ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bachelor's Thesis

Network Security Group, Department of Computer Science, ETH Zurich

A Comprehensive Test Environment For SCION Control Plane Algorithms

by Lorin Urbantat

Spring 2024

ETH student ID:	20-944-062
Email address:	lurbantat@ethz.ch
Supervisors:	François Wirz Jordi Subirà Nieto Prof. Dr. Adrian Perrig
Date of submission:	August 20, 2024

Abstract

SCION (Scalability, Control and Isolation On next-generation Networks) [13] control plane algorithms have a significant effect on a variety of aspects in a network such as performance, sustainability and many more.

Selecting paths, for example, is a problem that is hard to solve in live networks as they are unpredictable and change over time. At the same time, selecting paths can have a meaningful impact on network performance and sustainability.

In order to make it easier to select the best paths, we created a test environment allowing us to run control plane algorithms in a controlled manner with varying network topologies and conditions. In this thesis, we will delve into the various components required to create such an environment, as well as the challenges we encountered during the process.

Contents

1 Introduction

1.1 Motivation

While the early years of the internet were characterized by large-scale adoption and growth at all costs, more recently, such aspects as data governance and sustainability concerns over its carbon footprint and energy-efficiency have been coming into focus. In light of these changes, the kind of paths data is routed through on the internet can play a significant part in making the internet's future more sustainable while at the same time catering to data privacy concerns.

By choosing a path wisely, one can greatly influence the carbon footprint [23] and determine which infrastructure is used. This enables new approaches to data governance and sustainable networking. Additionally, effective path selection can improve network performance by allowing paths to be chosen based on traffic characteristics. For latency-critical data, a low-latency path can be selected, while data-intensive applications can be routed via a high-bandwidth path. These approaches can even be combined, enabling applications to send their latency-critical data over a low-latency path and their data-intensive data over a high-bandwidth path. [4]

The current internet architecture is based on the Border Gateway Protocol (BGP) [1]. BGP however does not allow for endpoints to select paths. In fact, path selection is almost completely intransparent to the endpoints and endpoints cannot control the path their data takes. Even Autonomous systems (ASes) only have limited control over the path. The only aspect in their control is where to forward data to. Additionally, BGP is subject to hijacking attacks [2], further complicating the path selection. This makes it impossible to choose paths and to optimize path selection according to specific metrics.

SCION (Scalability, Control, and Isolation On Next-Generation Networks) [13] is an internet architecture that aims to solve some of the most common problems with BGP. It allows endpoints to select paths. To this end, SCION uses a separate control and data plane. SCION path segments are associated with metadata that allow path selection based on specific metrics, such as carbon footprint, latency, bandwidth, reliability, and many more. However, to enable effective path selection according to configurable metrics, capable path selection algorithms are the prerequisite. These algorithms must be able to select paths based on the desired metrics and adapt to changing network conditions. A comprehensive test environment is necessary to develop path selection algorithms and subject them to testing under various conditions. This thesis focuses on creating such a test environment.

1.2 Problem definition

In order to run meaningful experiments on SCION control plane algorithms, a test environment that allows us to repeatably run experiments under configurable network conditions is needed.

The requirements for the test environment are:

- **Repeatability:** Running the same experiment multiple times yields the same results.
- **Configurability:** Network topology as well as network conditions can be configured. Network conditions include link properties such as bandwidth and latency, as well as the network load. In this context, network load refers to the amount of data sent over the network. This might be arbitrary internet traffic or synthetic traffic generated by the test environment.
- **Scalability:** Experiments can run on large-scale networks similar to the internet.
- **Extendability:** The internet is very heterogeneous. The test environment simulates this by being easily extendable with new components.

1.3 Organization

This thesis is structured into three main parts. Section 2 provides an overview of the technologies which form the context of this thesis. Section 3, discusses how the SCION topology tool was modified to generate Seed emulations. The extensions made to the Seed emulator in order to support all the needed functionality are introduced in section 4. This thesis closes with a discussion of the traffic generator we developed to generate traffic on the emulation in section 5. Implementing these three components will not only enable SCION developers to run meaningful tests on new control plane algorithms. What's more, it will also make it easier for network operators to test SCION in a controlled environment, thereby enabling further large scale SCION adoption.

2 Background

In this chapter, we provide an overview of the different technologies that this project is based on. Further, we also share which other technologies we evaluated but then discarded for this project.

We will start with a brief section about the SCION [13] internet architecture and continue talking about three different network emulators, namely NS-3 [11], Seed [21] and Kathara [8], which we considered using as a base for this project. Further, we will discuss advantages and disadvantages of each and motivate our final decision to go with Seed [21] for this project.

2.1 SCION

SCION is a path-aware routing protocol which aims to make inter-domain packet delivery secure, flexible and highly available. In what follows, we will give a brief overview of the SCION architecture and its main components.

SCION organizes Autonomous Systems (ASes) into so-called Isolation Domains (ISDs). ASes in an ISD agree on a common set of trust roots, or Trust Root Configuration (TRC). A subset of ASes in an ISD are so-called Core ASes which manage the trust roots and provide connectivity to other ISDs.

SCION distinguishes between inter-ISD and intra-ISD routing. Both use path-segment construction beacons (PCBs) to explore paths. A PCB is created by a core AS and then disseminated within an ISD or to core ASes in other ISDs depending on whether it is used to find inter-ISD or intra-ISD paths. These PCBs accumulate path and forwarding information in their hop fields as they traverse the network. This information is cryptographically protected, allowing endpoints to construct end-to-end paths by combining path segments. These paths are included in the packet header and used by intermediate routers to forward the packet. PCBs can also carry metadata about the path, such as the available bandwidth or latency, which is especially relevant for this project.

The SCION team and community [18, 10] has also developed a number of tools. We will use and extend some of these tools as part of this project.

2.1.1 bwtester

As a part of the scion-apps [10] repository, bwtester is a tool that allows for testing bandwidth and latency on a SCION network. It is a client-server application, which helps to measure the bandwidth and latency between two nodes in the network. It establishes two SCION UDP connections between the client and the server: one control connection and one data connection. The tool allows for specifying parameters such as duration, packet size, packet count and bandwidth. These can be specified separately for the client-server and server-client path due to the path-aware nature of the SCION protocol. We will use this tool not only to measure bandwidth and latency, but also to generate network load on our network.

2.1.2 SCION IP Gateway (SIG)

The SCION IP Gateway (SIG) [17] is a tool that allows sending IP traffic through a SCION network. This is accomplished by running SIG instances in different ASes of the SCION network. The nodes establish a session between each other and then allow tunneling IP packets through the SCION network. In detail, this works by encapsulating the IP packets in SCION packets and sending them through the SCION network. The remote SIG instance then decapsulates the SCION packet and forwards the IP packet to the destination effectively, acting as an IP router. This allows for the easy integration of existing applications that use IP into a SCION network. We will use SIG instances to allow generating IP traffic in our network.

2.1.3 Topology tool

The topology tool is part of the SCION tools [19]. It allows specifying network topologies in a custom *.topo* file format. These topologies are then compiled to a Docker compose deployment, which can be run. In this project, we will extend the topology tool to support compiling *.topo* files to Seed topologies.

2.2 Seed

Seed [21] is a network emulator that can be used to create network topologies. It represents essential network elements using Python classes, including BGP, DNS and even SCION. These are then compiled to Docker deployments. One can define network topologies using the Seed Python API. When running the Python file, Seed generates all the necessary Dockerfiles and configuration files to run the network. Other functionalities are implemented as services which can be installed on nodes. A web interface shows the network topology and allows for network monitoring. Mobile devices such as LTE or 5G, and IPv6 are not supported by the Seed network emulator. However, they both would not be highly relevant for this project.

For the purposes of this project, Seed thus proved to be the best choice regarding its feature set, SCION support and flexibility. It is based on docker containers and is less complex than NS-3.

2.2.1 Architecture overview

Seed is modular, allowing it to be easily extended and modified. The Seed core implements basic elements such as nodes, links, ASes, and more. It also includes the registry, a data structure that keeps track of all the elements in the emulation. This data structure can also be dumped to a file and re-loaded, supporting emulations to be easily saved and loaded.

Layers can be used to extend an emulation with extra functionalities that are not part of the Seed core. These include a base layer, a routing layer, ospf, bgp, and many more layers. The SCION compatibility is also implemented as a layer.

Once a Seed emulation is defined, it is rendered. The rendering step first establishes the dependency graph between layers and then generates all the configuration files necessary to run the emulation. This might include all the certificates and topology files necessary to run SCION for example.

Finally, when an emulation is rendered, it can be compiled using a Docker or Graphviz compiler. Throughout this project, we have used the Docker compiler. Compiling an emulation with the Docker compiler entails generating Dockerfiles for the containers and a docker compose deployment. For Docker images to build and run successfully, all files are put be in the right place.

Further to these basic functionalities, Seed also provides a variety of services that can be installed and run on nodes. Services are applications that can run on nodes in the emulation. Such services include the SCION-bwtester, for example. A web interface offers a map view [20] of the emulation and allows interacting with the nodes via CLI.

2.2.2 Scalability

In our case, a Seed emulation consists of a set of docker containers managed by docker compose. Each node is represented by a docker container. Depending on the use case, the number of nodes can vary, but there is at least one router per AS. For SCION emulations, two nodes are needed in each AS, one router and one acting as a SCION control service. Emulating a network with a number of ASes comparable to the internet is unfeasible because the network would have to run on the order of 60,000 docker containers on a single host. Seed has a distributed Docker compiler. This would make network emulation more feasible. However, the distributed Docker compiler is not actively maintained. At this point in time, network emulation is bottle necked by the number of docker containers that can be run on a single host.

2.3 Docker

Docker [3] is a containerization platform that supports deploying applications in containers. It effectively isolates applications from the host systems and other containers. In contrast to Virtual Machines (VMs), which also provide a similar abstraction and isolation, Docker containers share the host system's kernel. This makes running containers much more resource-efficient compared to VMs. Seed uses Docker as the underlying technology to run network nodes.

2.3.1 Scalability and overhead

Docker containers are lightweight and have little overhead compared to other virtualization solutions such as Virtual Machines. Though there is still some overhead, this is why the number of containers that can be run on a single host is limited. The limitations also result from the amount of resources each container needs. For example, if a given border router has to forward a lot of traffic, it will need more resources than a container only running a SCION control service. In our test, we were able to run around 60 containers on a single host with 4 CPU cores and 8GB of RAM.

2.4 tc

tc [25] is a Linux command line tool that allows to control traffic shaping, scheduling, policing and dropping in Linux. It has three different objects, qdiscs, classes, and filters.

Qdiscs or queuing disciplines, are associated with a network interface. When the kernel wants to send a packet to the network interface, it enqueues the packet in the qdisc and then gets as many packets as possible from the qdisc and hands them to the network interface driver. tc has a variety of different qdiscs, such as pfifo, bfifo, tbf, htb, fq_codel. Each qdisc has different properties and supports different use cases.

Qdiscs may contain classes. Classes are sub-queues of the qdisc. Traffic may be enqueued and popped from any of the sub-queues. One use case for a qdisc is a qdisc that prioritizes certain traffic over other traffic.

To categorize packets, qdiscs utilize classes and filters. These filters determine the appropriate class for each packet before it's placed in a queue. There is a wide range of filters, each tailored to specific use cases.

With these three mechanisms, tc allows for very fine-grained control over the network traffic. We will use it to simulate bandwidth limitations, latency cosntraints, and packet loss on our network links.

2.5 Related works

This section details related works that we considered to base this project on. Specifically NS-3 [11] and Kathara [8] are two network emulators that we considered using as a base for this project, instead of Seed. We decided for Seed because it best supported SCION and offered more flexibility being based on docker containers.

2.5.1 NS-3

"NS-3 is a discrete-event network simulator for internet systems, targeted primarily for research and educational use." [11]

In NS-3, the network topology is emulated completely. There are no nodes or links. Parts of the network are implemented as C++ objects. On the one hand, this allows for a detailed and fine-grained control of the network. On the other hand, applications need to be implemented on top of the NS-3 API. This makes it harder to use exisiting applications on top of NS-3 emulations especially compared to Seed [21] and Kathara [8]. This is because Seed and Kathara use docker containers to represent nodes which makes it easier to run existing applications on top of the emulations.

NS-3 has many existing modules for different network protocols and technologies. Such modules include WI-FI, LTE, CSMA, IP, TCP, UDP [12], and many more. They can be used to build and simulate complex network topologies and scenarios. Our use case builds on the support for SCION, albeit the support for SCION was not yet merged into the main project [24] at the time of writing. On top of that there is NetAnim which allows for the visualization of the network topology and the packets being sent.

2.5.2 Kathara

Like Seed, Kathara also uses docker containers to represents nodes in the network. Kathara further supports Kubernetes, which allows for the easy deployment of complex network topologies that cannot easily run on a single host using docker containers. Additionally, Kathara tools help generate network topologies or visualize the network [9]. However, at the time of writing this thesis Kathara did not have support for SCION or LTE. This was prohibitive for this project given that SCION support was critical. Implementing full SCION support was out of scope for this project.

2.5.3 Scapy

Scapy[14] is a packet manipulation library. It has a variety of functionalities to create, send, sniff and dissect packets with various protocols including SCION. When designing traffic and packet generation for this project, we considered Scapy[14]. Since Scapy[14] requires the manual creation of each packet that is put on the network, we had to disregard it. Creating packets manually theoretically provides a lot of flexibility in traffic generation. The complexity, however, was prohibitive for this project. The decision was to build a solution on top of other tools such as iPerf3[6] and the SCION-bwtester[15].

3 Topology tool extensions

The topology tool allows users to specify network topologies in a *.topo* file and then generate a docker compose environment from this file. We want to enable the topology tool to generate Seed emulations from *.topo* files. This requires some modifications to the topology tool.

3.1 Desired properties

The topology tool receives an input file in the *.topo* format.

Example file:

```
# Tiny Topology, IPv4 Only
ASes:
"1-ff00:0:110":
    core: true
    voting: true
    authoritative: true
    issuing: true
    mtu: 1400
"1-ff00:0:111":
    cert_issuer: 1-ff00:0:110
"1-ff00:0:112":
    cert_issuer: 1-ff00:0:110
links:
- {a: "1-ff00:0:110#1", b: "1-ff00:0:111#41", linkAtoB: CHILD}
- {a: "1-ff00:0:110#2", b: "1-ff00:0:112#1", linkAtoB: CHILD}
```

The file contains a list of ASes and a list of links. Each AS includes a set of properties specifying its parameters. Each link specifies the ASes it connects, the direction of the link, and the properties of the link. All of this information needs to be included in the Seed emulation.

However, the *.topo* file format does not include the key link properties needed for our use case: *latency*, *bandwidth*, and *loss*. What's more, we want to include metadata associated with the border router in the SCION beacons while running the Seed emulation. This will allow us to associate metadata with SCION path segments which gives clients more information about the paths making path selection based on metadata possible.

3.2 Implementation

3.2.1 Extending the topology file format

To include the desired properties in the *.topo* file, we extended the file format.

First, we added properties to both the ASes and the links. These include latency, bandwidth, and packet loss for the links between and within ASes.

Next, we added an optional border router property section to the *.topo* file. This section allows users to specify metadata, such as geographic location and a note that is considered by the border routers.

3.2.2 Modifying the topology tool

Following the amendments to the *.topo* file format to support Seed emulations, we modified the topology tool. The topology tool is written in Python and follows common object-oriented programming practices. Consequently, we wrote a `SeedGenerator` class, using the *.topo* file as input and return a Seed emulation. Then, we extended the existing topology tool to use this `SeedGenerator` class. The `SeedGenerator` class works by parsing the *.topo* file and then generating Python code which can be executed to generate the Seed emulation.

An alternative approach would have been to use Seed's Python API directly to generate the emulation instead of generating Python code. This could have resulted in more readable code. The downside would have been an unmodifiable Seed emulation. The chosen implementation allows the Seed code to be modified to fit other components.

Example

```
--- # Tiny Topology with border router properties
ASes:
  "1-ff00:0:110":
    core: true
    voting: true
    authoritative: true
    issuing: true
    mtu: 1400
  "1-ff00:0:111":
    cert_issuer: 1-ff00:0:110
  "1-ff00:0:112":
    cert_issuer: 1-ff00:0:110
links:
  - {a: "1-ff00:0:110#1", b: "1-ff00:0:111#41", linkAtoB: CHILD, mtu: 1280}
  - {a: "1-ff00:0:110#2", b: "1-ff00:0:112#1", linkAtoB: CHILD, bw: 500}
borderRouterProperties:
  "1-ff00:0:110#1":
    geo:
      latitude: 48.858222
      longitude: 2.2945
      address: "Eiffel Tower\n7th arrondissement\nParis\nFrance"
    note: "Hello World"
```

In this *.topo* file, there are three ASes: 110, 111, and 112. These are connected by two links. Using the topology tool on this *.topo* file results in Python code:

```
from seedemu.compiler import Docker, Graphviz
from seedemu.core import Emulator
from seedemu.layers import ScionBase, ScionRouting, ScionIsd, Scion, Ospf
from seedemu.layers.Scion import LinkType as ScLinkType

# Initialize
emu = Emulator()
base = ScionBase()
routing = ScionRouting()
scion_isd = ScionIsd()
scion = Scion()
ospf = Ospf()

# Create ISDs
base.createIsolationDomain(1)
```

```
# Ases
# AS-110
as110 = base.createAutonomousSystem(110)
scion_isd.addIsdAs(1,110,is_core=True)
as110.createNetwork('net0', prefix="10.4.110.0/24")
     .setDefaultLinkProperties(latency=0,bandwidth=0,packetDrop=0)
     .setMtu(1400)
as110.createControlService('cs_1').joinNetwork('net0')
as_110_br1 = as110.createRouter('br1').joinNetwork('net0')
as_110_br1.setGeo(Lat=48.858222, Long=2.2945, Address="""Eiffel Tower
7th arrondissement
Paris
France""")
as_110_br1.setNote('Hello World')
as_110_br1.crossConnect(111,'br1','10.3.0.2/29',
                        latency=0,bandwidth=0,packetDrop=0,MTU=1280)
as_110_br2 = as110.createRouter('br2').joinNetwork('net0')
as_110_br2.crossConnect(112,'br1','10.3.0.10/29',
                        latency=0,bandwidth=500,packetDrop=0)

# AS-111
as111 = base.createAutonomousSystem(111)
scion_isd.addIsdAs(1,111,is_core=False)
scion_isd.setCertIssuer((1,111),issuer=110)
as111.createNetwork('net0', prefix="10.4.111.0/24")
     .setDefaultLinkProperties(latency=0, bandwidth=0, packetDrop=0)
as111.createControlService('cs_1').joinNetwork('net0')
as_111_br1 = as111.createRouter('br1').joinNetwork('net0')
as_111_br1.crossConnect(110,'br1','10.3.0.3/29',
                        latency=0,bandwidth=0,packetDrop=0,MTU=1280)

# AS-112
as112 = base.createAutonomousSystem(112)
scion_isd.addIsdAs(1,112,is_core=False)
scion_isd.setCertIssuer((1,112),issuer=110)
as112.createNetwork('net0', prefix="10.4.112.0/24")
     .setDefaultLinkProperties(latency=0, bandwidth=0, packetDrop=0)
as112.createControlService('cs_1').joinNetwork('net0')
as_112_br1 = as112.createRouter('br1').joinNetwork('net0')
as_112_br1.crossConnect(110,'br2','10.3.0.11/29',
                        latency=0,bandwidth=500,packetDrop=0)
```

```
# Inter-AS routing
scion.addXcLink((1,110),(1,111),ScLinkType.Transit,
                a_router='br1',b_router='br1')
scion.addXcLink((1,110),(1,112),ScLinkType.Transit,
                a_router='br2',b_router='br1')

# Rendering
emu.addLayer(base)
emu.addLayer(routing)
emu.addLayer(scion_isd)
emu.addLayer(scion)
emu.addLayer(ospf)

# dump seed emulation to file before rendering
emu.dump("gen/scion-seed.bin")
emu.render()

# Compilation
emu.compile(Docker(internetMapEnabled=True,
                   internetMapClientImage="bruol0/seedemu-client"),
            './gen/seed-compiled')
```

3.3 Limitations

When implementing the `SeedGenerator` class, we ran into two limitations.

Firstly, Seed did not support several connections between different border routers of the same ASes (see figure 3.1). Notice how in figure 3.1, AS 111 and AS 110 are connected by two different links. This was not supported by Seed at the time of writing this thesis. While this is possible in the *.topo* file format, we would like to support several connections between different border routers of the same ASes in Seed as well. We extended the Seed emulator to support this. For more detail, please refer to the section 4.

Secondly, Seed does not support IPv6 at the moment. Since this is not essential for our use case, we excluded IPv6 from the topology tool, meaning that *.topo* files with IPv6 addresses are not supported. Nonetheless, we added a feature flag that allows users to treat *.topo* files with IPv6 as if they were IPv4.

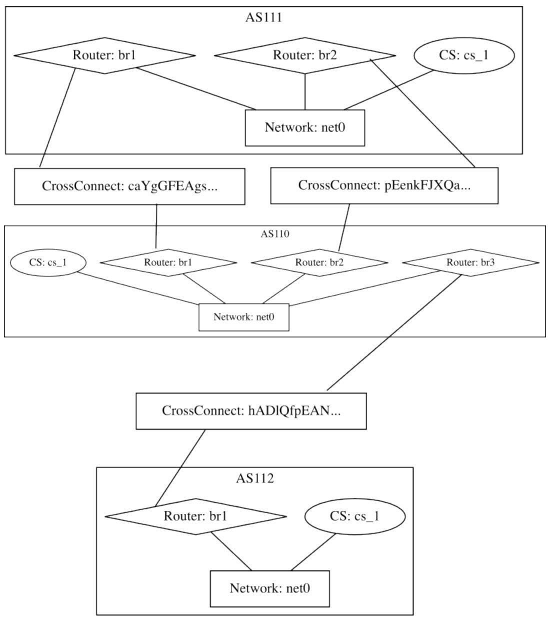


Figure 3.1: Topology that was previously impossible in Seed

4 Seed extensions

To allow our test environment to support the full range of topologies, we needed to extend the Seed emulator. This included extending the SCION routing, adding support for static and dynamic link properties, as well as adding metadata to the SCION beacons. This chapter discusses the necessary properties of our test environment and how we implemented them.

4.1 Test environment properties

SCION routing

In the Seed emulator, we require the SCION routing to support all topologies that can be defined in a *.topo* file. At the time of writing this, Seed only supports a subset of the possible topologies. To date, it is impossible to define topologies with multiple links between different border routers of the same two ASes. We will extend Seed to make the SCION routing as flexible as the *.topo* files allow.

Link properties

Our test environment should emulate link properties such as link delay, bandwidth and packet loss. Ideally, it should be able to statically define these properties for each link in the topology as well as change them dynamically while the test environment is running. While Seed already supports setting link properties, it is not as versatile as we need it to be. For instance, it is not possible to set link properties for cross-connect links. Moreover, changing link properties dynamically is not supported either.

SCION beacon metadata

All available SCION beacon metadata such as geographical location, link properties and notes about the border router should be added to the SCION beacons. The beacon metadata will enable path selection algorithms on the client to make more informed decisions about which path to take.

SCION IP Gateway support for Seed emulation

Real-world SCION networks often exist alongside BGP networks. Therefore, they can tunnel IP traffic through a SCION IP Gateway. To emulate a real-world network and tunnel IPv4 traffic through a SCION IP Gateway, it needs to be supported in Seed.

4.2 Implementation

4.2.1 Make the SCION routing more flexible

First, we needed to make the SCION routing more flexible by connecting specific border routers of ASes to specific border routers of other ASes. This will allow us to support all topologies that can possibly be defined in a *.topo* file.

Up to this point, it was possible to specify the ASes that were connected to each other. It was not possible to ascertain through which border routers they were connected. It was implementated as follows: In Seed, users specified a route by defining which ASes should beconnected to each other. Then, Seed dynamically selected the corresponding border routers that had a link to the other AS. Seed used these border routers to create the SCION routes. If there were several applicable border routers, the choice was arbitrary and depended on the order in which the border routers were stored in Seed's data structure.

In order to make the routing as flexible as we needed, we decided to add support for specifying which border routers were supposed to be connected. Specifying this is optional and if they are not specified, Seed falls back to the default behavior. This ensures backwards compatibility with Seed emulations that do not specify the border routers. This enables all the topologies that can be defined in a *.topo* file while preserving backwards compatibility.

4.2.2 Static link properties to links

As described in section 4.1, we want to define link properties such as link delay, bandwidth and packet loss. Ideally, Seed should be able to statically define these properties for each link in the topology as well as change them dynamically while the test environment is running.

Statically versus dynamically defining link properties requires different approaches. This results from the way Seed defines emulations and then compiles them to Docker compose deployments. Statically defined link properties are set before rendering and compiling the Seed emulation. The static link information is included in SCION beacons because it is known at the time of rendering. The SCION configuration files, including the *staticInfoConfig.json* file containing the metadata carried by the SCION beacons, are generated at the time of rendering. By contrast, dynamically defined link properties are set while the emulation is running and after the Seed emulation has been compiled to Docker compose deployments. As a result, the dynamic link information is not included in the SCION beacons because it is not known at the time of rendering the Seed emulation.

In order to support link properties in Seed, a lot of the work was already done because the underlying network class which represents links in Seed already has the necessary functionality.

In contrast to other link types in Seed, the networks representing cross-connect links are created at the time of rendering rather than before. This makes the network object inaccessible while defining the emulation, meaning that adding the capability to specify link properties for cross-connect links was outstanding. We added the link properties to the data structure which represents the cross-connect links for a given node. Subsequently, we set these properties on the network object, that is created during rendering.

Since all other links, such as AS-internal networks or InternetExchanges already allowed setting link properties on the underlying network objects, no further amendments were necessary.

The link properties that are set on the Seed network class are then compiled to a script that uses tc [25] to set the link properties on the nodes connected to a given network. Further, the link properties are also included in the SCION beacons as far as they are known at the time of rendering the Seed emulation. For further details on this, consider 4.2.3.

4.2.3 SCION path metadata in beacons

To add metadata to the SCION beacons, we need to include a file called *staticInfo-Config.json* on the control service of each SCION AS. The file contains the metadata to be included in the beacons [16].

First, we extended Seed to support all kinds of metadata that can be added to SCION beacons to date. We added support for the following metadata:

- AS note
- Border router geographic location
- Border Router note

What was left for us was to add the appropriate fields to the Seed classes representing the SCION ASes and border routers.

Other metadata, such as bandwidth, packet loss, and latency were already supported or added in the previous sections.

After we successfully implemented SCION beacon metadata support in Seed, we created the *staticInfoConfig.json* file for each SCION AS. We added a step during the Seed rendering process which collected all the information necessary and then created the file. One limitation encountered in this approach is that AS-internal link properties such as latency and bandwidth depend on the AS-internal routing and thus cannot always be known statically for all topologies. The way we alleviated this problem was by constraining inclusion of the metadata only to topologies where each pair of border routers within the same AS is always connected directly through a network. Then there is never more than one hop between two border routers of the same AS. This ensures that we can always statically determine the link properties between two border routers of the same AS.

4.2.4 SCION IP Gateway support for Seed emulation

Unlike previous extensions (beacon metadata 4.2.3, static link properties 4.2.2), which only required AS-level configuration changes, supporting the SCION IP Gateway (SIG) necessitates both AS-level modifications and running SIG instances on designated endpoint nodes.

We proceeded as follows: Firstly, we need to run an instance of the SIG on a node in each of the two ASes we wanted to connect. Secondly, we need to add the SIG configuration to the `topology.json` file in each AS.

To add the SIG configuration to the `topology.json` file in each AS, we modified the `ScionAutonomousSystem` class in Seed because this is where the `topology.json` file is generated.

We add a new `sig_configuration` field to the `ScionAutonomousSystem` class which contains the configuration for the SIG. We can then include this configuration in the `topology.json` file during the rendering step.

To run SIG instances in the ASes, we can make use of services. In Seed, services can be used to install and run software on the nodes. For more detail on services, refer to the section 2.2. To create a service, we create a `ScionSigService` class which inherits from the `Service` class. The `ScionSigService` then creates instances of the `ScionSIGServer` class. These `ScionSIGServer` instances are also referred to as virtual nodes (vnodes). These vnodes are then bound to physical nodes during rendering. In the `ScionSIGServer` class, we added methods to set the SIG configuration. Then, we generated the configuration files for the SIG instance and added the start command to the server. The start command can be executed when the node is started.

We can now create SIG instances in Seed emulations as follows:

```
# AS-153
# regular AS setup
as153 = base.createAutonomousSystem(153)
scion_isd.addIsdAs(1, 153, is_core=False)
scion_isd.setCertIssuer((1, 153), issuer=150)
as153.createNetwork('net0')
as153_cs = as153.createControlService('cs1').joinNetwork('net0')
as153_router = as153.createRouter('br0')
as153_router.joinNetwork('net0')
as153_router.crossConnect(150, 'br0', '10.50.0.3/29')


# we create a physical host for the SIG
as153.createHost("sig").joinNetwork('net0')

# we set the SIG configuration
as153.setSigConfig(sig_name="sig0",node_name="sig",
                    local_net = "172.16.12.0/24",
                    other = [(1,150,"172.16.11.0/24"),
                             (1,152,"172.16.14.0/24")])

# get the SIG configuration in order to add it to the SIG service
config_sig0 = as153.getSigConfig("sig0")
# we install the SIG service on a virtual node
# called sig153 and set the configuration
sig.install("sig153").setConfig(sig_name="sig0",config=config_sig0)

# bind the sig153 virtual node to the sig physical node
emu.addBinding(Binding('sig153',
                        filter=Filter(nodeName='sig', asn=153)))
```

This creates a SIG instance in AS-153 and binds it to the physical host, called `sig`. The equivalent configuration must also be added to the other ASes, in this case AS-150 and AS-152.

Challenges adding SIG support

Throughout the process of adding SIG support to Seed, we encountered a series of challenges. The challenges were mainly related to the fact that we wanted to support running multiple SIG instances in parallel in the same AS in order to maximize throughput. In our experiments, each SIG instance could only handle around 200Mbps of bandwidth. We decided to run several SIG instances in parallel to increase the bandwidth.

Although increasing the bandwidth as discussed is theoretically supported, we could not get it to work in a stable manner. In some instances, the SIG instances did not connect with each other. For some reason, one of the instances always worked while the others did not. Further, when we disabled the working instance, the other one started working. We tried a variety of different approaches to solve this problem: First, we tried different port assignments for the SIG instances because we thought that the SIG instances might use conflicting ports. Then, we tried putting the Seed instances on a different node to test if SIG instances conflict when they are using the same network interface or node. Still, this did not resolve the issues we encountered. We were not able to finally resolve this problem and settled on running one SIG instance per AS. This still allows us to establish connections with several other ASes but limits the bandwidth.

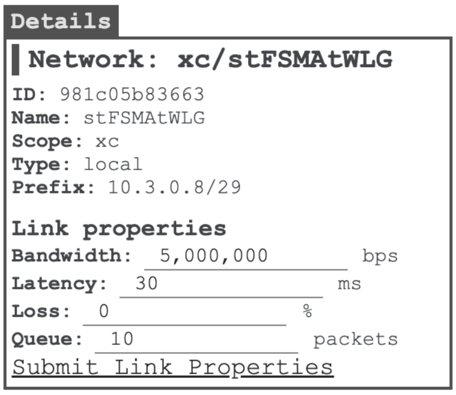
4.2.5 Dynamic link properties

Our initial hesitation to add dynamic link properties was based on their use and value for our project. Nevertheless, we added dynamic link properties to increase the flexibility of the test environment and make it more realistic. Adding this functionality will allow us to change bandwidth, latency and packet loss on the fly while the test environment is running. This also allows us to emulate links going down over time, by setting the bandwidth to 0.

We considered several approaches of adding dynamic link properties to Seed. The first one was to reuse an implementation by Tony John [7] which allowed dynamic link properties. This would have significantly reduced the amount of work needed to add this functionality. The downside of this approach was that the Graphical User Interface (GUI) used to set the link properties was completely separate from the map view Seed offers. This would have required using two different GUIs at the same time. Instead, we decided to add dynamic link properties to the map view of Seed instead:

We added controls to the information panel which is displayed when selecting a network. When selecting a network in the GUI, the network properties are fetched from the backend, first. The backend attaches to one of the docker containers that are connected to the network and executes `tc qdiscs show`. The backend then parses the output of this command and sends it to the GUI.

Once new properties are set in the GUI, they are sent to the backend, and it will then execute the `tc qdisc replace` command within all containers attached to the selected network. This will overwrite existing static and dynamic link properties with the new ones.



```
ID: 981c05b83663
Name: stFSMAtWLG
Scope: xc
Type: local
Prefix: 10.3.0.8/29

Link properties
Bandwidth:  5,000,000   bps
Latency:  30   ms
Loss:  0   %
Queue:  10   packets
```

Figure 4.1: Network Information Panel

4.2.6 Changing the TC configuration

After conducting a series of tests, we observed that bandwidth-limited links using tc sometimes went down for extended periods of time. This was regardless of whether traffic was on them or not. We realized that this was due to how Seed configured tc to use the tbf qdisc [27] for bandwidth and the netem qdisc [26] for latency and loss. Both interfered in such a way that traffic was not dropped but endlessly enqueued. This caused TCP to keep increasing the send window, completely overwhelming the link.

We started using only the netem qdisc for all link properties. This way, traffic that did not fit on the link or in the queue was dropped, causing the TCP to reduce the send window. The new behaviour benefited our project because it better reflects how bandwidth-limited links behave in the real world.

5 Traffic generator implementation

After successfully generating a Seed emulation, with additional functionality, from a *.topo* file, we will proceed by generating traffic in the Seed emulation as discussed in this chapter.

5.1 Desired properties

In order to build a test environment which accurately represents real-world networks, we need to generate traffic on the test network because no real-world network is devoid of traffic. Since this project focuses on testing SCION control plane algorithms, it follows that we need to generate SCION traffic. Further, we are interested in how SCION interacts with IPv4 traffic because IPv4 traffic is ubiquitous on the internet. In addition to SCION traffic, we will thus also generate IPv4 traffic both using BGP and the SCION IP Gateway.

For the purposes of this project, the application sending the traffic or the traffic content are of secondary importance. By contrast, the amount of traffic and the distribution of the traffic over time are highly relevant metrics. Thus, the traffic generator will support controling both the amount of traffic and distribution of traffic over time. We refer to traffic with a specific bandwidth that is generated at a certain point in time as **background traffic**.

Although this project mainly focuses on such background traffic, the test environment would benefit from generating traffic from specific applications as well. To investigate how different control plane algorithms interact with different applications, generating web traffic would be of particular interest because it is one of the most common types of traffic on the internet. Other types of application traffic such as video streaming traffic would also be interesting to investigate in future research, but lay outside the scope of this project.

5.2 Implementation

5.2.1 Possible approaches

There are two possible approaches to implement a traffic generator in a Seed emulation. First, the traffic generator can be implemented as a process running on each traffic-generating node. Several such processes can communicate with one another in order to synchronize when they start and stop generating traffic and which process should send traffic to which other process. Second, the traffic generator can be implemented by choosing a supervisor running on the host system which then coordinates the traffic generation.

The benefit of the first approach is that it can be implemented as a Seed service and is fully contained within Seed. However, the first approach is more complex because it requires the implementation of a communication protocol between the nodes. Also, Seed only supports one service per node. If we wanted to run a SIG instance at the same time as the traffic generator, we would have to use separate nodes, limiting test system scalability. What's more, this communication would be on top of the network and would add unpredictable background traffic to the emulation. By adding a separate network for the communication between the traffic generators, the background traffic issue could be avoided.

The benefit of the second approach is that it is simpler to implement because it does not require any synchronization between the nodes. Further, there is no traffic between the nodes because it can be implemented using the docker socket to execute commands in the nodes. One downside is that this approach requires running an extra supervisor that is not contained within Seed.

When writing this, it came to our attention that the seed labs team started working on their own traffic generator. They used the first approach and implemented it as a Seed service. However, the seed lab team traffic generator lacked key elements which are essential for our purposes. First, it did not support generating SCION traffic. Second, it had no mechanism to modulate the parameters of the traffic over time. Third, it was impossible to coordinate when the traffic generation should start or stop. Had we decided to use the seed lab team's traffic generator, it would have required customization. Regardless of customization, we would still have had to contend with the mentioned downsides.

After careful consideration, we decided in favor of the second approach. It is easier to implement and does not add any additional background traffic to the emulation.

5.2.2 Traffic generator architecture

As described before, the traffic generator [28] is based on the following principle. We will define a specific type of traffic with specific parameters at a specific point in time. The traffic generator will generate the specified traffic by executing a process inside the nodes using the Docker socket.

5.2.3 Specifying traffic

To specify traffic, we use `.json` with the following structure:

```
{
    {
        "traffic_patterns": [
          {
            "start_offset": "0m",
            "source": "1-110",
            "destination": "1-112",
            "mode": "bwtester",
            "specialized_parameters": {
              "cs": "10,?,?,10Mbps",
              "sc": "10,?,?,500kbps"
            }
          },
          {
            "start_offset": "10s",
            "source": "1-110",
            "destination": "1-111",
            "mode": "iperf",
            "sig": true,
            "parameters": {
              "duration": 30,
              "bandwidth": "10Mbps"
            }
          },
          {
            "start_offset": "0s",
            "source": "1-110",
            "destination": "1-112",
            "mode": "web",
            "parameters": {
              "duration": 30
            }
          }
        ]
      }
}
```

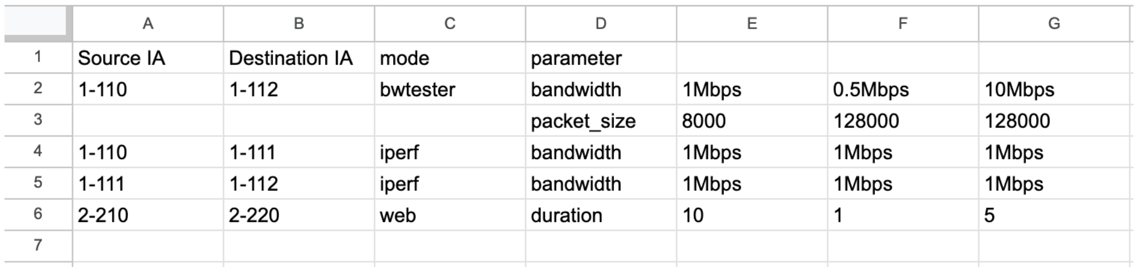
The above list of traffic patterns has the following parameters:

- **Start Offset:** This is the time offset from the start of the traffic generation when this pattern should start. This can be specified in seconds, minutes or hours

- **Source:** This is the IS-AS pair that the traffic should originate from
- **Destination:** This is the IS-AS pair that the traffic should be sent to
- **Mode:** This is the traffic mode. Possible values are (bwtester, iperf, web)
- **sig:** This is a boolean that indicates if the traffic should be sent through the SCION IP Gateway. This parameter only exists for the iperf and web mode.
- **Specialized Parameters:** These are command-line arguments that are directly passed to the underlying tool generating the traffic (e.g. iperf3, bwtester). Thus, the arguments depend on the mode.
- **Parameters:** These are standardized parameters and support (duration, bandwidth, packet size). Though the web mode only supports duration and will ignore bandwidth and packet size. Also note that one pattern can have either specialized parameters or parameters but not both.

5.2.4 Traffic matrix

In addition to specifying the traffic patterns in a `.json` file, we cancalso specify a traffic matrix. The traffic matrix represents the same information diffrently, helping to specify parameters over time. A traffic matrix looks as follows:



	A	B	C	D	E	F	G
1	Source IA	Destination IA	mode	parameter			
2	1-110	1-112	bwtester	bandwidth	1Mbps	0.5Mbps	10Mbps
3				packet_size	8000	128000	128000
4	1-110	1-111	iperf	bandwidth	1Mbps	1Mbps	1Mbps
5	1-111	1-112	iperf	bandwidth	1Mbps	1Mbps	1Mbps
6	2-210	2-220	web	duration	10	1	5
7							

Figure 5.1: Example of a Traffic Matrix

The matrix can be created in any spreadsheet application. The columns represent different source destination pairs over time. The lines represent the times when traffic was generated. By default, one column represents 10s. However, this can be modified using the *time_step* parameter in the traffic generator. In the background, this traffic matrix is compiled to a *pattern.json* file.

5.2.5 Starting the traffic generator

Starting the traffic generator works as follows.

1. The traffic generator has two mandatory arguments:
    a) a pattern file
    b) a Seed emulation dump (Refer to 2.2 for more information on dumping Seed emulations)
2. The traffic generator then parses and sort the traffic pattern file by start offset.
3. Next, it extends the Seed emulation with all the necessary parts, such as the extra AS for BGP routing or nodes that send/receive the generated traffic.
4. Next, it renders, compiles and starts the Seed emulation and waits for all the nodes to be operational.

The result is a command that looks something like this:

```
python traffic_gen.py -p ./pattern_sample.json -s ./scion-seed.bin
```

This command starts the traffic generator with the Seed emulation dump `scion-seed.bin` and the `pattern_sample.json` pattern file. Using additional optional arguments can help modify the behavior of the traffic generator. For a detailed list of all the arguments, refer to the traffic generator README [28].

Traffic generation

Once the emulation is started and all the nodes are operational, the generator will iterate through the patterns and launch processes in the nodes corresponding to the traffic mode and parameters. To launch these processes, the traffic generator expects a `Client` and a `Server` Class. These classes are responsible for handling the command line arguments as well as the shell commands to launch the processes. Each `Client/Server` class exposes a start function which attaches to the correct container and launches the correct program in the container. To extend the traffic generator with new traffic modes, a new `Client/Server` class can be implemented.

Web traffic

Compared to the bwtester and iperf traffic generation modes, the web mode is more complex. The web mode works as follows. It has a list of URLs which can be modified using an argument. The default is the 100 most visited websites [22]. The traffic generator clones the index of each of these websites using `wget` and then uses `nginx` to serve these on the traffic generation nodes.

For this purpose, an index of the indexes is generated according to the following template:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Links Landing</title>
</head>
<body>
    <h1>Links to Subdirectory</h1>
    <p><a href="link1">title1</a></p>
    <p><a href="link2">title2</a></p>
    <p><a href="link3">title3</a></p>
    .
    .
    .
    <p><a href="link100">title100</a></p>
</body>
</html>
```

This index serves as the landing page for the web traffic. On the other end of the web traffic mode, there is a client based on the python `requests` library. This client requests the index page to get the list of links. Then it will iterate through all the 100 links on the page and request them one by one in a round-robin fashion for the duration specified in the traffic pattern.

5.2.6 Adding BGP to traffic generation

To generate IPv4 traffic on our SCION network, BGP needs to be added to the network. SCION connections can be translated into BGP connections, with one exception. In SCION, core ASes forward traffic among one another without any limitations. If we translated this into BGP peering connections, we would run into a problem. In BGP, ASes do not forward traffic received from a peer to other peers whereas in SCION, core ASes do. There are two approaches to navigate this. The first one is to add a BGP peering connection between each pair of SCION core ASes. This, however, would add a lot of BGP peering connections to the network. The second approach is to add a new parent AS which acts as a provider for all SCION core ASes. This will ensure that there is connectivity between all SCION core ASes. The downside of both approaches is that the number of hops that traffic has to take to reach its destination is different for SCION traffic and IPv4 traffic. This might influence the results when testing IPv4 against SCION in our test environment. This is because SCION traffic is passed directly between core ASes and the number of hops depends on the topology whereas IPv4 traffic always takes either one (first approach) or two hops (second approach) when passing from one core AS to another core AS. We implemented the second approach because it keeps complexity at a minimum and has a smaller influence on scalability.

Figure 5.2 demonstrates how the BGP routing addition works.

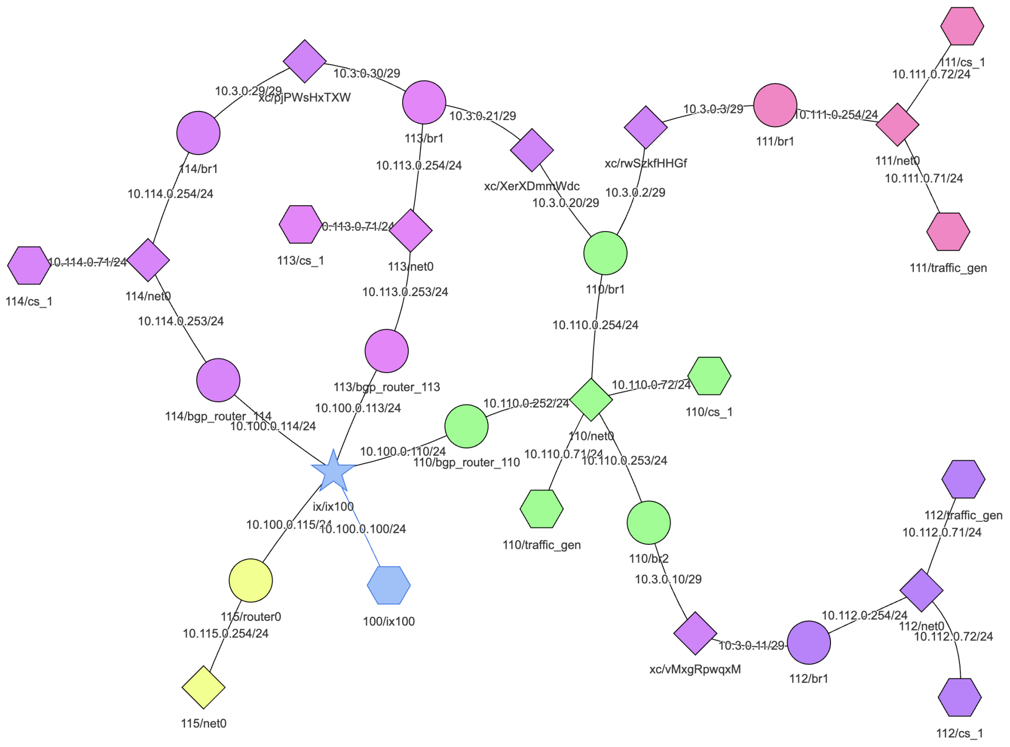


Figure 5.2: BGP Routing Example

There are four SCION ASes (110 (green), 111 (pink), 112 (purple), 113 (purple), 114 (purple)) all of which are in the same ISD. AS 110, 113, 114 are core ASes whereas AS 111 and 112 are children of AS 110. We can see how AS 110 is not directly connected to AS 114 but indirectly through AS 113. In SCION, this is works because AS 113 forwards the traffic to AS 114. In BGP, this would not work. This is why all three core ASes are connected to the AS 115 (yellow) which acts as a provider for all core ASes and ensures BGP connectivity between all core ASes.

6 Future work

Where we leave of we are at a point where it is possible to effectively test SCION control plane algorithms in an emulated environment. Going forward this will pave the way for reseach in novel path selection algorithms leading to a more sustainable and fairer internet. Especially the use of path metadata in the path selection process is an interesting area for future research.

Still there are several opportunities for future improvements. Firstly, more applications could be added to the traffic generator, including video streaming, file transfer, and other protocols that are commonly used on the internet. This would make the emulated network even more similar to real-world networks. Secondly, it would be worth investigating running several SCION IP Gateway (SIG) instances in parallel to enable a higher bandwidth. Lastly, a new compiler could be implemented for the Seed emulator. This would facilitate bigger emulations. One possible candidate for such a compiler would be a Kubernetes compiler. The emulations could be distributed to run on a cluster of machines. This would practically remove the size limit of Seed emulations allowing emulations at a scale that is closer to real-world networks.

7 Discussion

In the project described as part of is thesis, we made it possible to generate Seed emulations directly from *.topo* files. This makes Seed drastically more useful for SCION development. What's more, we extended Seed to support more of SCION's features. Furthermore, our project paved the way for easier testing of SCION control plane algorithms and other SCION features. This will enable faster development of SCION and help to position SCION as a viable solution for the future of routing on internet. To conclude, our projet made Seed more capable, thus putting it at the forefront in the available toolscape for network emulation.

8 Conclusion

In the project this thesis is based on, we set out to create a test environment which would enable us to test SCION control plane algorithms. This included modifying the SCION topology tool, extending the Seed emulator to support all the functionality we needed and creating a traffic generator to generate traffic on the emulated network. Reconsidering the goals of this project, we accomplished most of what we set out to do. The exceptions include the limited bandwidth we were able to put through the SIG and the limited number of different applications that traffic can be generated with. These limitations however are not critical and do not meaningfully impact the usefulness of The test environment. All in all, we succeeded in creating a capable test environment allowing us to test SCION control plane algorithms.

Acknowledgements

First and foremost, I would like to thank my supervisors, François Wirz and Jordi Subirà Nieto, for their support and guidance throughout the process of this thesis.

Next, I would like to express my gratitude to Prof. Dr. Adrian Perrig for giving me the opportunity to work on this thesis in the Network Security Group at ETH.

Further, I owe my thanks to friends and family for helping me at various stages of this thesis with their guidance and feedback.

Lastly, I need to mention that most of the Code produced in the process of this thesis as well as parts of the thesis itself were written with the support of the GitHub Copilot[5] extension and other Large Language Models.

Bibliography

[1] BGP4.AS. Border gateway protocol. https://www.bgp4.as/.

[2] Cloudflare. Bgp hijacking. https://www.cloudflare.com/en-gb/learning/security/glossary/bgp-hijacking/.

[3] Docker. Docker. https://www.docker.com, 2024.

[4] J. Gessner. Leveraging application layer path-awareness with scion. Master thesis, ETH Zurich, Zurich, 2021.

[5] Github. Github copilot. https://github.com/features/copilot, 2024.

[6] iperf3. iperf3. https://iperf.fr, 2024.

[7] T. John. Seed-emulator. https://github.com/netsys-lab/seed-emulator/tree/tjohn327/testbed, 2024.

[8] Kathara. Kathara. https://www.kathara.org, 2024.

[9] Netkit. Netkit lab generator. https://github.com/KatharaFramework/Netkit-Lab-Generator, 2024.

[10] netsec ethz. scion-apps. https://github.com/netsec-ethz/scion-apps/tree/master, 2024.

[11] nsnam. Ns-3. https://www.nsnam.org, 2024.

[12] nsnam. Ns-3 model library. https://www.nsnam.org/docs/release/3.41/models/ns-3-model-library.pdf, 2024.

[13] A. Perrig, P. Szalachowski, R. M. Reischuk, and L. Chuat. *SCION: A Secure Internet Architecture*. Springer, 2017.

[14] Scapy. Scapy. https://scapy.net, 2024.

[15] SCION. Scion bandwidth tester. https://github.com/netsec-ethz/scion-apps/tree/master/bwtester, 2024.

[16] scion. Scion path metadata. https://docs.scion.org/en/latest/manuals/control.html#path-metadata, 2024.

[17] scion. Sig documentation. `https://docs.scion.org/en/latest/manuals/gateway.html`, 2024.

[18] scion protocol. Awesome scion. `https://github.com/scionproto/awesome-scion`, 2024.

[19] scion protocol. Scion tools. `https://github.com/scionproto/scion/tree/master/tools`, 2024.

[20] seed labs. Seed client. `https://github.com/seed-labs/seed-emulator/tree/master/client`, 2024.

[21] seed labs. Seed-emulator. `https://github.com/seed-labs/seed-emulator/tree/master?tab=readme-ov-file`, 2024.

[22] semrush.com. Most visited websites. `https://www.semrush.com/blog/most-visited-websites/`.

[23] S. S. Seyedali Tabaeiaghdaei. Green routing. `https://scion-architecture.net/pages/scion_day_2022/slides/SCIONDay-GreenRouting.pdf`, 2022.

[24] S. A. Tabaeiaghdaei. ns-3-scion. `https://gitlab.com/SeyedaliTaba/ns-3-scion`.

[25] tc. tc. `https://man7.org/linux/man-pages/man8/tc.8.html`, 2024.

[26] tc. tc-htb. `https://man7.org/linux/man-pages/man8/tc-netem.8.html`, 2024.

[27] tc. tc-tbf. `https://man7.org/linux/man-pages/man8/tc-tbf.8.html`, 2024.

[28] L. Urbantat. Traffic generator. `https://github.com/Bruol/traffic_gen`.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.