POLITECNICO DI TORINO

Master's Degree in Computer Engineering
Academic Year 2022-2023

Politecnico
di Torino

Master's Thesis

Evaluation of SCION for User-driven Path Control: a Usability Study

Supervisors
Prof. Paola GROSSO
Prof. Fulvio VALENZA
M.Sc. Leonardo BOLDRINI

Candidate
Antonio BATTIPAGLIA

«E allora prendi e parti,
insegui i sogni
spacca i mondi, lascia indietro la mania
lascia indietro la follia,
le vecchie bare ed i ricordi;
parti a caccia di farfalle da far nascere nel cuore,
cerca nuvole lontane
che dipingano l'amore.»

Table of Contents

List of Tables

List of Figures

Acronyms

AD
Autonomous Domain

AIP
Accountable IP

AP
Attachment Point

AS
Autonomous System

ASN
Autonomous System Number

DoS
Denial-of-Service

D-DoS
Distributed Denial-of-Service

ISD
Isolated Domain

ISP
Internet Service Provider

ML
Machine Learning

PKC

Public Key Certificate

SCION

Scalability, Control, and Isolation On Next-Generation Networks

SCMP

SCION Control Message Protocol

TCB

Trusted Computing Base

TD

Trust Domains

UPIN

User-driven Path verification and control in Inter-domain Networks

VM

Virtual Machine

Chapter 1

Introduction

Citizens and governments are increasingly reliant on digital technologies, which have become deeply ingrained in the fabric of society [1]. These technologies, being built on the conventional Internet architecture, inherit some of its inherent limitations. These limitations encompass the lack of user control over the network and, consequently, a diminishing trust in digital systems [2]. The Responsible Internet paradigm strives to address these challenges by enhancing the transparency, accountability, and controllability of the Internet [2].

The UPIN (User-driven Path verification and control in Inter-domain Networks) project, rooted in the principles of the Responsible Internet, introduces a framework that empowers users to manage network behavior [3]. This initiative aims to seamlessly integrate with the existing Internet architecture while granting users a degree of control over network traffic. Achieving user control over network traffic, however, necessitates a fundamental shift in network architecture, distinct from the conventional approaches in current use.

SCION [4] (Scalability, Control, and Isolation On Next-Generation Networks) is an Internet architecture designed to address some of these issues by providing end-to-end users with route control, failure isolation, and explicit trust information. SCION mitigates the challenges that the Responsible Internet seeks to resolve and inherently offers robust security and resilience due to its sound design principles. SCION achieves this through the concept of Autonomous Systems (ASes) that are divided into autonomous routing sub-planes known as *trust domains*. These trust domains provide rich connectivity and ample choice of routes, thereby ensuring natural isolation of routing failures and preventing manual misconfigurations. Importantly, from our research perspective, trust domains empower end-users with substantial control over inbound and outbound traffic, establish meaningful and enforceable trust, and facilitate scalable routing updates.

This research endeavors to explore the potential and constraints of relying on a SCION network to offer users control over how their traffic traverses the network.

For user-driven path control to become a reality, understanding the characteristics of underlying paths is imperative. We delve into aspects such as the impact of user-selected paths with the lowest latency to a specific destination on available bandwidth within a SCION network. This examination allows us to gauge the performance implications of shifting network control from operators to end-users. The research unfolds by providing an insight into an existing SCION network and its capabilities, including the applications that run on it and how different paths are influenced by latency, bandwidth, and packet loss. We subsequently introduce the software that leverages these applications to assess the performance of available paths within the SCION network. This data is then stored in a database that can offer extensive insights into path characteristics. The database serves as a resource to provide users with optimal path selections based on performance metrics, geographical placement of network components, and the operators responsible for them. Specifically, this goal is achieved by providing a user with a path-recommendation system accessible through a UI.
The remaining part of this thesis is structured as follows. Chapter 2 elucidates the concepts of SCION and the UPIN project, equipping the reader with a general understanding of the state of the art. Furthermore, it offers an in-depth exploration of the problem and the scope of this work. Chapter 3 provides an overview of the experimental setup and its capabilities. In Chapter 4, we delve into the design considerations and the software implementation, which subsequently pave the way for a detailed exploration of the path selection process and performance analysis, as expounded in Chapter 5. Chapter 6 discusses the path recommendation system, and finally, Chapter 7 presents the conclusion of this work.

1.1 INDIS 2023

This work has been condensed in a research article accepted for the INDIS 2023 Workshop in Super Computing 2023 Conference held in Denver, Colorado. The paper, presented on the *12th of November 2023* by Leonardo Boldrini, is a collaborative effort by Antonio Battipaglia, Leonardo Boldrini, Ralph Konig, and Paola Grosso. The paper can be accessed at: ***Evaluation of SCION for User-driven Path Control: a Usability Study*** [5].

Chapter 2

Background

Before diving into the core of this work, we need to discuss the state-of-the-art, what we relied on and what was the challenge to address. We begin with Responsible Internet, a paradigm encapsulating the ethical side of internet infrastructure [2]. Transitioning to UPIN, a framework that aims to fulfill new trust requirements for a Internet user [3]. We explore how it empowers users in determining data paths across inter-domain networks. The architectural backbone of our exploration, SCION [4], takes center stage. Its scalability, control, and isolation principles redefine network dynamics. This chapter serves as a foundational tapestry, weaving Responsible Internet, UPIN, and SCION into the fabric of our research narrative.

2.1 The UPIN Framework

To contextualize the UPIN framework, we mention the Responsible Internet paradigm. This novel security framework is engineered to grant users unprecedented control over the network, encompassing metadata about the network's structure and operation, as well as the transportation of their data [2]. The existing internet infrastructure, fraught with inherent problems and limitations, presents challenges in transparency and control over data routing, rooted in its original design. Significant research has aimed to transcend these constraints, yielding both revolutionary approaches like RINA [6] and evolutionary solutions like SCION. However, since revolutionary approaches demand a complete redesign of the network architecture, their feasibility is hindered by the global dependence on the current internet infrastructure.
In this context, the escalating demand for a trustworthy internet, especially to support critical services such as smart energy grids and intelligent urban transport systems, is the backdrop against which the UPIN framework emerges [3]. This framework is a significant step forward, aiming to make inter-domain networks

more transparent, accountable, and controllable. With the growing need for improved internet security, UPIN brings a fresh approach to how users verify and control paths in these networks. Unlike revolutionary approaches, UPIN charts a feasible evolutionary path, presenting a framework that empowers users to define network behavior. In UPIN, users become the drivers of communication, specifying requirements ranging from simple Key Performance Indicators (KPIs) to advanced scenarios such as avoiding specific regions or jurisdictions. Future internet applications, especially those in critical sectors like smart energy grids, demand heightened levels of trust from the internet infrastructure.

2.1.1 UPIN Framework Components

The UPIN framework comprises several components, each tailored to meet specific requirements related to transparency, accountability, and controllability. Table 2.1 provides a comprehensive overview of these key UPIN components, outlining their distinct roles.

Component	Transparency	Accountability	Controllability
Domain Explorer	✓	✓	✓
Path Controller	×	×	✓
Path Tracer	✓	✓	×
Path Verifier	✓	✓	×
Frontend	✓	✓	✓

Table 2.1: UPIN components and their corresponding fulfillments.[3]

The core concept of the UPIN architecture revolves around UPIN-enabled domains, each implementing specific UPIN components: Domain Explorer, Path Controller, Path Tracer, Path Verifier, and Frontend. The framework doesn't mandate a specific data plane technology, allowing flexibility in implementation, with domains supporting different technologies such as Segment Routing or SCION. UPIN-enabled domains can coexist seamlessly with non-UPIN domains, enabling operations only within UPIN-enabled domains along the path.
The research described in this work will focus on the Path Controller component. It takes on the responsibility of establishing the right forwarding rules according to user preferences and dispatching them to all routers within its domain. It operates locally, meaning it influences only the nodes within its own domain, not those in other domains. In section 2.3 we will explore the research question that puts together the UPIN framework and SCION architecture.

2.2 SCION

SCION is a modern Internet architecture designed with a focus on giving end users solid control over managing routes, handling failures, and understanding trust in their end-to-end communications. This thoughtful design ensures that users have significant influence over both incoming and outgoing traffic. Built to thrive in challenging situations, SCION emphasizes reliability, transparent trust, and using multiple routes across different domains. Its design inherently includes strong resilience and security, following sound principles without relying on additional security patches. Moreover, SCION operates on the idea that only a few top-tier Internet service providers (ISPs) within a trusted domain are needed to ensure reliable end-to-end communication. This approach leads to a minimal Trusted Computing Base (TCB), a key strength of SCION. [5].

2.2.1 Function Properties

SCION operates on three foundational principles: isolating domains, allowing mutual control over path selection for both endpoints and intermediate ISPs, and establishing explicit trust for end-to-end communication [4]. Let's explore them:

- **Domain-based Isolation:** The choice of dividing the control plane into autonomous and independent domains ensures isolation, protecting routing in one domain from malicious activities in others. This feature provides security and scalability while maintaining reachability and path diversity across domains.

- **Mutually Controllable Path Selection:** SCION enables joint path selection between source and destination, significantly enhancing the controllability of routes from both the endpoints. This is done without creating conflicts with routing policies of ISPs.

- **Explicit Trust and Small TCB for End-to-End Communication:** Through the separation of mutually untrusting entities into different domains, each of them gains the autonomy to designate a unified root of trust (for instance, selecting from a subset of tier-1 ISPs) to foster trust among its Autonomous Domains (ADs). This grants endpoints the authority to consciously determine whom to trust for establishing dependable end-to-end communication. Importantly, ADs lacking trust in other domains hold no sway over an endpoint's path discovery and route computation. This configuration culminates in a compact TCB tailored for end-to-end communication reliability.

In the following paragraphs we will explore the fundamental functioning of SCION, breaking down its two main plane: Control Plane and Data Plane.

Control Plane

In SCION, Autonomous Domains establish communication paths using up-paths and down-paths for sending and receiving packets with Trust Domains (TD) Cores. The construction mimics a path vector, starting with TD Core ADs transmitting one-hop paths to their 1-hop customer ADs through path construction beacons. Customer ADs add themselves to the path, forwarding it to their customers and peers, creating a cascading effect. Endpoint ADs then choose from the paths received from providers or peers to create their own K (preferably maximally disjoint) up-paths towards the TD Core and down-paths for receiving packets. Once the path construction has come to an end, the Lookup operation is involved. Endpoint ADs share their chosen down-paths on the TD's Path Server, a service located in the TD Core. This server is queried by local and foreign ADs for routing information. SCION employs Accountable IP (AIP) for host and AD addressing, where each address corresponds to a public key. The TD Core issues a TD membership certificate for each AD address. When performing a name lookup, entering a human-readable destination name provides both the AIP address of the destination host and AD, along with the down-paths specific to the destination AD at the AD level [4].

Data Plane

In the process of forming a complete end-to-end communication path to a destination in SCION, the source AD undertakes path selection. Initially, it selects one of its up-paths to reach the TD Core and then queries the destination's down-paths through a name or address lookup. The source AD then picks one of the queried destination's down-paths to construct the entire communication path. For simplicity, the gateway in an endpoint AD typically makes the default path selection decision on behalf of the hosts in that AD, though a host can also negotiate with its provider AD for customized path selection policies.
In route joining, the source AD examines the paths to identify common ancestor ADs before combining one of its up-paths with one of the destination AD's down-paths. This process aims to find a "shortcut" path without traversing the TD core, enhancing efficiency.
Subsequently, during forwarding, when the source AD constructs the complete end-to-end communication path, each packet contains "opaque fields" generated by the transit ADs during path construction. These fields encode forwarding path information, specifying ingress and egress points at each transit AD along the end-to-end path. Within each AD, any internal routing protocol can be used to

determine a path from ingress to egress. The destination can either reverse the embedded path or query the source AD's Path Server for alternative paths to reach the source. Consequently, in SCION, packet forwarding between ADs eliminates the need for routing and forwarding tables [4].

2.2.2 Scalability

SCION attains routing scalability by confining route dissemination and computation within each Trust Domain independently. Specifically, only the TD Core initiates routing messages, namely the path construction beacons, and these messages circulate solely within the boundaries of the TD. This is in contrast to path vector or link-state protocols where any node in the network can generate routing updates that propagate throughout the entire network. The approach of limiting routing messages to the TD Core promotes scalability, permitting the TD Core to proactively deploy frequent path construction beacons. This proactive strategy ensures that each Autonomous Domain can acquire up-to-date path information within the TD.

2.2.3 Security Properties

Given the interesting properties of SCION, we might be concerned about its security features and how they are achieved. Hence, we will now discuss the most interesting ones:

- **Reflection DoS attacks**: In reflection attacks, an attacker manipulates return addresses to direct unwanted packets from one target (A) to another (B). Systems like AIP, which allow verifying address ownership, can add a protocol for authenticating return addresses, but this introduces complexity. SCION offers inherent protection against reflection attacks by integrating packet addressing with the traversal path. For instance, if a malicious AD (M) tries to inject an attack packet to A with a spoofed return path to a legitimate target AD (B), M would be on the return path itself, nullifying any advantage gained from the attack.

- **Data-Plane Attacks**: Malicious routers pose a threat by manipulating or discarding packets at the data plane, impacting both control messages like routing updates and data packets. SCION addresses this by providing multiple path choices to endpoint ADs, enabling them to avoid paths with detected poor performance. Additionally, each transit AD digitally signs itself into the path construction beacons, allowing the endpoint AD to identify every entity on the forwarding path and hold them accountable for potential misbehavior. Consequently, a malicious router dropping path construction beacons can

only disrupt the specific path or link containing it, offering no advantage as endpoints can choose alternative paths.

- **SPOF Elimination:** SCION's use of multiple Trust Domains removes the necessity for a single authority governing the entire Internet, mitigating deployment challenges and avoiding a *Single Point of Failure* (SPOF). For instance, DNSSec presently relies on a single root of trust for the entire Internet, whereas SCION employs a distributed approach where each TD maintains its own root-of-trust authority.

- **Prefix (AID/EID) hijacking**: SCION provides robust defense against prefix (AID/EID) hijacking. Each endpoint AD or endhost employs AIP as a self-certifying address, and their identities are isolated in different TDs, signed by the respective TD Core. Even if a malicious endpoint in one TD claims the same AID or EID as another in a different TD, the identifiers are scoped to their respective TDs, preventing collisions. Additionally, since the malicious endpoint lacks the private key for the public key used in deriving the AID, it cannot sign any valid statements for the AD.

- **Routing path falsification**: In SCION's path construction, each transit AD commits to the path construction beacon, signing locally announced links and preceding path information in an onion fashion. This design prevents a malicious AD from dropping specific ADs to manipulate the path. Instead, it can only discard the entire preceding path, prompting endhosts to choose alternative paths. The use of onion signatures further ensures that a malicious router cannot extract or splice segments from different paths and cannot modify previous hops in the path.

To sum up, SCION emerges as a robust Internet architecture, empowering end users with enhanced control over their communication paths. Its innovative design not only prioritizes security and reliability but also provides scalability and resilience. Its ability to offer diverse communication path sets and its strategic approach to minimizing the Trusted Computing Base make it a promising solution for the evolving needs of the internet landscape.

2.3 Exploring the Challenge

In our exploration we aim to bridge the aspirations of the UPIN framework with the architectural principles of SCION. As discussed in section 2.1, the UPIN project envisions a paradigm shift, putting users at the helm of controlling data paths within inter-domain networks. This ambitious goal encounters its technological frontier when interfacing with SCION, a network architecture renowned for its

scalability, control, and isolation features.
The UPIN project envisions a simple but thoughtful idea: enabling users to control how their data moves through inter-domain networks. This aims to establish a user-focused model where preferences, ranging from basic Key Performance Indicators (KPIs) to more detailed instructions like avoiding specific regions, direct the communication paths. This shift marks a departure from conventional models, putting end-users in the driver's seat of network communication. Though, it does not directly provide a specific technology to achieve the path control feature.

2.3.1 UPIN and SCION: Intersection

The intersection of the UPIN framework and the SCION architecture presents intriguing opportunities. The UPIN components, especially the Path Controller, are designed to operate within UPIN-enabled domains, steering data paths locally. Simultaneously, SCION aims to provide the feature of Path Control into Isolated Domains, prioritizing robust security measures. This intersection not only aligns with the overarching goals of both UPIN and SCION but also sets the stage for collaborative advancements in user-driven path control within secure, isolated network domains. The intricacies of this harmonious integration form a focal point for our exploration in the upcoming sections. Hence, the research question we want to answer with the development of this work is:

Can the SCION architecture be suitable for UPIN users?

2.3.2 Performance Evaluation: Navigating Latency, Bandwidth, and Data Loss

To answer the previous question, we focused on latency, bandwidth, and data loss as critical metrics to define efficient paths. SCIONLab [7], an experimental testbed of the SCION network, becomes our arena for this exploration. Gathering data from multiple paths, our analysis aims to discern patterns, trade-offs, and constraints that shape the landscape of user-driven path control in SCION. By investigating the possibilities and limitations of SCION for user-driven path control, we lay the foundation for our subsequent research. The research's emphasis on performance metrics, database storage of path data, and the query-driven path selection process informs our approach. As we delve into the complexities of user-driven path control within the SCION architecture, the synthesis of UPIN's vision and SCION's principles emerges as a focal point of our investigation. Moreover, we aim to provide a building block for the front-end component (look at table 2.1) of UPIN framework, with a Path Recommendation System that relies on the data obtained with the performance evaluation and, which is user accessible.

In the next chapters, we will dig deeper into the methodologies, experiments, and results that illuminate the challenge outlined here. Through this research, we aspire to contribute not only to the realization of user-driven path control but also to the broader discourse on the synergy between user-centric frameworks and robust network architectures.

Chapter 3

Experimental Setup

Before diving into the description and analysis of the software architecture, which characterizes the main component of this work, it is crucial to understand the underlying environment on top of which all the experiments were conducted and the software architecture is built.
The environment considered relies on **SCIONLab** [7], a next-generation testbed.

3.1 SCIONLab: A Next-generation Testbed

Network testbeds have been essential in advancing networking research and enabling scientific breakthroughs. These ad hoc environments provide researchers with a controlled platform to conduct experiments and evaluate novel network protocols, algorithms, and technologies. They are crucial to understand the intricacies of network behavior and exploring innovative solutions.

However, it is worth noting that the majority of existing testbeds is intended to experiment current Internet. While this focus has undoubtedly yielded valuable insights and advancements, there is a growing need to broaden the scope of testbeds to encompass emerging network paradigms and technologies.

For instance, next-generation networks that support new networking approaches like path-aware networking, multipath communication or novel security techniques, require specifically designed testbeds.

As widely discussed in the previous chapter, the next-generation network under evaluation is **SCION** [4]. Specifically, the idea is to explore its path-aware feature and its limitations. For this purpose, **SCIONLab** has been developed to enlarge research opportunities and experimentation with SCION.

3.1.1 Architecture

SCIONLab architecture is designed to provide a fully distributed SCION network infrastructure, made up by different Autonomous Systems organized in isolated domains. Users can define their own ASes and connect them to the SCION network, for running experiments. This global topology's main goal is to provide a variety of paths between different ASes to support multipath operations.
It is worth noting that a SCIONLab AS network typically is made up by a single host, unless differently specified. Simultaneously, it operates control plane services, border routers, and end host applications, hence in this work we will interchangeably use "ASes" and "hosts" to refer to network entities.

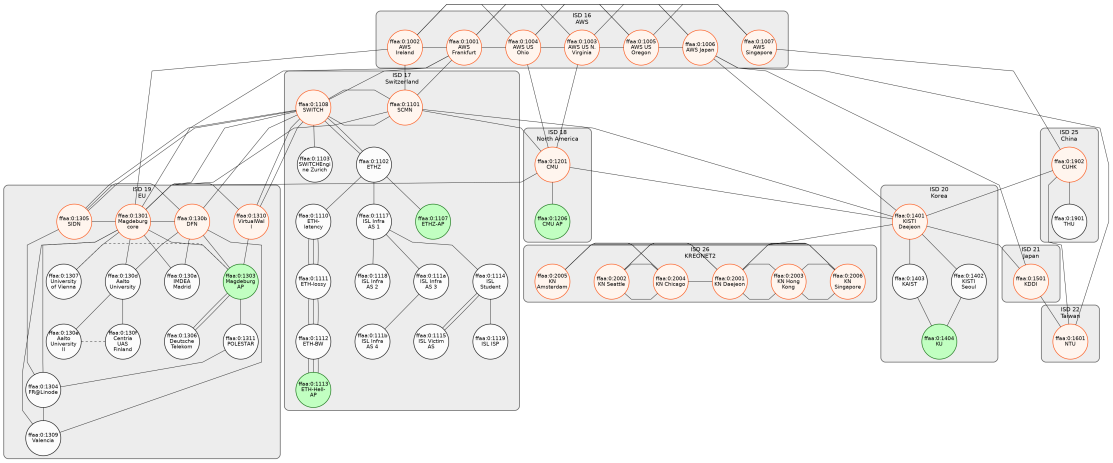

Figure 3.1: SCIONLab Topology[1]: in light orange there are Core ASes; Non-Core ASes are white colored; Attachment Points are green.

Figure 3.1 depicts global SCIONLab topology currently available (*we provide a bigger and clearer picture, in landscape version, of this topology in the Appendix A, figure A.1*). Every node in this topology represents an AS. Each AS is assigned a globally unique AS number (ASN) and a public/private key pair. This key pair is certified through the issuance of a public key certificate (PKC).
There are three different types of ASes:

- **Core ASes**: in the previous picture they are light orange colored. A Core AS is the *root of trust* inside the AS, which is the entity that signs PKC of other ASes in the same ISD.

[1]Source: `https://www.scionlab.org/topology`

- **Non-core ASes**: these are standard components of the SCIONLab infrastructure, having no specific role. They are white colored in the picture.
- **Attachment points (AP)**: these are the most interesting components of SCIONLab because they allow users to attach their own ASes. In this way it is possible to extend the global topology with the experimenters' computational resources. In the picture above 3.1, they are light green colored.

3.1.2 Initialization and Configuration

In order to start experimenting with SCIONLab, we had to define a couple of ASes to attach at some specific endpoints. Precisely, two ASes were created through SCIONLab web interface (`https://www.scionlab.org/`) and attached to `ETH-Hell-AP` and `ETHZ-AP`.

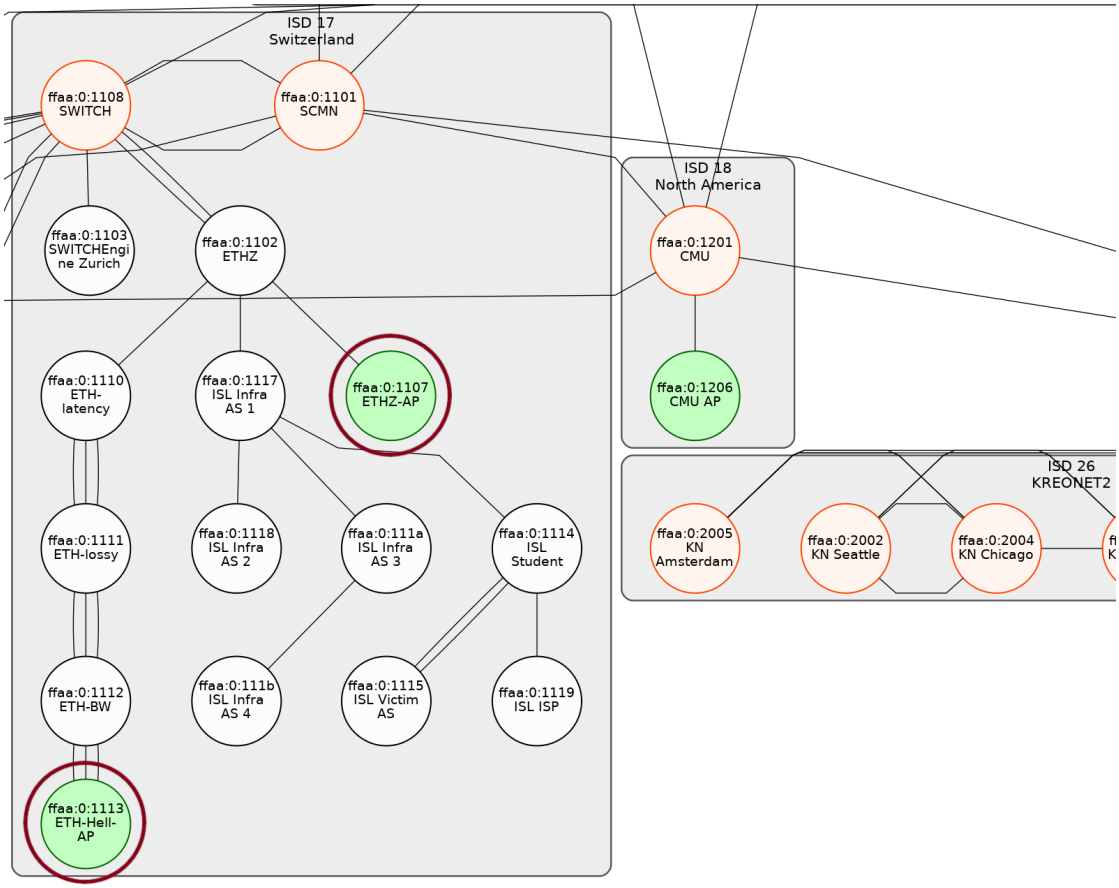

Figure 3.2: SCIONLab Partial Topology - Attachment Points Used

The choice of these attachment points was carefully reasoned, taking into account

that ETH-HELL-AP provides different link degradation levels, as the figure 3.3 shows. Specifically, for this attachment point there are links with increased latency, constrained bandwidth, and introduced packet loss. The other attachment point, ETHZ-AP, was instead selected to test intra-domain communication with the other AP.

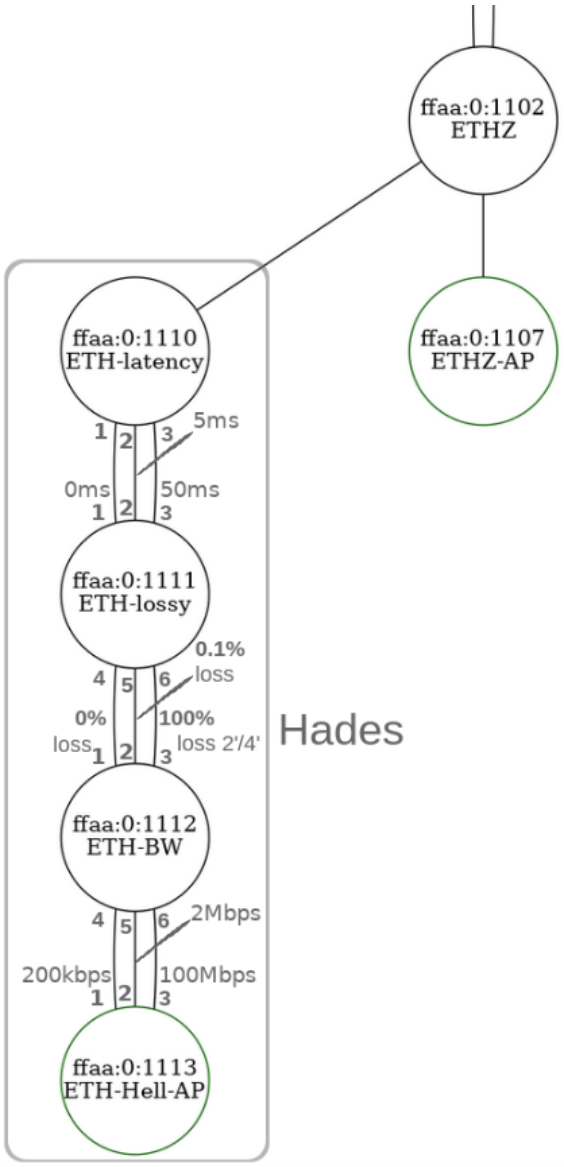

Figure 3.3: Partial Topology - Links with Degradation

Once this configuration phase was completed, a unique ASN was issued, along with cryptographic keys and public-key certificates. Subsequently, a Vagrant file

for each of the user controlled ASes was generated to instruct the configuration of two Virtual Machines (VMs). This file made the setup process lightweight by automating the installation of SCIONLAB services, relevant packages, and necessary configurations.
Finally, two fully configured VMs belonging to the global SCIONLab topology were ready to use.

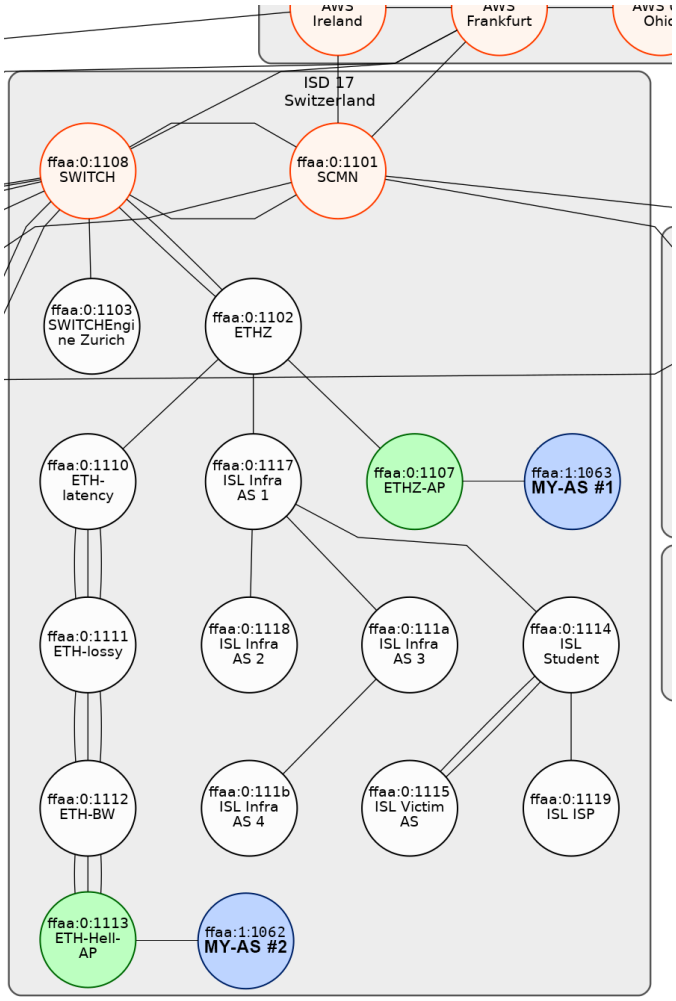

Figure 3.4: Partial topology (ISD 17) with new attached ASes, blue-colored.

3.1.3 Available Applications

The VM configuration process also installs a predefined set of SCION applications automatically. Additionally, specific ports are forwarded to the host system to provide different services, e.g., allowing access to the SCIONLAB AS from an

external end-host. The SCION apps available are highly relevant to understand what will be explained in the next chapter [4], therefore a subset of the most used commands will be presented following:

- **`scion address`**: this command returns the relevant SCION address information for the local host.
- **`scion showpaths`**: it lists available paths between the local and the specified AS. By default, the list is set to display 10 paths only, it can be extended using the `-m` option. Moreover, a really useful feature for this work, is the `--extended` option, which provides additional information for each path (e.g. MTU, Path Status, Latency info).
- **`scion ping`**: it tests connectivity to a remote SCION host using SCMP echo packets. When the `--count` option is enabled, the ping command sends a specific number of SCMP echo packets and provides a report with corresponding statistics. Furthermore, the real innovation is the `--interactive` mode option, which displays all the available paths for the specified destination allowing the user to select the desired traffic route.
- **`scion traceroute`**: it traces the SCION path to a remote AS using SCMP traceroute packets. It is particularly useful to test how the latency is affected link by link after every hop. Even this command makes interactive mode available.
- **`scion-bwtestclient`**: it is the only application presented in this work that is not installed by default in the VM. `Bwtestclient` is part of a bigger bandwidth testing application named `bwtester` which allows a variety of bandwidth tests on the SCION network. The application enables specification of the test duration (up to 10 seconds), the packet size to be used (at least 4 bytes), the total number of packets that will be sent, and the target bandwidth (e.g. 5,100,?,150Mbps specifies that the packet size is 100 bytes, sent over 5 seconds, resulting in a bandwidth of 150Mbps. The question mark `?` character can be used as wildcard for any of these parameters, in this case the number of packets sent. Its value is then computed according to the other parameters. The parameters for the test in the client-to-server direction are specified with `-cs`, and the server-to-client direction with `-sc`).

The usage of these scion commands will be further analyzed in the next chapter.

Chapter 4

Software Architecture

Following the exploration of the experimental setup, mainly of SCIONLab [7], it is possible to appreciate the software architecture built upon it.
As mentioned in previous chapters, the idea of this work is to assess **SCION** architecture's [4] suitability for a **UPIN** user [8]. To accomplish this, the most effective approach is to evaluate the SCION architecture in its designated testbed. Hence, a test-suite relying upon SCION built-in commands was developed to perform this evaluation. We will explore its design, implementation and assessment in this chapter.

4.1 Overview

In the *"Background"* chapter (2) we have discussed SCION features, especially in terms of path-awareness and path selection, as well as UPIN users' use-cases. With this software architecture we want to assess those aspects and test network performances such as: latency, bandwidth and packet loss.
The software relies on a 3-tier architecture: there is a client-server interaction model, along with a database where information are retrieved and stored by the client. In more detail, the two previously configured ASes in the SCIONLab network act as clients interacting with a pool of servers. These servers are globally distributed around the network and belong to different ISDs. The interaction model is really simple: a client which wants to test paths performances to reach different destinations, performs the following actions by running the test-suite:

1. **Paths Collection**: the client needs to gather information about all the possible paths to reach each destination, so in the first phase it will collect the list of paths available and their known characteristics.

2. **Paths Test Execution**: the client has to test, for each of the retrieved paths,

network performances in terms of latency, loss and bandwidth available.

3. **Stats Storage**: final step is the storage of the previous statistics. One entry for each path is inserted in the database and contributes to provide samples for path analysis and evaluation.

Figure 4.1 provides an overview of the software architecture and summarizes all the steps described above.

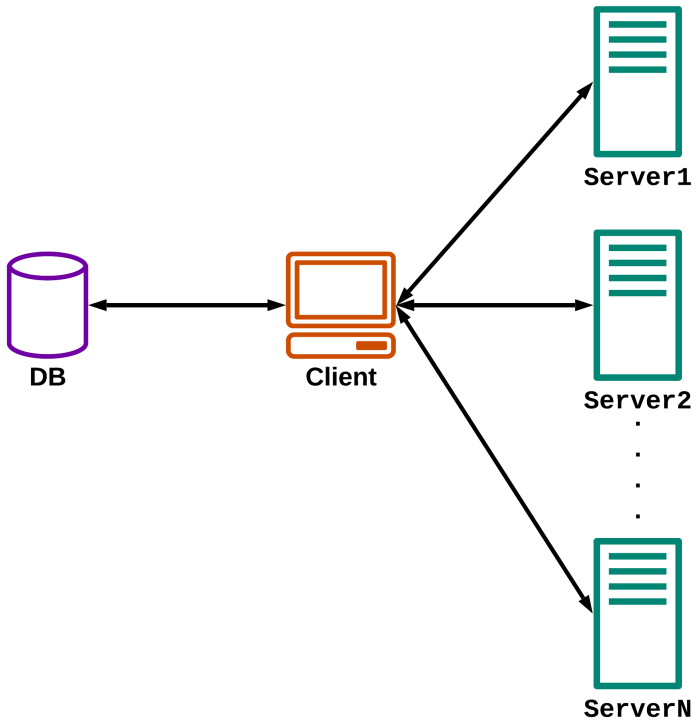

Figure 4.1: Overview of the software architecture: the client interacts with each server to gather information about paths and then stores them in the database

4.2 Technical Requirements

The first step in the development of software was to identify the main technical requirements in order to properly design the application. Certainly, these requirements have been examined considering the software running over a SCION network, that means a network over which it is possible to perform path selection. In this section the identified constraints are presented, followed by an explanation of design choices made to fulfill them.

4.2.1 Scalability

Since the test-suite is based on testing network performances, one of the most important requirements is scalability, which means the system's capability to adapt to a larger workload or user base without compromising performance, responsiveness, or reliability. In this particular case, scalability is a fundamental aspect that revolves around the test-suite's ability to handle a large and ever-growing quantity of data while maintaining optimal performance and responsiveness. As the test-suite is designed to constantly perform network measurements and evaluations, it inevitably generates a significant amount of data. This data includes information about path performances, network statistics, and other relevant metrics collected from a multitude of test runs.
The amount of data generated can grow in two different cases:

1. **Increasing the number of tests performed per destination**: this is the case in which we run our tests a higher number of times to enlarge the data set with more samples. Naturally, this is generally performed to improve the quality of data and perform a better analysis.

2. **Increasing the number of destinations under test**: we can increase the number of servers considered during the assessment. This leads to enlarge the number of paths under test and, subsequently, the amount of data produced grows with them.

By proactively considering scalability in the test-suite's design and implementation, it can confidently handle the dynamic nature of the SCION network, support ongoing performance evaluations, and accommodate future growth, providing valuable insights and analysis.

4.2.2 Fault Tolerance

Another aspect to take into account is fault tolerance, that is the ability of the software to continue functioning properly in the presence of faults or failures. In our case, we can identify many types of failure:

- **Data Loss**: since the application is based on retrieving and storing data in a database, and it relies on a dynamic network, sometimes it may happen that some data gets lost due to a malfunction in the network or in the software. The right approach is to try limiting this amount of data.

- **Server Failure**: as our source is not the only actor in this architecture, it is not the single point of failure. Destinations can be up or down and in some cases they could not answer at our requests. They are not under our

control but, on our side, we have to manage missing answers properly, to avoid deadlocks in our software.

- **Error Messages**: it can also happen that a server is not down but it provides a bad response. As in the previous case, we should handle also this kind of responses.

Fault tolerance is a key point of this architecture since, continuous measurements require continuous functioning.

4.2.3 Portability

Portability is a feature of a software application that describes how easily it can be transferred or adapted to different computing environments or platforms without requiring significant modifications. Our application is intended to be working on all the SCION-based networks, with minimal modifications required. Portability problems that may arise are:

- **Different Commands Specification**: the whole software architecture relies on the latest SCION built-in commands available in SCIONLab, hence, there might be updates or previous versions of SCION that may not recognize the commands used.

- **Flexibility to changes in metrics**: in the future, a user could desire to add more metrics for the assessment of a path. These metrics must be easy to integrate in the software.

Make a software portable is essential to speed its adoption and improvements, but also to provide a plug-and-play system that requires at most minor changes.

4.2.4 Security

Security plays a crucial role in this architecture, given the multitude of interactions that could potentially introduce vulnerabilities. Ensuring data integrity and authentication, managing database access, and safeguarding against Denial-Of-Service attacks are the baseline for a secure test-suite.
Let us now proceed to a more in-depth examination of these properties:

- **Data Authentication and Integrity**: As the software conducts measurements across the network and store them in a database, it is crucial to establish the legitimacy of data and verify whether any tampering has occurred. Upholding data authentication and integrity means to address this requirement.

- **Database Access Management**: Once data authenticity and integrity are established, it is equally important to perform access control to store, read and modify them. Only authorized users, following an authentication process, should be granted these privileges.

- **Denial-Of-Service Attacks**: The threat of DoS attacks looms large over systems that require continuous operation, as such attacks have the potential to disrupt services for considerable periods. The development of a resilient architecture capable of mitigating these attacks becomes indispensable.

We must consider security into every phase of the development process, starting from the design stage to avoid undesired consequences and future complications.

4.3 Design Choices

After a deep examination of main technical constraints, we can now delve into design choices taken to address them. A complete description regarding different component design and how to achieve properties previously described, will be discussed in this section.

4.3.1 Database Design

Since the software is based on retrieving and store data efficiently, the first thing to design was the database and take all the decisions needed to achieve good performance. At this purpose, the choice fell on a non-relational database (DB) because of two considerations:

- **Massive dataset organization**: a non-relational database can easily store huge quantities of data and query them, since it uses horizontal scaling and distributed architecture to handle large datasets efficiently. Scale and speed are crucial advantages of non-relational databases and can be strongly useful in testing systems, like the one considered here, since the amount of data produced is massive.

- **Flexible database expansion**: data is something that can dynamically change, a non-relational DB can absorb new data points, enrich the existing database with new levels of granularity and extend previous data. Since the test-suite is thought to be extensible and metrics to evaluate can change over time, a non-relational database is the right choice once again.

Among non-relational DBs, MongoDB has been selected for its usability and performances.

For the sake of clarity, groups of data logically recalling the same concept are named *collections*, they hold the same role of tables in a relational database, though in MongoDB, data structures belonging to the same collection can be heterogeneous. An entry in a collection, is instead called *document*.
The designed database is composed of the following collections:

- **`availableServers`**: it stores information about servers with which the test-suite can connect and perform performance tests. This collection was needed in the SCIONLab topology to exceed the limitation that not all the performances can be tested for each node. The problem is that it is possible to test bandwidth only over a subset of the ASes available. Hence, for completeness of testing, only those destinations for which it was possible to retrieve latency, bandwidth and loss were considered. Figure 4.2 illustrates the displacement of destination subsets within our topology (*landscape picture in Appendix B, figure B.1*). Notably, as we indicated earlier in subsection 3.1.1, the norm is for each AS to house only one host. However, in this specific case, certain ASes contain multiple servers.

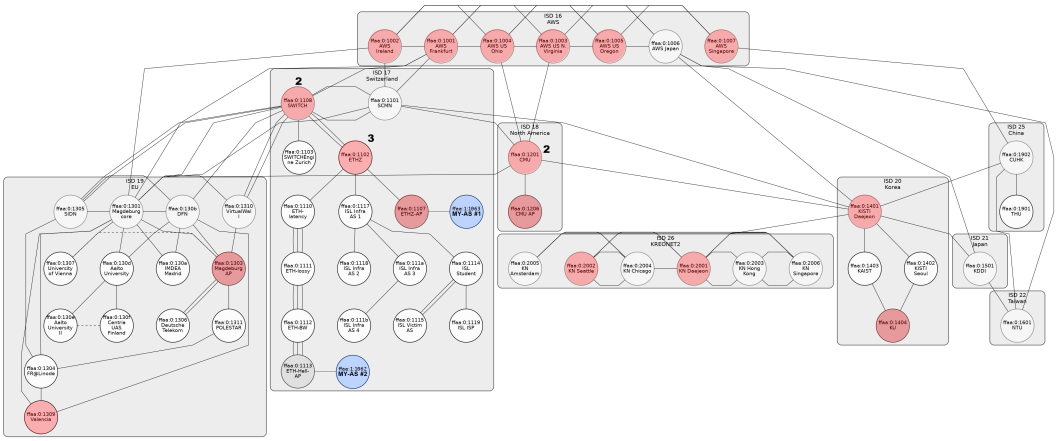

Figure 4.2: SCIONLab Topology: in red all the available ASes which contain servers that can be fully tested. Bold numbers over them state the amount of servers housed by that AS.

- **`paths`**: this collection holds the information gathered for each path to each destination in availableServers, it includes list of hops but also known characteristics.

- **`paths_stats`**: it collects all the statistics gained after the test-suite run but also further information related to the path (e.g. ISDs traversed, number of hops..., we will provide further details in the next sections).

The database schema, along with documents' structure, is provided below (4.3).

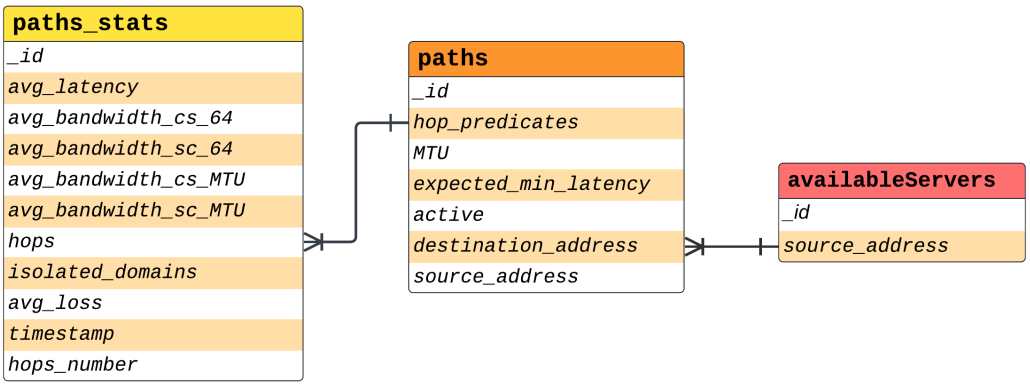

Figure 4.3: Database Schema presenting, from left-to-right, collection of paths' statistics, collection of each path for each server and servers considered for the assessment

Looking at the previous picture (4.3), it is possible to identify the described collections. *"AvailableServers"* collection has 2 fields: server's source IP address, along with an id. This identifier is a progressive integer and in our case it can be a number between 1 and 21, since we only have 21 destinations fully testable in our topology (4.2). Each *"paths"* document, instead, has its own identifier (_id), that is built by combining the server id and a progressive number for the path (e.g. a path whose id is `2_15` identifies the path `15` of the destination `2`). Other fields are used to describe some known properties about the path. Finally, each document of *"paths_stats"* collection has its own identifier built by combining the path identifier with a timestamp, in order to identify the measurement in time over a specific path for a specific destination. Other fields are performances related, such as: average latency, average loss or average bandwidth in upstream and downstream, considering packets of only 64 bytes or packets of MTU size.

4.3.2 Technical Requirements Design

Some actions have been taken to achieve scalability, fault tolerance, portability, and security properties. As previously described, the choice of a non-relational DB can strongly improve scalability in querying and storing operations as well as in distributing data automatically. In addition, another solution was to reduce I/O operations' overhead by preferring multiple insertions of path statistics to single ones. Naturally, there is a trade-off between fault tolerance and scalability in terms of insertions. Preferring multiple insertions means also that if a crash happens all the statistics are lost and not saved. On the other hand saving one

measurement at time decreases performances dramatically and makes the system less scalable. We decided to insert all the measurements after testing once all the paths for that destination. In this way, a loss of data can be negligible since one sample for each path would be lost without unbalancing the number of samples for each path; this leads to a growth in fault tolerance. At the same time, this reduces I/O overhead and allows the system to manage an increasing amount of data, making it resilient to potential overload. Moreover, since nodes can be up and down and sometimes they might be unreachable, the software architecture was provided with error handling to reduce crashes and keep the system working with a dynamic and fallible network.
Regarding portability, the software uses commands available in SCIONLab, therefore its usage over a different SCION based network may require a few adjustments to adapt them, though the whole architecture would mostly be the same. Furthermore, the choice of MongoDB enables effortless modification or addition of metrics, resulting in a highly extensible and portable system.
In terms of security, many solutions have been designed, though, they are not implemented yet. The primary focus has been given to:

- **Database Access Management**: the first thing to constraint is database access, particularly during the statistics' saving operation. Restriction is needed to avoid fake performances injection that may alter analysis and provide misleading results. A possible way of doing this is the usage of public key certificates to get write access to the DB.
  At the moment of this writing, this feature has not been implemented yet to make the interactions lightweight. The adoption of a Public Key Infrastructure (PKI) for db access management is a plan for the future.

- **Statistics Authentication and Integrity**: similarly, also produced measurements should be authenticated with a PKC to provide data integrity and authentication. This step is crucial in preventing the possibility of receiving counterfeit data, even if the database accesses are under control. Since our architecture relies on SCIONLab commands, we will assume that tampering protection and data authentication have been managed in those application development.

- **Denial-Of-Service**: DoS attacks can be really hard to manage. Fortunately, SCION's inherent properties offer some relief by minimizing the likelihood of DoS attacks. Its separation of control and data planes, along with message authentication with asymmetric cryptography, allow rejecting unverified incoming packets, thwarting DoS attacks effectively.

These solutions, just described, make the system nearly fully compliant to the specified requirements. Hence, the test-suite is scalable and can easily adapt to

a growing workload in terms of users or data. It guarantees a good level of fault tolerance, enabling the system to be resilient to server side errors (e.g. servers overload or malfunction) and limiting data loss. In terms of portability, our architecture requires minor changes to fully adapt to other SCION networks and is open to future improvements. Finally, security has been considered during all the steps of development and by virtue of SCION architecture it is already partially achieved. On this side, we will work to achieve a higher level that can fully satisfy our requirements.

4.4 Implementation

With a grasp of the fundamental aspects of the architecture, including its requirements and design choices, we have gained the foundational knowledge necessary to examine its implementation. In this section, we will focus on how the choices taken in the design phase reflect in the code, hence the most interesting code snippets will be analyzed. We will get a full understanding of the test-suite fundamental units and their mutual interaction to achieve the desired functioning.
The described implementation is publicly available at: `https://github.com/MrR0b0t14/SCION-Test-Suite`.

4.4.1 Test-Suite Units and Interactions

The test-suite is mainly composed of three components: a shell script and two python scripts. For the sake of clarity, a shell is a command-line interface that allows a user to interact with an operating system by typing in commands. A shell script, on the other hand, is a file containing a series of commands written in a shell scripting language, which is then executed by the shell interpreter. Shell scripts are used to automate tasks, perform system operations, and manipulate files and data using command-line utilities. They are especially useful for automating repetitive tasks, creating custom workflows, and managing system configurations. These are the main reasons for which a shell script is suitable for testing purposes and we have adopted it in our architecture. Precisely, the shell script that we used is written in **Bash** *(Bourne Again Shell)*, which is the default shell for most Unix-like systems, including Linux. Bash scripts are versatile and widely used for various automation tasks. Similarly, python scripts consist of sequences of instructions or commands written in the python language. Python is a versatile and widely used programming language known for its readability and high-level libraries that can simplify and help in the development process. Hence, a best practice is to write the main logic of a scripting application in python scripts and execute them from a shell script.
In summary, our test-suite leverages the combined strengths of shell and Python

scripting to achieve efficient and flexible testing processes. In the next paragraphs we will delve deeper in each of these units.

Bash Script

Summing up what we previously explained, the bash script can be seen as a container or wrapper of the python scripts. It provides a command line interface (CLI) to the user and execute other units to test each path and store the result. From the CLI, the user specifies the execution parameters and options necessary to define the behavior of the application at run-time.
We can observe three main parameters:

- **`<iterations>`**: it is an integer that specifies the number of times that tests must be executed for each path. Differently from the other arguments, this one is required and must be defined at execution time. This value is propagated to the **`run_tests.py`** script.
- **`--skip`**: this is an optional argument and can be specified to bypass the collection of paths to each destination and speed up the test execution. Its usage is meaningful only under two conditions: first, paths must have been collected at least once, and second, they must remain unchanged since the last collection. Its specification without meeting both of these conditions leads to either no results or inconsistent measurements, along with potential errors for those paths that have been disused.
- **`--some_only`**: it is another parameter useful to accelerate testing. Its application constraints the test execution to only the first destination. Therefore, all the paths of the first destination in the `availableServers` collection will be tested `<iterations>` times.
- **`-h, --help`**: as in any CLI it displays the "help message", explaining briefly all the parameters and the available options. By specifying this option, the application will not run and will discard all the other parameters and options.

This script, known as `test_suite.sh`, is located in the root directory of our repository (). An example of its usage could be the following:

```
./test_suite.sh 100 --skip
```

The result will be the execution of the tests, for each path available, 100 times and skipping the path collection phase.
For a deeper comprehension, let's briefly examine how the shell script works. The initial step involves checking for the presence of the help option, given its higher priority. Next, we need to account for all potential combinations (e.g.,

both `--skip` and `--some_only`, only one of them, or none). This is achieved using a series of *if statements* that covers all possible scenarios. In each case, two primary actions are executed: first, the collection of paths through the invocation of the `collect_paths.py` script, followed by the execution of tests using the `run_tests.py` script.
Below, we provide a listing of the entire code already explained.

```
#!/bin/bash
iterations=$1
args="$@"

display_help_info() {
    echo "Help information:"
    echo "Usage: ./test_suite.sh <iterations> [options]"
    echo "Arguments:"
    echo "  -h, --help: Display help information"
    echo "  <iterations>: Number of iterations to run the test suite"
    echo " --skip: Skip the path collection and run the test suite (optional but if present, MUST BE after <iterations>)"
    echo " --some_only: With this, the test suite will be run for maximum 2 paths for each destination (optional but if present, MUST BE after <iterations>)"
}

# Iterate through the arguments
for arg in $args; do
    # Check if the argument is '--help'
    if [[ "$arg" == "--help" || "$arg" == "-h" ]]; then
        # Display help information
        display_help_info
        exit 0
    fi
done
if ! [[ $1 =~ ^[0-9]+$ ]]; then
    echo "Invalid iterations argument. Argument is not a digit."
    display_help_info
else
    if [ -z "$iterations" ]; then
```

```
29          echo "Please provide the iterations number as an
     argument."
30          # Display help information
31          display_help_info
32          exit 1
33      fi
34
35      for arg in $args; do
36          # Check if the argument is '--skip'
37          if [[ "$arg" == "--skip" ]]; then
38              echo "Skipping the path collection..."
39              for arg in $args; do
40                  # Check if the argument is '--some_only'
41                  if [[ "$arg" == "--some_only" ]]; then
42                      echo "Running the test suite for
    maximum 2 paths for each destination..."
43                      python3 Tests/run_test.py -n $1 --
    some_only
44                      exit 0
45                  fi
46              done
47              echo "Running the test suite..."
48              python3 Tests/run_test.py -n $1
49              exit 0
50          fi
51      done
52
53      # Collect the paths
54      echo "Collecting all the paths..."
55      python3 collect_paths.py
56
57      for arg in $args; do
58          # Check if the argument is '--some_only'
59          if [[ "$arg" == "--some_only" ]]; then
60              echo "Running the test suite for only 1
    destination..."
61              python3 Tests/run_test.py -n $1 --some_only
62              exit 0
63          fi
64      done
65
```

```
66      echo "Running the test suite..."
67      python3 Tests/run_test.py -n $1
68      exit 0
69  fi
```

Paths Collection Script

One of the two sub-units of the bash script is the `collect_paths.py` component. It serves as the first internal script with the purpose of gathering information about all the paths available to reach each destination. The user does not interface directly with it, since the only part visible is the external wrapper. Even if its main role is to discover paths, it also performs other operations like data pre-processing, data insertion and deletion. Let's discuss these elements:

- **Paths Collection**: clearly, the main goal of this script is to discover and retrieve paths information to reach the desired destinations stored in the `availableServers` collection. Hence, the initial step involves querying the database and collect the set of destinations to test. For each of them the application spawns a sub-process that runs the SCION command:

  ```
  scion showpaths ---extended -m 40
  ```

  This command provides a maximum of 40 paths for each destination, ranked by hop count, along with all their details: hops predicates, MTU, path status and minimum latency expected over the path. From this output, we have decided to retain only paths with a number of hops at most equal to the minimum required plus one. This selection strategy is aimed at conserving time by excluding paths that are overly lengthy and fail to meet our latency criteria, we will deeply discuss latency in sub-section 5.1.1.
  The following snippet of code is a portion of the script, it describes the sub-process execution and the path filtering, a full version of this function is provided in the Appendix B with the listing B.1.

```
1 def path_info_building(server):
2     #execute scion showpaths command
3     cmd = f"scion showpaths {
    server_destination_address_sp} --extended -m 40"
4
5     proc = subprocess.Popen(cmd, shell=True, stdout =
     subprocess.PIPE, stdin=subprocess.PIPE)
6
7     # Read the output of the command and store it in
    a list
```

```
8       output = []
9       dirty_path_info = []
10      hops_number = 0
11
12      min_hops = 2000
13      while True:
14          line = proc.stdout.readline()
15          paths = re.match(r"\d+ Hops:", line.decode('
    utf-8').rstrip())
16          if paths:
17              hops_number = paths.group().split(" ")[0]
18              if int(hops_number) < min_hops:
19                  min_hops = min(min_hops, int(
    hops_number))
20                  print("Minimum Hops: " + str(min_hops
    ))
21              paths = False
22          if not line or int(hops_number) > min_hops+1:
23              break
24          output.append(line.decode('utf-8').rstrip())
25
```

- **Data Pre-processing**: this step is crucial to clean data and change their format to a suitable one. The output of the `showpaths` differs from what is expected as input for the testing commands. Precisely, the syntax of hop predicates interfaces varies, requiring a parsing operation before storing the paths in the db. For better comprehension, consider the following example of hop predicates from the output of `scion showpaths`:

  ```
  17-ffaa:1:1063 1>518 17-ffaa:0:1107 1>4 17-ffaa:0:1102 2>3
  17-ffaa:0:1108 12>7 16-ffaa:0:1001 5>3 16-ffaa:0:1004 6>4
  16-ffaa:0:1002
  ```

  In that format, egress interface is specified before the '>' symbol, while the ingress of the following hop after it. The data pre-processing maps the same predicates in the following sequence:

  ```
  17-ffaa:1:1063#0,1 17-ffaa:0:1107#518,1 17-ffaa:0:1102#4,2
  17-ffaa:0:1108#3,12 16-ffaa:0:1001#7,5 16-ffaa:0:1004#3,6
  16-ffaa:0:1002#4
  ```

  In this expression, interfaces are placed after the hop to which they refer to, separated by a comma.

- **Data Storage**: once data have been correctly pre-processed, a set of paths for each destination is ready to be stored. The storage operation involves two steps: first, the insertion of paths just gathered, second, the deletion of the ones no longer available.

After these steps, the paths collection (figure 4.3) is fully populated and we can now start the real testing phase with the last script.

Tests Execution Script

This last script, known as `run_test.py`[1], represents the core of the whole application. As with the paths collection script, also in this case the user does not interact directly with it, but the `iterations` parameter and eventually the `--some_only` option, are propagated from the bash script.
The structure of this script is quite simple, there are 3 nested `for` loops used to run tests over each path, for each destination, "`iterations`" number of times. To promote better insight, this is the structure of the script:

```
1 for i in range(iterations):
2     # ... some lines here
3     for server in available_servers:
4     #for each path in paths where path.
    destination_address == server.source_address
5         for path in paths:
6             if(path["destination_address"] == server["
    source_address"]):
7         #... continues here ...
```

The code inside this structure is executed for each path of each destination, one or many times, according the parameter specified by the user. In this block three functions are invoked, each of them generates a sub-process which tests one or more metrics of the path by running some SCION commands.
Specifically, the three sub-processes perform the following actions:

- **Latency and Loss Measurement**: this operation is performed by the first sub-process which executes the following command:

```
scion ping {server_address} -c 30 --sequence
'{hop_predicates}' --interval 0.1s
```

[1]Can be found under the `Test` folder in the root directory.

Basically, it 'pings' the destination using SCMP packets, measuring the latency and the packet loss. We set the interval between each packet to 0.1s and we send 30 packets (3s of measurement). The destination address and the sequence of hops determining the path to test, are dynamically set at each iteration of the three nested loops. The output values of the ping command, are: the average latency measured by the 30 packets sent (in milliseconds) and the packet loss percentage. These values are used, along with the next measurements, to build a `paths_stats` entry (figure 4.3).

- **Bandwidth Measurement with 64 bytes Packets**: this is the subsequent operation performed by the second sub-process which executes the following command:

  ```
  scion-bwtestclient -s {server_address} -cs
  3,64,?,12Mbps -sequence '{hop_predicates}'
  ```

  This is the bandwidth tester application available in SCIONLab. Also in this case we are dynamically specifying server address and hop predicates, but we are also adding, in this order, the time interval for which the bandwidth needs to be achieved (*3s*), the packet size to send over the path (*64 bytes*), a wildcard for the number of packets automatically computed by the application, and the desired bandwidth to achieve (in this case *12Mbps*). We defined these parameters for the client-server measurement only and, by default, they are used for the server-client too.
  At the end of its execution we will get the average bandwidth client-server sending 64 bytes sized packets and the average bandwidth server-client with the same packets.

- **Bandwidth Measurement with MTU bytes Packets**: this is the last operation performed by the third sub-process before the storage of the measurements. It executes the following command:

  ```
  scion-bwtestclient -s {server_address} -cs
  3,MTU,?,12Mbps -sequence '{hop_predicates}'
  ```

  As in the previous measurement, it performs the same operation but sending packets with the size of maximum transmission unit, resulting in two more measurements: average bandwidth client-server and server-client, with MTU sized packets.

After that all the measurements have been performed, it is useful to provide another information before the storage, it is the set of ISDs crossed by the packets. It can be interesting to enlarge the context of measurements with the ISDs and see if they influence somehow the performance. To get this set, the following function has been defined:

```python
#function that gets the ISDs from the hop predicates
def getISD(hop_predicates):
    hops = hop_predicates.split(" ")
    isds = []
    for hop in hops:
        if hop.split("-")[0] not in isds:
            isds.append(hop.split("-")[0])
    return isds
```

We have now gained all the desired information about our path, including latency, loss, bandwidth, and the set of traversed ISDs. The final step involves storing this data. This operation takes place after each path has been tested for a specific destination. As previously discussed in the Technical Requirements Design subsection (4.3.2), this storage approach enhances fault tolerance and reduces overhead in I/O operations.
In summary, the test-suite relies on SCION applications: ping and bandwidth tester. The software comprises three main components: a shell script that acts as a wrapper, and two Python scripts. The first Python script is responsible for collecting the paths to each destination and storing this information in the `paths` collection within our database. The second Python script exploits the collected information to test each available path in terms of latency, bandwidth, and packet loss. The results of these tests are then stored in the `paths_stats` collection.
With all the statistics now at our disposal, we can proceed to evaluate the results and draw our conclusions.

Chapter 5

Path Selection

Path selection is a crucial aspect for network users as it combines several significant concepts in networking. Firstly, an informed path selection helps users to avoid routes that may be congested or underperforming, contributing to overall network stability. Moreover, by having insight into available paths and their characteristics users can make informed decisions about which routes to select, potentially avoiding paths that might raise security concerns or involve untrusted nodes. Path selection also empowers users to optimize their network experience, since the selection of paths with favorable latency, bandwidth, and low packet loss leads to the satisfaction of their specific needs. Chapter 4 provides a detailed description of how the test-suite works and is structured. In this chapter we will delve into the application of the test-suite to assess the SCIONLab [7] network and perform results analysis. The objective of this analysis is to empower users with a comprehensive understanding of the provided tools, enabling them to make informed decisions regarding path selection.
Before diving into the path analysis, we need to understand on which nodes the experiments were conducted and which assumptions we made.
Firstly, we chose to perform the tests only from "`MY_AS#1`" (fig. 3.4). Although "`MY_AS#2`" has many links with different degradation levels (fig. 3.3) that might be interesting, these links were flattening the measurements over different destinations, since all the paths from that AS have in common the same initial hops. Hence, we opted for a source with a lower correlation among paths. Moreover, we picked a subset of 5 destinations from the 21 available in SCIONLab (figure 4.2). In the figure 5.1 they are marked with a blue star.

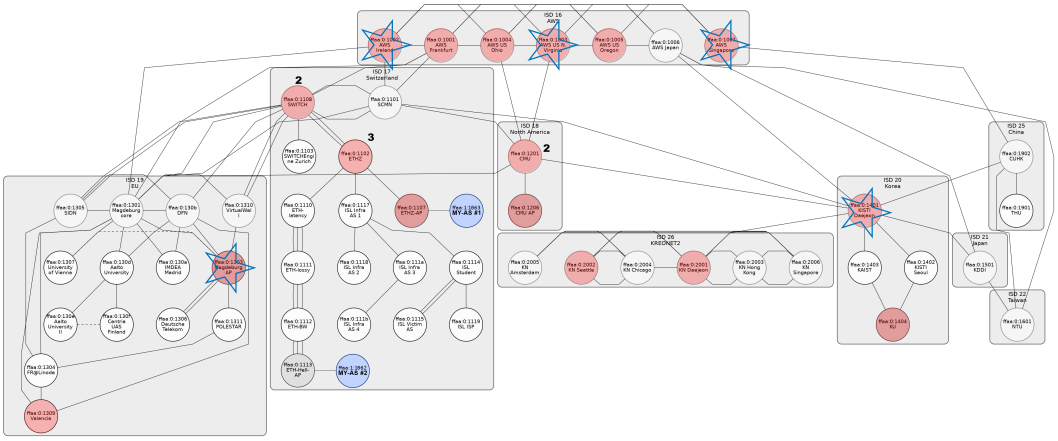

Figure 5.1: Tested Destinations in SCIONLab Topology. Selected servers are marked with a blue star, while all the servers available are red colored.

These servers were accurately selected from different geographical locations and, possibly, from different ISDs. The idea is to assess how much the geographical position and the belonging to a specific ISD can influence performance. Hence, the picked servers are placed in the following countries: Germany, Ireland, North Virginia, Singapore and Korea.
Even though the tests have been performed on this subset, we decided to perform a preliminary analysis about the reachability on the full server set. The picture 5.2 can be used to enrich the servers topology (4.2).

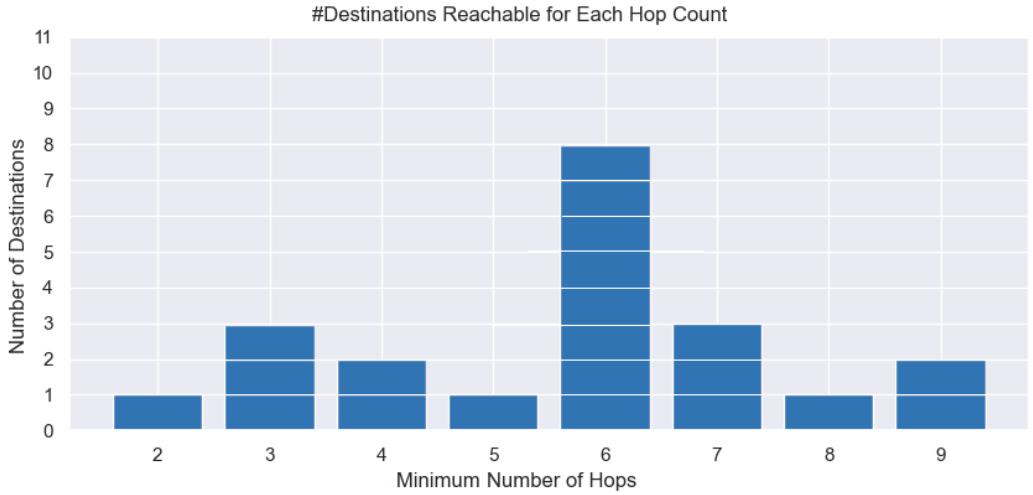

Figure 5.2: Server Reachability from `MY_AS#1`. In blue is displayed the number of destinations reachable requiring minimum a certain hop count.

Essentially, the illustration depicts the number of destinations that require a minimum number of hops to be reached. It offers valuable information regarding the average path length and the distribution of servers across the network. Notably, the average path length is 5.66 hops and about 70% of paths can be reached within 6 hops. This two data highlight the central position of our AS within the server distribution. This insight can easily be confirmed by looking again at the topology (4.2) displaying both our AS and all the servers.
In the next section we will look at performances details of the 5 destinations subset.

5.1 Path Analysis and Result Presentation

In the course of this analysis, the test-suite gathered a substantial dataset comprising approximately three thousand samples. This wide volume of data provides a robust foundation for our subsequent analysis, offering meaningful insights into path performance.
The visual representations, or plots, were generated using the capabilities of a *Jupyter Notebook*, formerly known as *IPython Notebook* [9], which is a versatile and interactive computational environment used for code execution, data visualization, and result presentation. To shape these plots, we exploited the *Seaborn* library [10], a Python data visualization library built on top of *Matplotlib* [11]. Additionally, we employed *Pandas* [12], a popular data manipulation library, to efficiently handle, clean, and transform the collected data, ensuring a smooth and accurate analytical process. Pandas supplies an abstraction for data manipulation with an integrated indexing: the `DataFrame` object. It is strongly useful for structured data management, efficient indexing and flexible data operations.
Following, we will provide multiple graphs about latency, bandwidth and packet loss and we will discuss their most interesting elements. This dissertation is essential to deeply understand why this test-suite is powerful in a path aware network.

5.1.1 Latency Assessment

The first property that we will assess is path latency. Latency refers to the time delay between sending a data packet from a source host to a destination one, and receiving a response. It measures the round-trip time for data to travel between two locations in a network. For instance, low latency is crucial for real-time applications like video conferencing and online gaming.
In a latency assessment, several key metrics are considered to provide a comprehensive understanding of the network performance. These metrics include variance, which indicates the degree of spread in latency values; outliers, representing unusually high or low latency values that can impact user experience; the average value, giving an overall measure of latency; and the median, which offers a representative

value that is less sensitive to extreme values. Examining these metrics collectively provides insights into the stability and consistency of latency in the network. At this purpose, we chose whisker plots to visually represent the distribution of latency values and highlight key statistical measures in a concise manner. A whisker plot is graphical representation useful to display the distribution of a dataset along with its central tendency and variability. Basically, the key components of a whisker plot are:

- **Box**: The box in the plot represents the interquartile range (IQR), which spans the middle 50% of the data. It is drawn between the first quartile (Q1) and the third quartile (Q3). The length of the box indicates the spread of the central portion of the dataset.
- **Whiskers**: The whiskers extend from the box and represent the range of the data outside the IQR. They typically extend up to a certain factor (usually 1.5 times) of the IQR. Data points beyond the whiskers are often considered potential outliers.
- **Median Line**: A horizontal line inside the box represents the median (Q2), which divides the data into two halves. It provides a measure of central tendency that is less affected by extreme values compared to the mean.
- **Outliers**: Individual data points outside the whiskers are plotted as separate points. They are potential anomalies that fall significantly beyond the typical range of the dataset.

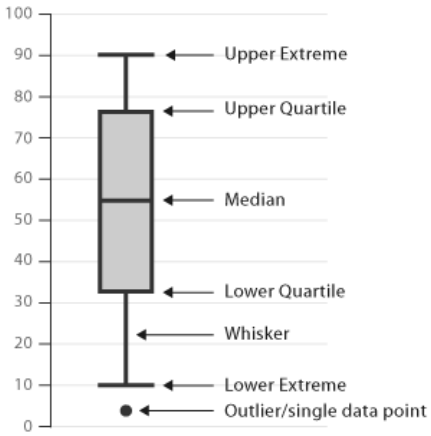

Figure 5.3: Whisker plot representation[1]

[1]Source: `https://datavizcatalogue.com/methods/box_plot.html`

Firstly, our assessment focused on the evaluation of average latency values for each path leading to the five destinations. Figure 5.4 illustrates the whisker plots of latency values for each path of destination `16-ffaa:0:1002,[172.31.43.7]`, which stands for the Ireland AS (top left AS in figure 5.1). On the x-axis, there are the path identifiers of routes having a number of hops less than or equal to the minimum plus one; while, on the y-axis, there are the average latency values. Hence, paths are categorized into two groups: 6 hops paths (in red) and 7 hops paths (in purple).

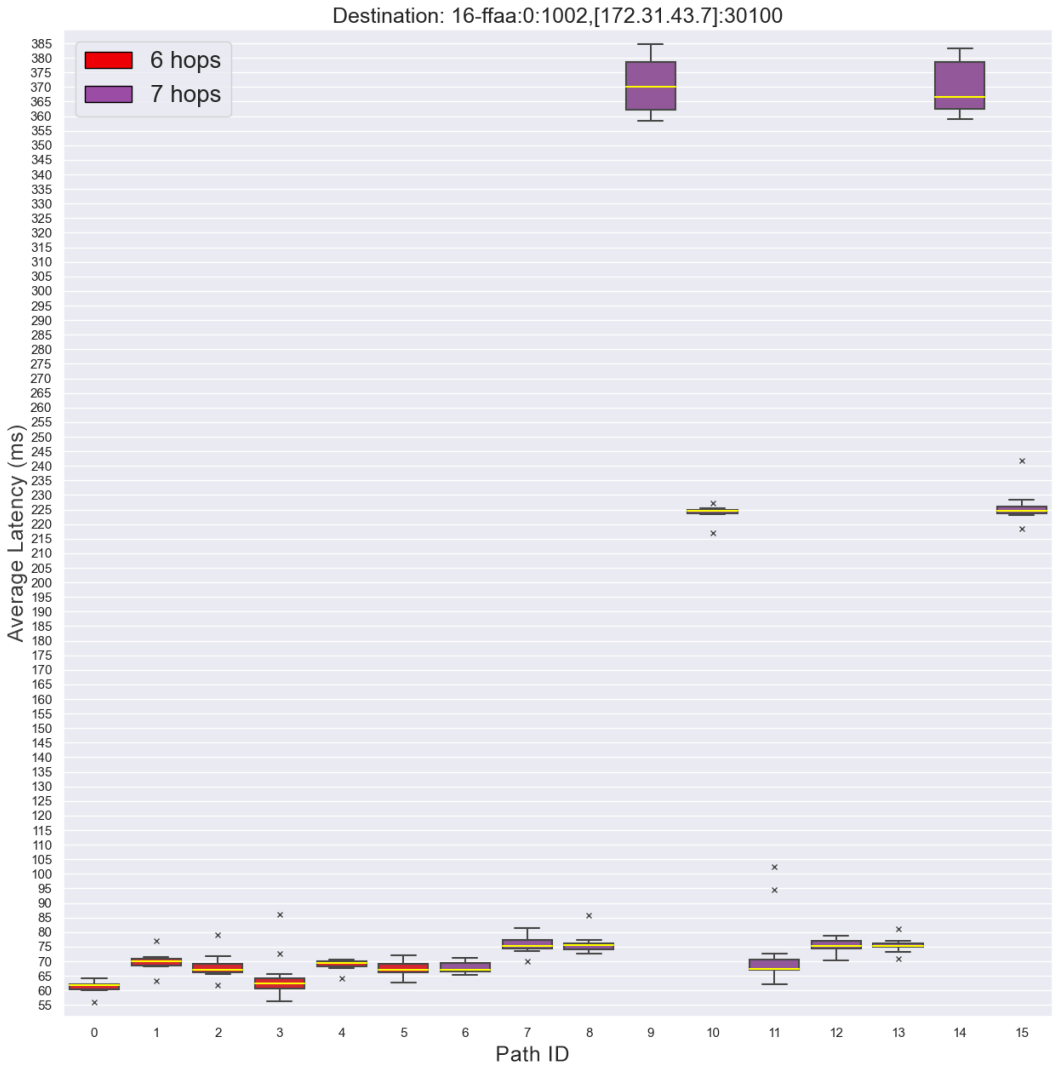

Figure 5.4: Average Latency Values measured for each path of destination `16-ffaa:0:1002,[172.31.43.7]` (AWS - Ireland). Box plots are split into 6 hops paths length, in red, and 7 hops paths length, in purple.

The most interesting aspect in this graph (5.4) is the clear separation of latency values into three main layers, each with nearly the same average values. From an accurate analysis of paths "10" and "15" we have observed that the second-last hop of both paths is placed in Ohio, USA (AS `16-ffaa:0:1004` in topology 5.1).


```
{
  _id: '1_10',
  hop_predicates: '17-ffaa:1:1063#0,1 17-ffaa:0:1107#518,1 17-ffaa:0:1102#4,2
17-ffaa:0:1108#3,12 16-ffaa:0:1001#7,5 16-ffaa:0:1004#3,6 16-ffaa:0:1002#4',
  MTU: '1432',
  expected_min_latency: '>176.4ms (information incomplete)',
  active: true,
  destination_address: '16-ffaa:0:1002,[172.31.43.7]:30100',
  source_address: '17-ffaa:1:1063,127.0.0.1'
},
{
  _id: '1_15',
  hop_predicates: '17-ffaa:1:1063#0,1 17-ffaa:0:1107#518,1 17-ffaa:0:1102#4,3
17-ffaa:0:1108#4,12 16-ffaa:0:1001#7,5 16-ffaa:0:1004#3,6 16-ffaa:0:1002#4',
  MTU: '1432',
  expected_min_latency: '>176.4ms (information incomplete)',
  active: false,
  destination_address: '16-ffaa:0:1002,[172.31.43.7]:30100',
  source_address: '17-ffaa:1:1063,127.0.0.1'
}
```


Figure 5.5: Paths `1_10` and `1_15`. Ohio (USA) AS `16-ffaa:0:1004` is highlighted with a red rectangle.

Moreover, if we consider paths "9" and "14" they both deviate towards an AS in Singapore (`16-ffaa:0:1007` in topology 5.1).


```
{
  _id: '1_14',
  hop_predicates: '17-ffaa:1:1063#0,1 17-ffaa:0:1107#518,1 17-ffaa:0:1102#4,3
17-ffaa:0:1108#4,12 16-ffaa:0:1001#7,4 16-ffaa:0:1007#2,5 16-ffaa:0:1002#6',
  MTU: '1432',
  expected_min_latency: '>330.7ms (information incomplete)',
  active: true,
  destination_address: '16-ffaa:0:1002,[172.31.43.7]:30100',
  source_address: '17-ffaa:1:1063,127.0.0.1'
},
{
  _id: '1_9',
  hop_predicates: '17-ffaa:1:1063#0,1 17-ffaa:0:1107#518,1 17-ffaa:0:1102#4,2
17-ffaa:0:1108#3,12 16-ffaa:0:1001#7,4 16-ffaa:0:1007#2,5 16-ffaa:0:1002#6',
  MTU: '1432',
  expected_min_latency: '>330.7ms (information incomplete)',
  active: true,
  destination_address: '16-ffaa:0:1002,[172.31.43.7]:30100',
  source_address: '17-ffaa:1:1063,127.0.0.1'
}
```


Figure 5.6: Paths `1_9` and `1_14`. Singapore AS `16-ffaa:0:1007` is highlighted with a red rectangle.

This observation suggests that paths with geographically diverse hops have a more

significant impact on latency than the sheer number of hops.
This pattern holds true for other destinations as well, such as the one in North Virginia (`16-ffaa:0:1007,[171.31.19.144]`) in figure 5.7. Here, the main clusters are two and the one with paths having higher latency (`"2_6"`, `"2_13"`, `"2_16"`, `"2_21"`, `"2_29"` and `"2_31"`) consists only of paths passing through Oregon, USA (`16-ffaa:0:1005` in figure 5.1), regardless of the number of hops.

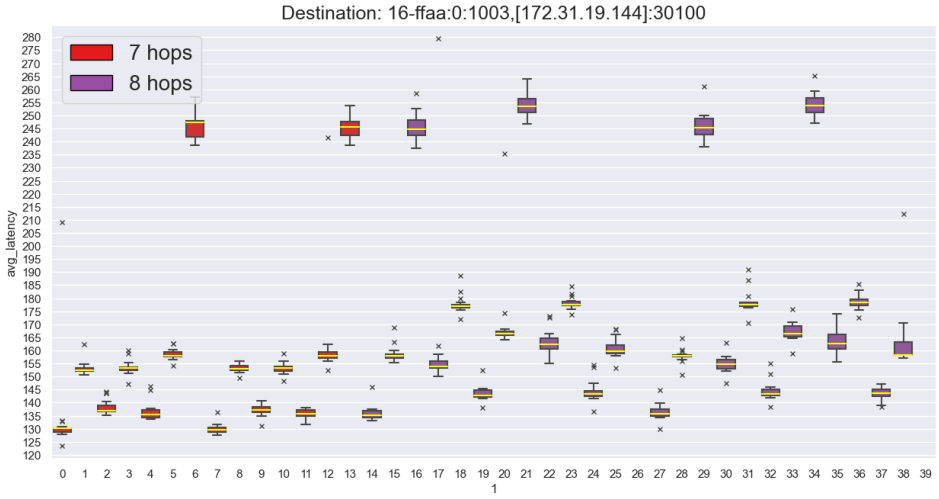

Figure 5.7: Average Latency Values measured for each path of destination `16-ffaa:0:1007,[171.31.19.144]` (AWS - US N. Virginia). Box plots are split into 7 hops paths length, in red, and 8 hops paths length, in purple.

Also in this case, there is a noticeable geographical separation between Northern Virginia and Oregon. As illustrated in figure 5.8 the distance between these two locations highlights the considerable geographical gap. Opting for paths connecting these regions involves traversing a more extended route, which translates to a longer physical distance, especially considering that our source is located in Switzerland.

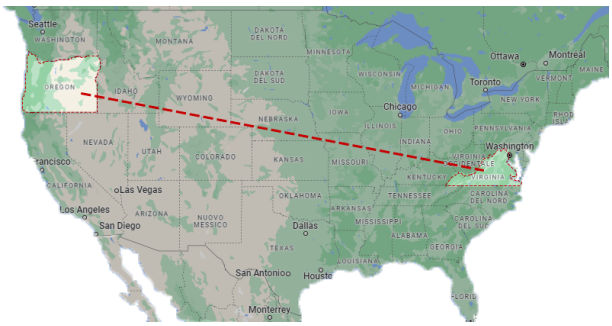

Figure 5.8: USA Map with Virginia and Oregon marked with a dotted red line.

The relation between hops location and latency is very useful to determine which paths should be discarded in a path selection based on low latency routes.
As a complement of our guess, we performed another analysis on latency by grouping, for each destination, paths traversing the same set of isolated domains and having the same hop count. Figure 5.9 shows a graph describing this analysis. The x-axis provides the different sets of ISDs traversed to reach the destination (AWS Ireland `16-ffaa:0:1002,[172.31.43.7]`, visible on the top). Once again, the interesting insight here is that only the hops number is not enough to determine the latency variance or increment. Indeed, by looking at the graph on the left side, in which the only difference between the two whiskers is in the number of hops and not in the set of ISDs considered, we can see a huge variance in latency values for the purple column. This may lead us to think that latency is affected also by the number of hops. Though, if we take out long distance paths like those passing through Singapore or Ohio (which are geographically far from the destination in Ireland) we can observe a smaller variance and comparable values, as observable from the right side of figure 5.9 (*landscape version at Appendix C figure C.3*).

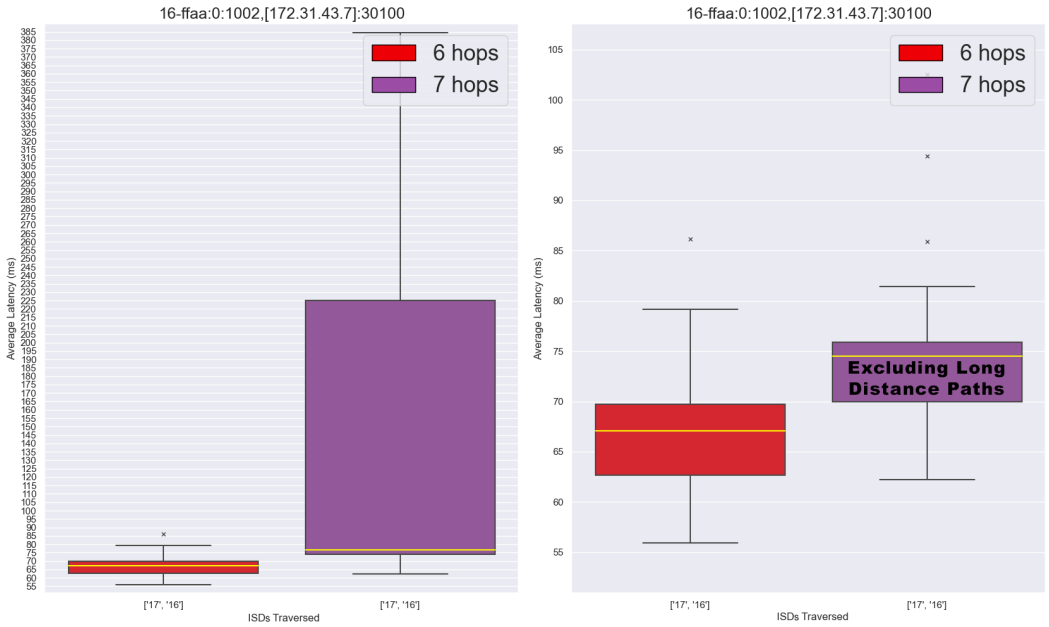

Figure 5.9: Average latency for each ISD set grouped by hop count. On the left side, the plot includes all the measurements. On the right side, long distance paths have been excluded from the second ISDs set.

Hence, the physical distance between hops confirms to be the predominant component in the latency assessment. Moreover, it is worth noting that removing long distance paths we do not only get a latency reduction but also a more compact

box plot. Therefore, it seems that ASes `16-ffaa:0:1007` and `16-ffaa:0:1004` introduce a wide jitter other than high latency peeks. This assessment helps us to exclude routes passing through these ASes for streaming audio and video services, as well as VoIP calls, in which latency consistency is more important than low latency values.

5.1.2 Bandwidth Assessment

The second parameter to evaluate for a conscious path selection is bandwidth. It is the maximum amount of data that can be transmitted through a communication channel or network connection in a given period. It represents the capacity or throughput of the channel and is typically measured in bits per second (*bps*), or its multiples like kilobits per second (*kbps*), megabit per second (*Mbps*) and so on. Bandwidth determines how quickly data can be transferred between devices or systems. A higher bandwidth indicates a larger data-carrying capacity, allowing more data to be transmitted in a shorter amount of time. It's essential for various network activities, such as web browsing, video streaming, file downloads, online gaming, and more. Higher bandwidth connections generally result in smoother and faster user experiences, particularly for applications that require the rapid transfer of large amounts of data. For this assessment, we will use again the whisker plots since they provide a full description of measurements.
In Section 4.4, we explained the kind of information gathered by the bandwidth tester in the path collection script. Essentially, we conducted two distinct tests: one with a lower bandwidth requirement of 12Mbps, and another requiring for 150Mbps. Each evaluation involved two scenarios: utilizing packets of both MTU size and 64 bytes size, and examining interactions from both client to server and server to client. The aim of this approach was to comprehensively assess network behavior from various angles, including upstream and downstream perspectives, as well as under different conditions such as high bandwidth demands, small packet transmission, and average usage at 12Mbps. Considering the first test at 12Mbps, we achieved a consistent trend across all five destinations. Figures 5.10 and 5.11 depict the average bandwidth values tested for two destinations: Magdeburg AP in Germany (AS `19-ffaa:0:1303,[141.44.25.144]` in topology 5.1) and KISTI Daejeon in Korea (AS `20-ffaa:0:1401,[134.75.250.114]` in topology 5.1). These graphs illustrate the distribution of bandwidth for each path (measured in Mbps), showcasing downstream measurements on the right and upstream measurements on the left. Additionally, each path is represented by two whiskers: the yellow whisker corresponds to bandwidth values obtained using MTU-sized packets, while the blue whisker represents values obtained with 64-byte packets. Regardless of a bigger variance for Korea destination in both upstream and downstream plots, we can see that all the paths act similarly for both destinations.

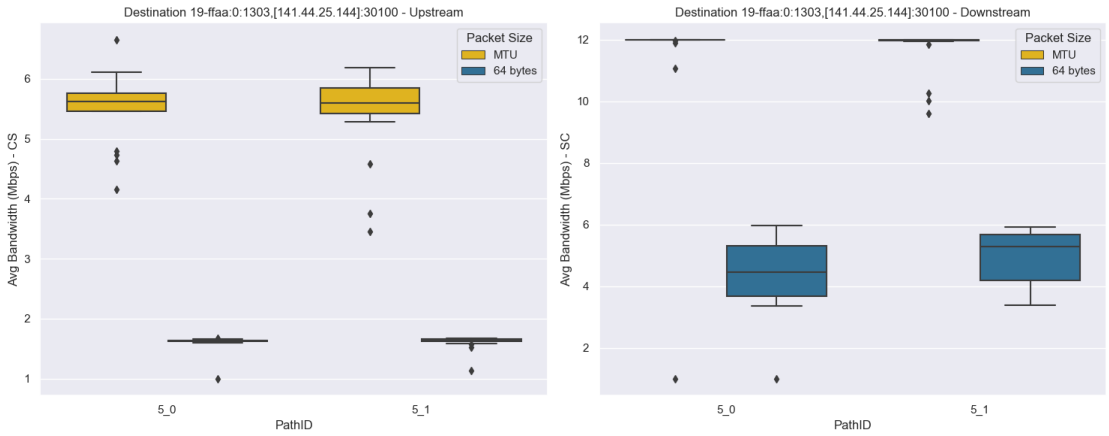

Figure 5.10: Average bandwidth values for each path, requiring a bandwidth of 12Mbps from and to a Germany Server (address on the top). On the left side there are the upstream measurements, while on the right side the downstream ones.

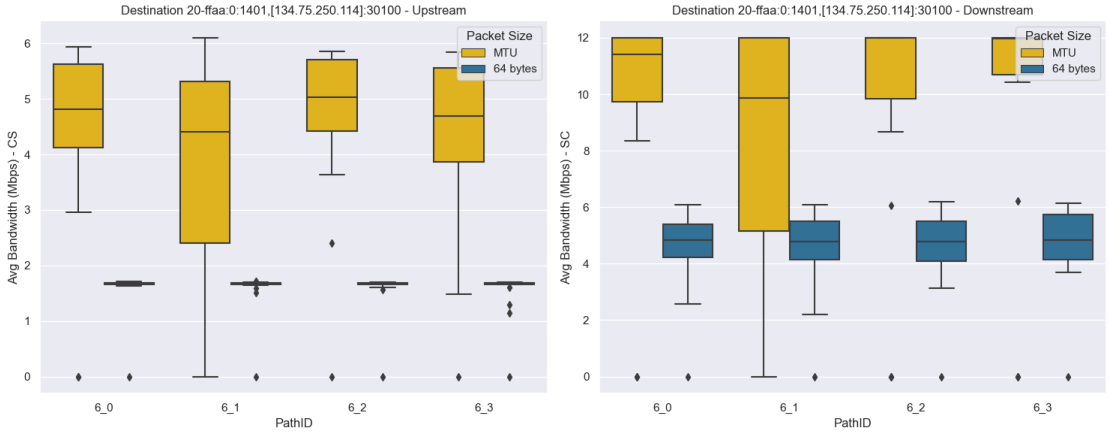

Figure 5.11: Average bandwidth values for each path, requiring a bandwidth of 12Mbps from and to a Korea Server (address on the top). On the left side there are the upstream measurements, while on the right side the downstream ones.

As we would expect, in upstream they achieve a lower bandwidth compared to the downstream counterpart. This phenomenon is in line with the internet's inherent asymmetry, where user data consumption typically exceeds data upload. Moreover, all the paths get a lower bandwidth by sending 64-byte packets compared to the MTU packets. This is an expected outcome, as using smaller packets increases the total packet count, subsequently amplifying the overhead of packet headers. Basically, the behavior of our network can be summed up with this: we achieve a higher bandwidth in downstream sending bigger packets, regardless the path we

take.
This trend reverses when we require a higher bandwidth such as 150Mbps, highlighting the limitations of bandwidth in SCIONLab network. Indeed, looking at pictures 5.12 and 5.13, showing the average bandwidth for the same destinations but with a requirement of 150Mbps, we can clearly see that the trend is not the same anymore.

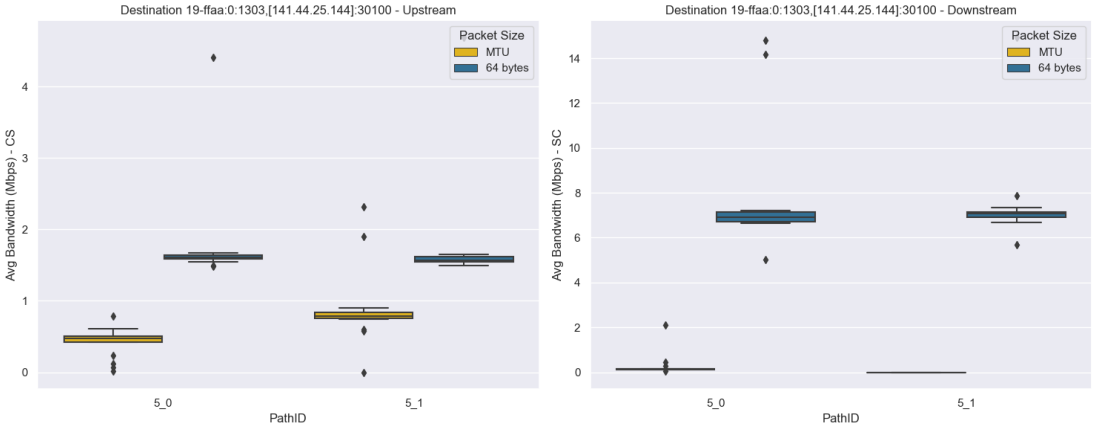

Figure 5.12: Average bandwidth values for each path, requiring a bandwidth of 150Mbps from and to a Germany Server (address on the top). On the left side there are the upstream measurements, while on the right side the downstream ones.

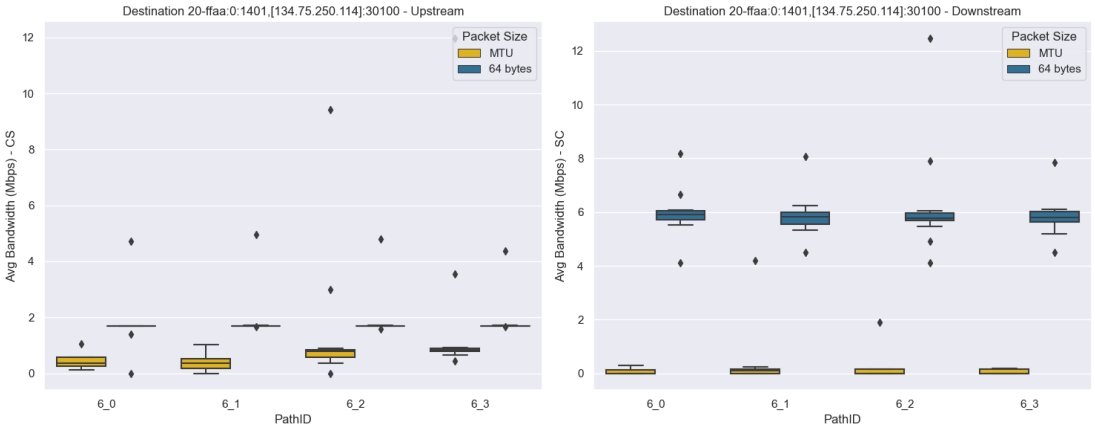

Figure 5.13: Average bandwidth values for each path, requiring a bandwidth of 150Mbps from and to a Korea Server (address on the top). On the left side there are the upstream measurements, while on the right side the downstream ones.

In the figure above (5.12 and 5.13), we observe an higher achieved bandwidth by sending smaller packets instead of bigger ones. This looks counter-intuitive because of the overhead of packet headers. Hence, our guess is that the network may not have sufficient capacity to handle the desired 150Mbps bandwidth for MTU-sized packets. Even though MTU-sized packets are more efficient in terms of payload-to-overhead ratio, the network may be congested or have limited capacity, causing packet drops and resulting in a lower achieved bandwidth. Indeed, dropping 64 bytes packets does not decrease the achieved bandwidth as dropping MTU-sized packets. This is an insight that requires further analysis even though it is suggested by all the destinations involved.

5.1.3 Packet Loss Assessment

The last but not the least assessment regards the packet loss ratio. Packet loss in networking refers to the situation where data packets transmitted over a network fail to reach their intended destination. It is a common occurrence in network communication and can be caused by various factors such as network congestion, hardware failures, signal interference, or software errors.
When a data packet is sent from a source to a destination, it traverses various network devices and links. If any of these devices or links are overloaded, faulty, or experiencing high traffic, they might drop or discard some packets to alleviate the congestion. This results in packet loss. Packet loss can have significant implications on network performance and the quality of user experience. It can lead to re-transmissions, delayed data delivery, and reduced throughput. In real-time applications like video conferencing or online gaming, even a small amount of packet loss can result in noticeable disruptions, such as frozen video frames or audio glitches. Monitoring and managing packet loss is essential for maintaining network reliability and performance. Hence, we want to assess this phenomenon to prevent a user by choosing routes with a high packet loss ratio. Figure 5.14 shows the average packet loss percentage for each path available to reach AWS destination in Northern Virginia, USA (AS `16-ffaa:0:1003,[172.31.19.144]` in topology 5.1). In the graph each path is represented with a different colored dot[2]. The dot size stands for the number of measurements having the same packet loss ratio.

[2]After an interval of 9 paths, colors are re-used.

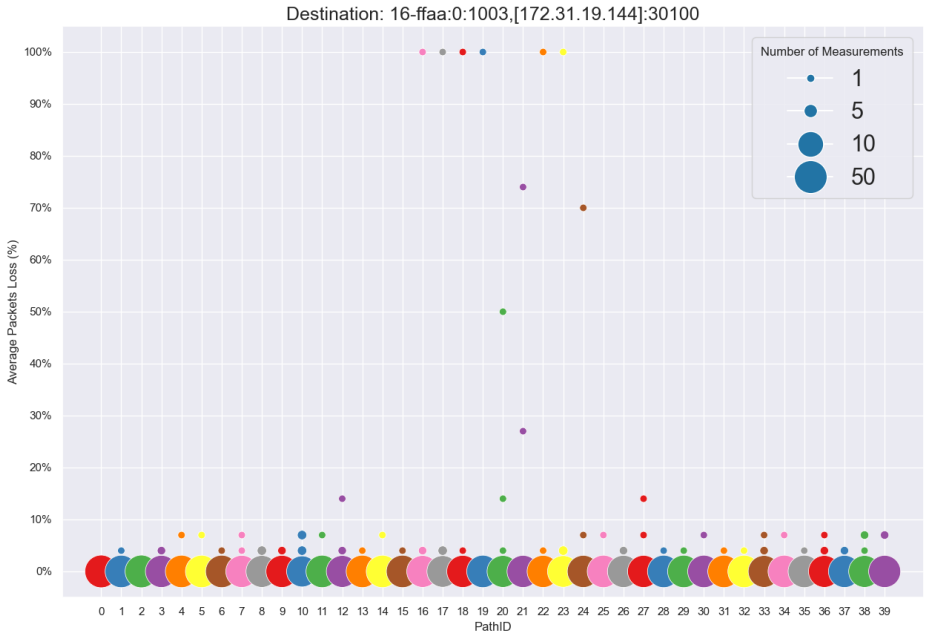

Figure 5.14: Average packet loss percentage for each path of AWS US N. Virginia AS. Each dot color represents a path and its size the number of measurements having the same loss ratio. Dots legend is on the upper right corner.

In Figure 5.14, we consistently observe that the majority of paths exhibits a loss ratio of 0%, with a few instances occasionally reaching almost the 10% mark. However, within this context, there are particular paths that notably register a complete 100% loss rate, as evidenced by paths `2_16`, `2_17`, `2_18`, `2_19`, `2_22`, and `2_23`. These instances of complete loss merit our attention, warranting exploration into potential factors at play, such as network congestion. By looking at the sequence of hops for each of these paths, a commonality emerges: the shared nodes are only those concentrated in the first half of the path. Moreover, since these measurements were carried out in succession (due to the consecutive nature of the paths), our hypothesis is that one or more of these common nodes experienced a period of congestion.
This distinct pattern repeats itself in Figure 5.15 as well. Here, the most of measurements similarly exhibits a 0% loss rate, with a few noteworthy exceptions. These consecutive isolated cases reinforce the notion of an uncommon network behavior deserving of further investigation.

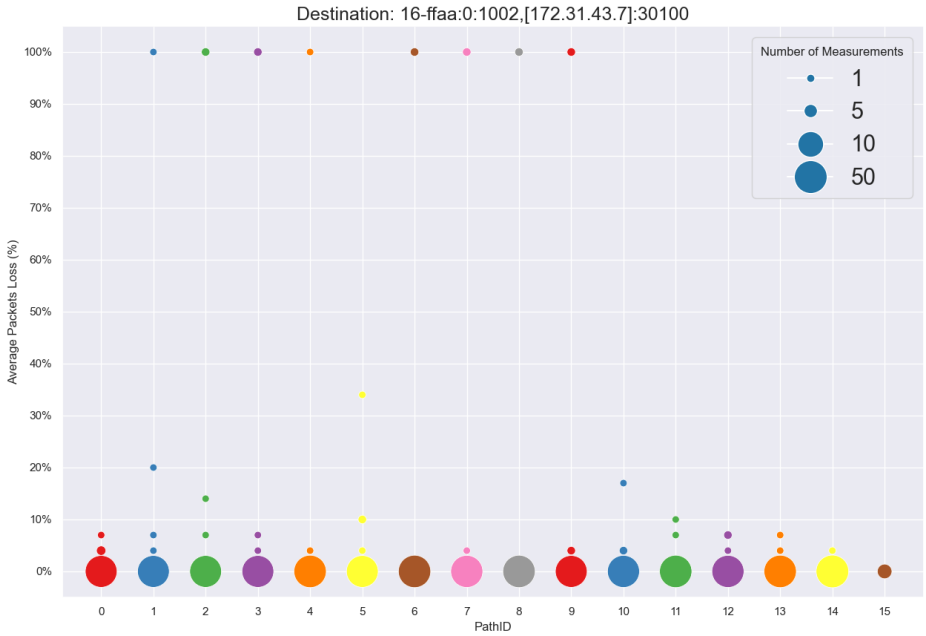

Figure 5.15: Average packet loss percentage for each path of AWS Ireland AS. Each dot color represents a path and its size the number of measurements having the same loss ratio. Dots legend is on the upper right corner.

5.2 Considerations

This path analysis concludes with several interesting insights, showcasing the role that the test-suite can play in fostering informed and thoughtful path selections. In terms of latency, we have experienced that the most effective element is the physical distance among the nodes building the path and that the number of hops or the ISDs set alone are not enough to determine a latency variation. We have then observed the bandwidth limitations of the SCIONLab network, that according to our hypothesis is due to an insufficient capacity. About packet loss, the network results to be stable with sporadic events of complete loss.
In a nutshell, the path selection feature of SCION, when coupled with a robust test-suite and data analysis techniques, blend into a powerful tool that helps to fulfill the controllability requirement of a UPIN user. Our goal is to satisfy this mandate not only through the implementation of a test-suite, but also by providing a user interface. Hence, the next chapter will discuss the details of the path recommendation system that has been developed using the collected paths data.

Chapter 6

Path Recommendation

The test-suite, detailed in Chapter 4, and the comprehensive analysis presented in Chapter 5, collectively offer a practical guide for users to make informed decisions when selecting the optimal path that aligns with their specific requirements.
However, it's important to note that, currently, SCION [4] allows users to choose paths without providing detailed information about their characteristics. Users are often left to make path selections based solely on destinations. This can lead to sub-optimal choices and a lack of awareness regarding the network's performance along those paths.
This is where the test-suite comes into play. It acts as a data-gathering tool, providing in-depth insights into the characteristics of the available paths, including factors such as latency, bandwidth, and packet loss. Though, to truly empower users, these insights need to be presented in a user-friendly and accessible manner.
The path recommendation system acts as a bridge between the wealth of data collected by the test-suite and the end users. With its user-friendly interface, it allows users to aggregate, navigate, and, most importantly, understand this data. It transforms raw statistics into actionable information, making it easier for users to make choices that align with their specific needs and preferences.
In this chapter, we will delve into the architecture and functionality of the path recommendation system. We will explore how it takes the complex data about path characteristics and distills it into a user-friendly format. This system represents the final piece in the puzzle, ensuring that users can harness the full potential of SCION to create a more efficient and responsive network experience.

6.1 Path Recommendation: Architecture

The path recommendation system has been developed as a web application, a choice made to ensure its broad accessibility to users and make the service platform independent. As for the path selection testing suite, it follows a 3-tier architecture but with different interactions compared to the ones described in Figure 4.1. There are three main components interacting:

- **Client or Front-end**: This is the user-facing part of the system, providing a straightforward and intuitive interface for users to interact with. Users can easily make requests and receive path recommendations through this component.

- **Server or Back-end**: The server plays a central role in the system's operation. It is responsible for handling user requests, querying the database for path information, and ensuring overall system efficiency, even under high traffic loads. The server's performance optimizations allow for a seamless and responsive user experience.

- **NoSQL Cloud-based Database**: This component represents the cloud-based NoSQL database, specifically MongoDB Atlas in our case. It serves as the repository for a comprehensive set of information about available paths and performances, retrieved through the usage of the test-suite discussed in Chapter 4. Its adoption enables efficient data storage and retrieval for the path recommendation system. The cloud-based nature ensures scalability, accessibility, and reliability of the database.

By dividing the system into these three distinct components, we've created a flexible and accessible tool that can effectively cater to a wide range of users' path selection needs. The client offers user-friendly interactions, while the server manages the underlying data efficiently, enabling the application to perform well even during periods of high demand. Moreover, the choice of a cloud-based database ensures that the system's data is stored securely in a remote database, facilitating data management, scalability, and accessibility.

The interaction among these components is illustrated in Figure 6.1. A user, engaging with the front-end, functions as a client, sending requests to the server. The server acts as a logic tier, querying the database to generate a response delivered back to the client.

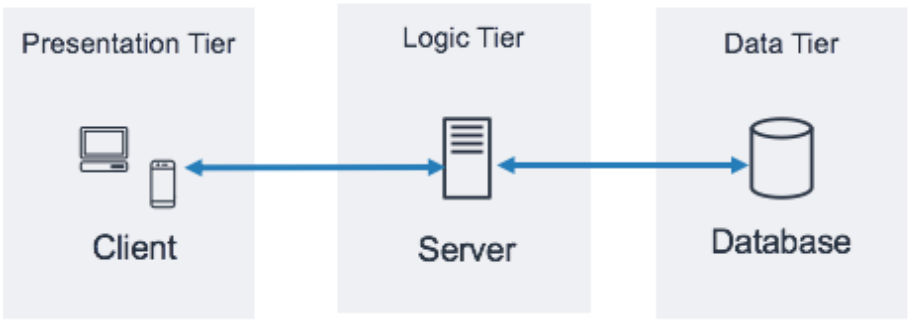

Figure 6.1: Path Recommendation System Architecture: A 3-Tier Architecture[1]where the client initiates a request, the server processes it by interacting with a database, and provides the corresponding response.

6.1.1 Technologies Involved

In order to ensure the intended functionality on both client and server, delivering the promised features, the path recommendation system employs an advanced technology stack. We will now dive into this stack for both client and server components.

Client Side

Client-side technologies predominantly focus on the user interface as they define the interface directly accessible to users. In the context of this application, we have embraced the following technologies:

- **React**: It is also known as React.js, stands as an open-source JavaScript library originating from Facebook [13]. It is particularly renowned for its application in crafting user interfaces (UIs) for web applications, especially in the context of single-page and mobile applications. React's reputation rests on its efficiency, simplicity, and its capability to produce highly responsive and interactive user interfaces. The decision to employ React in this application was driven by its key functionalities, which include:

[1]Source: `https://docs.aws.amazon.com/`

- **Component-Based Architecture**: React revolves around the concept of components. These components are independent, reusable building blocks of the user interface. Each component can represent a specific UI element, be it a button, form, or even an entire section of a web page. The strength of React lies in its capacity to seamlessly combine these components to create intricate and multifaceted user interfaces.
- **Virtual DOM (Document Object Model)**: React employs a virtual representation of the DOM. Rather than directly manipulating the real DOM, React updates only those components affected by changes in data. This approach offers a substantial improvement over updating the entire real DOM, which can be comparatively slower and less efficient.
- **Efficient Updates**: React incorporates a diffing algorithm that calculates the disparity between the previous virtual DOM and the new one. Subsequently, it updates solely the portions of the UI that have undergone alterations. This strategy effectively minimizes the workload imposed on the browser, leading to swifter updates and a more responsive UI.

- **Material UI**: Material-UI [14], often abbreviated as MUI, is a popular open-source user interface (UI) framework for building web applications. It provides a set of reusable, customizable, and well-designed UI components that follow the Material Design guidelines, a design language developed by Google. Material Design is known for its clean, modern, and visually appealing design principles. Among its key features, we can highlight that is React-based (it is a set of React components), it also enhances accessibility which is an important consideration for modern web applications and it is highly customizable, leading to the creation of a smooth and personalized application.

Server Side

The server-side is powered by a set of technologies that work together to ensure the system operates efficiently and smoothly:

- **JavaScript and Node**: JavaScript and Node.js form the foundation of the server-side. Node.js, a runtime environment, is particularly adept at handling server operations and facilitates the seamless communication between the client and the database.
- **MongoClient Library**: The server interfaces with MongoDB Atlas, a popular cloud-based database service. This interaction is made possible through the use of the MongoClient library, which acts as the bridge between the server and MongoDB. This enables the efficient retrieval and storage of data, a critical function in the system.

- **Express Framework**: To further enhance routing and service management, we've implemented the Express framework. Express is known for streamlining server responsibilities, providing a well-organized and efficient environment. It ensures that user requests are handled with precision, database interactions are optimized, and the overall system performs smoothly, even during peak usage periods.

These technologies, both on the client and server sides, work cohesively to deliver a robust and user-friendly path recommendation system. The client's REACT-based interface ensures that users can easily interact with the system, while the server, powered by JavaScript, Node.js, MongoClient, and Express, efficiently manages user requests, database interactions, and overall system performance, even during peak usage.

6.2 Implementation

In this section we will explore the implementation details of the path recommendation system, diving deep into both Front-end components and Back-end API. We will examine some code snippets and provide the page rendering.

6.2.1 Front-end components

The path recommendation system has been implemented as a single-page application, adopting a bottom-up approach. Leveraging React's component-oriented nature, we defined small, modular components that were seamlessly combined to achieve the desired behavior.
Before delving into a detailed analysis of each component, let's take an overview of the key elements:

- **Navbar**: The navbar serves as a central control hub, providing users with the functionality to access the menu. Its extensibility allows for the smooth addition of future features, ensuring adaptability as the system evolves.
- **Path Recommendation Form**: Accessible from the navbar, the path recommendation form empowers users to define specific requirements for the desired path. This component acts as a user-friendly interface, facilitating the input of criteria that guide the system in generating tailored path recommendations.
- **Path List**: The path list component acts as a wrapper for smaller components and is responsible for displaying every path that meets the requirements specified and submitted through the Path Recommendation Form. Its modular design allows for flexibility and scalability in presenting a comprehensive view of the available paths.

– **Single Path Component**: Each path within the path list is represented by an individual component. This component not only displays the relevant details of the path but also incorporates a command line block featuring a copy button. This feature proves handy for users who wish to employ a specific path in SCIONLab.

Figure 5.1 illustrates the appearance of the page upon initial opening (*landscape version of this picture is available at Appendix D, figure D.1*).

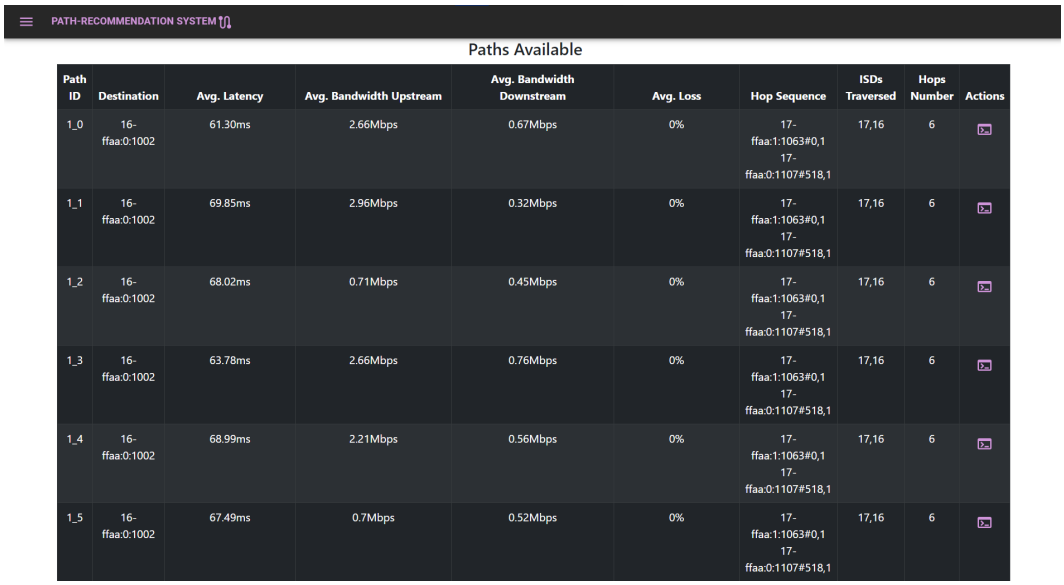

Figure 6.2: The front-end view of the Path Recommendation System upon opening, showcasing the initial list of paths retrieved from the database before any filtering is applied.

Navbar Component

On top of the front-end page (6.2), we introduce the Navbar component. As of the current implementation, its primary purpose is to facilitate access to the left-side menu. The Navbar component is defined in the following code snippet:

```
function MyNavbar(props){
    const setMenuShow = props.setMenuShow;
    const menuShow = props.menuShow;
    const form = props.form;
    const setForm = props.setForm;
    const pathsList = props.pathsList;
    const setPathsList = props.setPathsList;
```

```
8
9       return (
10          <Container className='p-0' fluid>
11              <AppBar position="sticky">
12                  <Toolbar variant="dense">
13                      <div style={{marginRight: "27.5%"}}
    className='mr-5 d-flex justify-content-center align-
    items-center'>
14                      /* Additional components or
    functionalities may be added here */
15                          <FormDrawer unfinished={props.
    unfinished} setUnfinished={props.setUnfinished}
    destinationsList={props.destinationsList}
    setDestinationsList={props.setDestinationsList}
    selectedDestination={props.selectedDestination}
    setSelectedDestination={props.setSelectedDestination}
     pathsList={pathsList} setPathsList={setPathsList}
    form={form} setForm={setForm} menuShow={menuShow}
    setMenuShow={setMenuShow}/>
16                          <Typography variant="button"
    color="secondary" component="div">
17                              <b>Path-Recommendation
    System</b><RouteIcon/>
18                          </Typography>
19                      </div>
20                  </Toolbar>
21              </AppBar>
22          </Container>
23      );
24  }
```

The component introduced above integrates essential features, notably the `FormDrawer`. This internal component, custom-designed by us, serves to smoothly reveal the form (detailed in 6.2.1) as a drawer from the left side of the page and facilitates interactions with form elements. To enhance styling and achieve a unified and visually engaging user interface, the component leverages the AppBar and Toolbar components from the Material-UI library [14]. The underlying code structure is organized around the effective use of props, enabling streamlined state management and seamless data exchange across various components. This approach contributes to the overall modularity and maintainability of the codebase.

Path Recommendation Form Component

The `PathRecommendationForm` function is a crucial component equipped with functionalities that enable users to customize and filter paths. This form leverages React's state management and integrates with the overall system through props. Here's a brief breakdown:

- **Form Handling**: The form manages various parameters such as latency, bandwidth, and other criteria relevant to path selection.
- **Filtering Mechanism**: Upon submission, the form triggers an asynchronous function (`handleSubmit`) that communicates with the server API to fetch paths based on the specified criteria. The form's dynamic nature allows users to input different values (e.g. minimum and maximum latency, bandwidth, loss...) influencing the filtering process. It's noteworthy that the available filters include the option to specify a set of Isolated Domains to avoid. This functionality empowers users to filter paths, excluding those associated with specific domains deemed untrustworthy.
- **Dropdown Menu**: The form includes a dropdown menu (`CustomizedMenus`) that facilitates the selection of a destination, enhancing user interaction and providing a user-friendly experience.
- **Event Handling**: Event handlers, such as `onChange` for text fields and the form submission (`onSubmit`), are incorporated to ensure a responsive and interactive form.
- **Modular Structure**: The code embraces modularity, utilizing props to manage state and communicate with parent components. This enhances maintainability and flexibility in adapting to future modifications.

This design emphasizes a user-centric approach, aiming to provide an intuitive and efficient means for users to tailor their path preferences. The following code snippet presents a condensed version of the original component to lighten the reading, while maintaining its core functionality.

```
function PathRecommendationForm(props){
    const form = props.form;
    const setForm = props.setForm;
    const setPathsList = props.setPathsList;
    const setUnfinished = props.setUnfinished;

    async function handleSubmit(event) {
        event.preventDefault();
```

```
9           try {
10              setUnfinished(true);
11              const paths = await API.loadFilteredPaths(
    form);
12              setPathsList(paths);
13              setUnfinished(false);
14              props.setMenuShow(false);
15            } catch (error) {
16              console.log(error);
17            }
18      }
19
20  return(
21  <>
22  <Box onSubmit={(event) => handleSubmit(event)} className
    ='mt-5' sx={{ '& .MuiTextField-root': { m: 2, width:
    '25ch' }, }} component='form'>
23      <div className='px-3 mb-3 d-flex flex-column justify
    -content-center'>
24          <Typography className='mx-5 text-center' sx={{
    fontSize: 24 }} variant="button" color='secondary'
    display="block" gutterBottom>
25              Filter Your Paths
26          </Typography>
27          <CustomizedMenus value={form.minLatency}
    onChange={(e) => setForm({...form, destination: e.
    target.value})} form={form} setForm={setForm}
    destinationsList={props.destinationsList}
    setDestinationsList={props.setDestinationsList}
    selectedDestination={props.selectedDestination}
    setSelectedDestination={props.setSelectedDestination
    }/>
28      </div>
29      <Divider className='mx-5' role="presentation" light>
30          <Chip variant='outlined' color='secondary' sx={{
     fontWeight: 'bold' }} label="Latency" className='
    font-weight-bold' />
31      </Divider>
32      <div className='px-3 d-flex justify-content-around'>
```

```
33          <TextField value={form.minLatency} onChange={(e)
     => setForm({...form, minLatency: e.target.value})}
    color='secondary' id="outlined-basic" label="min.
    Latency" variant="outlined" />
34          <TextField value={form.maxLatency} onChange={(e)
     => setForm({...form, maxLatency: e.target.value})}
    color='secondary' id="outlined-basic" label="max.
    Latency" variant="outlined"/>
35      </div>
36      //... Other params to submit
37      <div className='px-3 d-flex justify-content-center'>
38          <Button type='submit' color='secondary' variant=
    "outlined" endIcon={<TuneIcon/>}>Filter</Button>
39      </div>
40  </Box>
41  </>
42  )
43  }
```

When rendered, this component presents the following representation:

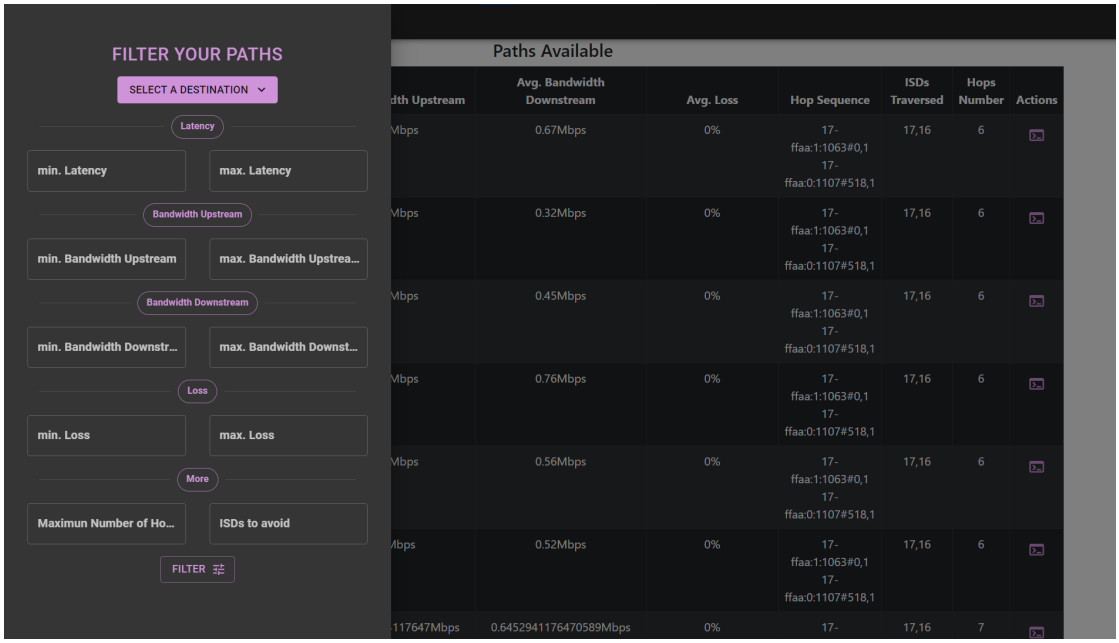

Figure 6.3: Path Recommendation Form: A user interface enabling users to filter the path list based on various criteria.

A wider and clearer version of the above picture has been provided in the Appendix D, with the figure D.2.

Path List Component

The `PathsList` component serves as the visual representation of a comprehensive table showcasing available paths. Its dynamic population of table rows encapsulates crucial details like Path ID, Destination, Average Latency, and all the other parameters defining a path. This table operates as a container for individual `PathRows`, internal components defining the rendering of each row. The `PathRow` itself acts as a wrapper for `PathData` objects, responsible for modeling the content of each cell within the table.
During the path-fetching process, the component incorporates a loading spinner, contributing to a more responsive user interface. The table's design emphasizes responsiveness and expansiveness, facilitating a user-friendly exploration of the available paths.
Following is presented a code snippet modelling the `PathList`.

```
function PathsList(props){
return <>
    <Row>
      <Col><h4>Paths Available</h4></Col>
    </Row>
    {props.unfinished ? <Box className='mt-5 w-100 h-100
    justify-content-center' sx={{ display: 'flex' }}>
                <CircularProgress sx={{color:"#AB47BC"}}
   size='10%'/>
              </Box> :
    <Table className='table-responsive m-auto mb-2 table
   -expandable' striped bordered hover variant="dark"
   style={{maxWidth: "75%", margin: 0}}>
        <thead>
            <tr>
                <th>Path ID</th>
                <th>Destination</th>
                <th>Avg. Latency</th>
                <th>Avg. Bandwidth Upstream</th>
                <th>Avg. Bandwidth Downstream</th>
                <th>Avg. Loss</th>
                <th>Hop Sequence</th>
                <th>ISDs Traversed</th>
                <th>Hops Number</th>
```

```
                    <th>Actions</th>
                </tr>
            </thead>
            <tbody>
            {

              props.pathsList.map((p) =>
                    <PathRow destinationsList={props.
    destinationsList} setDestinationsList={props.
    setDestinationsList} selectedCodeBlock={props.
    selectedCodeBlock} setSelectedCodeBlock={props.
    setSelectedCodeBlock} showCode={props.showCode}
    setShowCode={props.setShowCode} color='primary' key={
    p.id} path={p}/>)
            }
            </tbody>
      </Table>}
      <BasicModal selectedCodeBlock={props.
    selectedCodeBlock} setSelectedCodeBlock={props.
    setSelectedCodeBlock} showCode={props.showCode}
    setShowCode={props.setShowCode}/>
</>;
}
```

Furthermore, the component integrates a modal window (`BasicModal`) that enables additional actions on selected paths. Essentially, it provides users with a bash command that can be copied and executed directly within SCIONLab to perform a `traceroute` on the defined destination using the chosen path.
Figures 6.2 and D.1 display the table rendering, while the modal window is presented below in figure 6.4 and in the Appendix D with the landscape picture D.3.

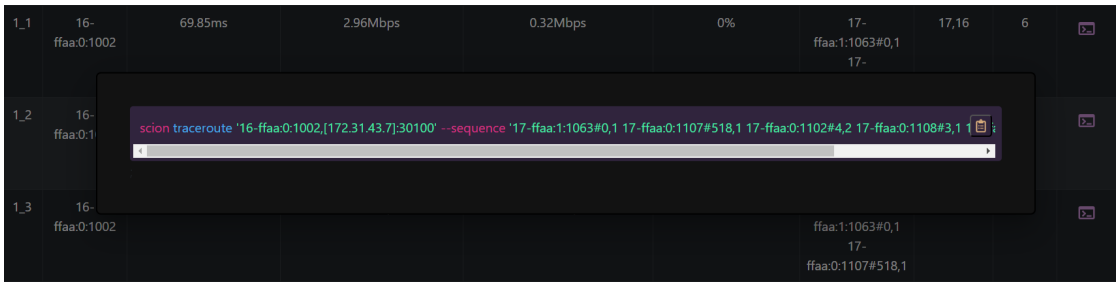

Figure 6.4: Modal for copying `traceroute` command with path-specific details.

6.2.2 Back-end APIs

As we delve into the backend of our path recommendation system, understanding the role of APIs in powering the visual and interactive components is crucial. These APIs serve as the communication bridge between the client and server, constituting the backbone of the application.
When the application initiates or the user triggers an operation requiring a server response, the client's API is invoked, responsible for reaching the appropriate server-side endpoint. Subsequently, the server undertakes computational tasks, processing the user-defined constraints by querying a MongoDB database in the cloud. Specifically, MongoDB Atlas is our cloud-based NoSQL database. It houses extensive information about available paths. Server-side queries play a pivotal role, managing operations like path filtering, detailed information retrieval, and performance metric calculations. The RESTful API endpoints establish routes for diverse operations, ensuring seamless communication between the front-end and back-end.
As an illustrative example, we will explore a single API connected to a server-side endpoint, aiming to provide a concise overview of the data flow.

API 1: Loading the list of Paths

The `loadPathsList()` function is an asynchronous operation initiated by the `PathList` component when the associated page is accessed, to load all the paths available on the database. It sends a request to the server-side endpoint, specifically targeting the `http://localhost:3001/api/paths/` route.
Upon receiving a successful response (*HTTP status 200*), the function processes the data, transforming it into a collection of "Path" objects. These objects encapsulate essential information needed to represent each path in the table. In the event of an unsuccessful response, an error is thrown, providing insights into the encountered issue. The following code provides the function described:

```
async function loadPathsList() {
    const url = SERVER_URL + '/paths/';
    try {
        let response;
        response = await fetch(url);

        if (response.ok) {
            const arr =  await response.json()
            const pathList = arr.map((p) =>
```

```
            new Path(p.id, p.destination, p.avgLatency,
    p.avgBandwidthCs64, p.avgBandwidthSc64, p.
    avgBandwidthCsMTU, p.avgBandwidthScMTU, p.avgLoss, p.
    timestamp, p.hopsNumber, p.hopsSequence, p.
    isolatedDomains));
            return pathList;
        }
        else {
            const text = await response.text();
            throw new TypeError(text);
        }
    } catch(err){
        throw err
    }
}
```

Upon execution, the server processes the request through the specified endpoint:

```
app.get('/api/paths/', async (_req, res) => {
  const destination = null;

  /* Invoking the database function to retrieve
   * paths based on the provided constraints
   */
  db.getPaths(destination, null)
    .then((paths) => {
      /* Responding with a JSON object containing
       * the retrieved paths on success
       */
      return res.status(200).json(paths);
    })
    .catch((err) => {
      /* Responding with a JSON object containing the
       * encountered error on failure
       */
      return res.status(500).json(err);
    });
});
```

This endpoint is tailored for handling GET requests, retrieving paths from the database based on specified constraints. In this specific case, both the destination and other filters are set to `null`, indicating that all paths are requested without

any filtering, and that all available destinations are acceptable. Upon successful retrieval, the paths are returned as a JSON object with a status code of 200. In case of an error during the database query, an error object with a status code of 500 is returned.

The `getPaths` function, referred to in this endpoint, is designed to retrieve all paths from the database and their associated statistics. Following, we provide a compact version of this function, describing the defined scenario, which means path retrieval without any filtering. The detailed listing of this function, which includes operations for various filtering scenarios, is available in Appendix D, listing D.1.

```
/*This function retrieve all Paths from the DB
  and their stats*/
  const getPaths = async (dest, filters) => {
    const pathsList = [];
    const regexPattern = dest ? "\\b${dest}#\\d+$" : "\\
   b.*#\\d+$";
    let pipeline = [];

    if(filters !== null){
      //Check the Appendix
    }else{
      pipeline = [
        {
          $match: {
            hops: {$regex: new RegExp(regexPattern)},
            avg_bandwidth_sc_MTU: { $ne: "Information
   not available" },
            avg_latency: {$ne : "0ms"}
          }
        },
        {
          $addFields: {
            idWithoutTimestamp: { $substr: ["$_id", 0, {
    $subtract: [{ $strLenCP: "$_id" }, 27] }] },
            pathNum: {$toInt: { $substr: ["$_id", 2, {
   $subtract: [{ $strLenCP: "$_id" }, 29] }] }},
            destNum: {$toInt: { $substr: ["$_id", 0, 1
   ]}}
          }
        },
        {
```

```
27          $addFields: {
28            avg_latency_number: { $toDouble: {$substr: [
    "$avg_latency", 0, { $subtract: [{ $strLenCP: "
    $avg_latency" }, 2] }] }},
29            avg_loss_number: { $toDouble: { $arrayElemAt
    : [{ $split: ["$avg_loss", "%"] }, 0] } },
30            avg_bandwidth_sc_MTU_number: { $toDouble: {
    $arrayElemAt: [{ $split: ["$avg_bandwidth_sc_MTU", "
    Mbps" ] }, 0] } },
31            avg_bandwidth_cs_MTU_number: { $toDouble: {
    $arrayElemAt: [{ $split: ["$avg_bandwidth_cs_MTU", "
    Mbps" ] }, 0] } },
32          }
33        },
34        {
35          $group: {
36            _id: "$idWithoutTimestamp",
37            avg_latency: { $avg: "$avg_latency_number"
    },
38            avg_bandwidth_sc_MTU: { $avg: "
    $avg_bandwidth_sc_MTU_number" },
39            avg_bandwidth_cs_MTU: { $avg: "
    $avg_bandwidth_cs_MTU_number" },
40            avg_loss: { $avg: "$avg_loss_number" },
41            hops: { $first: "$hops" },
42            hops_number: { $first: "$hops_number" },
43            isolated_domains: { $first: "
    $isolated_domains" },
44            pathNum: {$first: "$pathNum"},
45            destNum: {$first: "$destNum"}
46          }
47        },
48        {
49          $sort: {
50            destNum:1,
51            pathNum: 1
52          }
53        },
54        {
55          $project: {
56                _id: 1,
```

```
57                  avg_latency: { $concat: [{$toString: "
    $avg_latency"},"ms"]},
58                  avg_bandwidth_sc_MTU: { $concat: [{
    $toString: "$avg_bandwidth_sc_MTU"},"Mbps"]},
59                  avg_bandwidth_cs_MTU: { $concat: [{
    $toString: "$avg_bandwidth_cs_MTU"},"Mbps"]},
60                  avg_loss: { $concat: [{$toString: "
    $avg_loss"},"%"]},
61                  hops_number: 1,
62                  hops: 1,
63                  isolated_domains: 1,
64                }
65          }
66          ];
67        }
68
69      try {
70          const db = await connectToDatabase();
71          const collection = db.collection("paths_stats");
72          const paths = await collection.aggregate(
    pipeline).toArray();
73
74          for(const path of paths){
75              const hops = path.hops.split(" ")
76              const destination = hops[hops.length - 1].
    split("#")[0];
77              pathsList.push(new Path(path._id,
    destination, path.avg_latency, null, null, path.
    avg_bandwidth_cs_MTU, path.avg_bandwidth_sc_MTU, path
    .avg_loss, null, path.hops_number, path.hops, path.
    isolated_domains));
78          }
79
80          return pathsList;
81      } catch (error) {
82          throw error;
83      }
84 }
```

The function employs a MongoDB aggregation pipeline to process the data. The

pipeline includes operations such as matching the regex pattern for paths, calculating average metrics, sorting, and projecting the necessary fields.

6.2.3 Concluding Remarks

To encapsulate the knowledge gained in this chapter, let's revisit the foundational components and interactions that define the Path Recommendation System. The user journey commences with interactions on the user-friendly front-end, crafted with React.js, where constraints for path selection are defined. These constraints are then transmitted to the server via well-defined APIs. Utilizing Node.js and Express, the server processes these requests by querying a MongoDB Atlas database for pertinent path information. Subsequently, the server furnishes the client with computed paths, elegantly displayed in a tabular format on the front-end. This streamlined process ensures a responsive experience, enabling users to effortlessly filter paths based on diverse criteria.
This chapter aims to provide a holistic understanding of the system's robustness and its capability to deliver precise and relevant path recommendations. With the comprehensive details now at our disposal, the next chapter is poised to encapsulate our conclusions, offering a synthesized perspective on the entirety of the work presented thus far.

Chapter 7

Conclusion

In addressing the overarching question of how to empower users to control the paths of their data in a network, the UPIN project [8] delves into the potential and limitations of SCION for user-driven path control. This research explores various facets of SCION's performance, with a keen focus on latency, bandwidth, and data loss in SCIONLab, an experimental testbed.

Our findings confirm that latency in SCIONLab is predominantly influenced by the physical distance between nodes, surpassing the impact of factors like the number of hops or the ISDs traversed. Additionally, we've uncovered intriguing insights into the bandwidth limitations of the SCIONLab network, revealing a decrease in capacity when targeting higher bandwidth paths, a phenomenon warranting further investigation. Despite variations, packet loss remains relatively stable in most cases.

The software architecture, underpinned by a database housing data on numerous paths, plays a crucial role in our investigation, showcasing the implications of shifting control from network operators to end users on network performance.

While the current SCION path selection paradigm often leaves users reliant on endpoint destinations without detailed information on path characteristics, the introduced test-suite addresses this gap. Offering insights into factors such as latency, bandwidth, and packet loss. Then, the integration of a Path Recommendation System signifies a substantial advancement. It acts as a bridge, presenting this detailed data in a user-friendly and accessible manner. With its intuitive interface, the system allows users to aggregate, navigate, and comprehend this information, transforming raw statistics into actionable insights that align with users' specific needs and preferences.

Finally, the integration of a path selection feature in SCION, complemented by a robust test-suite and data analysis techniques, emerges as a powerful tool for fulfilling the controllability requirements of a UPIN user. Beyond the implementation of a test-suite, the introduction of a sophisticated Path Recommendation System marks a significant leap forward. This system not only empowers users to make informed choices based on network performance but also serves as a key avenue for future research.

7.1 Future Works

As we chart the course for future developments, several avenues present themselves to enhance and expand upon the existing capabilities of the Path Recommendation System. One prominent direction involves the integration of advanced Machine Learning (ML) techniques. By leveraging ML algorithms, we can analyze historical usage patterns, user preferences, and real-time network conditions to dynamically recommend the most optimal paths. This infusion of intelligence would contribute to a more adaptive and responsive system. Another critical focus for future enhancements centers on refining the user interface. Introducing a mapping feature that visually represents a partial network topology would empower users with a more intuitive understanding of the underlying network structure. Interactive elements, such as the ability to click on specific nodes, could significantly improve the user experience, making path selection more transparent and user-friendly. To streamline the user experience even further, we envisage integrating a Secure Shell (SSH) connection directly into the system. This feature would allow users to execute the application with the recommended path seamlessly. By facilitating direct interactions with the network, this integration aims to enhance the efficiency of network diagnostics and testing. Looking towards broader integration possibilities, we plan to extend the availability of the path recommendation system by providing it as an Application Programming Interface (API). This move would empower developers to seamlessly integrate path recommendations into their own tools and systems. By encapsulating the functionality within an API, we aspire to extend the utility of the system, making it a versatile tool for diverse applications beyond its standalone interface.
In summary, these envisioned future works underscore our commitment to advancing user-driven path control. By integrating machine learning, refining the user interface, introducing SSH connections, and providing API accessibility, we aim to evolve the Path Recommendation System into a more intelligent, user-friendly, and versatile tool.

Appendix A

Experimental Setup

Appendix A serves as a supplementary resource to enrich the content discussed in Chapter 3. Here, we offer more extensive details and additional visuals, enhancing your grasp of the experimental setup intricacies.

Figure A.1 is a landscape version of 3.1. We want to provide further details about it reiterating that the orange colored nodes represent the Core ASes which are the root of trust of other ASes within the same ISD. It is worth mentioning that in this topology, some ISDs have only Core ASes, like: *ISD 16 AWS* and *ISD 26 KREONET2*. This highlights their key role as bridges with other ISDs, rather than conventional clusters of ASes catering to end-users. Moreover, the green colored nodes are the Attachment Points of each ISD, used to allow users attaching their own ASes and contribute to the global topology with experimenters' computational resources.
The attachement points available for SCIONLab are:

- **Magdebourg AP**: this Access Point is placed in Germany, indeed it is within the ISD of Europe. Its SCION address is: `19-ffaa:0:1303`.
- **ETH-Hell-AP**: this Access Point is placed in Switzerland, within the ISD 17, and refers to the ETH University. Its SCION address is: `17-ffaa:0:1107`.
- **ETHZ-AP**: it is the second access point of the ETH and it is the one where we attached our AS to perform all the experiments. Its SCION address is: `17-ffaa:0:1113`.
- **CMU AP**: it is placed in North America, in ISD 18. Its SCION address is: `18-ffaa:0:1206`.
- **KU AP**: this Access Point is hosted in Korea, within the ISD 20. Its SCION address is: `20-ffaa:0:1404`.

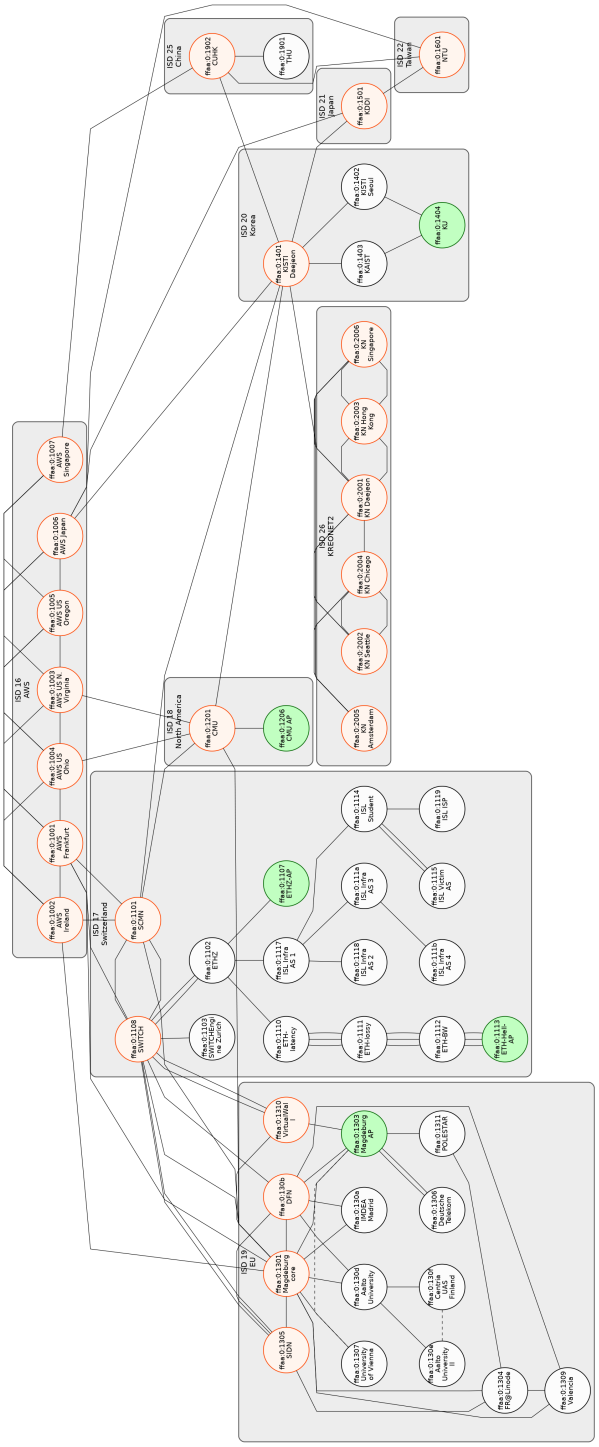

Figure A.1: SCION Topology - Landscape version of 3.1

Appendix B

Software Architecture

This appendix extends the insights presented in Chapter 4 by providing additional details and code listings. Here, we delve deeper into the intricacies of the software architecture, unraveling specific code snippets that were condensed for clarity in the main chapter. This supplementary section aims to offer a more comprehensive reference for those seeking a detailed exploration of the system's software foundations.

Listing B.1: Listing of function `path_info_building(server)`. Extended version of listing 4.4.1.

```
1  def path_info_building(server):
2      paths_to_be_inserted = []
3      scion_addr_cmd = "scion address"
4      scion_addr_proc = subprocess.Popen(scion_addr_cmd,
   shell=True, stdout=subprocess.PIPE, stdin=subprocess.
   PIPE)
5      source_addr = scion_addr_proc.stdout.readline()
6      source_addr = source_addr.decode('utf-8').rstrip()
7
8      server_destination_address_sp = server["
   source_address"].split(",")[0]
9
10     #execute scion showpaths command
11     cmd = f"scion showpaths {
   server_destination_address_sp} --extended -m 40"
12
13     proc = subprocess.Popen(cmd, shell=True, stdout=
   subprocess.PIPE, stdin=subprocess.PIPE)
14
```

```
15      # Read the output of the command and store it in a
    list
16      output = []
17      dirty_path_info = []
18      hops_number = 0
19
20      min_hops = 2000
21      while True:
22          line = proc.stdout.readline()
23          paths = re.match(r"\d+ Hops:", line.decode('utf
    -8').rstrip())
24          if paths:
25              hops_number = paths.group().split(" ")[0]
26              if int(hops_number) < min_hops:
27                  min_hops = min(min_hops, int(hops_number
    ))
28                  print("Minimum Hops: " + str(min_hops))
29              paths = False
30          if not line or int(hops_number) > min_hops+1:
31              break
32          output.append(line.decode('utf-8').rstrip())
33
34      # Join the lines into a single string
35      output_text = '\n'.join(output)
36
37      pattern = r"\[ ?(\d+)\] Hops: \[([^]]+)\]\s+MTU: (\d
    +)\s+NextHop: ([^\s]+)\s+Expires: ([^\n]+)\s+Latency:
     ([^\n]+)\s+Status: ([^\n]+)\s+LocalIP: ([^\n]+)"
38      matches = re.findall(pattern, output_text)
39      #Almost good path info but I need to change the
    format of the hops field
40      for match in matches:
41          path_info = {
42              "Path_ID": match[0],
43              "Hops": match[1],
44              "MTU": match[2],
45              "Latency": match[5],
46              "Status": True if match[6] == 'alive' else
    False,
47          }
48          dirty_path_info.append(path_info)
```

```
49
50      # Now I need to change the format of the hops field
51      for path in dirty_path_info:
52          old_hop_field = path["Hops"]
53          new_hop_field = convert_hop_predicates(
    old_hop_field.split(" "))
54
55          new_path = {
56              "_id": str(server['_id']) + "_" + str(path["
    Path_ID"]),
57              "hop_predicates": new_hop_field,
58              "MTU": path["MTU"],
59              "expected_min_latency": path["Latency"],
60              "active": path["Status"],
61              "destination_address": server["
    source_address"],
62              "source_address": source_addr,
63          }
64
65          paths_to_be_inserted.append(new_path)
66
67      return paths_to_be_inserted
```

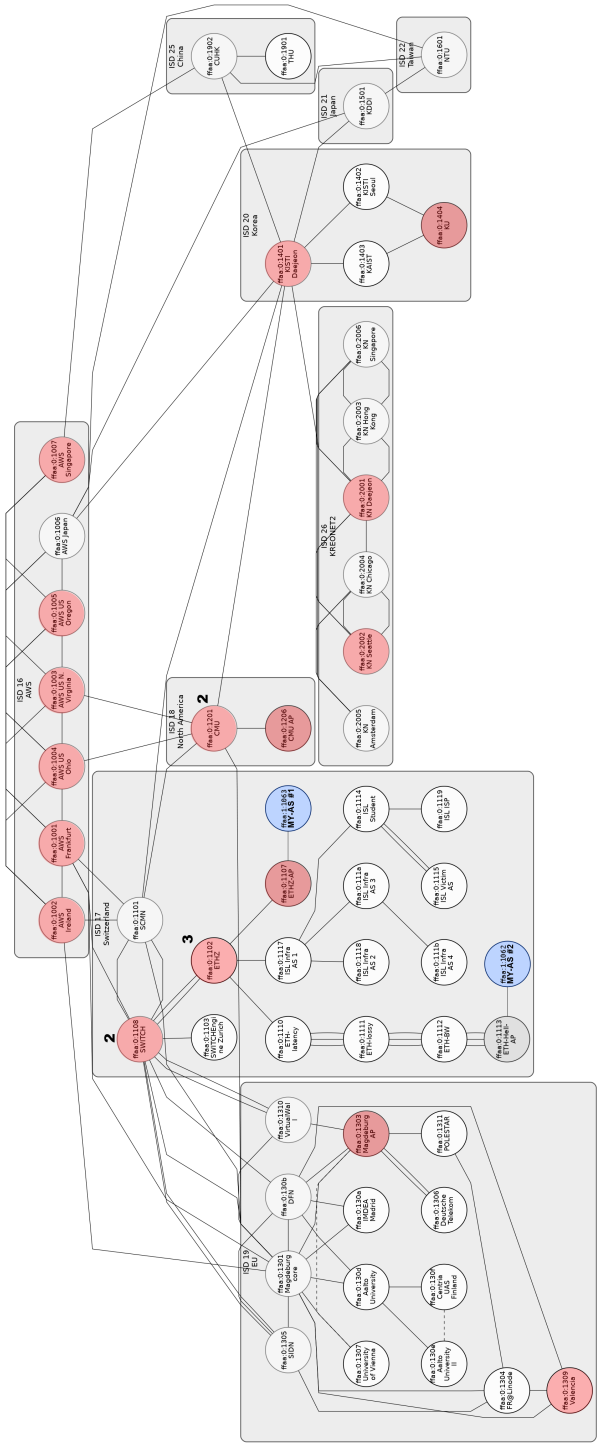

Figure B.1: SCION Topology with Available Servers - Landscape version of 4.2

Appendix C

Path Selection

Appendix C supplies additional information regarding the analysis of Path Selection feature, discussed in Chapter 5.

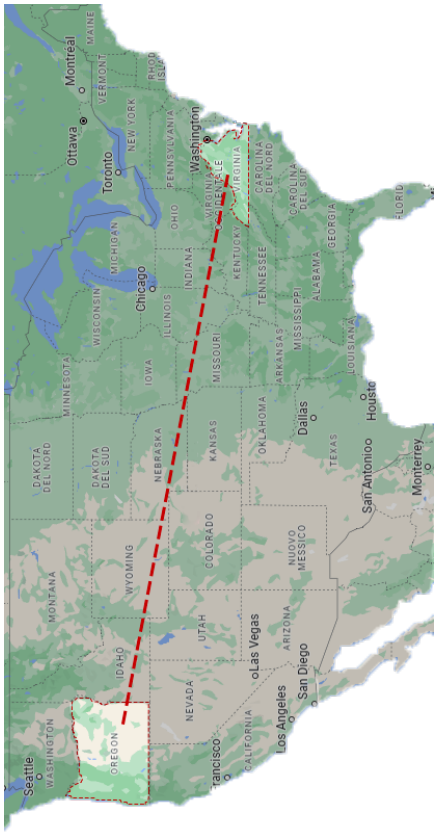

Figure C.1: USA Map enhancing the distance between Virginia and Oregon, both marked with a dotted red line. - Landscape version of 5.8

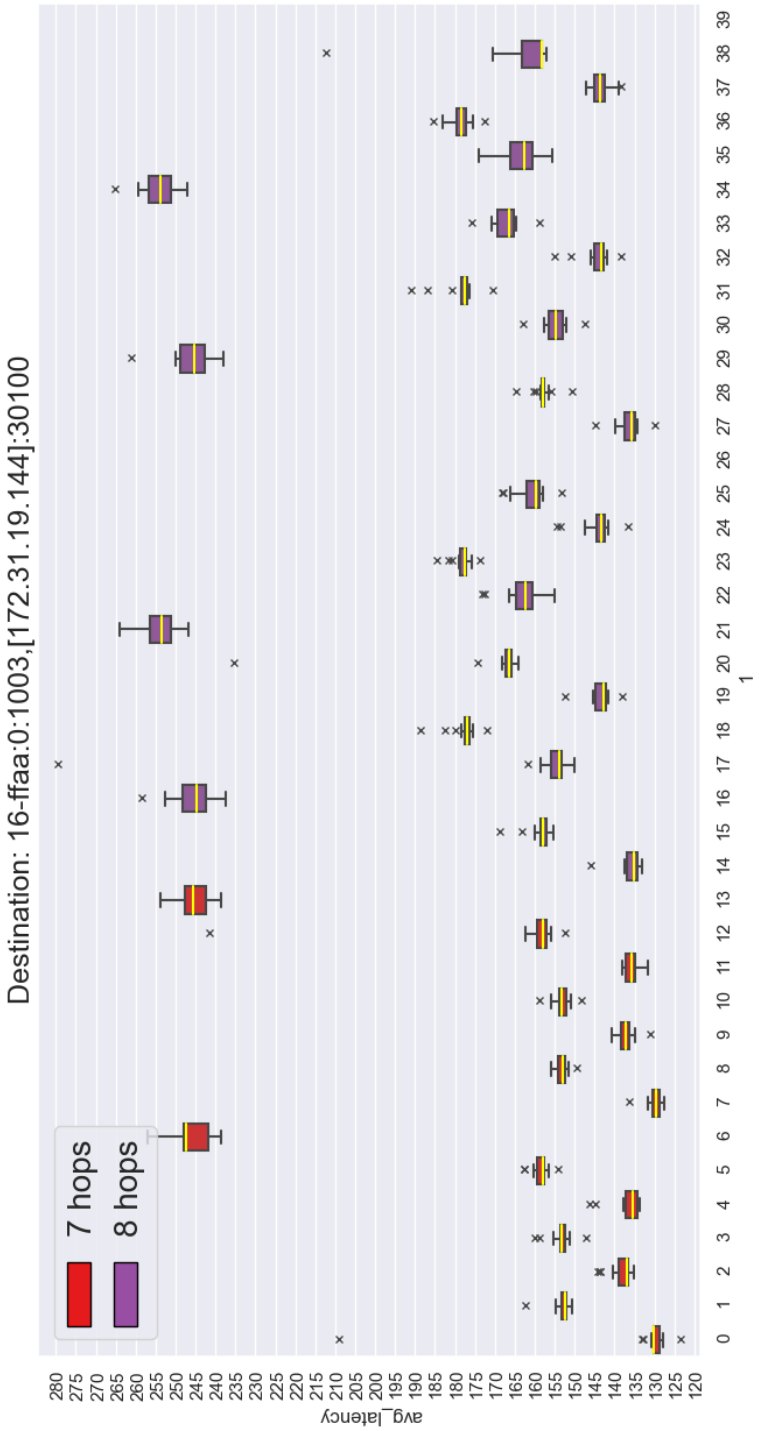

Figure C.2: Average Latency Values measured for each path of destination 16-ffaa:0:1007,[171.31.19.144] (AWS - US N. Virginia). - Landscape version of 5.7

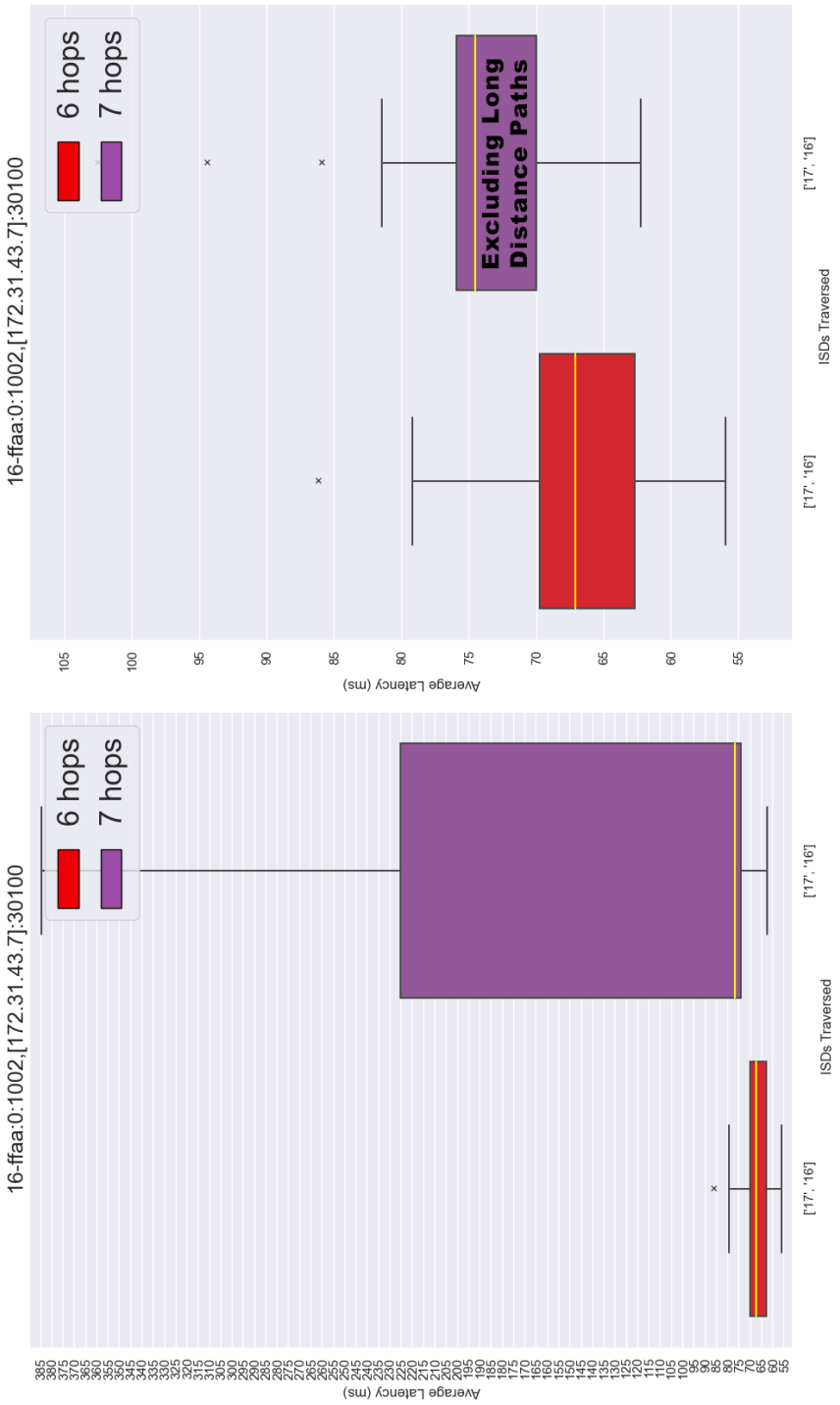

Figure C.3: Comparison of Average Latency between ISD Sets, with and without Long-Distance Paths. - Landscape version of 5.9

Appendix D

Path Recommendation

Appendix D complements the insights presented in Chapter 6 by offering additional details about the Path Recommendation System and its front-end. This section unveils intriguing code listings that were abbreviated for enhanced readability. Moreover, it features landscape images strategically placed to deepen your understanding of previously introduced concepts. This appendix is designed to provide a comprehensive reference for those seeking a more in-depth exploration of the system's intricacies.

Listing D.1: Listing of function `getPaths(destination, filters)`. Extended version of listing 6.2.2.

```
1 /*This function retrieve all Paths from the DB
2   and their stats*/
3 const getPaths = async (dest, filters) => {
4     const pathsList = [];
5     const regexPattern = dest ? `\\b${dest}#\\d+$` : `\\
    b.*#\\d+$`;
6     let pipeline = [];
7
8     if(filters !== null){
9       const minLatency = filters.minLatency === '' ? 0 :
     parseFloat(filters.minLatency);
10       const maxLatency = filters.maxLatency === '' ?
    Number.MAX_VALUE : parseFloat(filters.maxLatency);
11       const minBandwidthUpstream = filters.
    minBandwidthUpstream === '' ? 0 : parseFloat(filters.
    minBandwidthUpstream);
```

```
12      const maxBandwidthUpstream = filters.
    maxBandwidthUpstream === '' ? Number.MAX_VALUE :
    parseFloat(filters.maxBandwidthUpstream);
13      const minBandwidthDownstream = filters.
    minBandwidthDownstream === '' ? 0 : parseFloat(
    filters.minBandwidthDownstream);
14      const maxBandwidthDownstream = filters.
    maxBandwidthDownstream === '' ? Number.MAX_VALUE :
    parseFloat(filters.maxBandwidthDownstream);
15      const minPacketLoss = filters.minPacketLoss === ''
     ? 0 : parseFloat(filters.minPacketLoss);
16      const maxPacketLoss = filters.maxPacketLoss === ''
     ? Number.MAX_VALUE : parseFloat(filters.
    maxPacketLoss);
17      const maxNumberOfHops = filters.maxNumberOfHops
    === '' ? Number.MAX_VALUE : parseInt(filters.
    maxNumberOfHops);
18      const isdsToAvoid = filters.isdsToAvoid === '' ?
    [] : filters.isdsToAvoid.split(',');
19      const f = {
20        $match: {
21          avg_latency: {$gte : minLatency, $lte :
    maxLatency},
22          avg_bandwidth_sc_MTU: {$gte :
    minBandwidthUpstream, $lte : maxBandwidthUpstream},
23          avg_bandwidth_cs_MTU: {$gte :
    minBandwidthDownstream, $lte : maxBandwidthDownstream
    },
24          avg_loss: {$gte : minPacketLoss, $lte :
    maxPacketLoss},
25          hops_number: {$lte : maxNumberOfHops},
26          isolated_domains: {$nin : isdsToAvoid}
27        }
28      };
29      pipeline = [
30        {
31          $match: {
32            hops: {$regex: new RegExp(regexPattern)},
33            avg_bandwidth_sc_MTU: { $ne: "Information
    not available" },
34            avg_latency: {$ne : "0ms"}
```

```
35          }
36        },
37        {
38          $addFields: {
39            idWithoutTimestamp: { $substr: ["$_id", 0, { $subtract: [{ $strLenCP: "$_id" }, 27] }] },
40            pathNum: {$toInt: { $substr: ["$_id", 2, { $subtract: [{ $strLenCP: "$_id" }, 29] }] }},
41            destNum: {$toInt: { $substr: ["$_id", 0, 1]}}
42          }
43        },
44        {
45          $addFields: {
46            avg_latency_number: { $toDouble: {$substr: ["$avg_latency", 0, { $subtract: [{ $strLenCP: "$avg_latency" }, 2] }] }},
47            avg_loss_number: { $toDouble: { $arrayElemAt: [{ $split: ["$avg_loss", "%"] }, 0] } },
48            avg_bandwidth_sc_MTU_number: { $toDouble: { $arrayElemAt: [{ $split: ["$avg_bandwidth_sc_MTU", " Mbps" ] }, 0] } },
49            avg_bandwidth_cs_MTU_number: { $toDouble: { $arrayElemAt: [{ $split: ["$avg_bandwidth_cs_MTU", " Mbps" ] }, 0] } },
50          }
51        },
52        {
53          $group: {
54            _id: "$idWithoutTimestamp",
55            avg_latency: { $avg: "$avg_latency_number" },
56            avg_bandwidth_sc_MTU: { $avg: "$avg_bandwidth_sc_MTU_number" },
57            avg_bandwidth_cs_MTU: { $avg: "$avg_bandwidth_cs_MTU_number" },
58            avg_loss: { $avg: "$avg_loss_number" },
59            hops: { $first: "$hops" },
60            hops_number: { $first: "$hops_number" },
61            isolated_domains: { $first: "$isolated_domains" },
```

```
62            pathNum: {$first: "$pathNum"},
63            destNum: {$first: "$destNum"}
64          }
65        },
66        f,
67        {
68          $sort: {
69            destNum:1,
70            pathNum: 1
71          }
72        },
73        {
74          $project: {
75                _id: 1,
76                avg_latency: { $concat: [{$toString: "
    $avg_latency"},"ms"]},
77                avg_bandwidth_sc_MTU: { $concat: [{
    $toString: "$avg_bandwidth_sc_MTU"},"Mbps"]},
78                avg_bandwidth_cs_MTU: { $concat: [{
    $toString: "$avg_bandwidth_cs_MTU"},"Mbps"]},
79                avg_loss: { $concat: [{$toString: "
    $avg_loss"},"%"]},
80                hops_number: 1,
81                hops: 1,
82                isolated_domains: 1,
83              }
84        }
85        ];
86     }else{
87       pipeline = [
88         {
89           $match: {
90             hops: {$regex: new RegExp(regexPattern)},
91             avg_bandwidth_sc_MTU: { $ne: "Information
    not available" },
92             avg_latency: {$ne : "0ms"}
93           }
94         },
95         {
96           $addFields: {
```

```
97            idWithoutTimestamp: { $substr: ["$_id", 0, { $subtract: [{ $strLenCP: "$_id" }, 27] }] },
98            pathNum: {$toInt: { $substr: ["$_id", 2, { $subtract: [{ $strLenCP: "$_id" }, 29] }] }},
99            destNum: {$toInt: { $substr: ["$_id", 0, 1 ]}}
100         }
101       },
102       {
103         $addFields: {
104           avg_latency_number: { $toDouble: {$substr: [ "$avg_latency", 0, { $subtract: [{ $strLenCP: "$avg_latency" }, 2] }] }},
105           avg_loss_number: { $toDouble: { $arrayElemAt: [{ $split: ["$avg_loss", "%"] }, 0] } },
106           avg_bandwidth_sc_MTU_number: { $toDouble: { $arrayElemAt: [{ $split: ["$avg_bandwidth_sc_MTU", " Mbps" ] }, 0] } },
107           avg_bandwidth_cs_MTU_number: { $toDouble: { $arrayElemAt: [{ $split: ["$avg_bandwidth_cs_MTU", " Mbps" ] }, 0] } },
108         }
109       },
110       {
111         $group: {
112           _id: "$idWithoutTimestamp",
113           avg_latency: { $avg: "$avg_latency_number" },
114           avg_bandwidth_sc_MTU: { $avg: "$avg_bandwidth_sc_MTU_number" },
115           avg_bandwidth_cs_MTU: { $avg: "$avg_bandwidth_cs_MTU_number" },
116           avg_loss: { $avg: "$avg_loss_number" },
117           hops: { $first: "$hops" },
118           hops_number: { $first: "$hops_number" },
119           isolated_domains: { $first: "$isolated_domains" },
120           pathNum: {$first: "$pathNum"},
121           destNum: {$first: "$destNum"}
122         }
123       },
```

```
124         {
125           $sort: {
126             destNum:1,
127             pathNum: 1
128           }
129         },
130         {
131           $project: {
132                 _id: 1,
133                 avg_latency: { $concat: [{$toString: "
    $avg_latency"},"ms"]},
134                 avg_bandwidth_sc_MTU: { $concat: [{
    $toString: "$avg_bandwidth_sc_MTU"},"Mbps"]},
135                 avg_bandwidth_cs_MTU: { $concat: [{
    $toString: "$avg_bandwidth_cs_MTU"},"Mbps"]},
136                 avg_loss: { $concat: [{$toString: "
    $avg_loss"},"%"]},
137                 hops_number: 1,
138                 hops: 1,
139                 isolated_domains: 1,
140               }
141         }
142         ];
143       }
144      try {
145         const db = await connectToDatabase();
146         const collection = db.collection("paths_stats");
147         const paths = await collection.aggregate(
    pipeline).toArray();
148 
149         for(const path of paths){
150             const hops = path.hops.split(" ")
151             const destination = hops[hops.length - 1].
    split("#")[0];
152             pathsList.push(new Path(path._id,
    destination, path.avg_latency, null, null, path.
    avg_bandwidth_cs_MTU, path.avg_bandwidth_sc_MTU, path
    .avg_loss, null, path.hops_number, path.hops, path.
    isolated_domains));
153         }
154 
```

```
155             return pathsList;
156         } catch (error) {
157             throw error;
158         }
159 }
```

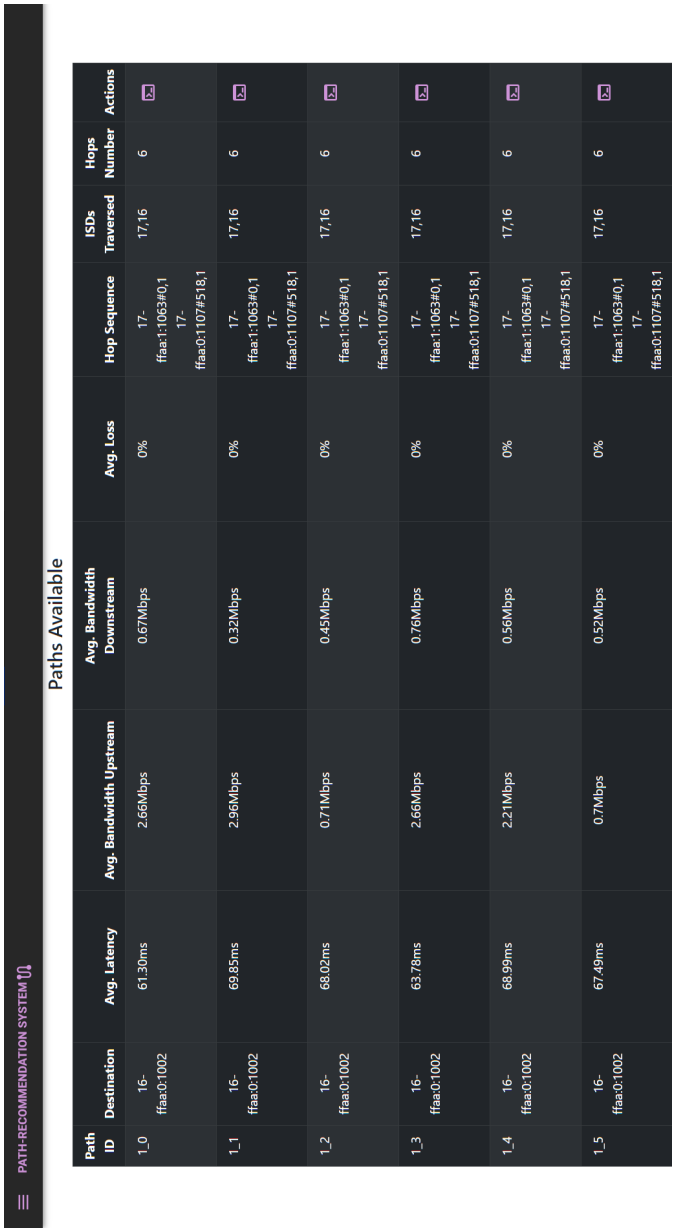

Figure D.1: Front-end view of the Path Recommendation System upon opening, displaying retrieved paths from the Database. - Landscape version of 6.2

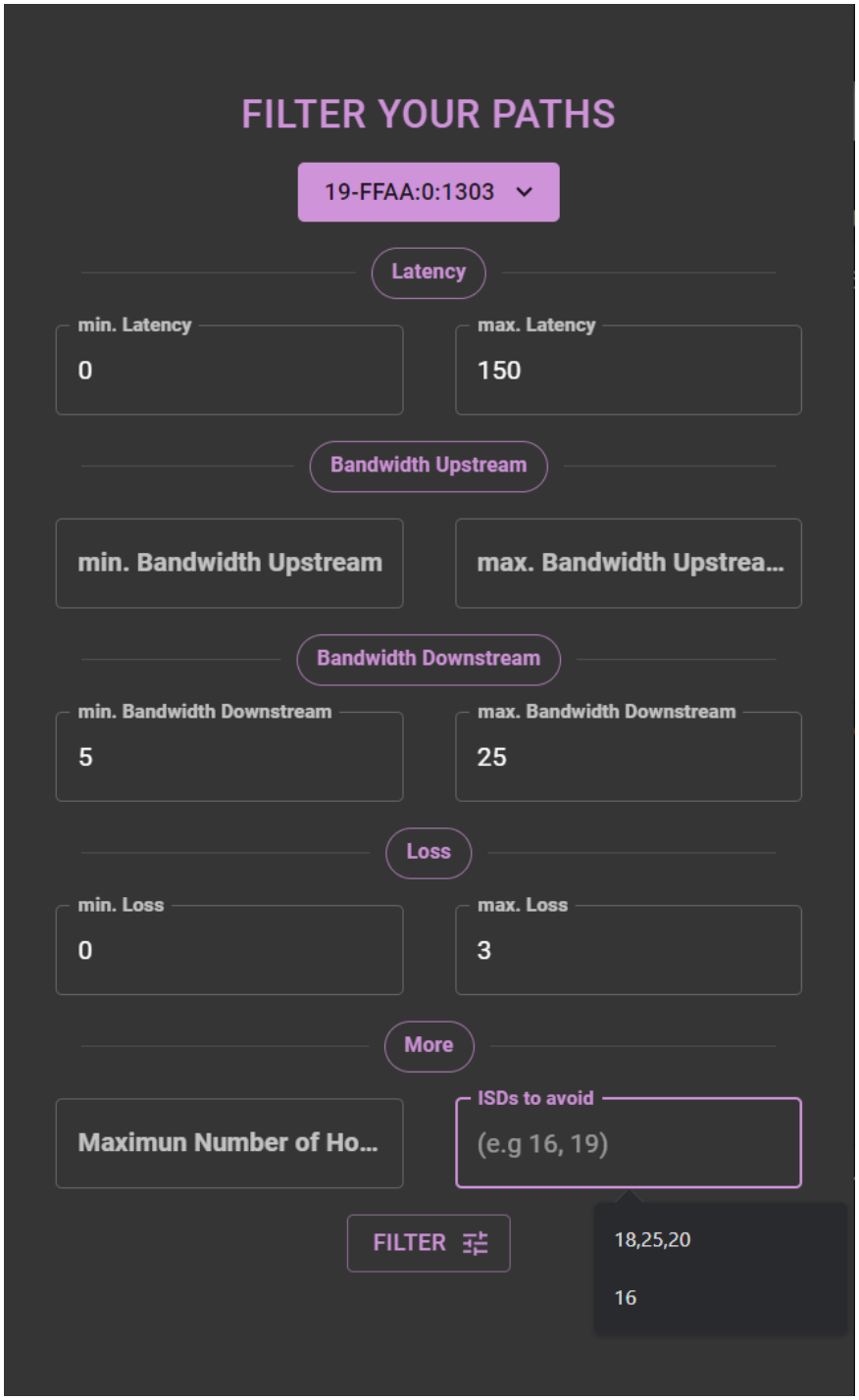

Figure D.2: Path Recommendation Form: A user interface enabling users to filter the path list based on: destination, latency, bandwidth, loss, number of hops and ISDs to avoid.

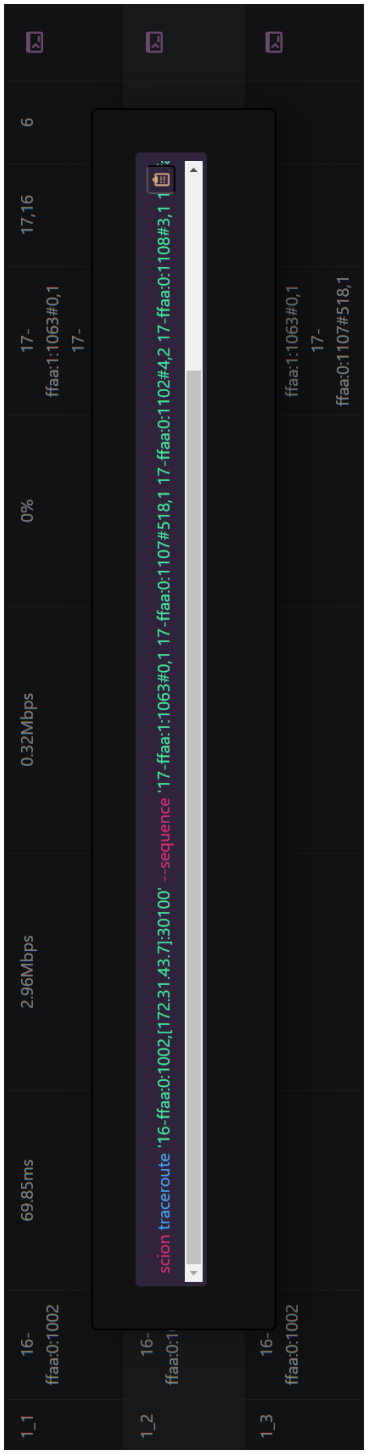

Figure D.3: Modal Window appearing in response to user interaction with a path. It facilitates the copying of the `traceroute` command, complete with pre-defined information specific to the selected path. - Landscape version of 6.4

Acknowledgements

Ho riservato questo spazio per ringraziare coloro che hanno fatto parte, in un modo o in un altro, di questo percorso. Mettetevi comodi, sarò prolisso.

Un primo e doveroso ringrazimento va fatto senz'altro, al mio relatore, il **Prof. Valenza**. Dal primo giorno ha saputo supportarmi al meglio, mostrandosi sempre disponibile e dimostrandomi sostegno e fiducia durante tutto il percorso di tesi. Se oggi sono qui, a raccogliere i frutti di questo lavoro, è anche grazie a lei.

Ringrazio **Paola** e **Leonardo**, coloro che mi hanno accolto, supervisionato e aiutato durante il mio soggiorno ad Amsterdam. Dal primo giorno mi hanno fatto sentire a casa. Mi hanno spronato a portare avanti un lavoro che fosse frutto di una mia ricerca, spesso rivolgendosi a me come ad un collega più che ad uno studente, credendoci più di quanto non facessi io stesso. Da questa collaborazione ho avuto anche la possibilità di vedere pubblicato ciò per cui ho lavorato 5 mesi. Non smetterò mai di ringraziarvi abbastanza per questa opportunità.

Ringrazio la mia mamma e il mio papà: **Teresa** e **Francesco**, travi portanti della mia vita. Giorno dopo giorno, avete contribuito con un mattoncino a rendermi la persona che sono oggi. Avete festeggiato i miei traguardi e mi avete aiutato a rialzarmi dopo una caduta. Grazie a voi ho sperimentato cosa significa donare totalmente sé stessi a qualcuno. Vi devo ogni cosa.

Ringrazio mio fratello, **Alessandro**, per essere diventato parte centrale della mia vita, rendendosi la persona su cui fare affidamento senza mai dubitarne. La tua fiducia nei miei confronti e il tuo chiedere un mio consiglio anche quando non ne so nulla a riguardo, mi spronano a fare meglio nell'aiutarti a raggiungere ogni cosa che possa renderti felice in questa vita.

Ringrazio i miei nonni: **Nino**, **Antonietta**, **Lucio** e **Rina**. Siete stati i miei capi ultras negli ultimi 24 anni, gioendo per ogni evento della mia vita ed esaltando le

mie capacità ben oltre la realtà. Il vostro sostegno è stato la mia forza.

Ringrazio mia zia **Manuela** per essermi stata accanto nonostante la distanza, non facendomi mai mancare affetto o un appoggio a cui affidarsi. Mi hai trattato da adulto sin da quando adulto non lo ero affatto, aiutandomi a crescere e ad affermarmi nel *"mondo dei grandi"*.

Ringrazio mio zio **Mario**, la persona che ha creduto nel mio percorso e nelle mie scelte durante tutto il tragitto. Le sue parole di saggezza, il suo impegno e l'interesse che ha messo nel cercare di aiutarmi e di non farmi mai mancare nulla, mi hanno aiutato a raggiungere questo obiettivo.

Ringrazio la persona con cui condivido la *"cella"* da cinque anni e mezzo: **Federica**. I ringraziamenti per lei potrebbero sostituire il contenuto di questa tesi. Sei in grado di emozionarti e commuoverti per i miei traguardi più di quanto io possa mai fare. L'amore, la pazienza e la gentilezza che mi dimostri ogni giorno mi rendono una persona migliore.

Ringrazio i miei cugini: **Lucio** ed **Emanuele**. Negli ultimi anni ho particolarmente sentito l'affetto che mi avete dato e la voglia di starmi accanto. In maniera diversa siete stati un modello da seguire per me.

Ringrazio **Jacopo**, per aver condiviso questo percorso con me. L'ansia di un esame, o per l'attesa di un voto, la fatica per eccellere in qualcosa e la felicità per aver raggiunto un traguardo, sono tutte esperienze che abbiamo fatto insieme. In quanto amico, mi hai aiutato ogni volta che sono stato in difficoltà, non solo accademica, sacrificando il tuo tempo per farmi superare un ostacolo. In quanto ingegnere, l'impegno che metti in un lavoro e la meticolosità con cui lo svolgi sono stati fonte di ispirazione per me, mi hai reso un ingegnere migliore.

Ringrazio **Lele**, per la sua incrollabile stima nei miei riguardi. Nella nostra amicizia hai saputo ricoprire ogni ruolo, dalla spalla su cui piangere per una delusione, all'amico di sempre, quello che puoi chiamare alle 4 del mattino e con cui puoi ridere fino a smettere di respirare. Nonostante i tuoi mille impegni, hai sempre trovato il modo per alleggerire un po' il mio zaino, durante questa camminata.

Ringrazio **Donato**, per la sua innata dote all'ascolto. Hai sempre dato ampio spazio alle mie parole, porgendomi le giuste domande e donandomi preziosi consigli. Hai dato valore alla mia breve esperienza in questa vita, chiedendomi ugualmente un parere anche quando il tuo sguardo poteva arrivare al di là del mio. Il rapporto di condivisione che abbiamo costruito mi aiuta a migliorare e crescere ogni giorno.

Ringrazio **Benny** e **Stefano**, per essere stati miei *"compagni di viaggio"* per un po'. Anche se le nostre strade accademiche si sono separate due anni fa, abbiamo continuato a vivere l'amicizia, condividendo insieme momenti di felicità ma anche di tristezza. Avervi accanto nonostante la distanza fisica mi ha fatto vivere questa esperienza con più serenità, con la consapevolezza che le nostre strade avranno sempre modo di ricongiungersi.

Ringrazio **Michele** e **Rocco**, per continuare a far ancora parte della mia vita dopo 12 anni, sebbene la distanza. In maniera diversa, condividiamo moltissime cose ed obiettivi. Nel corso degli anni abbiamo continuato a motivarci e spingerci oltre l'obiettivo, forse a volte dimenticandoci di fermarci a festeggiare. A voi devo proprio questo tipo di sostegno, mi spronato ad inseguire i miei sogni.

Ringrazio *"gli amici del quarto piano"*: **Akhil**, **Caterina**, **Davide**, **Jabroot** e **Luana**. Siete riusciti ad alleggerire la distanza da casa, rendendo quel piccolo angolo di Torino un luogo accogliente in cui è bello stare. Le cene insieme, le arance lanciate e le porte scassinate, le porterò sempre con me.

Infine, ringrazio ogni altra persona con cui ho avuto modo di confrontarmi lungo questo cammino, ogni altro amico che mi è stato accanto. Negli ultimi otto mesi ho vissuto in quattro città diverse, in tre stati diversi ed ho cambiato otto volte casa in meno di sei mesi. Non avrei mai potuto farcela senza avere accanto tutti coloro che mi hanno sostenuto.

Se mi facessero la solita e famosa domanda: *"Cos'è più importante: il viaggio o la destinazione?"*, mi sentirei di rispondere, senza alcun dubbio: *"La compagnia"*.

Grazie ancora per essere stati la mia.

Bibliography

[1] Tomi Dufva and Mikko Dufva. «Grasping the future of the digital society». In: *Futures* 107 (2019), pp. 17–28 (cit. on p. 1).

[2] Cristian Hesselman et al. «A responsible internet to increase trust in the digital world». In: *Journal of Network and Systems Management* 28.4 (2020), pp. 882–922 (cit. on pp. 1, 3).

[3] Rodrigo Bazo, Leonardo Boldrini, Cristian Hesselman, and Paola Grosso. «Increasing the Transparency, Accountability and Controllability of multi-domain networks with the UPIN framework». In: *Proceedings of the ACM SIGCOMM 2021 Workshop on Technologies, Applications, and Uses of a Responsible Internet*. 2021, pp. 8–13 (cit. on pp. 1, 3, 4).

[4] Xin Zhang, Hsu-Chun Hsiao, Geoffrey Hasker, Haowen Chan, Adrian Perrig, and David G. Andersen. «SCION: Scalability, Control, and Isolation on Next-Generation Networks». In: *2011 IEEE Symposium on Security and Privacy*. 2011, pp. 212–227. DOI: 10.1109/SP.2011.45 (cit. on pp. 1, 3, 5–7, 11, 17, 48).

[5] Antonio Battipaglia, Leonardo Boldrini, Ralph Koning, and Paola Grosso. «Evaluation of SCION for User-Driven Path Control: A Usability Study». In: *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*. SC-W '23. Denver, CO, USA: Association for Computing Machinery, 2023, pp. 785–794. ISBN: 9798400707858. DOI: 10.1145/3624062.3624592. URL: https://doi.org/10.1145/3624062.3624592 (cit. on pp. 2, 5).

[6] Vincenzo Maffione, Francesco Salvestrini, Eduard Gras, Leonardo Bergesio, and Miquel Tarzan. «A Software Development Kit to exploit RINA programmability». In: May 2016. DOI: 10.1109/ICC.2016.7510711 (cit. on p. 3).

[7] Jonghoon Kwon, Juan A. García-Pardo, Markus Legner, François Wirz, Matthias Frei, David Hausheer, and Adrian Perrig. «SCIONLAB: A Next-Generation Internet Testbed». In: *2020 IEEE 28th International Conference*

on Network Protocols (ICNP). 2020, pp. 1–12. DOI: 10.1109/ICNP49622.2020.9259355 (cit. on pp. 9, 11, 17, 34).

[8] Leonardo Boldrini, Rodrigo Bazo, Cristian E.W. Hesselman, and Paola Grosso. «UPIN - A shift in network control from operator to end user». English. In: ICT Open 2021, ICT.OPEN2021 ; Conference date: 10-02-2021 Through 11-02-2021. 2021. URL: https://www.ictopen.nl/ (cit. on pp. 17, 66).

[9] Fernando Pérez and Brian E. Granger. «IPython: a System for Interactive Scientific Computing». In: *Computing in Science and Engineering* 9.3 (May 2007), pp. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53. URL: https://ipython.org (cit. on p. 36).

[10] Michael L. Waskom. «seaborn: statistical data visualization». In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021 (cit. on p. 36).

[11] J. D. Hunter. «Matplotlib: A 2D graphics environment». In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55 (cit. on p. 36).

[12] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134 (cit. on p. 36).

[13] Accomazzo Anthony, Murray Nathaniel, and Lerner Ari. *Fullstack React: The Complete Guide to ReactJS and Friends*. Fullstack.io, 2017. ISBN: 0991344626 (cit. on p. 50).

[14] Material-UI Developers. *Material-UI*. 2023. URL: https://mui.com/ (cit. on pp. 51, 54).