (bgp-4). Technical report, IETF, 2005.

[Rol09] Roland Kirstein. Doping, the Inspection Game, and Bayesian Monitoring. 2009.

[SKM06] A. Sahoo, K. Kant, and P. Mohapatra. Improving BGP Convergence Delay for Large-Scale Failures. In *International Conference on Dependable Systems and Networks (DSN'06)*, pages 323–332. IEEE, 2006. doi:10.1109/DSN.2006.41.

[SW11] Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-Wesley, Boston, Mass. and London, 4th ed. edition, 2011.

[ZDG+14] Haojin Zhu, Suguo Du, Zhaoyu Gao, Mianxiong Dong, and Zhenfu Cao. A probabilistic misbehavior detection scheme toward efficient trust establishment in delay-tolerant networks. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):22–32, 2014. doi:10.1109/TPDS.2013.36.

Statement of Authorship / Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit selbstständig und außschließlich unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.
Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder einer anderen Prüfungsbehörde vorgelegt oder noch anderweitig veröffentlicht.

Unterschrift

Datum