Gateway Architectures for Interaction between the Current Internet and Future Internet Architectures

Piergiacomo De Marchi

Padova 2017

Relatore: Prof. Maria Elena Valcher
Co-Relatori: Prof. Adrian Perrig
Prof. Nicola Dragoni

SCION

University of Padova
Department of Information Engineering
Via Gradenigo 6/b,
35131 Padova, Italy
Phone +39 049 827 7500
info@dei.unipd.it
www.dei.unipd.it

Summary (English)

It is challenging for future Internet architectures to be deployed, given the dominance of today's Internet. A way to facilitate deployment of new Internet architectures is to provide a Gateway to transparently facilitate communication between the current and the future Internet. Moreover, a Gateway can exploit properties of the future Internet architectures to provide improved functionality even for legacy-to-legacy communication.

In this project, we design, analyze, and implement a gateway for the SCION secure Internet architecture. First, the SCION Gateway enables communication between legacy IPv4 or IPv6 hosts and SCION hosts, and second, it enables legacy IP traffic to be encapsulated and transported over the SCION network, hence obtaining SCION security properties even for legacy traffic. In this work, we develop the SCION Gateway and we analyze the security implications/benefits for legacy traffic to be interfaced with SCION. We analyze then how the SCION Gateway can provide DDoS defense properties for the legacy hosts it serves, without requiring any change to the legacy network besides obtaining SCION network access.

We show that the interfacing between legacy Internet and SCION is possible and efficient through the SCION Gateway, without requiring any major network changes. We then analyze and demonstrate the security benefits that can be achieved for the legacy network and hosts. We anticipate that the SCION gateway can provide beneficial properties for real deployments.

Preface

This thesis was prepared within the Network Security Group of the ETH Zürich University in fulfilment of the requirements for acquiring a double M.Sc. in Computer Science Engineering from the University of Padova and the Technical University of Denmark.

The thesis deals with the necessity of a new gateway architecture for the interaction between the current Internet and the future Internet architectures and emphasizes how this must be scalable and secure. It also deals with the design, implementation and evaluation of a new type of application-level gateway which interconnects the Internet with the SCION architecture.

The thesis consists of an extensive analysis of today's Internet problems and the requirements a new network architecture should satisfy to solve them, an extensive background on the working principles of old and new gateway technologies and finally, an exhaustive design of a secure and scalable gateway solution for the Internet-SCION interconnection. The thesis also describes the implementation choices made during the development process of the aforementioned gateway system and the evaluation results obtained.

This written document is delivered together with an electronic archive which contains all the developed software (`https://drive.google.com/open?id=13opRbxUmdw7vgjhBtW7UnFyJF9UN9Tix`), and a demo video that shows a few tests carried out for one of the common scenarios (`https://drive.google.com/open?id=1ujqHT8Z_Rcd-6npMQmsccPRTkyTpu8V8`).

Supervisor:	Maria Elena Valcher	*meme@dei.unipd.it*	(UNIPD)
Co-Supervisors:	Adrian Perrig	*adrian.perrig@inf.ethz.ch*	(ETH)
	Nicola Dragoni	*ndra@dtu.dk*	(DTU)

Padova, 4-December-2017

[illegible]

Piergiacomo De Marchi

Acknowledgements

Gratus animus virtus est non solum maxima, sed etiam mater virtutum omnium reliquarum.

– Cicero, *Orationes Pro Cn. Palatio 80*

I would like to express my special thanks and gratitude to my family that always supported me as much as they could, particularly during the academic years. My sister Arianna, my first supporter and my best friend, always there to show me that there is a way out when things go wrong. My beloved mom Mariarosa, constantly caring about me, listening to all my needs and treating me as her crowning jewel. My father Antonio, my hero and my inspiration to do better and better. Dad you taught me how to become a man and to love your neighbor as you love yourself and I wish you were here today. I made you a promise that day, that I would have made you proud of me, and I hope to be a step closer to keep my word after today.

Contents

1 Introduction

The success of the Internet architecture is far beyond the initial expectations and is massively influencing all the aspects of the modern society. However, security, scalability and availability of this architecture do not match its importance and frequently lead to serious consequences that in certain cases we can not even quantify.

Patches to the Internet have been often constrained by its intrinsic design so in the last decade researchers have proposed alternative solutions for a future Internet architecture. NEBULA [1], NIRA [2] and XIA [3] are only a few examples which address a subset of today's Internet problems.

SCION (Scalability, Control and Isolation on Next-generation networks) is the only future Internet architecture proposal that is under active deployment and use, and its aim is to solve security, scalability and availability issues that plague current networks. SCION achieves properties of: high availability, transparency, scalability and support for heterogeneous trust – preserving today's Internet economic aspects unaltered – thanks to the *Isolation Domains (ISDs)* concept [4] [5] [6] [7]. An ISD constitutes an interconnected logical set of *Autonomous Systems (ASes)* administered by a subset of them called *ISD Core*. The ISD is governed by a policy called *Trust Root Configuration (TRC)* negotiated by the ISD Core.

The SCION project counts among its collaborators and sponsors the European Research Council (ERC), Google, KDDI, the National Science Foundation (NSF), Swisscom and SWITCH. Such a new Internet architecture seems really promising and efficient, but much more than this is needed in order to overcome the current status quo of the worldwide networks. Virtually all today's electronic devices are connected to the Internet. Stopping or changing this is not a step than can happen overnight, so in order to successfully deploy SCION architecture a good interoperation with the current legacy Internet and all its working principles and protocol stacks is required.

In this thesis work we present the SCION-IP Gateway (SIG), the key that enables communications between SCION and the legacy IP world. Specifically, it enables IP end-hosts to transparently benefit from SCION network and obtain improved security and availability over their communications. Furthermore, being immune to IP spoofing and thanks to an ad-hoc authenticated IP-mapping system, the SIG provides also DoS and DDoS defense proprieties for the legacy hosts it serves.

We show then that the interconnection between SCION and the legacy Internet is possible and efficient through the SIG without introducing any infrastructure changes, apart from setting the SIG's IP address as default gateway for all the end-hosts within the AS the SIG belongs to.

In this introductory chapter, we first analyze more in details today's Internet vulnerabilities that plague our networks and that are the motivations for the adoption of a new architecture [7] (Section 1.1). We later explain how the SCION architecture addresses some of these problems [7] (Section 1.2). We then describe the motivations to have an interconnection between SCION and Internet that led to the choice of designing and implementing the SIG (Section 1.3). Finally, in the last sections, we briefly illustrate the contribution to this research work (Section 1.4) and provide the overall structure of the thesis (Section 1.5).

1.1 Today's Internet Problems

"Internet Hourglass" is the widely accepted definition of the current Internet architecture. The similarity between the shape of the Internet architecture and the shape of an hourglass is immediately recognizable by having a look at Fig.1.1. The Internet has been evolving a lot over the last 40 years: from the very first packet switching researches in the 1960s, the introduction of the TCP/IP stack and Ethernet LANs in the 1970s, the development of DNS and TCP congestion control in the 1980s, the BGP, CIDR and NATs adoption in the 1990s, the

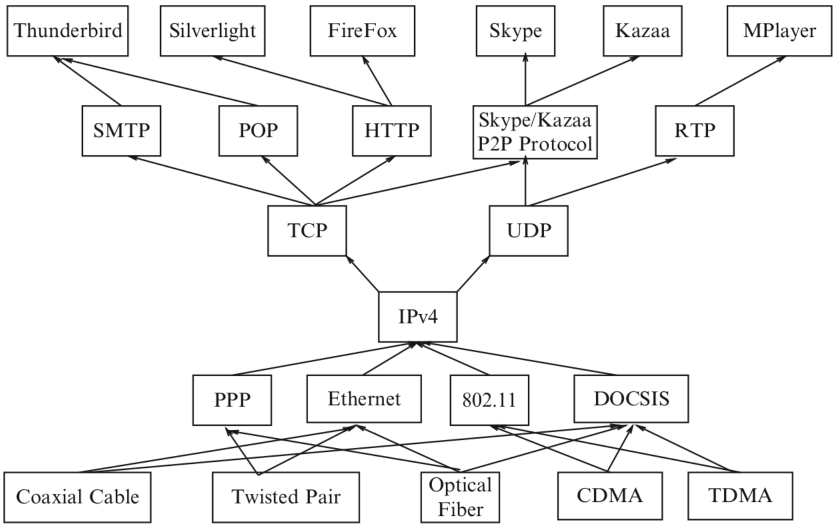


Figure 1.1: An (incomplete) illustration of the hourglass Internet architecture (reprinted from [8]).

advent of the last decade new access technologies (wireless, broadband, LANs, WiMAX, etc.) and, naturally, the unceasing expansion of the application-layer protocols and services (VoIP, peer-to-peer, video streaming, social networks, multiplayer gaming, etc.). As a matter of fact though, only parts of the Internet protocol stack (application and physical layers) have changed significantly – and keep changing – while the core protocols (network and transport layers) have been ossified[8].

This discrepancy, together with the continuous expansion of the Internet, has led to numerous issues and vulnerabilities of the current architecture. It would require an entire book to mention all of them, so in this section, as in the official SCION book [7], we only present an overview of the salient ones that motivate the adoption of a future Internet architecture.

1.1.1 The Internet Protocol (IP)

IP is the principal communication protocol of the Internet that allows packets forwarding between the end hosts. The first version (IPv4) was defined in 1981 and extended by IPv6 in 1998 [9]. IP routes packets from source to destination choosing a single path that is unknown to the end hosts. The forwarding process is done only on the basis of the next-hop and the complete path cannot be tracked or influenced either by the end hosts, or by the middle routers. This approach is trivial, but it comes with different drawbacks:

- **Lack of separation between routing and forwarding:** Forwarding of IP packets only depends on the forwarding tables on routers which change dynamically and can alter working paths and even break them.
- **Lack of transparency and control:** Being able to select and verify a path is a desirable property, for example, in case an end host wants to avoid the sent packets to be routed through an adversary or an untrusted network, or in case it wants to choose the favourite path according to a specific metric. Unfortunately, IP does not provide this and does not provide multiple paths utilization either.
- **Stateful routers:** Forwarding tables maintained on routers to select the next-hop has drawbacks. Performing a route table lookup for each packet is very time consuming, therefore high-performance networking equipment usually relies on Ternary Content-Addressable Memory (TCAM) hardware that is extremely expensive and energy-consuming. Furthermore, the memory size of the forwarding tables saturates the capacity of the TCAM hardware. Finally, routers that keep track of network communication states can suffer from Denial-of-Service (DoS) attacks that aim to exhaust these states [10].

1.1.2 The Border Gateway Protocol (BGP)

BGP is the routing protocol adopted by the Internet that connects independent networks or ASes as Internet Service Providers (ISPs). Each AS advertises to the outside world its reachability transmitting the list of the IP address *prefixes* it owns through a BGP update message. Such BGP updates accumulate the identifiers of the ASes through which they have passed, and they contain a list of the advertised prefixes together with the next-hop to reach them. Each AS collects such messages to create its locale routes. Two main types of business relationships between ASes exist: the *customer-provider* relationship (where one AS pays another to forward its traffic) and the *peering* relationship (where two ASes agree that connecting to each other without payment is mutually beneficial). ISPs then select the best routes based on *polices* that reflect these business relationships [11]. BGP comes with numerous drawbacks:

- **Outages:** Control plane and data plane are not separated on the Internet, so the forwarding process may stop during route changes – that can derive from attacks to the routing system. Furthermore, when BGP update messages are shared, it may require up to tens of minutes for the network to converge to a stable state [12] and this can lead to traffic interruptions.
- **Lack of fault isolation:** BGP is a global distributed protocol and the BGP updates messages are spreaded all over the world among all the ASes. Due to lack of hierarchy or isolation of different areas, a single faulty BGP speaker can affect the routing of all the globe (as in the AS 7007 incident [13]).
- **Lack of scalability:** The amount of messages BGP has to handle is proportional to the number of global destinations. This plays against the scalability of the system and also prevents BGPSec (a proposal for a secure BGP) from frequently disseminating freshly signed routing updates.
- **Single path:** At the end of the BGP process, only one single path is chosen to reach a given destination. Although some protocols allow to use multiple paths in parallel, BGP does not provide path control to end-hosts and does not provide multiple AS-level paths.

1.1.3 Lack of Authentication and Trust

Authentication is another important feature that the original Internet protocols lacked and it is becoming more and more fundamental since attackers exploit its absence in order to inject malicious packets to attack the network.

Today's Internet most used authentication infrastructures are: RPKI that provides the roots of trust for BGPSec; TLS that allows web severs authentication; and DNSSEC that provides authentication for DNS. However, the current situation is still unsatisfactory starting from the fact that all the mentioned protocols suffer the compromise of a single entity (i.e., if a single entity is compromised, so is the whole chain of trust). BGPSec and DNSSEC both rely on a single or a very small set of roots of trust, while for TLS any one among the hundreds of Certificate Authorities (CAs) can issue a certificate for any domain. Furthermore, the Internet Control Message Protocol (ICMP) does not even present an authentication method and allows injection of fake or spoofed packets. Finally, the Internet also lacks of an infrastructure that enables two end-hosts to establish and share a secret key for end-to-end encryption and authentication, and mainly relies on Trust-On-First-Use (TOFU) approaches [14].

1.1.4 Attacks

In this section some attacks against which Internet has little or no protection are presented.

1.1.4.1 Prefix Hijacking

Since BGP does not provide a fault isolation mechanism, many Internet outages are caused by a malicious or erroneous announcement of IP prefixes. This problem is called prefix hijacking. The most known example happened in February 2008 when Pakistan's internal censorship attempt turned out in a global outage of YouTube for about two hours [15].

A similar attack is prefix redirection, where an attacker eavesdrops the traffic towards a destination by hijacking its prefix to receive its packets, but also engineers BGP updates such that the packets finally reach the intended destination. A striking case is reported in [16] where the attacker redirected the traffic to make a detour across another continent.

The overall problem is exasperated by the fact that defining BGP routing policies is a very complicated and manual task, hence error-prone.

1.1.4.2 Spoofing and DDoS Attacks

ICMP is a very common protocol to send error or diagnostic messages and is used by tools as *ping* and *traceroute*, but the ICMP packets are not authenticated and their source address can be easily spoofed, possibly leading to DoS and DDoS attacks [17] [18], or the disconnection of two BGP routers [19]. Regular IP packets are also not authenticated and their origin can also be spoofed.

The most glaring examples of DDoS attacks are the one against Estonia in April 2007 that disrupted many of their critical infrastructures [20] and the very recent one on Dyn's DNS infrastructure [21] [22] that produced an unprecedented amount of traffic – exceeding 1 Tbps – using Internet-of-Things (IoT) devices.

1.1.4.3 Forged TLS Certificates

Even a single compromised trust root can be used to create and distribute rogue TLS certificates. A famous case is the one that involves the Iranian government who used a forged certificate for Google and Yahoo services in order to perform a Man-In-The-Middle (MITM) against its citizens. The attack was supposed to be mounted on the DigiNotar CA, who signed the certificates [23] [24]. CAs hold a significant power since they can produce valid certificates. However, browsers and OS vendors hold even more power since they control which CAs are trusted by default.

1.1.5 Transition to a New Architecture

Changing two fundamental Internet protocols as IP and BGP is an extremely hard task, but necessary in the long run. Obviously, the transition will not happen overnight and the only way to promote it is to provide methods and tools that can gradually replace these old protocols and lead the evolution towards the desired future network's properties. The re-designing of a clean-state architecture needs then to fix the problems that affected the old architecture, maintaining, though, a high level of interoperability with the old functionality and mechanisms that were adopted by its predecessor.

1.2 Goals of SCION

In this section we present what are the high-level goals of a secure Internet architecture as proposed by the official SCION book [7]. We illustrate why they are important and how they can be achieved. We also briefly describe which are non-goals for an inter-domain point-to-point communication architecture excluded by the design.

1.2.1 Availability in the Presence of Adversaries

SCION aims to be a point-to-point communication infrastructure highly available even in the presence of distributed adversaries: as long as there is an attacker-free path, this should be discovered and used with guaranteed bandwidth.

Availability in presence of adversaries is very challenging. An *on-path adversary* may drop, delay, or alter packets that should be forwarded, or even inject additional packets. An *off-path adversary* may setup hijack attacks to control the traffic through the elements under its control, and then perform on-path attacks. The off-path attacker could attract traffic through the peripherals under its control by announcing forged paths or promoting attractive network metrics. Conversely, an attacker may also render some paths out of its control less attractive, for example using congestion techniques. An adversary controlling a large botnet may perform significant Distributed Denial-of-Service (DDoS) attacks to congest some links or the resources of some network elements. Finally, an adversary could also interfere with the discovery process of genuine paths (e.g., by flooding the control plane with bogus paths).

1.2.2 Transparency and Control

SCION aims to provide transparency and control for the forwarding paths and the trust roots.

1.2.2.1 Transparency and Control over Forwarding Paths

Network path transparency means that endpoints know and can verify the forwarding path taken by network packets. This is very helpful when transmitting sensitive data because one can choose which ISDs to traverse and which not.

By ensuring transparency, also path control can be achieved. This enables ASes to control the incoming paths segments through which they are reachable. Given such segments, senders can then create end-to-end paths. Path control brings many beneficial features:

- **Separation of control plane and data plane:** To enable the path control, the control plane (used to determine paths) has to be separated by the data plane (used to forward packets). This separation enhances availability since the forwarding can be affected by routing changes or other control-plane operations.
- **Enabling of multipath communication:** The sender can choose multiple paths to forward its packets. This also enhances availability [25].

- **Defending against network attacks:** If the path is carried in the packet's header (which is one way to achieve path control), the receiver can reverse it to return the response to the original sender thus mitigating reflection attacks. Path control can also mitigate DoS and DDoS attacks by circumventing malicious network entities or congested links.

Path control also presents some fragile aspects that need to be handled properly:

- **Respecting ISPs' forwarding policies:** Senders' freedom of choosing paths can not violate ISPs' policies that have then to provide a set of policy-compliant paths that senders can choose from.
- **Preventing malicious path creation:** A malicious sender may exploit path control, for example in order to create a network loop that increasingly consumes network resources.
- **Scalability of path control:** Source routing does not scale because a source would need to know the overall network topology. To make path control scale, a source can choose only among a small set of paths.
- **Permitting traffic engineering:** Fine-grained path control would inhibit ISPs from performing traffic engineering, so SCION provides end-hosts path control at the granularity of AS on the level of ingress/egress interfaces allowing ISPs to fully control internal paths. ISPs can further perform traffic engineering based on per-path bandwidth allocations, which can be encoded in the forwarding information.

1.2.2.2 Transparency and Control over Trust Roots

Roots of trust are nowadays used for the verification of Internet entities as a web server public key in a TLS communication, or a DNS response in DNSSEC. Because of intermediate CA certificates, that are not explicitly listed but implicitly trusted, in today's Internet it is not always possible to know all the trust roots an end-host or user is trusting. Independent studies have counted over 300 roots of trust in the TLS PKI [26], but because of lack of transparency they may be more. SCION aims to provide transparency of trust roots which means an end-host or user can always know the complete chain of trust it needs to rely upon for the validation of a specific certificate. Control over trust roots instead allows users to decide which roots of trust they want to rely upon.

1.2.3 Efficiency, Scalability, and Extensibility

Security and availability usually come at the expenses of efficiency and scalability. High performances and scalability, however, are necessary for a new Internet architecture to succeed in the current socioeconomic environment. Therefore, SCION seeks *high efficiency*, meaning that the packet forwarding has to be at least as efficient as the current IP forwarding system. Furthermore, SCION seeks *improved scalability* compared to the current Internet architecture, in particular with respect to BGP protocol and the size of the routing tables.

To achieve this, SCION's approach is to avoid storing state information on routers whenever possible, and to encode them on packets' headers, instead (cryptographically protecting them). This considerably reduces the complexity of router systems and accelerates the forwarding process. In fact, it was observed that modern block ciphers as AES can be computed faster than performing memory lookups. For example, on current PC platforms, computing AES requires about 50 cycles while fetching a byte from the main memory about 200 cycles [27]. Moreover, achieving high parallelism in hardware block ciphers is cheap thanks to their simplicity and small number of gates, while the same parallelism on high-speed memory systems is complex and expensive. Avoiding state information on routers not only improves the network efficiency, but also prevents state-exhaustion attacks [10] and state inconsistencies across routers.

Efficiency and scalability are favoured also by having the end-hosts assisting with the network-layer functionality such as path selection. This results in a simpler forwarding plane and more efficient routers. Furthermore, end-hosts are the entities on the network that know better their internal state, bit-error recovery, duplicate suppression, delivery of acknowledgments and preferred or undesirable paths so, by having more control, they can contribute to the efficiency of the network.

SCION aims also to be an *extensible* architecture that means new functionalities should be easily built and deployed.

1.2.4 Support for Global but Heterogeneous Trust

Given the plenty of different jurisdictions and interests of today's Internet, an important challenge is to scale the authentication of entities (e.g., ASes, DNS servers, or domains for TLS) to a global environment.

Both monopoly and oligopoly RPKI models do not scale well because mutually distrusting entities do not agree on a single root of trust (monopoly model), and because the security of many roots of trust depends on its weakest point. SCION then seeks to support *a global environment with heterogeneous trust.*

1.2.5 Deployability

If early adopters can not obtain enough benefits migrating from Internet to SCION, initial deployment is unlikely to happen. In this respect, SCION has to guarantee already to its first deploying ISP a competitive advantage with respect to its competitors. To this end SCION provides: high-availability under control-plane or data-plane attacks (e.g., providing built-in DDoS defenses), path transparency and control, trust-root transparency and control, high efficiency, robustness to configuration errors, fast recovery from failures, high forwarding efficiency and multipath forwarding.

To encourage the migration, SCION needs also to provide an efficient interconnection to the today's Internet entities (see Section 1.3), like web servers, RPKI authorities and CAs, in order to allow users to keep benefiting from the information currently available on the Internet network.

The migration should require minimal infrastructure changes and the deployment should be possible re-utilizing current ISPs' infrastructures (only requiring the upgrading or the substitution of few border routers). Also the configuration settings of the new architecture can not drastically differ from the current one in order to reduce the amount of personnel training.

Finally, economic aspects are imperative. ISPs should be provided with new business models and adapted selling strategies and the migration cost should be reduced to the minimum. SCION should not disrupt current Internet business models, but maintain the same topology and business relationships (e.g., support peering).

1.2.6 Non-Goals

Some aspects are not current goals of SCION, but may become such in the future thanks to improvements or layers above the communication one (under development). Multicast or efficient content dissemination is not part of the basic infrastructure due to the extra complexity it would introduce. Software vulnerabilities of network components are also out of the scope of the SCION design. Malicious Internet content is preferably addressed by a layer above the communication infrastructure and hence also not analyzed during the design of SCION.

1.3 Motivations behind a SCION-Internet interconnection

Changing fundamental network protocols as IP and BGP is not an easy step that can happen overnight, but in the long run, technological evolution is necessary and inevitable. Consequently, SCION needs to provide a set of models and tools that can favor the transition towards the desired properties. Among them the SCION-IP Gateway (SIG), that is the topic of this thesis work, is a fundamental block.

The purpose of the SIG is a full interconnection between SCION and Internet networks. Any SCION AS that deploys a SIG service can communicate with the legacy IP world and with legacy IP hosts inside other SCION ASes. The SIG takes care of encapsulating the IP traffic into the SCION one at the sender side, decapsulating the SCION traffic into the original IP packets at the receiver side and correctly routing the information to the right legacy end-host through the SCION network. The SIG also takes care of mapping in which SCION AS a certain destination IP address belongs to in order to provide correct routing information through SCION.

Such a gateway is extremely important during the transition to a future Internet architecture for specific reasons:

- **Current data is on the Internet:** At the present time, the entire global information resides on servers connected only to the Internet network and this will remain as such for many years onward. It will take a considerable time to have these servers connected also to a new future Internet architecture as SCION or to have all the information replicated on new SCION-servers. If SCION users want to keep accessing this information a gateway module is needed between the two architectures.

- **SCION limited resources and time:** As every new project and business, SCION starts from a state of disadvantage compared to current network service providers and network equipment manufacturers. The SCION team has limited resources and an inexperience dictated by time compared to them, this implies they can not address all the network issues from the very beginning, and they can not provide to end users an analogous SCION system/feature for all the available Internet ones. For example, SCION bases its time synchronization on Network Time Protocol (NTP) Internet servers and does not provide their SCION counterpart, yet. Deploying all the corresponding features will require time, meanwhile SCION has to benefit from some Internet services through the SIG gateway.

- **Economical inferiority of SCION:** Economically speaking SCION can not yet compete with current Internet companies and their self-sustainable business models, so it needs to go alongside with them and not contrast them. In this way SCION can attract the interest of these companies rather than nourish their rivalry. Through the SIG gateway, SCION definitely encourages Internet companies to collaborate toward a better and more secure global connectivity infrastructure by providing an interconnection with itself and exporting some of its newly introduced benefits.

1.4 Thesis Contribution

The primary contribution of this thesis is the design (Chapter 4), development (Chapter 5) and evaluation (Chapter 6) of a new application-level gateway technology which allows the interconnection of the Internet network with SCION, the only future Internet architecture proposal that is under active deployment and use, whose aim is to solve security, scalability and availability issues that plague current networks.

The thorough investigation of gateways technologies and architectures, their history, principles and techniques is also a substantial contribution, as the subject is quite old and the related literature dates back to the 1980s and is almost not existent (Chapter 3).

The SCION architecture [7] is an extremely huge project, and its implementation is composed of more than 600 MB of *Python* and *go* software libraries developed by the Network Security Group of the ETH University. The SCION implementation is an academic proof-of-concept at the moment, so there is still no documentation written for it. Most of the time during the first three months of this thesis work was spent to fully understand SCION's design and implementation.

A big portion of time during the fourth month instead was used to deepen my understanding of DoS and DDoS attacks and their countermeasures, in order to design and evaluate a SIG's defense mechanism for the encapsulated legacy traffic (Section 4.6 and Section 6.4).

During this thesis project I learnt the *Python* programming language from scratch, together with the two syntaxes for the *Vagrant* and *Ansible* configuration files. Approximately 2500 lines of code were written (for an amount of 1,5 MB of software produced) for a complex multithreading system which threads all access the same variables, buffers and data structures. A good knowledge of the *Python* multithreading principles and the behaviour of its data structures in a concurrent context was fundamental during the development phase.

1.5 Thesis Structure

The rest of this document is organized as follows.

In chapter 1 we provide an overview of the SCION problem space extensively explaining today's Internet problems and the requirements a new Internet architecture should fulfill to solve them. In the chapter it is also emphasized the complexities of the transition to a new architecture, as SCION, which motivates the need to develop the SCION-IP Gateway (SIG) module for the SCION-Internet interconnection.

In chapter 2 we provide an extended background on gateway technologies and the motivations behind the willingness of interconnecting different networks. In the chapter different gateway technologies are also discussed and some examples of gateway systems are presented.

In chapter 3 a high level overview of the SCION architecture is presented to better understand the following chapters. SCION aims to solve security, scalability and availability problems that plague the today's communication networks. In the chapter, SCION's key concepts and working principles are described, together with some minor details that are necessary for the comprehension of the following chapters.

In chapter 4 we describe how the SIG module that interconnects the SCION network with the legacy IP world was designed and in which scenarios it works or does not work. The SIG-related challenges introduced by the proposal of a new Internet architecture are discussed, together with the way the SIG service intends to address them. The security benefits introduced by our solution, even for legacy-to-legacy traffic, are finally described.

In chapter 5 the prototype implementation details are discussed, together with the tools and technologies needed to develop the system. The topology of the development and testing environment and the way the codebase was structured are also described.

In chapter 6 an extensive evaluation of the developed prototype is illustrated. The most salient aspects tested are the interprocess-communication performance between SCION and legacy hosts, the scalability issues the SIG has to face when the SCION network grows and the benefits introduced by the legacy DoS and DDoS mitigation mechanism provided by the SIG.

Finally, in chapter 7 some conclusions over the proposed solution are drawn and some brief insight in the future possibilities that the solution opens are given.

2 Background on Gateway Technology

The motivations for building computer communication networks – programs and data exchange and sharing, remote access to resources, live updates, entertainment, etc. – are also the reasons for interconnecting networks. This is a consequence of the observation that the power of a communication system depends on the number of potential participants. Interconnecting different networks not only increases the number of the connected end-users and end-systems, and hence the power of the communication system, but can also provide extra functionality and benefits to the cooperating networks and their participants.

In this chapter we first introduce some concepts involved in computer communication networks and describe how these networks provide an interprocess communication facility (Section 2.1). We then compare the datagram and virtual circuit approaches to exchange traffic between processes (Section 2.2). We discuss the gateway device and its possible degree of participation in providing various levels of end-to-end- service (Section 2.3). We finally present some examples of gateway technologies proposed and used over time since the 80's (Section 2.4).

2.1 Interprocess Communication

When discussing computer networks communication, it is important to notice that the communication takes place, on request, at the process level (i.e. computer programs in execution) and not at the host level. Processes are the main actors of the communication network and they are the ones that send and receive the exchanged data. They run in host computers or other forms of hosts.

The protocols used for enabling the communication compose an interprocess communication system [28]. Fig.2.1 shows how combining the network and the host network interface can be seen as providing an interprocess communication system.

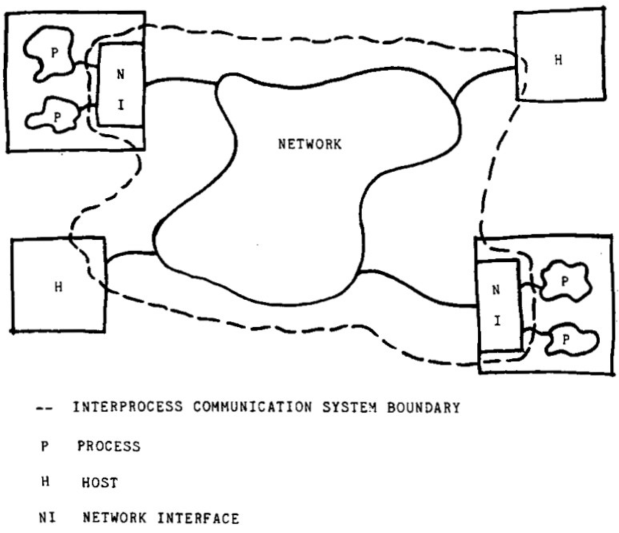


Figure 2.1: Communication network (reprinted from [28]).

A new host that connects to an existing network, and wants to be enabled to communicate, must implement the protocol layers that match the communication protocols used by such a network. The new host must then join the interprocess communication system to enable its own processes to communicate with other processes running in different hosts of the considered network.

The interconnection of different networks strictly requires that the processes running in the hosts of the interconnected networks make use of a common interprocess communication system. This can be achieved in different ways:

by converting the networks to a new interprocess communication system, by converting one or more levels of a network protocol to new protocols compatible with other networks, or by translating messages between pairs of interprocess communication systems at their points of contact [28].

2.2 Datagrams and Virtual Circuits

Commonly, there are two main approaches to provide an interface for packet-switched interprocess communications: through individual *datagrams*, or through a higher level service in the form of a *virtual circuit* [28] [29].

Datagrams are independent single messages. They are not acknowledged so they are intrinsically unreliable. In packet-switched networks large files are split into fixed size datagrams that are sent in-order, but can arrive out-of-order at the destination point. Datagrams are simple to implement since networks and gateways do not need to record and update state information. Due to their unreliable format structure, each datagram must carry complete address information to be sent over a communication network. Processes exchange datagrams through simple send and receive operations.

Virtual circuits are reliable by design thanks to acknowledgment techniques and always provide in-order delivery. They are difficult to implement because networks and gateways need to record and update state information. Virtual circuits are established through a message exchange step with the goal of setting up the circuit; when use terminates, another message exchange step is used to tear down the circuit. Data are transmitted using a short form of address or a virtual circuit identifier in place of a complete address, this is possible thanks to the state tracking performed by the network's devices as already discussed. To use a virtual circuit a process must first create it. Also this time, data is exchanged between processes through simple send and receive operations. Finally, the process must terminate the circuit.

Datagrams provide a transaction-oriented service while virtual circuits provide a connection-oriented service and both are needed in a communication network environment. Datagrams are more efficient for transaction-oriented information requests as Internet Videos, Voice Communication, Notifications of new Emails, etc. The most known example of datagram network is the Internet which uses the IP network protocol. Also the User Datagram Protocol (UDP) at the transport level of the OSI model is a very famous example. Virtual circuits are useful in connection-oriented contexts as remote Terminal Access and File Transfer between computers. Well known examples of virtual circuits networks are the Asynchronous Transfer Mode (ATM) and the General Packet Radio Service (GPRS).

2.3 Gateways Basic Concepts and Degree of Gateway Participation

Two or more communication networks are interconnected via a device, or a pair of devices, called gateway. Such a device appears simply as a host in that network (Fig.2.2).

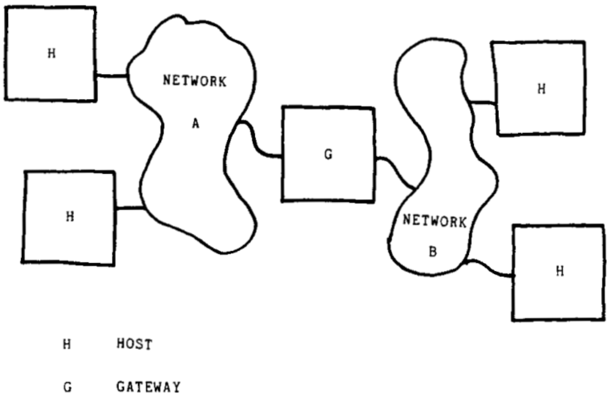


Figure 2.2: Interconnected networks (reprinted from [28]).

Some gateways simply take the messages coming from one network; unwrap them from that network's packaging and read their content; compute a routing function based on that content or the content of the previous package header; wrap that content in the other network's packaging; and finally send the new packages over the other network using the previously computed routing function. Since different networks may use different telecommunications technologies, such as POTS, SS7, Next Generation Networks (2G, 2.5G, 3G and 4G/LTE radio access networks) or PBX systems, this type of gateway is called *media-conversion gateway.* The most known example is Voice over Internet Protocol (VoIP) media-conversion gateway which performs the conversion between Time-division multiplexing (TDM) voice to a media streaming protocol, such as the Real-time Transport Protocol (RTP), that is a network protocol for delivering audio and video over IP networks [30].

Other gateways instead translate the protocol used in one network to the one used in another network. They replace messages from one network with different messages, that share the same protocol semantic, and send the latter over the other network. This type of gateways is called *protocol-translation gateway.* The simplest and most commonly used protocol-translation gateway is the one that converts between Modbus Remote Terminal Unit (RTU) and Modbus TCP [31].

The distinction between media-conversion and protocol-translation gateways is of one degree: the former fill the gap between the differing link and physical layer protocols of the networks they interconnect, while the latter fill the gap between differing network and higher layer protocols of the networks they interconnect.

The protocol-translation approach is not a trivial solution and it raises several issues. The complexity of its design seems to be inversely correlated with the protocol layer to be translated (the lower the protocol layer, the higher the complexity) [28].

At the lowest levels (physical and link levels), protocol translation does not cause any problem thanks to the hop-by-hop nature of the transport flow. It has to be taken into account, though, that different protocols have a different impact on reliability, throughput and delay characteristics of the overall communication system.

At the network and transport levels, the management of message size, addressing and flow control become crucial for the performances of the interprocess communication system. Since the message size can vary over time during the communication, one must take care of fragmentation and reassembly of messages. That is, the division of a long message into smaller pieces for their transmission over a small message size network, and the reconstruction of the original message at the reception of all these pieces on the destination side. Address translation between different networks also complicates the protocol translation and it becomes a hard problem when a network (or a transport level protocol) presents a larger address space than the network (or protocol) it has to be translated into. To further complicate the problem, when end-to-end flow control mechanisms are used – that is the common case for many transport level protocols – many other difficulties arise if the units controlled by a protocol and the other differ from each other. For example, if a protocol processes ASCII symbols and the other binary octets. Even more issues arise when different flow control models are used. For example, if the two protocol have different buffer space allocations and different transmission rates, or the sender makes use of the Stop-and-Wait approach and the receiver makes use of the Selective-Repeat one.

At higher levels, even more difficulties arise because of the increased state information used and the lower likelihood of a one-to-one translation of the different protocols' messages. Furthermore, a higher level protocol message is further multiplexed by all the lower levels which add their overhead information. Translating a higher level protocol means then also separately translating all the information coming from these lower levels.

A gateway can be seen as a single entity connected to both the networks it interconnects, or can be seen as having one "half" in each network [32][28] as shown in Fig.2.3. These gateway halves can be either independent computer devices, or extensions of the ordinary switching nodes, or software applications running on one of these previous two solutions (application-level gateways).

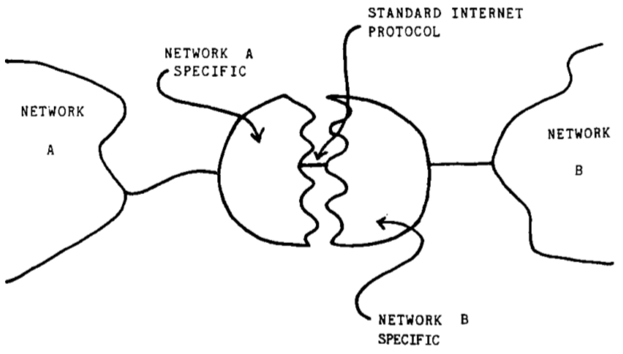


Figure 2.3: Gateway halves (reprinted from [28]).

2.4 Gateways: From Past to Present

In this section we briefly describe some examples of interconnected networks that made or make use of a gateway system to obtain interprocess communication. Some of these technologies were used in the past, others are more recent. Notice that, this section is not meant to be an extensive list of gateway technologies.

2.4.1 Interconnection of X.25 Networks

X.25 is one of the oldest packet-switched services available and it was developed before the Open Systems Interconnection model (OSI model) in 1984 by ITU Telecommunication Standardization Sector (ITU-T), formerly called Comité Consultatif International Téléphonique et Télégraphique (CCITT). X.25

is a protocol suite for packet-switched Wide Area Network (WAN) communication. An X.25 WAN consists of Packet-Switching Exchange (PSE) nodes as networking hardware, and Integrated Services Digital Network (ISDN) connections, leased lines, or old telephone connections as physical links. X.25 was widely adopted during the 1980s by telecommunication companies and financial transaction systems as Automated Teller Machines (ATMs). The X.25 family of protocols has been replaced to large extent by simpler protocols, particularly the Internet Protocol (IP), but it is still in use for niche applications as within the payment industry [33].

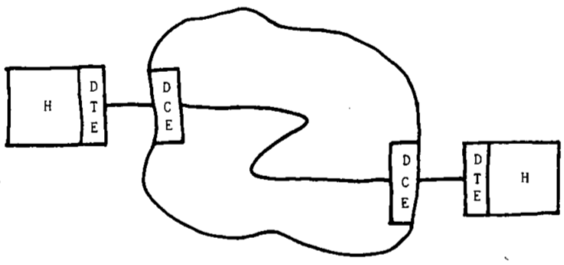


Figure 2.4: PDN virtual circuit (reprinted from [28]).

Public Data Networks (PDNs) following the CCITT X.25 Recommendation [34] have to be interconnected with each other according to the interface specified in CCITT Recommendation X.75 [35]. The X.25 Recommendation specifies that the interface between customer's equipment, called Data Terminal Equipment (DTE), and the network equipment, called Data Circuit-Terminating Equipment (DTE), relies on a virtual circuit approach which provides an interface for transport level protocols [28]. Fig.2.4 shows an X.25 PDN virtual circuit.

The interface provided by the X.75 Recommendation is similar. The equipment available at both sides is this time called Signaling Terminal Equipment (STE) and the STE-STE interface behaves as the DTE-DCE one. The STE-STE interconnection is a split-gateway, as the one of Fig.2.3, and each half of it is a physical device managed by PDN connected to itself. Fig.2.5 shows an example of four PDN networks connected by STE-STE interconnections.

Interconnecting different PDN's via X.75 gateways results in a series of consecutive virtual circuits, as shown in Fig.2.6, where each block is independent and preserves its own flow control and error recovery procedures.

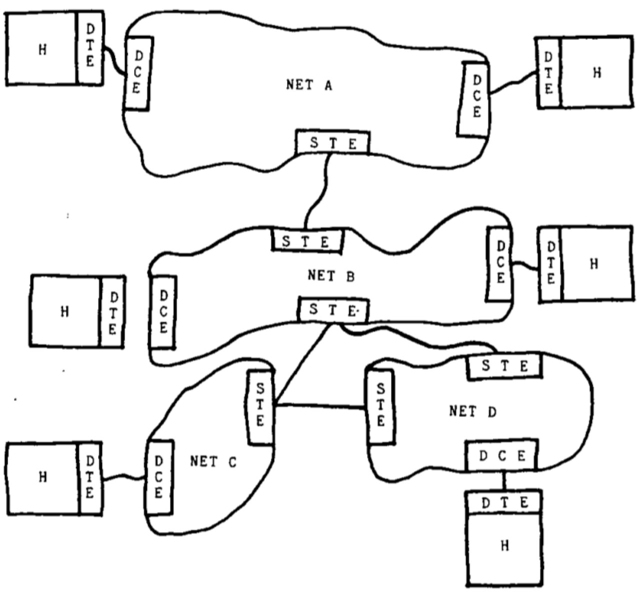


Figure 2.5: Interconnection of PND's (reprinted from [28]).

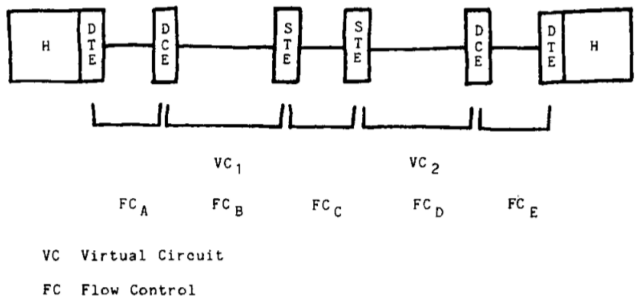


Figure 2.6: PDN transmission path (reprinted from [28]).

2.4.2 Interconnection of ARPA Research Networks

The research sponsored by the Advanced Research Projects Agency (ARPA) on networks interactions resulted in the development of an Internet protocol (IP) [36] and a Transmission Control Protocol (TCP) [37] to support the same functionality of PDN's X.25/X.75 services [28]. The IP is a level-3 datagram protocol of the OSI model, while TCP stands at level-4 protocol. The union of interconnected networks using these protocols is called an Internet. The networks used are of different kinds (e.g., the ARPANET [38], satellite networks, radio networks and cable networks) and we refer to them with the general connotation of Local Networks, even though they may extend over oceans and entire continents. The interface to a local network is called Local Network Protocol (LNP). In Fig.2.7 an end-to-end datagram connection through a local network is represented.

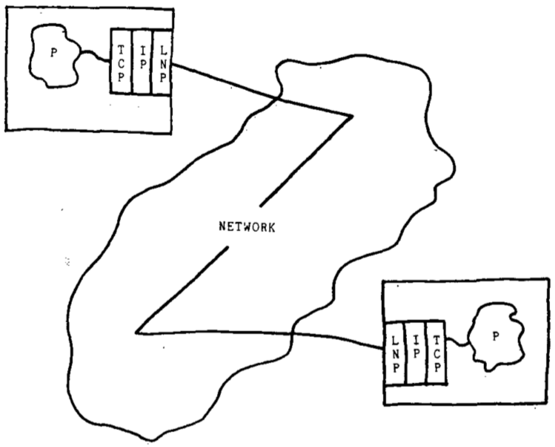


Figure 2.7: End-to-end connection (reprinted from [28]).

In the ARPA model two networks connect to each other through a single gateway device that is a host shared among two or more networks, as shown in Fig.2.8. Each host communicates with the gateway within its same network as it would do with any other host within it. The information needed to route the message is enclosed in the IP packet's header, while the message is carried in the IP packet's payload. A sending host first builds such an IP datagram and then forwards it to the selected gateway in its own network, wrapped within a Local Network packet. The receiving gateway receives such datagram, unwraps it to extract the IP packet and analyzes its IP header to determine the next gateway (or destination host) address in one of the networks it is directly connected. The gateway then wraps again the IP datagram into a Local Network packet and sends it to the extracted address.

The IP protocol does not provide flow control or error control for the transferred messages since only the IP headers are checksummed. The simplicity of the IP protocol allows a gateway to be simple and to be implemented also in small devices. The gateway does not even need to keep state information due to the datagram approach used by the IP protocol that does not require connections or virtual circuits. To provide a reliable end-to-end service ARPA Internet makes use of the TCP protocol that ensures in-order delivery, flow control,

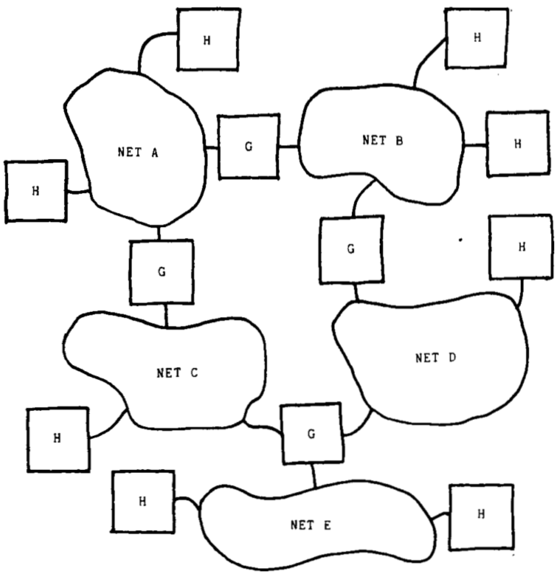


Figure 2.8: ARPA model of interconnected networks (reprinted from [28]).

positive acknowledgments with timeout and retransmission, sequence numbers, etc. Fig.2.9 shows an example of ARPA interprocess communication system where the virtual circuit and the flow control between the two processes are present only over the TCP protocol.

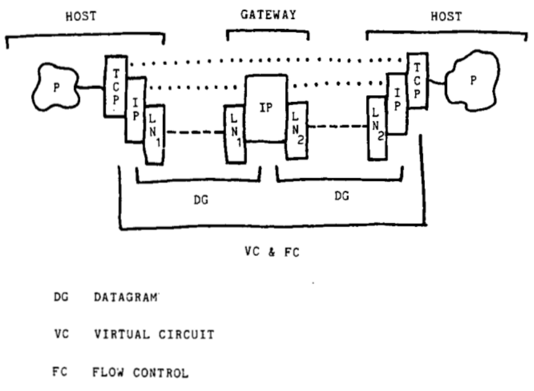


Figure 2.9: ARPA model of transmission path (reprinted from [28]).

2.4.3 Interconnection of IoT Networks and the Internet

A typical Internet of Things (IoT) architecture can be divided into three domains: sensing domain, network domain and application domain [39], as shown in Fig.2.10. The sensing domain is fundamental for the IoT architecture, it enables "things" to interact which each other. It is composed of "smart things", as embedded system devices, and its purpose is to collect information about physical targets using technologies as WSNs, RFID, NFC, FieldBus, Barcode and Zigbee. This information may be preliminarily processed before being sent to the network domain. Generally speaking the sensing domain's task is to enable low power consumption, low costs and protocol lightweight. Network domain is built over existing Internet infrastructures, such as WiFi, PSTN, 2G, 3G, LTE and Satellite. Its main aim is to transfer the information from the sensing domain to the destination host. Application domain is the one responsible of processing the information and provide the needed services. The information from the network domain is handled and various services are provided to the end users.

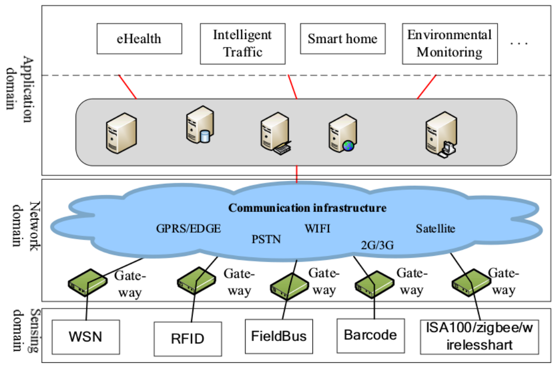


Figure 2.10: Three domains of IoT architecture (reprinted from [39]).

The IoT gateway that bridges the information between the sensing domain and the network domain is one of the most important components of the overall IoT architecture. It acts as proxy and every one is different from each other, based on the applications and requirements. However there are some common pattern among them: they implement multiple interfaces since they need to connect different sensing technologies (e.g, Bletooth, Zigbee, Wifi) with different networking technologies (e.g, PSTN, 2G/3G, LTE, LAN, DSL); they perform protocol-translation between two different sensing domain protocols or between a sensing domain protocol and a network domain protocol; they need to be managed by back-end IoT servers and to manage the "smart things" attached to them.

In [40] a proposal for a novel Smart IoT gateway is presented and its architecture is shown in Fig.2.11. The gateway respects the common patterns described above. It is responsible for protocol-translation and data fusion of different

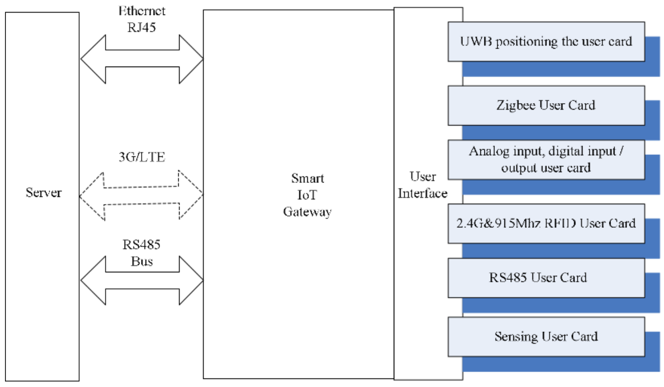


Figure 2.11: The architecture of Smart IoT gateway (reprinted from [40]).

sensor data and introduces three important benefits. Firstly, it presents slots for pluggable modules that adopt different communication protocols. Secondly, it presents unified external interfaces for a rapid and quick software development. Finally, it introduces flexible protocols to translate data coming from different sensors into a uniform format.

2.4.4 Interconnection of Wireless Sensors Networks and the Internet

A Wireless Sensor Network (WSN) is a wireless network that consists of spatially distributed autonomous embedded systems using sensors to cooperatively monitor physical or environmental conditions [41]. Due to the minimalist nature of the sensors involved, called motes, data storage capacity is very limited. Motes usually make use of low power radio transreceivers, therefore also their communication range is limited. WSNs are usually placed in dangerous or sensitive locations, so there is the need for a reliable way to retrieve data from them, monitor and configure them remotely, under any circumstances. WSN gateway technologies serve to this purpose [42].

Wireless radio technologies, like IEEE 802.11, can not be used directly with WSNs because of their high power consumption. Low-power and low-rate technologies, such as IEEE 802.15.4 or proprietary technologies, are used instead and they usually operate in 433MHz or 868/916MHz frequency bands [43]. Furthermore, also the communication protocols used by WSNs should be designed to fit their limited resources, because Internet protocols would introduce too much overhead. Consequently, WSNs typically make use of proprietary communication protocols. To connect such networks to the wide-area networks, such as WANs, LANs and cellular networks a gateway usually needs two interfaces, one for the wide-area network and one for the WSN.

To serve this purpose different proposals have been suggested. For example, the modular application-level gateway in [43] contains the protocols of both the networks and the protocol-translation phase happens at the application layer. This WSN gateway allows re-configuration of services and protocols to best adapt to different heterogeneous wide-area networks and different combination of protocols that may be used. The modular gateway is shown in Fig.2.12. Its functionalities can be divided into control plane and data plane [43]. Data

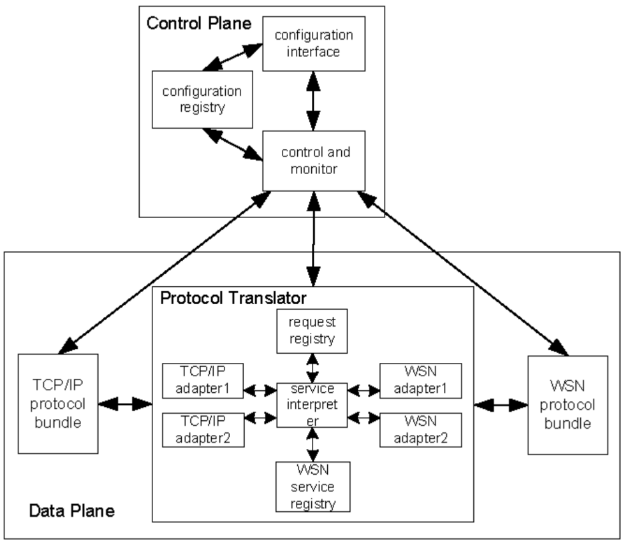


Figure 2.12: The modular WSN gateway architecture (reprinted from [43]).

plane entities administer the user access to the WSN, providing them with different interfaces to the WSN. Through the control plane, the user who has administrative privileges can configure and update the data plane entities. The control plane can also monitor the data plane status.

Another WSN gateway proposal focuses instead only on the WSN-GSM interconnection [41] and is shown in Fig.2.13. This gateway approach makes use of a

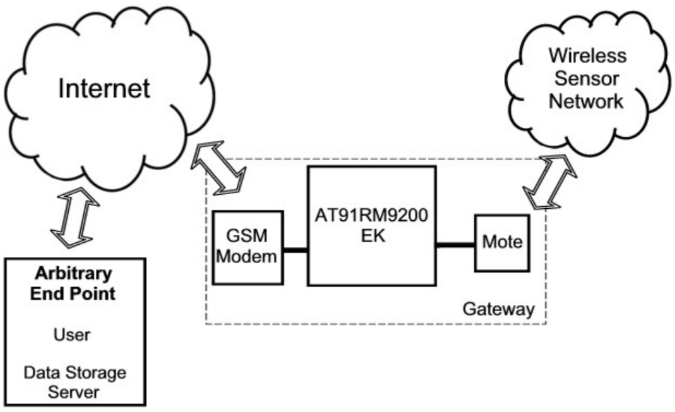


Figure 2.13: System hardware structure (reprinted from [41]).

Global System for Mobile Communications, originally Groupe Spécial Mobile, (GSM) modem for the Internet connection and a mote to connect the gateway to the WSN. Non-volatile memory and AT91RM9200-EK evaluation kit (EK) form the core of the system. This gateway also uses the protocol-translation approach. Packets from the WSN and complying with protocol IEEE 802.15.4 are first captured by the mote, then converted to the USB protocol, and finally to the General Packet Radio Service (GPRS) one and sent through the Internet via the GSM modem. Packets from the Internet follow the inverse procedure.

3

Backround on SCION

This chapter is a high level overview of the next generation architecture SCION [4] [6] [5] [7] whose goals consist of solving security, scalability and availability problems that plague the current communication networks, and they have been already extensively described in Section 1.2. SCION provides a point-to-point communication framework to achieve such properties and the infrastructure should remain available even in presence of powerful adversaries. As long as an attacker-free path is discovered between two communicating endpoints, this should be discovered and used with guaranteed bandwidth for the communication. In the following we give a summary of SCION's key concepts and mechanisms as the *Isolation Domains (ISDs)* and *Trust Root Configuration (TRC)* (Section 3.1), the control plane and all its fundamental SCION's network elements (Section 3.2), the data plane and the structure of SCION packets (Section 3.3), and the infrastructure's security aspects introduced (Section 3.4). In the last section we introduce some specific background concepts needed for the comprehension of Chapter 4 (Section 3.5). These are the *SVC addresses*, the *RAINS* naming service and the *SCION Daemon*.

3.1 Overview

An **Autonomous System (AS)** is a collection of networks that are maintained and administered as a single entity and internally connected using common routing protocols. In SCION, a set of ASes is logically grouped into an **Isolation Domain (ISD)**, a key concept of the SCION architecture to provide transparency and an heterogeneous trust environment. Every ISD is administered by a set of ASes called the ISD Core, and each AS within this set is called a Core AS. Each ISD is associated to a human-readable, globally unique namespace and contains a configuration file, called the **Trust Root Configuration (TRC)**, which states which roots of trust the ISD relies on to validate bindings between entities and their public keys or addresses. To join an ISD, an AS needs first to accept the ISD's TRC file. Subsequently, all the ASes within the same ISD need to agree on the entities that operate the trust root(s) and determine an internal policy. ISDs may also overlap or their topology may present a hierarchical structure which comprises sub-ISDs. Fig.3.1 shows a simple example of such topology and its building blocks.

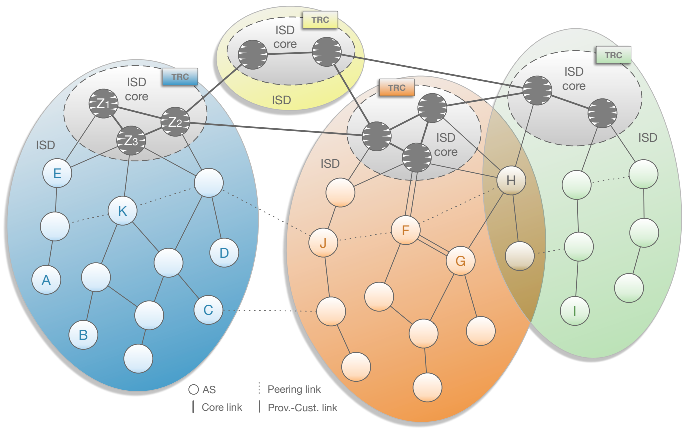


Figure 3.1: Autonomous systems (ASes) grouped into four ISDs. The core ASes are connected via core links. Non-core ASes are connected via customer-to-provider or peering links. AS H participates in two ISDs. (reprinted from [7]).

Compared to today's Internet, SCION is split into smaller logical pieces (the ISDs) which operate independently relying only on the locally accepted roots of trust. From a security perspective SCION is designed with the principle of privilege separation, that means all the control mechanisms are separated from the data plane as described in the following.

3.2 Control Plane

The control plane's task is to discover the available paths, communicate these to the end-hosts and route the traffic through them accordingly. SCION makes use of two different level of routing, intra-ISD and inter-ISD, which both use **Path Construction Beacons (PCBs)** to explore routing paths possibilities. A PCB is built by a Core AS and then floods it either within an ISD (to discover intra-ISD paths), or amongst Core ASes (to discover inter-ISD paths). Traversing different ASes, the PCBs collect cryptographically protected AS-level path information. Such information is chained together to create a transmission path segment which traverses a sequence of consecutive ASes. PCBs allow hence the traversed ASes to learn the path segments to reach their Core ASes and allow each Core AS, in turn, to acquire information on how to reach other Core ASes. With this mechanism, border routers do not need to maintain inter-domain routing tables anymore because all the needed AS-level path information is carried by the packets. This concept is called **Packet-Carried Forwarding State (PCFS)**. Unlike today's Internet where a packet is forwarded according to routing tables, in SCION it is forwarded according to a fixed path information, contained in the packet itself, that comprehends all the routing information needed.

In the following the SCION architecture's fundamental elements and the procedures to build, use and revoke a SCION path are described.

3.2.1 Beacon Server

Beacon Servers (BSs) are the elements responsible for path discovery and dissemination of PCBs information. BSs in a Core AS periodically generate intra-ISD PCBs and flood them to all the ASes within the same ISD. BSs in non-Core ASes receive such PCBs and propagate them to their own customer ASes. During inter-ISD propagation, PCBs are flooded along policy-compliant paths in order to discover multiple paths between any pair of Core ASes. To ensure consistency of the paths information, local servers run a fault tolerant protocol to agree on a master Beacon Server which, in turn, periodically generates a policy-compliant set of PCBs to announce the paths through which it wants the ASes to be reached and forwards it to all its customer ASes.

3.2.2 Path Server

Path Servers (PSs) are the elements responsible for storing mappings from AS identifiers to sets of such announced path segments and, similarly to today's DNS systems, they are organized as a hierarchical caching system. Each AS, with the help of the master BS, selects a set of path segments through which it wants to be reachable and upload it to a PS in the ISD core. PSs learn path segments by extracting such information from the PCBs fetched by local BSs. A PS keeps updating the registered segments to face possible link failures that may happen, segments expiration or better segments that may become available.

3.2.3 Certificate Server

Certificate Servers (CSs) are the elements responsible for the validation of the received information by using cached copies of TRCs retrieved from the ISD Core, and other ASes' certificates. They manage keys and certificates for securing intra-ISD communication. CSs are queried by BSs whenever they need to validate the authenticity of a PCBs.

3.2.4 Border Router

Boarder Routers (BRs) are the elements responsible for the connection between different ASes in SCION and their main task is to forward packets. BRs handle control packets and data packets differently. When a control packet is received, it is immediately forwarded to the appropriate server. When a data packet is received instead, the BR checks if the final destination is within the same AS or not. If this is the case, the BR simply forwards the packet to the end-host, otherwise it forwards it to the next BR, selected according to the path information carried by such a packet.

3.2.5 Path Construction

A SCION path is a combination of bidirectional and reversible multiple path segments. Path segments between Core ASes and non-Core ASes are either called **up-segments** (*towards* the ISD Core), or **down-segments** (*from* the ISD Core to a non-Core AS). In case the destination AS is in another ISD, a **core-segment** between the two ISD Cores is needed to achieve end-to-end communication.

Establishing an end-to-end communication is enabled by a combination of up to three path segments. The source end-host first learns the up-segments to reach its Core AS by querying the local PS which, in turn, forwards the resolution request to the Core PS. In case the destination is *within* the same ISD, the Core PS answers sending up to m down-segments to the local PS. If the targeted host is *outside* the ISD, the Core PS queries all the down-segments from the Core PS within the destination ISD, before responding to the local PS. In both cases, the local PS responds to the source end-host with up to k up- and down-segments and possibly also core-segments, if needed.
Special cases may apply if the up- and down-segments intersect at a non-Core AS or a peering link is present between the two segments. In the former case, a shorter path where the extraneous part of the path is removed is chosen. In the latter case, a shortcut path via the peering link is used.
After the selection of a valid path, this is encoded in the SCION packet header. The final destination end-host can respond either inverting the path stored within the received packet, or choosing a new path using the same procedure, described above.

Possible link failures are not automatically resolved and must be handled by the end-hosts, possibly requesting new paths. On the other hand, PCBs are disseminated every few second, hence new paths are quickly established and available to be fetched. This approach used by SCION greatly masks possible link failures. The availability of a link is continuously checked using SCMP messages (Section 3.4.3) and, in case a link breaks, a new PCB dissemination is triggered. Furthermore, since multipath communication is in place, another way end-hosts can use to face link failures is simply switching to another available path.

3.2.6 Path Revocation

Unlike DNS, where revocation relies only on TTL-based revocation, SCION provides an explicit path revocation mechanism. Apart from expiration, revoking a path should be allowed only with proper authorization to prevent an adversary from disconnecting an entire AS from the global connection by repeatedly revoking the paths to that AS. Revocation in SCION is based on AS interface revocation and works as follow:

- If an interface has to be revoked by an AS, the corresponding BS informs all end-hosts to make them aware of the revoked interfaces within an AS.
- If an end-host receives a packet containing the revoked interface, it issues a revocation message which is propagated along the reversed path contained

in the header of such packet received. Additionally, the end-hosts along the path forward the message to their local PS, so that the affected paths get removed.

- BS in downstream-ASes have to be informed about the revocation, because they should not be able to register new paths containing the revoked interface and they also need to deregister affected paths.

3.3 Data Plane

After discovering the paths, the data plane is responsible for forwarding the packets using the discovered paths. BRs forward SCION packets according the AS-level path information carried in their headers, without the need to inspect the final destination's address or consult any routing table. Only the BR in the destination AS needs to inspect the destination address of a SCION packet to forward it to the local end-host.

A fundamental aspect of the SCION data plane design is the separation of the locator (path towards the destination AS) and the identifier (the destination address). In this way, an AS can choose any type of identifier arbitrarily for its hosts as long as its BRs can interpret where to forward a specific packet.

3.3.1 SCION packets

SCION packets are structured as in Fig.3.2. A SCION packet is composed of a

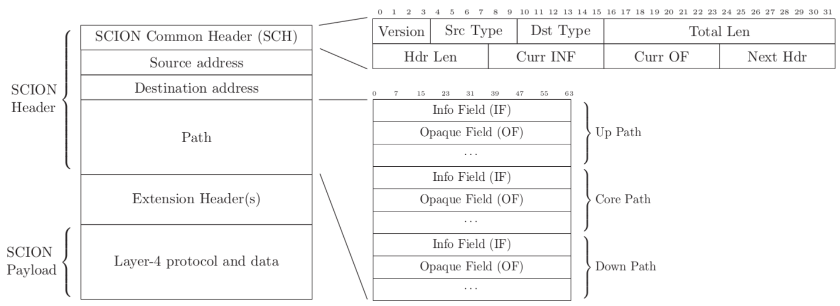
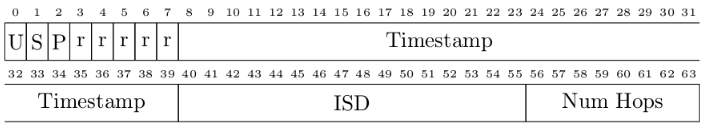
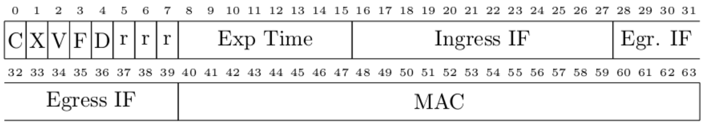


Figure 3.2: SCION packet details. On the left side, there is the high level layout of a SCION Packet. The SCION Common Header (SCH) and the SCION Path are shown in details on the right side.

payload, which encapsulate a layer-4 protocol (e.g., TCP or UDP) message, and a SCION header. The header, in turn, consists of a mandatory SCION common header (SCH), which encodes the most important fields as the packet length, the type of addresses used and the current position in the chosen path. After the SCH, the SCION header embeds the source and destination addresses, that may be optional in case the packet's context is unambiguous without address (e.g., forward packets on loopback). Next, the used SCION path is enclosed, which consists of up to three parts (Up-, Down- and – possibly – Core-Path). Each path segment presents an Info Field (IF) that is shown in Fig.3.3(a). It specifies



(a) Info Field (IF)



(b) Opaque Field (OF)

Figure 3.3: SCION Path Fields in detail.

the type of the path and contains information needed for the Opaque Field (OF) validation, such as the length of the path segment, the ISD's identifier and the timestamp of the ISD which initiated the propagation of the path. Each path segment also contains an Opaque Field (OF) (shown in Fig.3.3(b)), for each AS traversed by the considered path, this field contains information needed by BRs for packet forwarding, like ingress and egress interfaces, expiration time and a Message Authentication Code (MAC). The expiration time expressed by a 1-byte value is relative and the absolute expiration time is computed as follows (in seconds):

$$Timestamp + ((1 + ExpTime) \cdot \left\lfloor \frac{24 \cdot 60 \cdot 60}{256} \right\rfloor) \tag{3.1}$$

Thus, the minimal lifetime of an OF is around 5 minutes, while the maximum lifespan is around 24 hours.
The last SCION header's field is for the optional Extension Header(s) that provide extra parameters to a SCION packet.

3.4 Security Aspects

SCION presents an arsenal of security mechanisms to protect its network from malicious ASes and to provide a secure control plane. Following an overview of the most relevant ones. Their details can be found in papers [4] and [44].

Similarly to BGP [45], every AS signs the PCBs it forwards to enable their validation by all the entities. To achieve path correctness also the information within each PCFS would need to be cryptographically protected, but signatures would hamper the forwarding performance. Thus, every AS makes use of a secret symmetric key that is shared among BSs and BRs, this one is used to efficiently compute a Message Authentication Code (MAC) over the per-AS forwarding information. The MAC field is also called **Hop Field (HF**) and its structure only depends on the AS itself, without need of coordination among different ASes. The HF is computed over ingress and egress interfaces and has an expiration time. The specified interfaces uniquely identify the links to the previous and following connected ASes. If a router is connected via the same outgoing interface to three different ASes, three different egress interfaces identifiers are assigned. The HF's expiration time can have the granularity of seconds or hours, depending on the kind of path. In the following of this overview section we only consider long lasting paths with an HFs that presents an expiration time of the order of 12 hours.

3.4.1 Algorithm Agility

SCION is developed in algorithm agility, meaning that the algorithms can be easily and quickly replaced. The validation of the HF is per-AS, meaning that an AS can independently update its keys or cryptographic mechanisms without the need to agree on those with neighboring ASes. Multiple signatures are supported, so an AS can readily deploy a new signature system and add new signatures as well.

3.4.2 Authentication

SCION authentication is based on digital certificates, which bind identifiers with public keys and carry digital signatures that are verifiable by the roots of trust. The challenges that SCION aims to solve are achieving trust agility to enable flexible selection of trust roots, provide resilience to compromised private keys, and efficient key revocation.

In today's architectures, participants mutually agree on either a single (*monopoly model*) or several (*oligopoly model*) roots of trust which are equally and fully trusted. The main problem of the monopoly model is that it represents a single point of failure, while the one of the oligopoly model is it exposes several points of failure and is as much secure as its weakest point. SCION allows each ISD to define its own roots of trust using TRC files. In this way, compromising a private key associated with a trust root can not help to forge certificates outside the ISD. The TRC has a version number, a list of public keys which identifies the roots of trust, and policies to specify how many signatures are needed to perform a specific action.

3.4.3 SCION Control Message Protocol (SCMP)

The control plane includes the **SCION Control Message Protocol (SCMP)** that is an adaptation of the ICMP protocol to the SCION architecture and is also authenticated. Introducing authentication for SCMP messages is very challenging, since the naive approach of simply using signatures would create bottlenecks at routers in case many SCMP messages would be created in response to a link failure. For this and other purposes, SCION deploys an efficient key derivation mechanism called **Dynamically Re-creatable Key (DRKey)** [46].In DRKey, each AS makes use of a local secret key known to BRs in order to derive on-the-fly a per-AS secret key using an efficient pseudorandom function (PRF). Modern hardware implementations of block ciphers are even faster than DRAM memory lookups, thus, deriving suck key results in a speedup over fetching it from the memory. In order to validate an SCMP message received, the destination AS can fetch the derived key through an additional request message from the originating AS, which is protected by a relatively slow symmetric operation. However, local caching ensures such key is fetched infrequently. The result of the SCMP protocol is secured network control messages with minimal overhead.

3.5 Important Background Concepts

In this section some features of the SCION architecture are explained more in details because they are needed for the comprehension of some concepts expressed in chapter 4.

3.5.1 SVC address

In SCION, a **Service address (SVC address)** is a 2-bytes field used by SCION applications to register their UDP, SSP and TCP sockets. The SVC field specifies the kind of service that is bound to that socket (e.g, path service, SIG service, beacon service, certificate service, etc.), and for non-SCION services the SVC field is set to `None`. For example the SVC for a beacon server is `0x0000` and for a path server is `0x0001`.

This service addressing scheme greatly facilitates control plane anycast communication. If, for example, a BS wishes to register segments with a remote AS's PS, it does not need to know the actual address of a remote PS. Instead, it is only required the SCION SVC of the PS (i.e., `0x0001`), such that the SCION border router in the remote AS can select an alive instance of the PS service to deliver the packet to.
To implement this primitive, all the SCION BRs keep running a *discovery service* within their ASes to keep lists of alive instances for all the supported services. In this way, when a BR detects a packet directed to one of the supported services, an alive instance of that service is selected pseudo-randomly and the packet is sent to it.

3.5.2 Another Internet Naming Service (RAINS)

The details of the RAINS Protocol and the RAINS Client Protocol are specified in an Internet-Draft [47]. RAINS is fundamentally a message-exchange protocol whose goal is name resolution. Briefly, a client sends queries to a specific query-server, that is listening on an SVC address (see Section 3.5.1), and expects responses. The query-server may also send queries to other query-servers and/or authority-servers to receive information about a given name. A query-server forwards assertions, shards, and zones to other servers to respond to specific queries, or based on some other interest presumed by the query-server.

At the current status of SCION, query- and intermediate- servers are organized into a hierarchical cache. Every AS runs a service anycast query-server, and the queries that can not be served out of that query-server's cache are delegated to the service anycast query-server of one or more upstream ASes.

3.5.3 SCION Daemon

SCION provides a background process for its end-hosts and provides an API for applications and libraries to interact with its control plane. End-hosts can use the daemon to connect to SCION and interact with other SCION hosts. The SCION Daemon provides functionality for path lookup, name resolution, trust management, topology information and extension services.

An application which is using the SCION Daemon to connect to the SCION network can use the daemon's API to fetch the available path(s) toward a specific destination. These can then be used to send messages over SCION to such destination.

4 Design

In this chapter we describe how the SIG module that interconnects the SCION network with the legacy IP world was designed and in which scenarios it works or does not work. Part of its design was first proposed by the ETH Zürich Network Security Group [7] and was further improved during this research work.

In the following, we first describe the challenges introduced by the proposal of a new Internet architecture that are related to the SIG (Section 4.1): ensuring maximum interoperability with the current Internet through minimal infrastructure changes, enabling transparent operations for legacy end-hosts in the IP domain, preventing downgrade attacks to the legacy Internet. We then introduce the SCION-IP Gateway (SIG) explaining the way it addresses the presented challenges and the benefits it introduces in terms of security, even for legacy-to-legacy traffic (Section 4.2). We show all the possible real use case scenarios case-by-case demonstrating how the maximum interoperability is achieved (Section 4.3). Next, we discuss some rare cases not addressed by the current design since the design goals were to enable the common communications in an efficient way (Section 4.4). Finally, we explain how the SIG service also introduces benefits for the legacy traffic that passes through and helps mitigating DoS and DDoS attacks (Section 4.6).

4.1 Problem Space

First, we describe the requirements for the SCION-IP Gateway on interconnecting the two networks.

4.1.1 IP-in-SCION Encapsulation

Transporting legacy IP traffic over the SCION network obviously requires to encapsulate the IP traffic into SCION packets.

The tunneling protocol used by SCION is intentionally chosen *non-reliable* to avoid issues with stacking retransmission timers [48][49]. In fact, wrapping a reliable protocol with another reliable one can cause retransmission storms in case of packet loss and this can lead to link congestion and could also be used to set up a DoS attack. In addition, if a reliable tunneling protocol is used and the lower layer retransmission timer is slower than the upper one, when the lower layer starts loosing packets it increases its timeout. At this point the upper layer, which has a faster timeout, will not receive a timely acknowledgement and will queue a retransmission. Having the upper layer timeout still faster than the lower layer's one implies that the upper layer will queue up more retransmissions faster than the lower layer can process them, leading to a stall of the upper layer connection.[49].

Another consideration about the encapsulation protocol regards its isolation from the IP traffic. Recalling that the SCION packet payload size is variable (depends on path length, MTU and SCION extensions used in its header [7]) and recalling that IP-path MTU discovery only allows to decrease the MTU (there is no mechanism to increase it again), the IP MTU for a SCION encapsulated connection will decrease over time as the path changes, which results in bandwidth wastage. The SCION encapsulation protocol should then isolate the IP traffic from the underlying SCION MTU.

4.1.2 Routing and Connectivity

A complete interoperability between the two networks results in SCION to be transparent to all the legacy IP hosts communicating with each others, that means all these hosts will keep communicating in the standard way, dictated by the current legacy Internet, without being aware that SCION is working in between. Proper interoperability also implies that routing must be fully supported between two legacy IP hosts independently of the fact that they are both inside two different SCION ASes or if one of the two is inside a legacy (i.e. non SCION) AS.

Due to this transparency, the same routeability rules are preserved for public and private IP ranges(RFC1918) [50]. The hosts in SCION ASes that want to be reachable by legacy hosts in other SCION or legacy ASes must have public IP addresses.

4.1.3 Addressing

Legacy hosts do not support SCION's name resolution service (RAINS) and they only rely on the legacy name resolution service (DNS). DNS service, though, does not provide any interoperability with SCION addresses and legacy hosts do not know how to route their traffic to a SCION AS – they are not even aware that SCION is working in their between as already mentioned. The immediate consequence is that it is responsibility of the SIG to make sure that the legacy addressing of hosts inside SCION ASes is possible only using their bare IP address.

4.1.4 Support for Layer-4 Protocols

We already mentioned in Section 4.1.1 the isolation between the encapsulation protocol used and the IP traffic. In particular, due to the numerosity of the different layer-4 Internet protocol solutions (as SCTP, L2TPv3, ICMP[2], etc.), any interoperability solution adopted by SCION must be layer-4 agnostic, i.e., it must work for any layer-4 protocol used in the legacy Internet.

4.1.5 Support for SCION-only ASes

Some ASes (as AS E in Fig.4.1) may decide to be connected directly to both SCION and legacy Internet, others (as AS F in Fig.4.1) may decide to be only connected to SCION, as in the case of a bank institution that needs to benefit from all the SCION security improvements and wants to abandon the Internet weaknesses. Both cases should be fully supported be the SIG design.

4.2 The SCION-IP Gateway

The SCION-IP Gateway (SIG) is responsible for the interconnection between SCION and Internet networks. Every SCION AS that wants to communicate with the legacy IP world or wants to connect with the legacy hosts inside other SCION ASes must deploy at least one SIG service. This service takes care of encapsulating the IP traffic into the SCION one at the sender side, decapsulating the SCION traffic into the original IP packets at the receiver side and correctly routing the information to the right legacy end-host through the SCION network. The SIG also takes care of mapping in which SCION AS a certain destination IP address belongs to in order to provide correct routing information through SCION.

All the legacy traffic into (or out of) a SCION AS goes through the SIG service. This gives an extremely important role on the communication to the SIG that must then be fast, fault-tolerant (i.e. capable to handle packets-delay and packets-loss) and reliable also in case of faulty or malicious injected packets or power outages.

In this section we describe in details the actual design of the SCION-IP Gateway. The description is split in subsections, each one of them either addresses one of the requirements previously described in Section 4.1, or introduces new consequent considerations.

4.2.1 Routing

Recalling that the SIG[1] must be transparent to the legacy end-hosts, there are two possible cases for a SCION AS: either it has a direct connection to the legacy Internet or it has not.

In the simplest case, when the SCION AS has a direct connection to the Internet (as the AS E in Fig.4.1), all the outgoing legacy traffic is sent to the SIG simply setting its IP address as default-gateway on all the legacy hosts inside the AS – read Section 4.5 to understand what happens when more than one SIG service is present inside the same AS. For the incoming legacy traffic instead, the AS's IP border routers are used to advertise on the outside the AS's local IP addresses. The SIG IP address is then set up as the next-hop for all those IP border routers in such a way that all the incoming legacy traffic, that obviously has to go through such border routers, is forwarded to the SIG.

[1]In the following the term "SIG" refers to "SIG service" and is used for brevity.

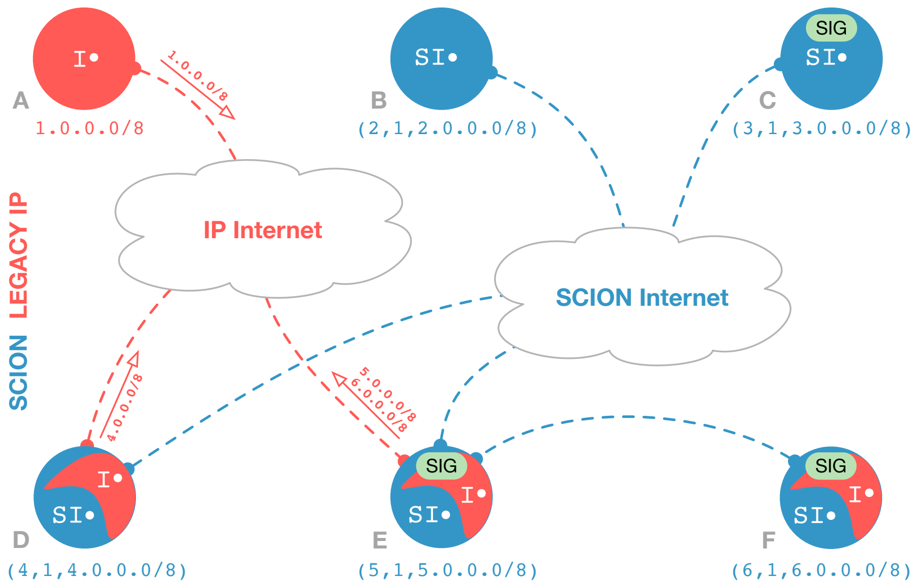


Figure 4.1: Types of networks we consider, differing in whether they connect to the SCION Internet, and whether they deploy a SCION-IP Gateway (SIG) service. The SCION components appear in blue, and the legacy IP components appear in red (reprinted from [7]).

In the more complicated case, when the SCION AS has not a direct connection to the Internet (as the AS F in Fig.4.1), the interconnection with the legacy world has to be offered by another SCION AS (as the AS E in Fig.4.1) through its SIG service. All the outgoing legacy traffic from F is routed to the local SIG that encapsulates it into SCION packets and forward them to the SIG service in the SCION AS E. The SIG in E then decapsulates them into the original IP packets and sends these ones on the legacy Internet through its direct connection to it. For the incoming legacy traffic, the SIG in E advertises on the outside legacy world F's IP addresses. It then encapsulates the legacy traffic directed to F into SCION packets and route them to the SIG in F that, in turn, decapsulates them and forward to the proper local IP end-hosts.

The second case (when the SCION AS has not a direct Internet connection) introduces a further difficulty. The SCION AS under consideration (in our case the AS F in Fig.4.1) has to be previously *associated* with another SCION AS that presents a direct link to the Internet (in our case the AS E in Fig.4.1) that, in turn, has to accept the task to mediate between the SIG in F and the Internet. This requires mutual trust and rigid regulations between the owners of the SCION ASes E and F. Agreements among them may be regulated by a specific global or local authority, but also on private bases. The latter, perhaps through a *customer-provider* or a *peering* relationship (see Section 1.1.2), is in line with the SCION decentralized approach that aims to discourage the usage of a single central authority. For availability reasons, the AS F may also have more than one association with other SCION ASes – and possibly more associations with different SIG services inside SCION AS E, if present – in case the SIG in AS E simply crashes, or in case of bureaucratic problems between the institutions that own the ASes E and F.

An **Outgoing Associations Config** table, where the records are SIG service address and respective encapsulation port pairs, is not needed by a SCION AS without direct Internet connection (as AS F). A simpler list containing only the associated AS identifiers is enough instead (see Fig.4.2). SIG service address or encapsulation-port may change for a considered associated AS (as AS E for example), but the SIG in F would realize it within 500ms – in the case it wants to communicate with AS E. This happens through the SIG negotiation procedure better described in Section 4.2.5, where the SIG in F uses the SIG SVC address for AS E to learn the new address(s) or encapsulation-port(s) of its SIG service(s).

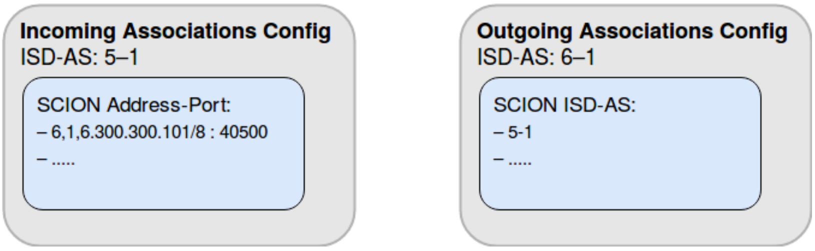


Figure 4.2: Outgoing associations list for AS F (without direct Internet connection) and incoming associations list for AS E (with direct Internet connection). Both the lists may have more than one record.

An **Incoming Associations Config** table, where the records are SIG service address and respective encapsulation port pairs, is needed instead by a SCION AS with direct Internet connection (as AS E). Knowing only the associated AS identifier is not enough in this case, otherwise every host from an associated AS could pretend to be a valid SIG service to the SIG in E. The Incoming Associations Config table for AS E can be seen in Fig.4.2. Also in this case, SIG service address or encapsulation-port may change for a considered remote associated AS (as AS F for example). Updating the Incoming Associations Config table of the SIG in E requires more effort this time. As long as the SIG in E is receiving traffic from the associated AS F, it has to query the SIG SVC address for AS F every 60s and, possibly, update its own Incoming Associations Config table. This, in order to sanitize the table's entries relative to AS F and avoid impersonation attacks or simply believing that another host is a valid SIG in AS F due to delays after some network's updates in AS F.

Keeping the Incoming and Outgoing Associations Config tables up to date is very important to avoid traffic drop and, in the worst case scenario, the complete disconnection of a SCION AS from the Internet network or impersonation attacks.

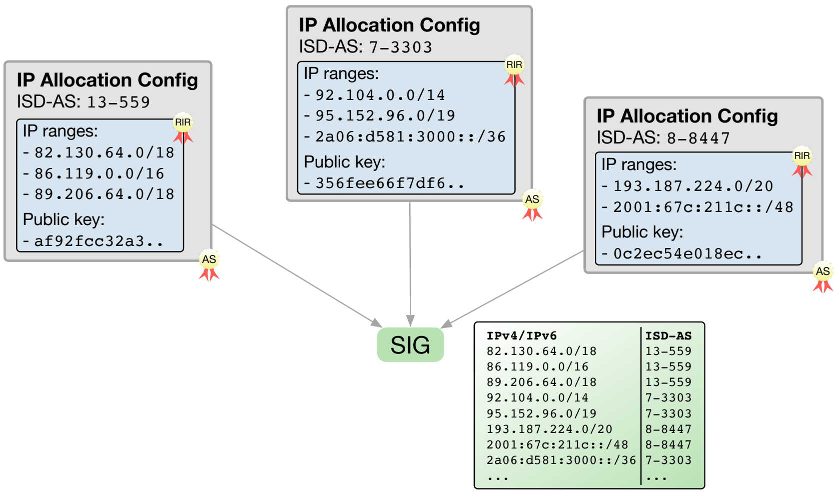
Notice that, AS E may have more than one SIG service running, but may want to disallow one or more of them from serving the traffic coming from the SIG service of the associated AS F. This can be in order to prioritize, through specific SIG(s) in E, local traffic (coming from within AS E), or remote traffic coming from other associated ASes, for performance reasons. In this scenario, any SIG service in AS F has to be aware to which SIG service in AS E can send its encapsulated traffic and to which not. Every SIG in AS E must then have a filtering rule that allows, or impedes, to itself to answer to any SIG SVC address call coming from AS F.

4.2.2 Mapping Legacy IP Addresses to SCION ASes

When the SIG service receives an outgoing legacy IP packet from the SCION AS in which it operates, it has to determine the SCION AS the destination IP belongs to, if any. A mechanism that maps from legacy public[2] IP to SCION AS is needed and it must also be verifiable, in such a way none AS can claim an IP address space it does not own (either maliciously or due to misconfiguration). The goal is achieved by having each SCION AS exporting its own **IP Allocation Config (IAC)** using its certificate server. The IAC contains the list of all the IP ranges the AS owns and the public key of that AS. Fig.4.3 shows the IACs of three different random ASes.

For origin validation purposes [51][52], the list of the IP address ranges inside the IAC is signed by the corresponding issuing RPKI authority [53] that can be the IANA, a Regional Internet Registry (RIR), a Local Internet Registry (LIR), or a National Internet Registry (NIR). The IAC also encloses the proper SCION ISD-AS identifier and timestamps (used also as IAC table version), that are omitted in Fig.4.3 for brevity's sake. Every IAC is finally signed by the SCION AS's private key, which is the private key corresponding to the public one contained in the signed IP prefixes list.

[2]Recalling that only public IP addresses are mapped as explained in Section 4.1.2.



IPv4/IPv6	ISD-AS
82.130.64.0/18	13-559
86.119.0.0/16	13-559
89.206.64.0/18	13-559
92.104.0.0/14	7-3303
95.152.96.0/19	7-3303
193.187.224.0/20	8-8447
2001:67c:211c::/48	8-8447
2a06:d581:3000::/36	7-3303
...	...

Figure 4.3: IACs of three ASes collected by the SIG service, and compiled to a mapping from IP address ranges to SCION ISD-AS numbers (reprinted from [7]).

The SIG service periodically needs to fetch the updated IACs of all the remote SCION ASes, with a procedure described in Section 4.2.2.1, and also has to quickly offer its own updated IAC table to remote ASes that need to fetch it. In case a new SIG service comes on-line it has to immediately start polling remote IACs, and pushing its own for other ASes to speed up the discovery process of itself. Whenever the SIG fetches a new IAC it has to verify its signature. Based on all the received IACs the SIG builds a local mapping of IP ranges to SCION ASes. This mapping is made available for queries by local SCION clients, in case they want to learn in which SCION AS a given IP address is living.

If an outgoing legacy IP packet arrives to the SIG service with a destination IP address that is not in the local mapping, it is assumed to be in a legacy (i.e., non-SCION) AS and then routed to the legacy Internet. In case of a SCION AS with direct Internet connection (as the AS E in Fig.4.1), the IP packet is directly forwarded through this direct link. In case of a SCION AS without a direct Internet connection (as the AS F in Fig.4.1), the IP packet has to be first encapsulated into a SCION packet by the SIG in F, then forwarded to the associated SIG in E, decapsulated there and finally routed to the Internet through E's direct link to it.

4.2.2.1 IAC Polling

The SIG needs to regularly fetch all the possible updated IAC tables from every remote AS in order to build and update its own mapping of IP ranges to SCION ASes. In the following, two proposals are analyzed and the second one is the one adopted by the SIG service. The first one has a relevant rule on explaining the reasons behind the adoption of the second solution.

The simplest way someone can think about to solve this IACs fetching problem is to have the SIG polling all the SIG services in other remote ASes on a 24 hours bases. If, on the one hand, this method certainly does not introduce any complexity and infrastructure changes for the overall SCION architecture, on the other hand, it introduces scalability issues – and for this reasons it is the one not adopted for the SIG service development. A small set of 400'000 ASes (among the 2^{32} possible ones) querying each other over 24 hours would already cause $\sim 1.9 \cdot 10^6$QPS traffic, only considering the ideal case (see following example for details). This is a serious traffic waste, particularly when considering that extremely few prefix changes normally happen among all the global ASes (the CIDR report for the 20th of June 2017 [54] shows that only 11'449 prefix changes happened in the previous 7 days). It has to be considered, though, that ASes granularity is much higher in SCION than the current Internet so prefix changes could happen more frequently than today, but this still would not justify the enormous traffic used to globally fetch the IACs. This reflects in the opposite instead: a 24 hours update frequency may not be enough anymore and lowering it would highly increase the already high dedicated traffic – that in our example it is already $\sim 1.9 \cdot 10^6$QPS for only a small set of 400'000 ASes).

Example Recalling that the current SCION implementation uses 12 bits for the ISD identifier and 20 bits for the AS identifier [7], a global amount of 4'096 ISDs and 1'048'576 ASes per ISD may be found in the SCION network. Assuming only a small subset of all the possible ASes, say 400'000, and only one SIG service per each AS. Every SIG service needs to fetch updates for every single IAC belonging to a remote SIG service, that means that SIG needs to send 399'999 queries. Every SIG service needs to interrogate every other SIG service every 24 hours, this means an overall amount of $\sim 160 \cdot 10^9$ queries, that reflects into $\sim 1.9 \cdot 10^6$QPS. This is a huge number of queries over the network and it is only the ideal case. In fact, some SIG services may be down and reissue queries and some queries may be lost and resent. These issues rapidly increase the $\sim 1.9 \cdot 10^6$QPS value.

The second solution proposed requires more development effort and few changes to the current CSs in the SCION infrastructure, but the performance improvements are worthy the changes. This is why this is the solution adopted for the SIG service development. The SIG regularly pushes the IAC table version to the local CS that, in turn, registers it with every Core AS in the same ISD. This push is done through a keep-alive message that piggybacks the entry (ISD-AS, IAC ver). The SIG can always poll one Core AS in every ISD to fetch updates (possible new IAC version) and the response received is an entry (ISD-AS, IAC ver). In case a newly received IAC version defers from the previously stored one, the SIG can directly query the respective AS itself to fetch its updated IAC. The sending frequency of the keep-alive message to the local CS is 500ms. The SIG polls one Core AS in every ISD with a frequency of 5 hours, as suggested by [52] when RPKI data is used as input to operational routing decisions. With this second approach, the number of queries is largely reduced since the SIG only needs to contact one Core AS per ISD and globally there are many less ISDs than ASes. Furthermore, queries are much lighter in this case since the response now is an entry (ISD-AS, IAC ver) instead of all the IAC itself. Finally, a polling frequency of 5 hours instead of 24 hours allows to have a more updated mapping of IP ranges to SCION ASes and hence fewer routing errors.

It should be noted also that, whenever an AS leaves the SCION network, all the other ASes should update their local mappings removing the entries associated with the leaving AS. A mechanism to implement this does not introduces further complexity to be implemented. Whenever an AS leaves the SCION network, it immediately stops to pushing its own IAC version to the local CS that, it turn, stops to registering it with every Core AS within the same ISD. If a CS does not receive a keep-alive message which piggybacks the entry (ISD-AS, IAC ver) for a specific AS within 2000ms, it discards the last received (ISD-AS, IAC ver) entry received from such AS. Whenever other ASes will poll the CS within a Core AS, they will realize they can not poll this IAC version anymore and they will decide to discard all the prefixes that were associated with that IAC table.

4.2.2.2 Populating Local IAC

Automating the process of mapping the IP prefixes inside an AS and keep them up to date is not trivial. IP address blocks are not assigned to ASes, IP prefixes and ASes numbers are assigned to organizations or people. This means that, if some IP prefixes are used on the public internet, there is no guarantee that the AS that used to announce them is assigned to the same organization those prefixes were assigned to [55].

Several suggestions for ASes lookup tools have been made, but they come with some drawbacks:

- Querying an Internet Routing Registry (IRR) database is a good practice, but these records are far away from being accurate [56][57][58].
- Some of the RIR databases provide accurate information thanks to proper developed tools, but only relative to the allocations they have made. The RIPE RIS project [59][60] is a good example that makes use of a whois server [61].
- Live lookups up of ASes announcements on the public Internet done by checking the BGP route table only tell who is currently announcing a particular IP prefix on a particular AS, but this is not enough information, while live lookups done through external route servers/looking glass (like *traceroute.org* [62]) are very dependent on third party websites and very cumbersome.

For all the previous considerations it has been decided that the local IAC table of a SIG has to be manually maintained by the AS's system administrator. Since the IP prefixes or AS changes already require the human interaction, manually updating the entries of the IAC only introduces a negligible amount of work (particularly because, as already shown, the global weekly changes of prefixes are very few and rare [54]).

4.2.3 Encapsulation

The SIG encapsulation protocol designed is built on top of the UDP/SCION protocol [7], that means it is wrapped in a non-reliable protocol to satisfy the considerations done is Section 4.1.1.

To respect also the other considerations done in Section 4.1.1 and isolate the IP traffic from the SCION MTU, the SIG converts the incoming IP packets into a byte stream to a specific remote SIG service (see Fig.4.4). In this way the number of bytes sent always corresponds to the current SCION MTU size and does not depend on the IP packets' size (some IP packets may be split in pieces and others may be merged together inside the same SIG packet in order to respect the SCION MTU size). A one-to-one conversion from IP packet to SIG packet would not be efficient, while the stream of bytes solution always maximizes the number of bytes that can be sent, respecting the SCION MTU. A stream contains original layer-3 (i.e., IP) and above contents for the encapsulated IP packet(s).

To rise the performances of the buffering phase, the SIG scans the destination address of each incoming IP packet received and build and maintains a different

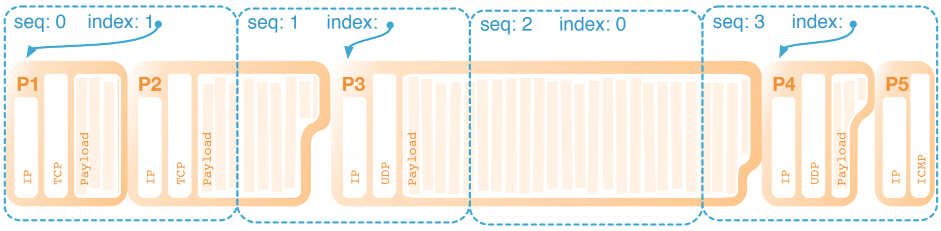


Figure 4.4: Encapsulation of five IP packets (reprinted from [7]).

unique byte stream per each different remote SIG service it wants to communicate to. This means that all the traffic between two SIG services belonging to two different SCION ASes is transported in the same stream.

To optimize transmission time and SIG's memory usage, the encapsulated IP packets are stored in the byte stream without boundaries and padding information.

Each SIG payload starts with a SIG header (as shown in Fig. 4.5) that comprehends: a four-byte sequence number, a two-byte index field and two unused bytes.

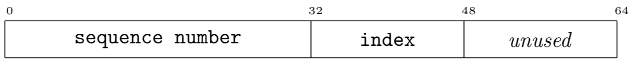


Figure 4.5: Format of the SIG header (reprinted from [7]).

- The `sequence number` is used for packet reordering and detect packet loss by a receiving SIG service. It starts from zero for a given direction of the traffic and pair of SIG services, and monotonically increases by one every SIG packet sent. It resets when the value reaches 2^{32} or when the SIG restarts. Fig.4.4 shows how the sequence number increases by one per each of the five UDP/SCION packets that encapsulate the five IP packets.
- The `index` field allows the receiver to resynchronize in case of packet loss. It points to the next start of an encapsulated packet inside a SIG payload, if any (in Fig.4.4 both the cases are shown). The index is multiplied by eight to get the byte offset from the beginning of the SIG payload. The index value is zero if there are no encapsulated packets that start in this SIG payload (see packet with sequence number 2 in Fig.4.4). The index value is one when an encapsulated packet starts an the start of the SIG payload.

Thanks to sequence number and index field, packet reordering and packet loss can be detected and the receiving SIG can resume operation at the start of the next correctly received encapsulated packet.

4.2.4 Decapsulation

The SIG decapsulation phase is reciprocal to the encapsulation one, but it requires more considerations and design effort due to the complications that may arise due to SIG packet(s) delay or loss, or due to possible packet injection attacks. In this section we explain how the QoS of the encapsulated legacy traffic is improved through the reordering of the received SIG packets (before decapsulating them) by the receiving SIG service and through a timeout mechanism that decides when declaring a SIG packet lost. It follows an analysis of three network behavioural problems, that can disrupt the SIG communication, and their respective solutions: intermittent packet loss, IP header sanitization and light authentication between communicating SIG services.

4.2.4.1 Reordering

Reordering the SIG packets at the receiving side is imperative [63] [64]. IP packets received at the sending side may be received out of order by the sending SIG, and hence stored out of order in the proper byte stream. This means they may also be sent out of order to the receiving SIG, and finally to the destination legacy end-host. If the receiving SIG service does not reorder at least the SIG packets, an extra layer of entropy on the order of the legacy IP packets received by the destination legacy end-host would be further introduced. This would then cause lot of extra timeouts and retransmissions for the encapsulated legacy traffic.

The SIG always reorder the received SIG packets, based on their sequence numbers, before decapsulating them. If a a SIG packet with sequence number X is received, but one ore more immediately preceding ones (with, for example, sequence numbers: $(X-1) \mod 2^{32}, (X-2) \mod 2^{32}, (X-3) \mod 2^{32}, ...)$ are not, the SIG waits a timeout to expire before considering all the delayed packets lost and resume the reordering and decapsulation process from the packet with sequence number X. The timeout is measured from the reception of the packet with sequence number X and is equal to $\frac{1}{2}RTT$ of the path in use (see Section 4.2.4.2 for more details).
As a SIG packet's sequence number is a value $\mod 2^{32}$, and it is unpredictable how long it can take for the sequence numbers $(X-1) \mod 2^{32}, (X-2)$

mod $2^{32}, (X-3) \mod 2^{32}, ...$ to be reused, the respective packets can not be marked as lost for a long period, but considered as such only in the precise moment their timeouts expire. The consequence is that the SIG can not keep track of the lost packets, but should still drop SIG packets that are received after their timeout expired. This situation is unlikely to happen during the normal behaviour of the network, but it could be caused by malfunctioning equipment or packet injection attacks (see Section 4.2.4.5 for details and proposed solution).

4.2.4.2 Timeout and RTT

The timeout after which the receiving SIG declares a SIG packet lost also has important implications on the QoS. If the timeout is too short, many genuine SIG packets would be declared lost too early. If the timeout is too long, the delay introduced, waiting for the delayed SIG packet(s), on forwarding the decapsulated legacy IP datagrams to their destination(s) would rapidly increase. Both the situations would cause many retransmissions on the encapsulated legacy traffic, and in certain cases also the interruption of legacy services.

In case a SIG packet is not received, the receiving SIG waits a timeout equal to $\frac{1}{2}RTT$ of the SCION path in use before declaring it lost. The timeout is measured starting from the correct reception of a SIG packet with sequence number greater than the one expected.

The way the SIG measures and update the Round-Trip Time (RTT) is analogous to the way TCP does it [65]. The value we actually consider is the Smoothed Round-Trip Time (SRTT) that is shown in (4.1) when the first RTT measurement R is made:

$$SRTT = R \tag{4.1}$$

and in and (4.2) for every consequent RTT measurement R' (that happens per each SIG packet received):

$$SRTT = (1-\alpha) \cdot SRTT + \alpha \cdot R' \tag{4.2}$$

α is a parameter that determine how fast we forget the past, the smaller it is the little influence the past has on the current value. The value recommended by RFC [65] is $\alpha = \frac{1}{8}$.

To measure the RTT relative to the path that is used by the sending SIG, the receiver SIG service has first to reverse that path. It then sends a probe message to the sending SIG service and measures the time it takes to receive the respective answer. The probe message is a simple SCION Control Message Protocol (SCMP) (see Section 3.4.3). What complicates this procedure is that SCION provides multipath communication by default, that means possibly having many paths between two communicating SIG services, and hence one different RTT for each. The receiving SIG should then probe the sending SIG through each of the used paths to measure all the different RTT values.

4.2.4.3 Intermittent Packet Loss

In the case that SIG packets are lost with a rapid intermittence, the receiving SIG experiences the reception of many IP fragments that can not be recomposed in the original IP datagram(s). These fragments then do not carry information, but are still transmitted and consume bandwidth. Fig.4.6 shows an example of this scenario.

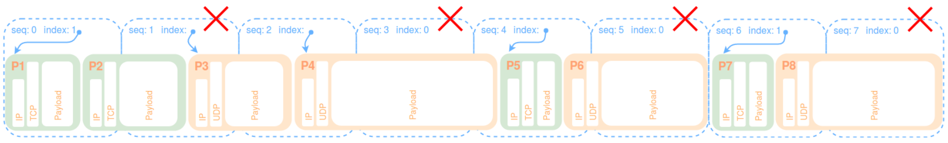


Figure 4.6: Example of intermittent packet loss where SIG packets with sequence numbers: 1, 3, 5, 7 are lost. Green IP packets come from host 5.5.5.5, while orange ones comes from 5.5.5.10. Red crosses symbolize lost packets.

When the receiving SIG realizes an intermittent packet loss is occurring, it could drop the entire block of affected SIG packets, but this would not reduce the bandwidth usage on the SCION network and these packets may encapsulate IP packets coming from other different legacy hosts, and it might be that none – or very few – of the packets sent by some specific hosts have been lost, as explained in the following example.

Example In Fig.4.6 some SIG packets received by the SIG service in SCION AS F are shown. The green traffic comes from host 5.5.5.5 in AS E, while orange traffic comes from 5.5.5.10, also in AS E. Since packets with sequence numbers: 1, 3, 5, 7 are lost, the SIG service in F can not reconstruct any of the IP packets coming from 5.5.5.10, but can reconstruct all the ones coming from 5.5.5.5, except for packet $P2$. This means that the traffic coming from 5.5.5.5 is almost not affected from the intermittent packet loss and possibly should not be dropped.

What the receiving SIG can do when it realizes an intermittent packet loss is in place is to communicate it to the sending SIG that has to immediately choose and start using one or more alternative available communication SCION path(s) (recalling the SCION multipath feature).

4.2.4.4 IP Header Sanitization

The SIG service fully relies on the received IP packets' structure to populate its byte streams. If malfunctions or crafted packet injections occur and no countermeasures are taken, the SIG service can be put on the knees.

If a malformed IP packet with a wrong **Total Length field** is received encapsulated into a SIG packet by the receiving SIG service, it can cause loss of synchronization between sending and receiving SIG services. For space reasons the SIG header contains only one index field (not one for each IP datagram encapsulated in that SIG packet), and there are no boundaries between encapsulated IP datagrams in the same SIG packet so, the only way the SIG has to identify the start of the next encapsulated packet is to use the value of the Total Length field of the current encapsulated packet. The Total Length field of the IP packet pointed by the index field of the SIG header is used as offset (from the first byte of the pointed IP packet) to identify the start of the second encapsulated IP packet, if any. The Total Length field of the second IP packet is used as offset (from the first byte of the second IP packet) to identify the start of the third encapsulated IP packet, if any. So on and so forth. In case of malformed Total Length field(s) the receiving SIG is helpless, so the solution must come from the sending SIG that must check that the Total Length field of each legacy IP packet received must coincide with its real length, if not the IP packet must be dropped. Explicitly checking the correctness of the Total Length field is a consequence of the fact that relying only on the correctness of the IP's CRC may not be enough due to a reasonable hash collision probability [66] [67].

4.2.4.5 Dropping Old Packets

The SIG service fully relies on the sequence numbers and index fields of the received SIG packets for the resynchronization and decapsulation operations. If considerable delays or crafted replay attacks [68] occur and no countermeasures are taken, the SIG service can be completely disrupted.

During the normal behaviour of the communication between two SIG services, a delayed packed should either arrive within $\frac{1}{2}RTT$ at the receiver side or never arrive. Only in extraordinary conditions such a packet can arrive after $\frac{1}{2}RTT$. These are: failures of communication equipment (like border routers, for example), or replay attacks. Recalling that the sequence number of a SIG packet is a value $\mod 2^{32}$, when the SIG service receives a SIG packet with a sequence number different than the one expected, it can not understand if this packet is an extremely delayed one (whose $\frac{1}{2}RTT$ timeout already expired) or a new one. If it is a very delayed packet and the SIG adopts the usual reordering and decapsulation process, synchronization between sending and receiving SIG services is lost. Every SIG header has then also to include a `timestamp` field to prevent this to happen. The SIG header of Fig.4.5 becomes hence the one of Fig.4.7. The 64-bits `timestamp` field is the one used by the **Network Time**

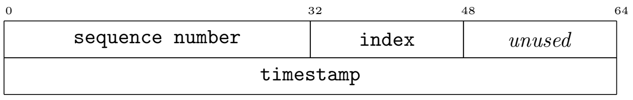


Figure 4.7: Format of the SIG header with timestamp field added.

Protocol (NTP). It consists of 32-bit part for seconds and a 32-bit part for fractional second, it presents a time scale that rolls over every 2^{32} seconds (136 years) and has a theoretical resolution of 2^{-32} seconds [69].

If a SIG packet with timestamp older than a certain range is received, it is immediately discarded. For our purposes this range is approximately $\frac{1}{2}RTT$. This approach allows to avoid the usage of a *replay cache* (like the one proposed by the Kerberos authentication system [70]), where all the packets received are stored, that may become very problematic [71] [72]. Security now depends on SIG services' clocks remaining roughly synchronized and on the trust of the mechanism which sets the system clock.

4.2.4.6 Light Authentication between Communicating SIG Services

The SIG service fully relies on the sequence numbers and index fields of the received SIG packets for the resynchronization and decapsulation operations. If crafted packet injections occur and no countermeasures are taken, the SIG service can be completely disrupted.

In the case a SIG packet is received with a sequence number that considerably differs from the one expected, but which timestamp has an acceptable value, the receiving SIG accepts it as a genuine one and assumes all the packets with sequence numbers in the middle as in late. Two examples are provided in the following.

Example If the expected sequence number is 22, but the received SIG packet carries the sequence number 4’200’000’000 is received, the SIG service assumes that all the packets with sequence numbers in the range [22, 4’199’999’999] are in late. If such packets are not received within $\frac{1}{2}RTT$ from the reception of packet with sequence number 4’200’000’000, they will be considered lost.

Example If the expected sequence number is 22, but the received SIG packet carries the sequence number 10 is received, the SIG service assumes that all the packets with sequence numbers in the range [22, 4’294’967’295] ∪ [0, 9] are in late. If such packets are not received within $\frac{1}{2}RTT$ from the reception of packet with sequence number 10, they will be considered lost.

This scenario is unlikely to happen, but may be due to a faulty behaviour of the network equipment. If so, it means that the packets with sequence numbers in the middle have been lost for real and the receiving SIG has simply to resume from the latest correctly received packet. If network faults are not involved, it means someone maliciously injected one or more crafted packets into the SCION network targeting the SIG. In this second case, the SIG must be able to protect itself.

If we consider the first example provided, the goal of the attacker is to have the SIG dropping as many packets with sequence numbers in the range [22, 4’199’999’999] as possible. In fact, very few of these packets will be received within $\frac{1}{2}RTT$ from the reception of packet with sequence number 22. The result is a resource depletion DoS attack which falls into the category of malformed packets [18] [73].

This attack can disrupt the overall communication between two SIG services in two different SCION ASes, and what is most important is that such attack can be also performed by an off-path attacker within the SCION network. In fact, everyone that knows the receiving SIG’s address can send a SIG packet to it with a crafted SIG header’s sequence number field. The solution is to authenticate the communication between two SIG services and, due to the high amount of traffic shared, this has to be done in a light and fast way. Symmetric cryptographic operations, such as computation and verification of a **Message Authentication Code (MAC)** for each SIG packet is the most efficient one and the **Keyed-Hash Message Authentication Code (HMAC)** is the NIST suggested algorithm to use [74].

Running HMAC on the entire SIG packet is not necessary, but suggested to ensure also the integrity of the message. If HMAC is run over all the SIG packet, considering that the SCION maximum length of a packet is 65’535 bytes, the fastest hashing algorithm to use is SHA-256 (see Fig.2 in [75]). To accelerate the computation of the MAC, the SIG can decide to run HMAC only over: the source ISD-AS (32-bit), the destination ISD-AS (32-bit), the SIG header comprising the timestamp (128-bit) and a random nonce to introduce some entropy (64-bit). In this case the fastest hashing algorithm to use is SHA3-256 (see Fig.1 in [75]). In both the cases the SIG header has to include a 256-bit field for the MAC in its header (see Fig.4.8 and Fig.4.9). Only in the second case a 64-bit field for the nonce is needed (see Fig.4.9).

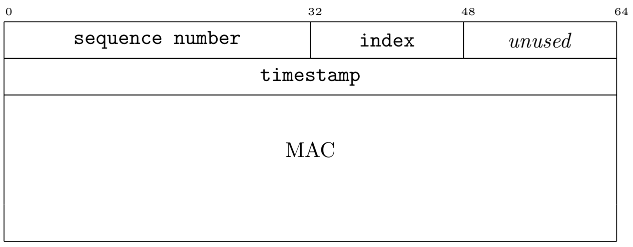


Figure 4.8: Format of the SIG header with MAC field added. MAC algorithm run over the entire SIG packet. HMAC with SHA-256 used.

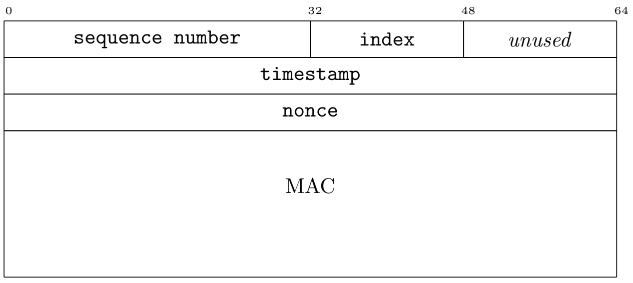


Figure 4.9: Format of the SIG header with nonce and MAC fields added. MAC algorithm run over: source ISD-AS, destination ISD-AS, SIG header comprising the timestamp and nonce field. HMAC with SHA3-256 used.

Embedding a MAC field in the SIG header, the origin of every SIG packet can be now authenticated by the receiver and the injection attack previously explained is no longer viable, assuming that an attacker does not know the symmetric secret key used by the two SIG services to compute the MACs. This symmetric key is securely set up and derived by both the SIG services using the SCION DRKey mechanism explained in Section 3.4.3.

4.2.5 SIG Negotiation

When the SIG service wants to sent encapsulated traffic to another AS, it needs first to determine destination address and port of the SIG service(s) in that AS. Such information can be retrieved contacting the SIG SVC address for that remote AS, but this is not the intended procedure for high volume traffic, as the one produced by the encapsulated traffic[3].

Instead, the sending SIG service first queries the remote SIG SVC address for that AS, and a remote service answers with its own address, SCION control port and encapsulation port. The latter is used because otherwise the encapsulated traffic could not be distinguished from the SCION control one. The sending SIG then uses the retrieved information to directly contact to remote SIG service.

As long as the sending SIG service wants to communicate with the remote SIG, it sends a keep-alive message to it every 500ms. If the respective response is not received within one second, the sending SIG will use the SIG SVC again and will start sending the encapsulated traffic to the new address it receives as response. This allows failover in case the remote SIG instance becomes unreachable.

As already mention in Section 4.2.2.1, the keep-alive message piggybacks an entry (ISD-AS, IAC ver) thanks to which the remote SIG service can also quickly realize if the IAC table of the sending SIG changed and, possibly, update its own local mapping of IP ranges to remote SCION ASes.

If there is no SIG service in the remote AS (or there is not any instance of it running), the remote border router will answer with an SCMP `Unknown Host` in response to the SIG SVC query.

4.2.6 Client Protocol Negotiation

End-hosts that support also SCION architecture can always decide which connection try first when presented with a domain to resolve or a bare IP address, possibly necessitating additional lookups. Purely legacy end-hosts instead have only the Internet connection available and do not have access to RAINS data [47] that means they can not extract information from SCION addresses. In this section we only consider the SCION-enabled clients and their behaviour. Notice that where we mention SCION hosts, it is implied a dual-stacked host

[3]Border routers' processing is heavier for packets with SVC destinations because they have to select a specific service instance.

with both IP and SCION network connectivity. Also note that for local traffic within the same AS, the SIG is not involved.

4.2.6.1 Without DNS/RAINS

A SCION client faced with a SCION address will try a SCION connection first. If this fails, it can then try a legacy IP connection. When a SCION client is faced with an IP address of a remote AS instead, it queries the IAC of a local SIG service to retrieve the corresponding ISD-AS. If a mapping is found, the client will try a SCION connection. If no mapping is found, the client will attempt a legacy IP connection through encapsulated traffic.

4.2.6.2 With DNS/RAINS

When a SCION host is presented with an host name as destination and not a SCION or IP address, it will perform both RAINS and DNS lookups on that host name. The SCION client will always prefer the RAINS answer over the DNS one for positive responses. This means that, if both RAINS and DNS have an entry for a name, the client uses the RAINS answer. If instead there is no entry provided by RAINS, the client can fall back using the DNS answer, if any. If the RAINS answer is a bare IP (i.e., not a SCION address), the client should treat it as a legacy host. If the answer is a SCION address, or if it is an IP address coming from DNS, the SCION client treats it as if it had been given directly that address (see Section 4.2.6.1).

4.2.6.3 Protocol Mismatch

It can happen that a client tries to connect with a protocol the destination service does not support. Errors must then be properly handled and the client must be informed appropriately.

- **IP client $\rightarrow$ SCION service.** The destination host – it has to be recalled that it is dual-stacked – will generate an error depending on the protocol used (`ICMP port unreachable` for UDP, for example). This reply will be routed to the source (if the destination is a remote AS, the SIG will be used), and the client will be informed by its operating system.
- **SCION client $\rightarrow$ IP service on IP host.** The destination host will generate an `ICMP port unreachable` reply that will be routed to the source

in the same way of the previous case discussed. The ICMP error does not contain enough information to make the client's application or socket understand the issue, but in this case the destination host is not dual-stacked and can not provide more information to a SCION service. This error is applied indistinctly to all the SCION connection attempts to the destination host.

- **SCION client → IP service on SCION host.** The destination host will generate an `SCMP port unreachable` reply which will be routed back to the source client as normal SCION traffic, and the client will be informed by its operating system.

4.3 Common Scenarios (Life of a Packet)

We next discuss the common scenarios that occur during the translation between SCION and IP. We refer to Fig.4.1 for the description and for clarity we will use the color blue for the SCION components and red for the legacy IP ones.

1) IP host in SCION AS → IP host in SCION AS
We start with the case of two IP hosts in two different SCION ASes communicating with each other. In Fig.4.1, this could be E.I → F.I or vice versa[4]. We do not consider communication with AS D in this scenario since it does not deploy a SIG service.
We assume that the client host E.I connects to a web server F.I and it knows the corresponding IP address, say 6.6.6.6 (possibly through a DNS lookup). The web server listens on port 80 and makes use of TCP as transport protocol. E.I has E's SIG service set as default gateway.
Once packets from E.I arrives to the local SIG service (in AS E), the address and encapsulation port of the destination SIG (in AS F) are needed. E's SIG service then queries its aggregated IAC table (see Fig.4.3) to identify the ISD-AS associated with the IP address of F.I and send a SIG SVC message to it. A remote SIG instance in AS F will answer with its address, say (6,1,6.0.0.1), a SCION control port P and an encapsulation port Q. E's SIG will then encapsulate E.I's packets as shown in FIg.4.4 and send them to (6,1,6.0.0.1) on the port Q.
If no SIG service is present in the remote AS (as for AS D for example), the remote border router will answer with an `SCMP Unknown Host` as response to the SIG SVC message and E.I's packets will be sent via legacy Internet.

2) IP host in SCION AS w/ IP connectivity → IP host in IP AS

[4]Notice that E.I denotes a legacy IP host inside the SCION AS E.

In this case, a connection from an IP host in a SCION AS with direct IP connectivity (as AS E) to an IP host in a non-SCION AS (such as AS A) is established. E's SIG service, that is the default-gateway for the legacy hosts in AS E, tries to map the destination IP address, say 1.1.1.1, to the respective ISD-AS identifier. Naturally it fails because AS A is not aware of SCION.
E's SIG then needs to fall back to IP routing. As AS A has surely announced its IP prefixes through its BGP-speaking border routers, the SIG service in AS E can use its direct Internet connection to send (non-encapsulated) IP packets to AS A reaching then the host with address 1.1.1.1.

3) IP host in IP AS → IP host in SCION AS w/ IP connectivity
This case represents the reverse direction of the connection of the previous case: a connection from an IP host in a non-SCION AS (such as AS A) to an IP host in a SCION AS with a direct link to the Internet world (such as AS E). As above, since AS E has surely annunced its IP prefixes through its BGP-speaking border routers, AS A sends (non-encapsulated) traffic to AS E.
The incoming IP packets at AS E are routed from the border routers to E's SIG service that, in turn, will forward them to the proper legacy destination hosts within AS E as they are without further processing.

4) IP host in IP AS → IP host in SCION AS w/o IP connectivity
We now extend Case 3 above assuming that AS A connects to a SCION AS that has no direct Internet connectivity (as AS F). In this case, AS F relies on some other SCION AS with direct IP connectivity (as AS E) to advertise its IP prefixes on the legacy IP world making them available for legacy ASes.
Neither the client in AS A, nor AS A itself realize that AS E is used as proxy, thus send the IP packets destined for the legacy host in F, say F.I with IP address 6.6.6.6, to AS E. E's border router forwards such proxy traffic to E's SIG service, which then encapsulate it and forwards it to a SIG service instance in AS F. The negotiation protocol is similar to the one described in Case 1, but this time it takes into account the restrictions imposed by the Outgoing Associations Config and Incoming Associations Config tables of AS F and AS E respectively (see Section 4.2.1).
The return traffic is covered in the explanation of Case 5.

5) IP host in SCION AS w/o IP connectivity → IP host in IP AS
This last case models the return traffic of Case 4 above. The destination address of A.I, say 1.1.1.1, does not appear in the default-gateway's IAC table that tells to the SIG to which ISD-AS send the encapsulated traffic. E's SIG service uses its direct IP connectivity to forward the (decapsulated) return traffic, coming from F's SIG, to the legacy AS A.

4.4 Cases Not Covered

Some possible combinations of IP and SCION networking are not covered by the functionality of the SIG service. These are rare cases or ones that bring only minimal gain in supporting them.

4.4.1 SCION Hosts without IP Stack

A SCION host without the IP communication stack is a special case that adds lot of complexity when communicating with IP hosts. The SIG service described is designed for dual-stacked SCION hosts as default case and SCION hosts without the IP stack are currently not supported.

4.4.2 Transparent Layer-3 Translation (AS Level)

The SIG service could provide transparent layer-3 translation only under the following conditions:

- To connect to the remote side a fixed maximum payload size (meaning either fixed path, or a small payload size) is needed. See the above Encapsulation requirement Section 4.2.3 for more details.
- The SIG service understand and can translate the L4 protocol used (so it can update the checksum, for example)
- All the traffic above layer-3 is completely L3-independent (i.e., not something like IPSEC in AH mode, nor something that embeds layer-3 addresses).

The SIG service does not support these three conditions for general traffic, so this is a not covered case.

4.4.3 Transparent Layer-3 Translation (Service Level)

Supporting the connection between hosts in the IP Internet world and hosts in SCION ASes, the translation should be done on the SCION host, but this would not introduce any benefit over a simply IP connection hence it is not covered.

4.5 Need of many SIG services per AS

The SIG service is the core component that allows legacy traffic communication among remote SCION ASes. If it becomes unavailable for any reason, the entire incoming and outgoing legacy traffic for its AS is disrupted. If we assume, for example, a critical scenario where a SCION AS without direct Internet connectivity (as the AS F in Fig.4.1) belongs to a Bank institution that wants to benefit from SCION features, and its SIG service is not working or not reachable, the result is that none from the Internet world can communicate with that Bank institution anymore with obvious economic implications for the bank. This critical scenario brings to light the problem of having a single SIG service in an AS because it would become a single point of failure for the SCION-Internet interconnection for that AS.

To be resistant against DoS and DDos, power outages or simply network links interruptions, each AS should deploy more than one SIG service. This would not only guarantee reliability and fault tolerance of the SIG service, but would also prevent its congestion in case of heavy traffic conditions since the traffic could be balanced between the different SIG instances within the local AS.

As mentioned in Section 4.2.1, a legacy host inside a SCION AS has the local SIG service set as default-gateway. Since no infrastructure changes are desired for the legacy hosts that want to benefit from the SCION architecture through the SIG, nothing more than setting their default-gateway should be needed by them to communicate with a local SIG service, even if multiple instances of it are running inside the local AS. First Hop Redundancy Protocols (FHRP) [76] can help a legacy host inside a SCION AS to choose the appropriate address to set as default-gateway. On Cisco equipment, there are some different protocols options to choose from, including Hot Standby Router Protocol (HSRP), Virtual Router Redundancy Protocol (VRRP) and Gateway Load Balancing Protocol (GLBP) [77]. HSRP and VRRP provide default-gateway redundancy having one router operating as the active gateway router and one or more other routers held in standby mode. GLBP enables all the available gateway routers to share the workload and be active simultaneously. GLBP in particular would allow to dynamically balance the outgoing traffic from an AS among the different SIG services and to avoid to congest a specific one. To balance also the incoming encapsulated traffic to a remote AS, the SCION border routers could provide a smart way to answer to SIG SVC queries (i.e., answer them with the address of the SIG service that is currently less congested).

4.6 Introduced Benefits for the Legacy Traffic: DoS Mitigation

Interconnecting the legacy Internet with the SCION network through the SIG service also brings benefits to the legacy world. More specifically, the SIG's design provide a way to mitigate DoS and DDoS attacks carried out through the encapsulated legacy traffic transported between one SCION AS to another. Thanks to the fact that the encapsulated traffic between two SIG services in two different ASes is authenticated (see Section 4.2.4.6) and the fact that every AS publishes its IAC table (which contains the list of the legacy IP addresses the AS contains) and this list is signed by the SCION AS's private key (see Section 4.2.2), the receiving SIG can always identify from which AS the encapsulated traffic is coming from. In case a DoS attack is in place from one AS to another remote one, through the encapsulated traffic, and special agreements are in place between the owners of the two ASes, the owner of the receiving under attack AS may require the help of the owner of the sending AS to stop such attack. Having the sending SIG service dropping the malicious legacy IP packets would allow to stop the attack near its origin (i.e., within the AS where the attacker is residing), with benefits in terms of the overall bandwidth for both SCION and Internet networks.

It has to be noticed that, despite every SIG packet is authenticated through the usage of a MAC (see Section 4.2.4.6), IP spoofing could trick the receiving SIG into thinking that the provenance of the malicious encapsulated IP packets received is another AS than the correct one. A trivial mechanism to prevent this is to use ingress traffic filtering [78] on the sending SIG that has then to drop all the IP packets with a spoofed source IP address before encapsulating them. Impeding IP spoofing is also fundamental to prevent powerful DDoS attacks as the amplification attacks, where the attacker sends a packet with the spoofed address of the victim to the broadcast IP address of a network, causing then all the hosts of the targeted network to send a reply to such victim [18].

5 Implementation

In this chapter we provide the implementation details of the SIG prototype developed during this thesis work. We describe tools and technologies needed during the development phase, the algorithms used and the protocols, data structures and file formats chosen.

Due to time constrains and the high interaction of each single module of the SIG system with the others, following an *agile model* for software development was not possible, so we used the *waterfall model* approach instead. A prototype that allows the communication between two ASes through their SIG services was implemented and further evaluated, but not all the features of Section 4.2 were deployed due to external impediments and time limitations, considered the size of the project (see Section 5.5).

In the following, the background knowledge, tools and technologies needed to design and develop the SIG service are introduced (Section 5.1) and the topology of the development and testing environment is described (Section 5.2). We then explain how the codebase was structured to efficiently develop the SIG and respect the design requirements (Section 5.3). Next, we briefly provide an overview of the main software blocks developed that compose the SIG service, and we show some scripts of the most interesting *Python* methods developed (Section 5.4). Finally, all the limitations imposed by external factors on the software development for this thesis work are explained (Section 5.5).

5.1 Tools Needed

During this thesis work a significant previous knowledge about the SCION design and its implementation was needed, together with different tools to develop and evaluate the SIG module.

SCION is an extensive design [7] than involves all the major blocks of the network infrastructure (beacon servers, path servers, border routers, certificate servers, etc.). Its implementation is composed of more that 600 MB of *Python* and *go* software, developed by the Network Security Group of the ETH University, that can be found at `https://github.com/netsec-ethz/scion`.

For this thesis work some time was spent in learning the *Python* programming language, the *Vagrant* tool and the *Ansible* automation platform together with their syntaxes to write the configuration files required to set up the virtual machine environments needed. A considerable time was also used for an extensive reading and understanding of the implemented SCION software in the *GitHub* repository due to the absence of its documentation. Knowledge about working with `pcap` files and live monitoring was required so the *Python* library `pcapy` had to be learned. Forging packets was also necessary so the *Python* library `scapy` had to be learned.

During this project, approximately 2500 lines of code were written (for an amount of 1,5 MB of software produced) for a complex multithreading system which threads all iterate concurrently on the same variables, buffers and data structures. Buffering the legacy IP traffic before converting it to SCION packets (as explained in Section 4.2.3) extremely increased the complexity of the system and required a good knowledge of the *Python* multithreading principles and the behaviour of its data structures in a concurrent context. Finally, a considerable time has been spent researching and learning about DoS and DDoS attacks and countermeasures with the goal of finding a way to protect the SIG service from bandwidth and resource depletion attacks [18].

5.2 Topology of Development and Testing Environment

To better understand the following sections, we already describe the topology of the testing environment adopted during this thesis work. Naturally, for each connection we needed two communicating ASes, each one deploying at least one border router (that can be a legacy or SCION border router depending on the kind of the AS), a SIG service (if and only if it is a SCION AS), and at least one active legacy IP host per each AS. A block diagram of such environment is shown in Fig.5.1. The environment is composed of three ASes and more

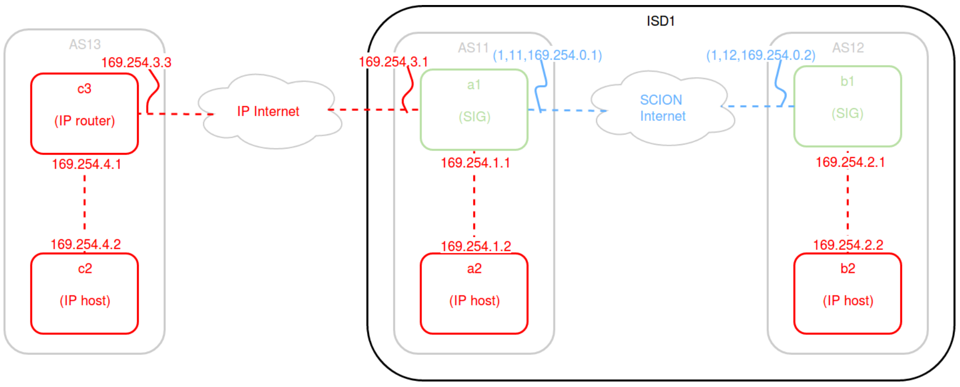


Figure 5.1: Topology of the testing environment adopted. It comprises two SCION ASes (i.e., AS11 that is a Core AS and AS12, that is a non-Core AS) within the same ISD (i.e., ISD1) and a non-SCION AS (i.e., AS13). Every AS contains one legacy IP host (namely a2, b2, c2). The two SCION ASes also contain a SIG service (namely a1, b1), while the non-SCION AS contains a legacy IP border router (namely c3).

specifically two SCION ASes, namely AS11 and AS12, and one non-SCION AS, namely AS13. AS11 and AS12 belong to the same SCION ISD, namely ISD1, and communicate through the SCION network. AS11 presents also a direct connection to the Internet network. AS13 is only connected to the Internet instead. Every As contains a legacy IP host (more specifically, AS11 contains a2, AS12 contains b2 and AS13 contains c2), while only the two SCION ASes deploy a SIG service (more specifically, AS11 deploys a1 and AS12 deploys b1). AS13 deploys a legacy IP router, namely c3, for enabling Internet connection. Fig.5.1 also shows the addresses of the network interfaces used by each entity. Our goal was to make the legacy hosts a2, b2 and c2 communicate with each other through the usage of the SIG services a1 and b1.

Naturally, all the three ASes deploy also all the other network components needed for the communication as SCION and non-SCION border routers, SCION and non-SCION certificate servers, SCION beacon servers, SCION path servers and so on. These components are not represented in Fig.5.1 and will be later kept out of discussion for the sake of brevity.

5.3 Structuring the Codebase

When it comes to maintainability and readability having a good codebase's structure is imperative and a good categorization is needed. All the software developed for this thesis work can be found in the `vagrant` directory in the *GitHub* SCION repository of the Network Security Group (NETSEC) of the ETH University (`https://github.com/netsec-ethz/scion`).

Inside the `vagrant` directory can be found a directory for each of the fundamental entities of the topology in Fig.5.1 (a1, a2, b1, b2, c2), each one contains the necessary files for that entity to operate:

```
vagrant
└─ sig
   ├─ a1
   ├─ a2
   ├─ b1
   ├─ b2
   └─ c2
```

Entities a1 and b1 are SIG services and their directory structure are the same. We then only represents a1's one:

```
a1
├─ pcapy-0.11.1
│  └─ ...
├─ sig.gen
│  ├─ ISD1/AS11
│  │  ├─ br1-11-1
│  │  │  ├─ topology.yml
│  │  │  └─ ...
│  │  ├─ bs1-11-1
│  │  │  ├─ topology.yml
│  │  │  └─ ...
│  │  ├─ cs1-11-1
│  │  │  ├─ topology.yml
│  │  │  └─ ...
│  │  ├─ endhost
│  │  │  ├─ topology.yml
│  │  │  └─ ...
│  │  ├─ ps1-11-1
│  │  │  ├─ topology.yml
│  │  │  └─ ...
│  │  ├─ sb1-11-1
│  │  │  ├─ topology.yml
│  │  │  └─ ...
│  │  └─ ...
│  ├─ dispatcher
│  │  └─ ...
│  └─ ...
├─ Vagrantfile
└─ gw.py
```

`pcapy-0.11.1` contains the library used by the SIG for sniffing the IP traffic. `sig.gen` contains topology and configuration files for the entity a1; this directory has to be renamed `gen` and copied within the `scion` folder on the host where a1 is desired to be run (see Section 6.1). Among the rest, inside the `sig.gen` folder it is present a directory for each network component inside the AS (beacon servers, border routers, certificate servers, path servers and SIBRA servers), every one of these directories include the `topology.yml` file with the local topology of the AS as shown in listing 5.1.

```
BeaconServers:
  bs1-11-1:
    Addr: 127.0.0.10
    Port: 30078
BorderRouters:
  br1-11-1:
    Addr: 127.0.0.9
    Interface:
      Addr: 169.254.0.1
      Bandwidth: 1000
      IFID: 18
      ISD_AS: 1-12
      LinkType: CHILD
      MTU: 1280
      ToAddr: 169.254.0.2
      ToUdpPort: 50000
      UdpPort: 50000
    Port: 30093
CertificateServers:
  cs1-11-1:
    Addr: 127.0.0.11
    Port: 30059
Core: true
ISD_AS: 1-11
MTU: 1400
PathServers:
  ps1-11-1:
    Addr: 127.0.0.12
    Port: 30092
SibraServers:
  sb1-11-1:
    Addr: 127.0.0.13
    Port: 30078
Zookeepers:
  1:
    Addr: 127.0.0.1
Port: 2181
```

Listing 5.1: `topology.yml` topology file for a1.

`Vagrantfile` is the configuration file used by the *Vagrant* tool to set up a Virtual Machine (VM) using *VirtualBox* according the specifics imposed by itself. These specifics are mainly related to network configuration and memory usage, a1's

`Vagrantfile` is shown on listing 5.2 and shows how the network setup respects the one of Fig.5.1.

```ruby
# -*- mode: ruby -*-
# vi: set ft=ruby
Vagrant.require_version ">= 1.9.1"

Vagrant.configure(2) do |config|
  config.vm.hostname = "a1"
  config.vm.box = "ubuntu/xenial64"
  # Link between a1 and b1
  config.vm.network "private_network", ip: "169.254.0.1",
    virtualbox__intnet: "gwnet"
  # Link between a1 and a2 (endhost)
  config.vm.network "private_network", ip: "169.254.1.1",
    virtualbox__intnet: "gwnet-a"
  # Link between a1 and c2 (endhost)
  config.vm.network "private_network", ip: "169.254.3.1",
    virtualbox__intnet: "gwnet-c"
  config.vm.synced_folder '.', '/vagrant'
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
  end
end
```

Listing 5.2: Vagrantfile of a1.

`gw.py` is the *Python* implementation of the SIG service that can be consulted in the digital archive provided together with this written work.

Entities a2, b2 and c2 are legacy IP hosts and their directory structure are the same. We then only represents a2's one:

```
a2
├─ Vagrantfile
│
└─ ansible.yml
```

The `Vagrantfile` of a2 is presented in listing 5.3.

```ruby
# -*- mode: ruby -*-
# vi: set ft=ruby
Vagrant.require_version ">= 1.9.1"

Vagrant.configure(2) do |config|
  config.vm.hostname = "a2"
  config.vm.box = "ubuntu/xenial64"
  # Link between a1 and a2 (endhost)
  config.vm.network "private_network", ip: "169.254.1.2",
      virtualbox__intnet: "gwnet-a", auto_config: false
  config.vm.synced_folder '.', '/vagrant'
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "256"
  end
  config.vm.provision "shell", inline: <<-SHELL
    apt-get update
    apt-get install -y python
  SHELL
  config.vm.provision "ansible" do |ansible|
      ansible.playbook = "ansible.yml"
      ansible.galaxy_role_file = "../requirements.yml"
      ansible.galaxy_roles_path = "../roles"
  end
end
```

Listing 5.3: `Vagrantfile` of a2.

The `ansible.yml` configuration file instead is used by the `Vagrantfile` one to set up the network interface of a2 and set a1 as a2's default-gateway (listing 5.4).

```
- hosts: all
  become: true

  roles:
  - role: dresden-weekly.network-interfaces
    network_manage_devices: no
    network_interfaces:
    - device: enp0s8
      auto: true
      method: static
      address: 169.254.1.2
      netmask: 24
      up:
      - ip route add 169.254.2.0/24 via 169.254.1.1
      - ip route add 169.254.4.0/24 via 169.254.1.1
```

Listing 5.4: `ansible.yml` file of a2.

5.4 SIG Software Blocks

In this section the *Python* script `gw.py` (i.e., the core of the entire implementation process) is decomposed into blocks (`IP_Receiver`, `SCION_Sender`, `IP Sender`, `Delayed Packets Sender` and `SCION Receiver`) that are further analyzed individually for a better understanding of their functioning. All these blocks are concurrent threads of the `gw.py` script, they all run simultaneously and interact with the same variables, buffers and data structures. For a better management of the `gw.py` script and in order to prevent errors, all these threads can be simultaneously killed by the `threading.Event()` object called `run_event` shared by all of them.

5.4.1 IP Receiver

The `IP_Receiver` thread's task is to capture the IP packets received by the SIG service itself, check their IP destination address and buffer them into the correct byte stream as explained in Section 4.2.3. In our environment only two SIG services are present (i.e., a1 and b1), this means that a1 only preserves a byte stream of IP packets directed to b1 and vice versa. This byte stream in the `gw.py` script is called `ip_buf` and is a `deque` thread-safe list-like container with memory-efficient appends and pops operations.

The SIG service makes use of the `pcapy open_live` utility to capture incoming IP packets in transit on its legacy Internet interface. Whenever an IP packet is received, its source IP address is first checked. If it does not belong to the prefixes owned by the local AS and written in the SIG's local IAC table, it means the packet has a spoofed source address and the SIG drops it. Notice that, due to limitations of the implemented SCION software (see Section 5.5), the IAC table was simply hard-coded in the `gw.py` script. If the source IP address is a genuine one instead, the SIG further processes it and checks if its Total Length field matches the real length of the packet to avoid the issues explained in Section 4.2.4.4. Only if the Total Length field is correct, the packet is deprived of its Ethernet header and appended as sequence of bytes in `ip_buf`, otherwise it is dropped. The SIG service does not need to encapsulate ARP packets hence these ones are always dropped.

5.4.2 SCION Sender

The `SCION_Sender` thread's task is to extract the information bytes from `ip_buf`, encapsulate them inside SCION packets and send them to the correct remote SIG service, using the SCION path(s) fetched from the SCION Daemon 3.5.3. Recalling that, a1 and b1 have only one byte stream each (as mentioned in Section 5.4.1), it is not needed a priority mechanism that chooses from which `deque` container extract the bytes to be encapsulated and sent over the only one SCION interface available. However, such mechanism could have used a Round-robin scheduling mechanism or could have prioritized the oldest packets received based on stored timestamps.

When the `SCION_Sender` thread is first created, it requests to the SCION Daemon the SCION path(s) for the only available remote SIG service (i.e., a1 requests the path(s) for b1 and vice versa). Listing 5.5 shows the methods `_get_path` and `_try_sciond_api` needed to request the path(s). In a real life scenario the path(s) should be frequently updated with consequent `_get_path()` requests, but in our environment this is not needed since the topology available comes from the static configuration file `topology.yml`, as already described in Section 5.3.

```python
def _get_path(self, api, flush=False):
    """Request path via SCIOND API."""
    path_entries = self._try_sciond_api(flush)
    path_entry = path_entries[0]
    self.path_meta = path_entry.path()
    fh_info = path_entry.first_hop()
    fh_addr = fh_info.ipv4()
    if not fh_addr:
        fh_addr = self.dst.host
    port = fh_info.p.port or SCION_UDP_EH_DATA_PORT
    self.first_hop = (fh_addr, port)


def _try_sciond_api(self, flush=False):
    flags = lib_sciond.PathRequestFlags(flush=flush)
    start = time.time()
    while time.time() - start < API_TOUT:
        try:
            path_entries = lib_sciond.get_paths(self.dst.isd_as,
    src_ia=self.addr.isd_as, flags=flags, connector=self.
    _connector)
        except lib_sciond.SCIONDLibError as e:
            logging.error("Error during path lookup: %s" % e)
            continue
        return path_entries
    logging.critical("Unable to get path from local api.")
    kill_self()
```

Listing 5.5: `_get_path` and `_try_sciond_api` methods needed by the SIG service with address `self.addr.isd_as` to request the path(s) for the remote SIG service with address `self.dst.isd_as`.

Whenever the size of the `ip_buf` is greater than the current SCION MTU, a number of bytes equal to the SCION MTU is extracted from the `deque` container to build the payload of a SIG packet. The SIG header is also built, with the information state kept by the `SCION_Sender`, and appended to the SIG payload. The SIG packet created is then encapsulated as payload of a regular SCION packet and sent through the path(s) previously fetched. Time constraints impeded to implement the improvements of Section 4.2.4.5 and Section 4.2.4.6, so the SIG header used for this implementation is the one of Fig.4.5. The `SCION_Sender` thread increment by one the `sequence number` field of the

SIG header every time a packet is sent, while the `index` field is calculated by the method `_offset_next_encap_pck()`. Listing 5.6 shows the methods `_create_payload()` and `_offset_next_encap_pck()` used to create the SIG packet (i.e., the SCION packet's payload) and to calculate the `index` field of the SIG packet's header.

```
def _create_payload(self, spkt):
    format = '!IHH%ss' % SCION_PAYLOAD_LENGTH
    pld = bytearray()
    for i in range(0, SCION_PAYLOAD_LENGTH):
        pld.append(self.buf.popleft())

    self.index = self.offset + 1
    encap_pck = pack(format, self.sn, self.index, self.unused,
    pld)

    print('sn: %s\nindex: %s\nencap_pck: %s' % (self.sn, self.
    index, encap_pck))

    # calculate the index field
    if self.no_encap_counter > 0:
        # no encapsulated packets start in this payload
        self.offset = -1
        self.no_encap_counter -= 1
    else:
        self._offset_next_encap_pck(pld)
        self.index = self.offset

    return PayloadRaw(encap_pck)


def _offset_next_encap_pck(self, payload):
    previous_offset = self.offset

    while self.offset < SCION_PAYLOAD_LENGTH:

        ip_header = payload[self.offset:self.offset +
    IP_header_length]
        if len(ip_header) != IP_header_length:
        # Extract missing header
            tmp = bytearray()
            for i in range(0, (IP_header_length - len(ip_header))
    ):
                tmp.append(self.buf.popleft())
            ip_header = ip_header + tmp

            # Re-append previously extracted peace of header
            for i in range (len(tmp)):
                self.buf.appendleft(tmp[len(tmp)-1-i])

        # Extract Total Length field of IP packet
        iph = unpack('!BBHHHBBH4s4s', ip_header)
        ip_length = iph[2]

```

```
45          self.offset = self.offset + ip_length
46
47      # Update offset and counter for packets with index=0
48      self.no_encap_counter = int((self.offset - (
        SCION_PAYLOAD_LENGTH - previous_offset))/
        SCION_PAYLOAD_LENGTH)
49      self.offset = self.offset % SCION_PAYLOAD_LENGTH
```

Listing 5.6: `_create_payload()` and `_offset_next_encap_pck()` methods used to create the SIG packet (that is the SCION packet's payload) and to calculate the `index` field of the SIG packet's header respectively.

Identifying the correct value of the `index` field for a SIG packet's header is not a trivial process. The `index` field is always equal to `self.offset + 1` (line 7), where `self.offset` is a variable used to determine the ending point of the currently analyzed IP encapsulated packet inside `ip_buf`. At the first interaction (i.e., with `sequence number` equal to zero), the value of the `self.offset` variable is zero, and it is updated by the `_offset_next_encap_pck()` method every time a new SCION packet is created. `_offset_next_encap_pck()` iterates until `self.offset` is equal or grater than the payload length of the considered SCION packet (line 29), in that moment `self.offset` points to the end of the last IP packet that can be totally, or partially, encapsulated within the SCION packet under construction. At every iteration of the while loop (lines 29–47), the `self.offset` value is updated summing to it the total length of the currently considered IP packet. This length is extracted from the Total Length field of such IP packet's header (lines 31–45). It can happen that the currently analyzed IP packet's header is not fully comprised in the sequence of bytes extracted from `ip_buf`, in this case the missing part has to be extracted, analyzed to find the necessary Total Length field and, finally, re-appended to `ip_buf` (lines 33–41). Once the while loop ends, `self.offset` modulo SCION_PAYLOAD_LENGTH is calculated and will be used to set the `index` field of the next SCION packet. In case an encapsulated IP packet's length is grater than twice or more the SCION_PAYLOAD_LENGTH value, `self.no_encap_counter` keeps track of the number of future created SCION packets that will carry an `index` field equal to zero.

5.4.3 SCION Receiver

The `SCION_Receiver` thread's task is to receive and store the SCION packets directed to the SIG service itself through its SCION network interface. It keeps track of the expected SIG's header's `sequence number` through the `self.sn_expected` variable. Listing 5.7 shows `SCION_Receiver`'s `_packet_put()`

function. If a SCION packet received matches such expected `sequence number`, it is immediately stored within a dictionary of received SCION packets called `self.spcks_dict[]` (lines 25–26), otherwise, `SCION_Receiver` checks how late the package has arrived and decides whether discard or keep it, according the discussion done in Section 4.2.4.2 (lines 6–23).

```
1  def _packet_put(self, packet):
2      spck = packet.pack()
3
4      sn, index, unused, payload = self._unpack_scion_pck(spck)
5
6      # if packets lost
7      if sn != self.sn_expected:
8          current_time = datetime.datetime.now()
9
10         if (sn in self.delayed_spcks) and ((self.delayed_spcks[sn
   ][1]-current_time).total_seconds() <= HALF_RTT):
11             # delayed packet arrived in time. Store it
12             self.spcks_dict[sn] = SCION_Packet(sn, index, unused,
    payload)
13
14         elif not(sn in self.delayed_spcks):
15             # the packets lost/delayed can be more than 1
16             while(self.sn_expected <= sn):
17                 print('packet with sn= %s was lost' % self.
   sn_expected)
18                 self.lost_or_delayed.insert(0,(self.sn_expected,
   current_time))  # insert on the left ---> TO BE REMOVED USING
    pop()
19                 self.delayed_spcks[self.sn_expected] = (True,
   current_time)
20
21                 self.sn_expected = (self.sn_expected + 1)%
   4294967296
22
23             self.spcks_dict[sn] = SCION_Packet(sn, index, unused,
    payload)
24
25     else:
26         self.spcks_dict[sn] = SCION_Packet(sn, index, unused,
   payload)
```

Listing 5.7: `_packet_put()` method of the `SCION_Receiver` thread that decides whether store or drop the received SCION packet.

More specifically, if the SCION packet just received has already been declared as in late (i.e., inserted in the dictionary `self.delayed_spcks[]` together with its expected arrival time, using its `sequence number` as key), and its expected arrival time does not defer more that $\frac{1}{2}RTT$ from the current time, the packet is kept and stored in `self.spcks_dict[]` (lines 10–12). If such SCION packet is not in `self.delayed_spcks[]`, it means it is a new one[1] to be stored in `self.spcks_dict[]`, and the packets with `sequence numbers` between the expected one and the one of the packet just received are either in late or lost and must be inserted in `self.delayed_spcks[]` (line 14–23). `self.lost_or_delayed[]` is a list object used to store the expected arrival time of a packet that is in late (or may be lost), this list is fundamental for the `Send_Delayed_Packets` thread to quickly retrieve the oldest not-received packet without the need of scanning all the `self.delayed_spcks[]` dictionary. In all the other cases, the received SCION packet is dropped.

Using the dictionary `self.spcks_dict[]` to store the received SCION packets requires more memory and is less performing than a buffer approach, but greatly facilitates the reordering process of the packets – particularly in case of delayed or lost packets. To quickly identify the needed packets, the keys of this dictionary are the `sequence numbers` of the stored SCION packets.

5.4.4 IP Sender

The `IP_Sender` thread's task is to decapsulate the SCION packets received, stored in the `self.spcks_dict` dictionary, and send the original legacy IP packets to the correct legacy destination through the SIG's Internet interface. `IP_Sender` maintains a state information variable called `sn_to_send` that represents the `sequence number` of the current SCION packet to be decapsulated and sent. This one is increased by one, according the modulo 2^{32} operation, at every interaction in such a way the SIG can only send packets in the order they have been received (i.e., not introducing further entropy on their order for the end destination).

The sending procedure used by the `IP_Sender` thread is not trivial and requires some implementation choices to be introduced:

- `self.splitIP_tail[]` is a dictionary whose element `splitIP_tail[SN]` contains the last fragment of the last encapsulated IP packet in the SCION packet with `sequence number` equal to `SN-1`, or is `None` when `index` is equal to 0 or 1 for the SCION packet with `sequence number SN` (see Fig.5.2).

[1]Notice that, it could also be a really old one that is extremely in late or a replayed one injected by an attacker, as explained in Section 4.2.4.5. The implemented version of the SIG header has not a timestamp value so the SIG can not detect and drop this. We assume then to work in an environment where this scenario can not happen.

- `self.splitIP_head[]` is a dictionary whose element `splitIP_head[SN]` is a tupla and contains either a value `True` and the first fragment of the last IP packet encapsulated in the SCION packet with `sequence number` equal to `SN`, or a value `False` and an intermediary fragment when `index` is equal to 0 for the SCION packet with `sequence number SN`, or `None` when the last IP packet encapsulated in the SCION packet with `sequence number` equal to `SN` ends within the SCION packet itself (see Fig.5.2).

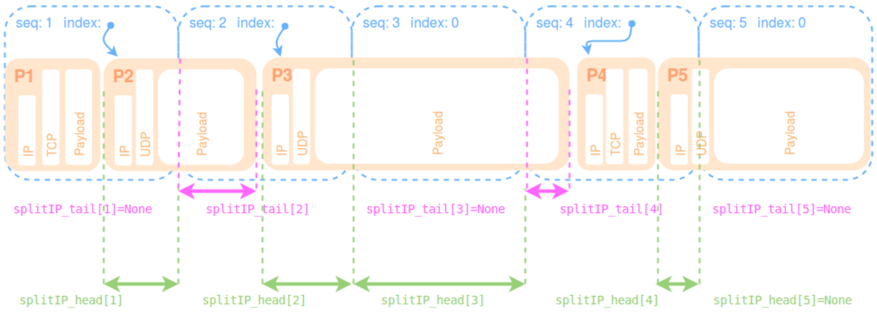


Figure 5.2: Possible values for the elements of the dictionaries `self.splitIP_tail[]` (in purple) and `self.splitIP_head[]` (in green). The arrows show the portion of bytes that composes a dictionary's element. Some elements may be `None`. Notice that an element of `self.splitIP_head[]` is a tuple, but their second element that is a Boolean is not represented for sake of brevity. `self.splitIP_head[3][1]= False`, while the Boolean is `True` for all the other elements of `self.splitIP_head[]` in this image, except for `self.splitIP_head[5]` that is `None`.

Listings 5.8 shows the sending procedure used by `IP_Sender`. At every interaction, it tries to decapsulate and send the IP packets in the SCION packet with `sequence number` equal to `sn_to_send` (that in the listing is indicated only with `sn`). If the `index` field of such SCION packet is equal to 0, its payload is inserted in `self.splitIP_head[sn]` together with a `False` value that indicates that no IP packets start in such payload (lines 6–12). In this case no IP packets can be decapsulated and sent since the SCION packet's payload embeds only one of the fragments that compose an original IP packet, and the other fragments are still missing.

```python
def _send_procedure(self, sn):
    spck = self.dict[sn]
    index = spck.index
    payload = spck.payload

    # no IP packets starts in this payload
    if index == 0:

        self.splitIP_head[sn] = (False, payload)  # tuple: (
IP_pck_starts_in_this_elem, part_of_payload_of_fragmented_pck
)

        # packet processed, time to remove it from dictionary
        del self.dict[sn]
    else:
        self._send_previous_fragmented_ip_pck(spck)
        self._send_ip_pcks_in_this_encap_pck(spck)


def _send_previous_fragmented_ip_pck(self, spck):
    sn = spck.sn
    index = spck.index

    if self.splitIP_head.get((sn-1)%4294967296) is not None:
        if index == 1:
            # if the previous IP pck had index=0 it has not been
sent yet and it has to be sent
            pld = b''
        else:
            # when index>1 part of the previous IP pck is in the
current encap pck
            pld = spck.payload[:index]

        iterator = (sn-1)%4294967296
        retrieved_fragments = []
        while(True):
            if self.splitIP_head.get(iterator) is None:
                # part of pck has been lost
                pld = None
                break
            elif self.splitIP_head[iterator][0] == True:
                # starting point of the IP pck found
                pld = self.splitIP_head[iterator][1] + pld
                retrieved_fragments.append(iterator)
                break
            elif self.splitIP_head[iterator][0] == False:
                # starting point of the IP pck not found
                pld = self.splitIP_head[iterator][1] + pld
                retrieved_fragments.append(iterator)

            iterator = (iterator - 1) % 4294967296

        if pld is not None:
            ethernet = scapy.Ether(src='08:00:27:e2:f7:59', dst='
08:00:27:0b:3c:de', type=0x0800)
            scapy.sendp(ethernet / scapy.Raw(pld), iface=ETH_LEGACY_IP)

            for i in retrieved_fragments:
                # remove from dictionary fragments sent correctly
                del self.splitIP_head[i]

    else:
        # it means that the packet with sequence number = SN-1 was already sent or lost/in late
        self.splitIP_tail[sn] = spck.payload[:index]


def _send_ip_pcks_in_this_encap_pck(self, spck):
    sn = spck.sn
    index = spck.index
    pld = spck.payload

    self.offset = index -1

    while self.offset <= SCION_PAYLOAD_LENGTH:
        ip_header = pld[self.offset:self.offset + IP_header_length]

        if len(ip_header) != IP_header_length:

            self.splitIP_head[sn] = (True, ip_header)
            # packet processed, time to remove it from dictionary
            del self.dict[sn]
            break

        else:
            iph = unpack('!BBHHHBBH4s4s', ip_header)
            ip_len = iph[2]

            ip_pck = pld[self.offset:self.offset + ip_len]
            if self.offset + ip_len <= SCION_PAYLOAD_LENGTH:
                ethernet = scapy.Ether(src='08:00:27:e2:f7:59', dst='08:00:27:0b:3c:de', type=0x0800)
                scapy.sendp(ethernet/scapy.Raw(ip_pck), iface=ETH_LEGACY_IP)

                self.offset = self.offset + ip_len

            else:
                self.splitIP_head[sn] = (True, ip_pck)
                # packet processed, time to remove it from dictionary
                del self.dict[sn]
                break
```

Listing 5.8: `_send_procedure()` method of the `SCION_Receiver` thread that decapsulates and send the legacy IP traffic.

If `index` is different from 0, procedures `_send_previous_fragmented_ip_pck()` and `_send_ip_pcks_in_this_encap_pck()` are called (lines 13–16).

`_send_previous_fragmented_ip_pck()` scans backward all the previous IP fragments stored in `self.splitIP_head[]` to retrieve all the ones, if any, associated with the first IP fragment encapsulated in the considered SCION packet (lines 32–47). If these fragment(s) are present, the full IP packet is rebuilt and sent over the Internet interface, while the retrieved fragment(s) are removed from the dictionary `self.splitIP_head[]` (lines 49–55). For sending the packets the *scapy* library is used to update their Ethernet header information. If `IP_Sender` realizes that one of the fragment(s) that should compose the original IP packet is not retrievable, the `_send_previous_fragmented_ip_pck()` method is exited. Such missing fragment could be lost or only delayed, so `IP_Sender` lets the decision to whether discard all the associated fragments or not to the thread `Send_Delayed_Packets`.
If the IP fragment immediately preceding the first one encapsulated in the considered SCION packet is not retrieved, it means that the SCION packet preceding the one under consideration could be delayed. The first IP fragment encapsulated in the SCION packet under consideration is then inserted in `self.splitIP_tail[]` for a future analysis by the `Send_Delayed_Packets` thread (lines 57–59).

`_send_ip_pcks_in_this_encap_pck()` scans instead all the legacy IP packets encapsulated within the considered SCION packet, decapsulates them and send them over the Internet interface. Starting from the value of the `index` field as offset, the header of the first encapsulated IP packet is parsed to retrieve its Total Length field that is used to determine the starting point of the next encapsulated IP packet. This mechanism is used to retrieve all the IP packets encapsulated within the SCION packet under consideration. Every one fully retrieved (i.e., not fragmented) is sent using the *scapy* library in the same way used by `_send_previous_fragmented_ip_pck()` (lines 84–88). In case the last parsed IP packet encapsulated is not fully retrieved, but only a fragment of it is present inside the considered SCION packet, this fragment is inserted in `self.splitIP_head[]` (lines 72–77 and 90–94). Further iterations of `IP_Sender` will try to merge it with the missing fragments possibly retrieved within the next received SCION packets.

5.4.5 Delayed Packets Sender

The `Send_Delayed_Packets` thread's task is to scan the dictionary of SCION packets arrived in late and, if they are received within $\frac{1}{2}RTT$ from their expected reception time, decapsulate them into the original IP packets and send these last ones to the correct legacy destination through the SIG's Internet interface.

Listings 5.9 shows the `_run()` method of `Send_Delayed_Packets` that is the core of all the thread's functioning.

```python
def _run(self):
    print('Send_Delayed_Packets Started')

    while (self.run_event.is_set()):
        current_time = datetime.datetime.now()

        if self.lost_or_delayed:
            sn, arrival_time = self.lost_or_delayed.pop() # return and remove delayed pck with shortest time to live
            if (current_time - arrival_time).total_seconds() > HALF_RTT:
                # packet is too much in late, discard preceding fragments and it (N.B. pop() already done)
                self._discard_preceding_fragments(sn)
                self._discard_following_fragments(sn)

                if sn in self.dict:
                    #notice that if the pck was lost, self.dict[sn] is empty
                    del self.dict[sn]
                if sn in self.delayed_spcks:
                    # notice that if the pck was lost, self.delayed_spcks[sn] is (False, None)
                    del self.delayed_spcks[sn]

            else:
                # try to send this pck and related fragments

                success_with_preceding = False
                success_with_following = False
                if sn in self.dict:
                    #notice that if the pck was lost, self.dict[sn] is empty
                    success_with_preceding = self._send_preceding_fragments(sn)
                    success_with_following = self._send_following_fragments(sn)

                if success_with_preceding and success_with_following == False:
                    # not all fragments sent, but still time to try so reinsert tuple in previous position
                    self.lost_or_delayed.append((sn, arrival_time))
                else:
                    # all fragments sent properly

                    if sn in self.dict:
                        # notice that if the pck was lost, self.
        dict[sn] is empty
39                      del self.dict[sn]
40                  if sn in self.delayed_spcks:
41                      # notice that if the pck was lost, self.
        delayed_spcks[sn] is (False, None)
42                      del self.delayed_spcks[sn]
43
44      print('***** Send_Delayed_Packets sender exited *****')
45      sys.exit(1)
```

Listing 5.9: `_run()` method of the `Send_Delayed_Packets` thread that decides when to discard or send the decapsulated packets.

`Send_Delayed_Packets` first checks if `self.lost_or_delayed[]` is not empty that means one or more SCION packets have been received in late and stored within `self.delayed_spcks[]`. If `self.lost_or_delayed[]` is not empty, its oldest element (i.e., the one with the shortest time to live) is retrieved with a `pop()` operation (lines 7–8). It can be noticed that the usage of the list `self.lost_or_delayed[]` allows to quickly retrieve such element, without the need to scan all the `self.delayed_spcks[]` dictionary. If the difference between the current time and the arrival time for the retrieved packet is greater than $\frac{1}{2}RTT$, the packet is too late and must be discarded together with all the IP fragments associated with it (lines 9–20). For this purpose the `_discard_preceding_fragments()` and `_discard_following_fragments()` methods are invoked. If the difference between the times is equal or lower, `Send_Delayed_Packets` tries instead to decapsulate the current SCION packet, retrieve its preceding and following IP fragments, if any, recompose possibly split IP packets, and send all the retrieved legacy traffic through the SIG's Internet interface (lines 21–42). The methods `_send_preceding_fragments()` and `_send_following_fragments()` are called for this purpose. If all the IP packets are sent correctly, the related entries are removed from `self.delayed_spcks[]`.

`_send_preceding_fragments()` is analogous to the method `_send_previous_fragmented_ip_pck()` used by the thread `IP_Sender`, while `_discard_preceding_fragments()` is its simplified version where the fragments are only retrieve (and not recomposed) and are not sent, but deleted.

`_send_following_fragments()` is analogous to the method `_send_ip_pcks_in_this_encap_pck()` used by the thread `IP_Sender`, while `_discard_following_fragments()` is its simplified version where the fragments are only retrieve (and not recomposed) and are not sent, but deleted. Differently from `_send_following_fragments()`, `_discard_following_fragments()` does not use the Total Length field of the IP headers to retrieve the next encapsulated IP packet, but interacts over the `self.splitIP_tail[]` and `self.splitIP_head[]` dictionaries as shown in Listing 5.10.

```python
def _discard_following_fragments(self, sn):

    if self.splitIP_tail.get((sn + 1) % 4294967296) is not None:
        # in this case the next fragment is the tail

        # there can be only one following IP fragment
        del self.splitIP_tail[(sn + 1) % 4294967296]

    elif self.splitIP_head.get((sn + 1) % 4294967296) is not None:
        # in this case the next fragment is a middle fragment of a pck that had index=0

        iterator = (sn + 1) % 4294967296
        retrieved_fragments = []
        while (True):
            if (self.splitIP_head.get(iterator) is None) or (self.splitIP_head[iterator][0] == True):
                # either the next pck is lost or it contains the tail of the fragmented IP

                if (self.splitIP_tail.get(iterator) is not None) and (self.splitIP_head[iterator][0] == True):
                    del self.splitIP_tail[iterator]
                break
            elif self.splitIP_head[iterator][0] == False:
                # starting point of the IP pck not found
                retrieved_fragments.append(iterator)

            iterator = (iterator + 1) % 4294967296

        if retrieved_fragments:
            for i in retrieved_fragments:
                del self.splitIP_head[i]
```

Listing 5.10: `_discard_following_fragments()` method of the `Send_Delayed_Packets` thread that discards the following IP fragments associated with the SCION packet under consideration. With "following" is ment the fragments belonging to SCION packets with `sequence number` greater than the one of the considered SCION packet.

Using `Send_Delayed_Packets`, together with the `self.lost_or_delayed[]` list and the `self.delayed_spcks[]` dictionary, is fundamental in terms of performances. As already explained, for each packet delayed or lost it has to be counted the time it has expired until it reaches $\frac{1}{2}RTT$ and the packets is declared lost or discarded. It is obviously not possible to keep a thread running per each packet delayed or lost since this could lead to a CPU depletion DoS attack [18]. The list `self.lost_or_delayed[]` is then used to store the times in which SCION packets have arrived to the SIG, while the thread

`Send_Delayed_Packets` keeps scanning such a list checking if the missing packets have arrived in time or not.

5.5 Limitations

The SCION architecture is a recent project and its current implementation is a prototype that can emulate a customized topology of the SCION architecture on local-host. Different blocks of the architecture are still under development, or not yet implemented, and this obviously has had some repercussions on the development of the SIG system. Also the high level of complexity of this thesis project and the overall SCION architecture, together with the time constraints imposed, impeded to implement all the features designed for the SIG system and explained in chapter 4. Furthermore, being still an academic proof-of-concept, there is no documentation yet for the more than 600 MB of *GitHub* SCION code. This also slowed down a bit the development of the SIG system. Finally, three major upgrades – among the frequent ones performed by the 27 contributing software developers –, of the *GitHub* SCION repository's software, imposed significant changes on the SIG implementation during the months, leading to a considerable delay on its final implementation.

SCION does not yet deploy the SIG SVC messages and any form of keep-alive messages mechanism needed by the SIG services to push updates of their local IAC table to the local CS and the Core ASes in the same ISD (see Section 4.2.2.1). This made impossible the implementation of the IAC upgrading system and the SIG negotiation mechanism (see Section 4.2.5).

Currently the RAINS system (Section 3.5.2) is under development, therefore the Client Protocol Negotiation of Section 4.2.6 could not be implemented either.

As already mentioned, the SCION prototype can emulate a customized topology of the SCION architecture on local-host. The basic one used and described in Section 5.2 already counts 15 different emulated hosts running on local-host. These, together with all the processes needed by *VirtualBox* to emulate them and run the private networks of Fig.5.1, resulted in a high CPU usage that did not allow to emulate further ASes, or hosts within them, on the machine used[2]. The available scenarios to perform our tests have then been limited (read Chapter 6), and the number of `ip_buf` buffers per each SIG service has been limited to one impeding to develop a priority-mechanism that chooses from

[2]The machine used for running the environment and performing our tests was a Lenovo MT 3493 Desktop PC mounting an Intel Core i3-3220 CPU @ 3.30GHz ×4 with 8 GB of RAM. The machine was running Ubuntu 16.04 LTS Operating System

which buffer to extract the legacy traffic to be encapsulated and sent over the only one available SCION network interface.

Finally, the MCU of the SCION link between a1 and b1 in Fig.5.1 comes from the static configuration file `topology.yml`, so testing any relation between latency aspects and a variable MTU could not be performed in Chapter 6.

6 Evaluation

In this chapter the SIG service designed in Chapter 4 and developed in Chapter 5 is evaluated according all the different scenarios of Section 4.3. The tests performed have a specific focus on the evaluation of the SIG service's interprocess communication performance (as the interprocess latency and goodput of the communication), the system's scalability issues (as the discovery time of new, or "old", SCION ASes that may join, or leave, the SCION network), and the performance of the introduced legacy DoS and DDoS mitigation mechanism (see Section 4.6).

AS already explained, all our tests were performed on local-host according to the topology of Fig.5.1. Running our tests on a single machine has lead to results that may be different from the ones that can be measured on a real network, where all the border routers, hosts and SIG services are running on different devices. In fact, latency aspects may defer between local-host and a real network since are topology- and distance-dependent. However, even if the performance obtained on local-host may differ from the ones of a real physical network, the main behaviours of the SIG service and the main performance differences between the different scenarios of Section 4.3 are still observable.

In the following, we first describe the environment setup procedure (Section 6.1), where we provide the details of how to install and run the tools needed for our test-bed and we explain how they work. We then describe the methodology followed to setup each one of our evaluation tests and their results. We start with the evaluation of the SIG's communication performance (Section 6.2), where we observe an interesting correlation between the interprocess latency and the sender's transmission rate, and we provide details on how to measure the communication goodput for all the different scenarios. Next, we evaluate the scalability issues, as the discovery time of a new SCION AS that joins the SCION network (and the time required to propagate and fetch its IAC table) and the discovery time of an AS which leaves the SCION network (Section 6.3). Finally, we quantify the benefits introduced by the SIG service for the encapsulated legacy traffic on the mitigation of DoS and DDoS attacks against legacy IP hosts (Section 6.4).

6.1 Environment Setup

The topology of the testing environment has been already shown in Fig.5.1. In this section we describe the procedure to set up such testing-bed on local-host.

The first step is to install the last release of *VirtualBox* on the machine used for the tests. If the machine makes use of the UEFI firmware instead of the traditional BIOS, the secure boot could impede the execution of *VirtualBox*. If so, there will be the need to generate a public-private key pair, which will be used to sign the kernel modules, and to execute the *Machine Owner Key (MOK)* utility to import and enroll such public key so that it can be trusted by the system. This procedure is extensively explained online on the *askubuntu* pages[1]. Once *VirtualBox* is correctly installed, the last releases of *Vagrant* and *Ansible* tools have to be installed.

The network configuration for each entity of Fig.5.1 (a1, a2, b1, b2, c2, c3) is stored in its own folder (see Section 5.3) in the `Vagrant` and `ansible.yml` files. To create and run a *VirtualBox* VM for one of the mentioned entities, say a1, the commands in Listing 6.1 have to be run within the directory `vagrant/sig/a1`.

```
vagrant up
vagrant ssh
```

Listing 6.1: Vagrant commands used to create and run a VM.

Since the VMs running a1 and b1 deploy also SCION connectivity, the SCION architecture must be imported and run on these two entities. This is done

[1]VirtualBox + Secure Boot + Ubuntu = fail: `https://askubuntu.com/questions/900118/vboxdrv-sh-failed-modprobe-vboxdrv-failed-please-use-dmesg-to-find-out-why`

following the procedure described in the `README.md` file of the SCION *GitHub* reository[2]. Briefly, the SCION repository must be cloned within the two VMs, the required packages with their dependencies must also be installed and the SCION topology must be generated and imported within each VM. The topology file used for our environment is shown in Listing 6.2. After the generation of such topology, the resulting `gen` folder must be copied in both a1 and b1 within `/go/src/github.com/netsec-ethz/scion/`. Finally, the directories `/go/src/github.com/netsec-ethz/scion/gen/ISD1/AS12` and `/go/src/github.com/netsec-ethz/scion/gen/ISD1/AS11` must be deleted from a1 and b1, respectively.

```
--- # TwoASes Topology
defaults:
  zookeepers:
    1:
      manage: false
      addr: 127.0.0.1
ASes:

  1-11:
    core: true
    mtu: 1400
  1-12:
    cert_issuer: 1-11
CAs:
  CA1-1:
    ISD: 1
    countryName: US
    stateOrProvinceName: Minnesota
    localityName: Minnetonka
    organizationName: my company
    organizationalUnitName: my organization
  CA1-2:
    ISD: 1
    countryName: US
    stateOrProvinceName: Minnesota
    localityName: Minnetonka
    organizationName: my company
    organizationalUnitName: my organization
links:
  - {a: 1-11, b: 1-12, ltype: PARENT, mtu: 1280}
```

Listing 6.2: Topology file used to generate the testing environment.

To run SCION within a1 and b1, the command of Listing 6.3 should be run within the `/go/src/github.com/netsec-ethz/scion/` directory on both the VMs.

```
./scion.sh run
```

[2]SCION: `https://github.com/netsec-ethz/scion`

Listing 6.3: Command to run the SCION infrastructure with the created topology.

At this point all our hosts are configured and running. The last step to complete our setup is to enable the network forwarding on c3 that is our only legacy router. This is done with the bash commands in Listing 6.4, which sets to 1 its `ip_forward` variable and enables the IP packets forwarding between its two interfaces (`enp0s8` and `enp0s9`) through its `iptables`.

```
echo "1" | sudo tee /proc/sys/net/ipv4/ip_forward
sudo iptables -A FORWARD -i enp0s8 -o enp0s9 -j ACCEPT
sudo iptables -A FORWARD -i enp0s9 -o enp0s8 -j ACCEPT
```

Listing 6.4: Commands to enable network forwarding on c3.

We can now start the `gw.py` scripts within a1 and b1 with the commands of listing 6.5 and start performing our evaluation tests.

```
sudo python3 /vagrant/gw.py -a 169.254.0.1 -p 30100 -d 1 -s 11

sudo python3 /vagrant/gw.py -a 169.254.0.2 -p 40500 -d 1 -s 12
```

Listing 6.5: Commands used to start the SIG services in a1 and b1 respectively.

6.2 Communication Performance

In this section, the outcome of the tests performed to measure the interprocess communication performance of the different scenarios of Section 4.3 are reported and further analyzed.

Despite the fact we described five different scenarios in Section 4.3, when it comes to test the communication performance, Scenarios 2) and 3) are equivalent to each other and Scenarios 4) and 5) are equivalent to each other too. Thus, from now onward we refer to Case A) for the performance of Scenario 1), to Case B) for the performance of Scenarios 2) and 3), and to Case C) for the performance of Scenarios 4) and 5).

More precisely, the three Cases refer to the connectivity between two hosts in the following contexts:

- **Case A)** *IP host in SCION AS* $\longleftrightarrow$ *IP host in SCION AS*
- **Case B)** *IP host in SCION AS w/ IP connectivity* $\longleftrightarrow$ *IP host in IP AS*
- **Case C)** *IP host in IP AS* $\longleftrightarrow$ *IP host in SCION AS w/o IP connectivity*

6.2.1 Simple TCP Communication

In this section, a trivial test to check the functioning of a TCP connection between two legacy IP hosts, which connect to each other through SCION using the SIG's encapsulation service, was performed for Case A), Case B), and Case C).

The tests were conducted simply running the `netcat` utility on the client and server hosts. The connections tested have been between hosts a2↔b2 (Case A)), a2↔c2 (Case B)) and b2↔c2 (Case C)). The command ran on the server side is shown in listing 6.6 for the host a2, while the command ran on the client side is shown in listing 6.7 for the host b2. In the listing 6.7 is also shown how we generated the 1 KB size file `example.txt` with random content.

```
sudo netcat -l 80 > log.txt
```

Listing 6.6: `netcat` server listening on port 80 to save the received information on file `log.txt`.

```
base64 /dev/urandom | head -c 1024 > example.txt
sudo netcat 169.254.2.2 80 < example.txt
```

Listing 6.7: Generate a random file of 1 KB size and run `netcat` client which connects to host `169.254.2.2` on port 80 to transfer the generated file `example.txt`.

Every time the file `example.txt` was correctly copied into the `log.txt` file on the server side.

6.2.2 Latency Aspects

For this section, the interprocess latency between different pairs of hosts of the topology in Fig.5.1 was measured according to Cases A), B) and C). These tests were performed using the *ping utility* [79]. Their output was then parsed and the results plotted into the graphs of the following subsections.

For every pair of hosts three runs of the ping utility were performed for an amount of time of 10 minutes each. For every run, a different sender's transmitting frequency was used (the "ping" requests were sent with intervals of 200, 500 and 800 ms, respectively). All the experiments were conducted in absence of packet loss.

6.2.2.1 Case A)

During these tests the interprocess latency between the hosts a2 and b2 was tested. The host a2 was used to send the "ping" requests, while the host b2 to answer with the "pong" responses.
Listings 6.8 shows the three commands used to run the three tests and their output is shown graphically in Fig.6.1, Fig.6.2 and Fig.6.3.

```
ping -c 3000 -i 0.2 169.254.2.2 > ping_a2_to_b2_interval_02.txt
ping -c 1200 -i 0.5 169.254.2.2 > ping_a2_to_b2_interval_05.txt
ping -c 750 -i 0.8 169.254.2.2 > ping_a2_to_b2_interval_08.txt
```

Listing 6.8: ping commands used to run the three tests of Case A).

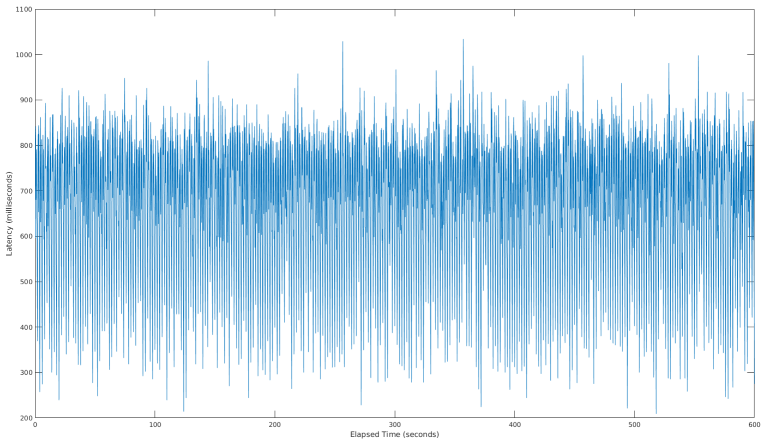


Figure 6.1: Interprocess latency between hosts a2 and b2 using a time interval between each request of 0.2s.

For the test shown in Fig.6.1, performed with a time interval of 0.2s, the minimum, average and maximum latency values are respectively 209.639, 671.952 and 1034.003 ms, while the mean deviation is 153.724 ms.

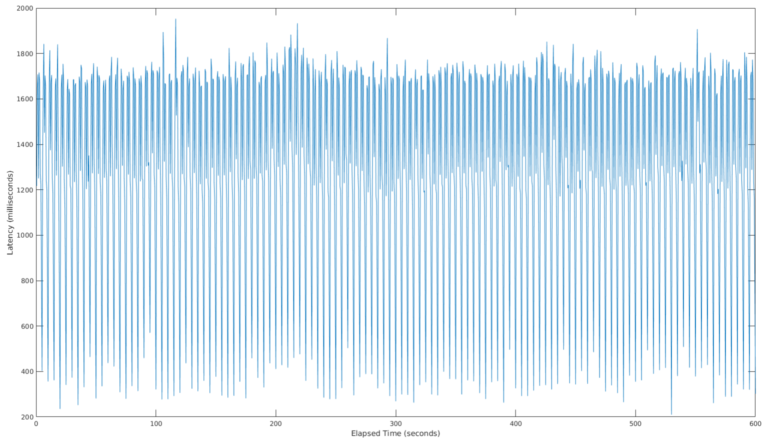


Figure 6.2: Interprocess latency between hosts a2 and b2 using a time interval between each request of 0.5s.

For the test shown in Fig.6.2, performed with a time interval of 0.5s, the minimum, average and maximum latency values are respectively 210.050, 1301.160 and 1953.530 ms, while the mean deviation is 433.151 ms.

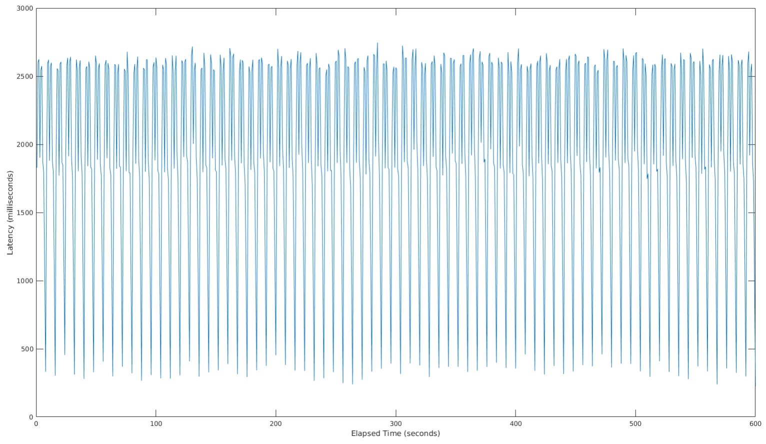


Figure 6.3: Interprocess latency between hosts a2 and b2 using a time interval between each request of 0.8s.

For the test shown in Fig.6.3, performed with a time interval of 0.8s, the minimum, average and maximum latency values are respectively 223.045, 1921.754 and 2746.816 ms, while the mean deviation is 716.697 ms.

6.2.2.2 Case B)

During these tests the interprocess latency between the hosts a2 and c2 was tested. The host a2 was used to send the "ping" requests, while the host c2 to answer with the "pong" responses. Notice that, the legacy border router c3 that is present in AS13 is set as default-gateway for the host c2.
Listings 6.9 shows the three commands used to run the three tests and their output is shown graphically in Fig.6.4, Fig.6.5 and Fig.6.6.

```
ping -c 3000 -i 0.2 169.254.4.2 > ping_a2_to_c2_interval_02.txt
ping -c 1200 -i 0.5 169.254.4.2 > ping_a2_to_c2_interval_05.txt
ping -c 750 -i 0.8 169.254.4.2 > ping_a2_to_c2_interval_08.txt
```

Listing 6.9: ping commands used to run the three tests of Case B).

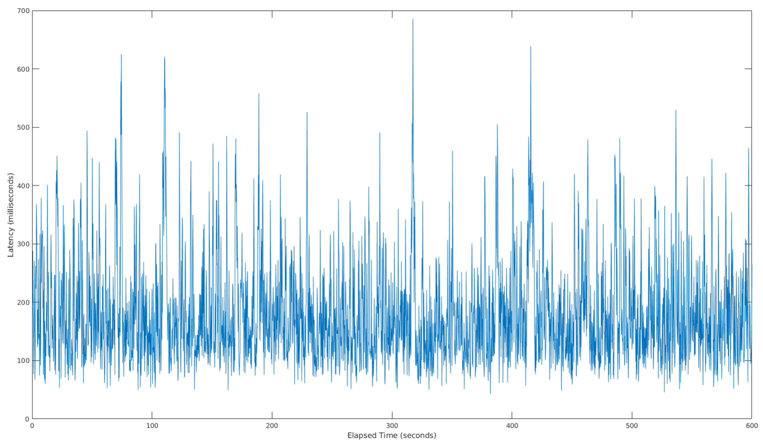


Figure 6.4: Interprocess latency between hosts a2 and c2 using a time interval between each request of 0.2s.

For the test shown in Fig.6.4, performed with a time interval of 0.2s, the minimum, average and maximum latency values are respectively 43.411, 181.861 and 686.728 ms, while the mean deviation is 89.126 ms.

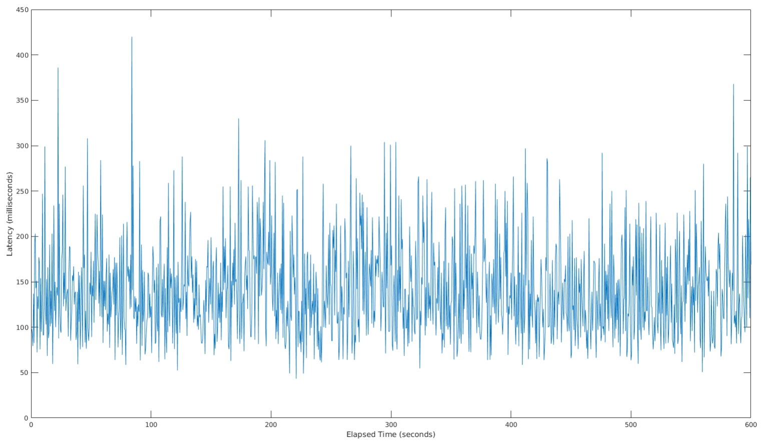


Figure 6.5: Interprocess latency between hosts a2 and c2 using a time interval between each request of 0.5s.

For the test shown in Fig.6.5, performed with a time interval of 0.5s, the minimum, average and maximum latency values are respectively 43.287, 141.696 and 420.978 ms, while the mean deviation is 53.099 ms.

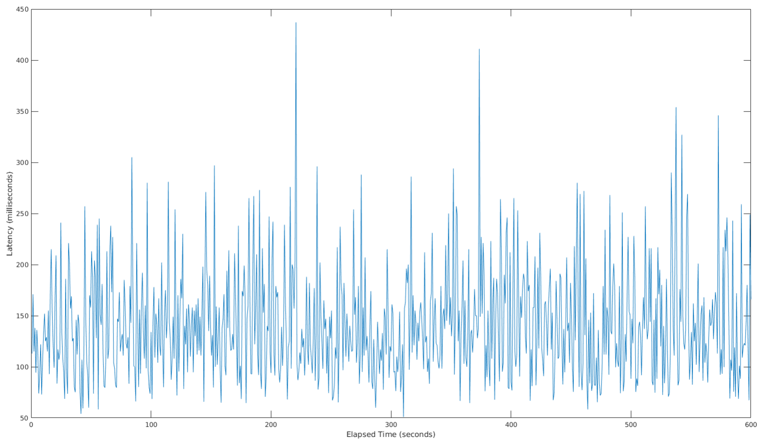


Figure 6.6: Interprocess latency between hosts a2 and c2 using a time interval between each request of 0.8s.

For the test shown in Fig.6.6, performed with a time interval of 0.8s, the minimum, average and maximum latency values are respectively 50.984, 142.115 and 437.495 ms, while the mean deviation is 54.284 ms.

6.2.2.3 Case C)

During these tests the interprocess latency between the hosts b2 and c2 was tested. The host c2 was used to send the “ping” requests, while the host b2 to answer with the “pong” responses. Notice that, the legacy border router c3 that is present in AS13 is set as default-gateway for the host c2. The presence of c3 between the communication is what differentiates the tests performed for Case A) and Case C).
Listings 6.10 shows the three commands used to run the three tests and their output is shown graphically in Fig.6.7, Fig.6.8 and Fig.6.9.

```
ping -c 3000 -i 0.2 169.254.2.2 > ping_c2_to_b2_interval_02.txt
ping -c 1200 -i 0.5 169.254.2.2 > ping_c2_to_b2_interval_05.txt
ping -c 750 -i 0.8 169.254.2.2 > ping_c2_to_b2_interval_08.txt
```

Listing 6.10: ping commands used to run the three tests of Case C).

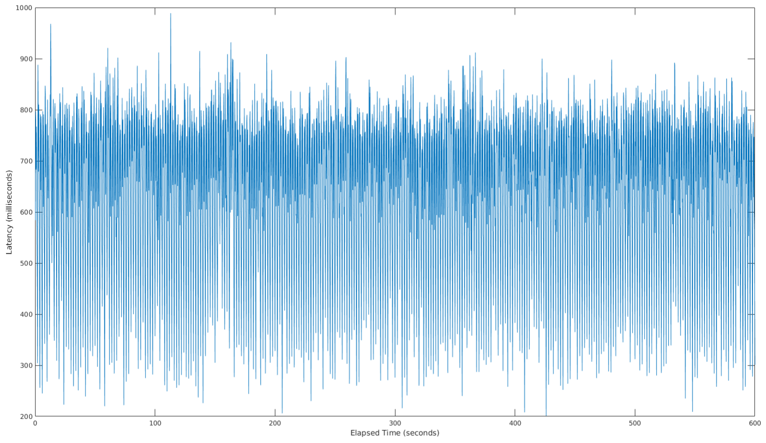


Figure 6.7: Interprocess latency between hosts c2 and b2 using a time interval between each request of 0.2s.

For the test shown in Fig.6.7, performed with a time interval of 0.2s, the minimum, average and maximum latency values are respectively 201.864, 639.901 and 989.419 ms, while the mean deviation is 155.357 ms.

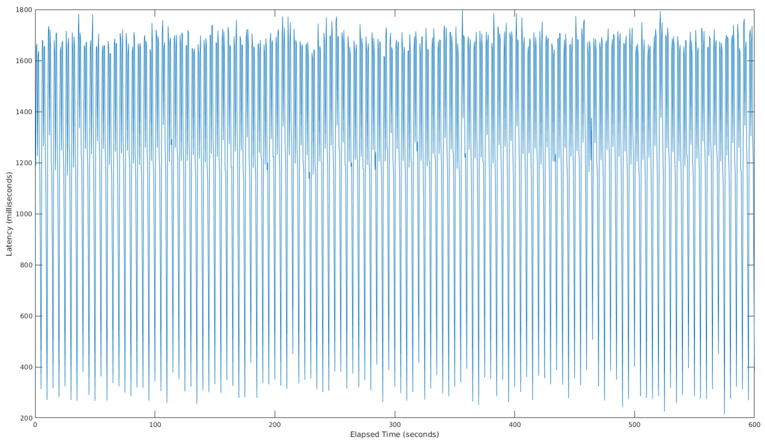


Figure 6.8: Interprocess latency between hosts c2 and b2 using a time interval between each request of 0.5s.

For the test shown in Fig.6.8, performed with a time interval of 0.5s, the minimum, average and maximum latency values are respectively 215.229, 1269.475 and 1800.543 ms, while the mean deviation is 431.746 ms.

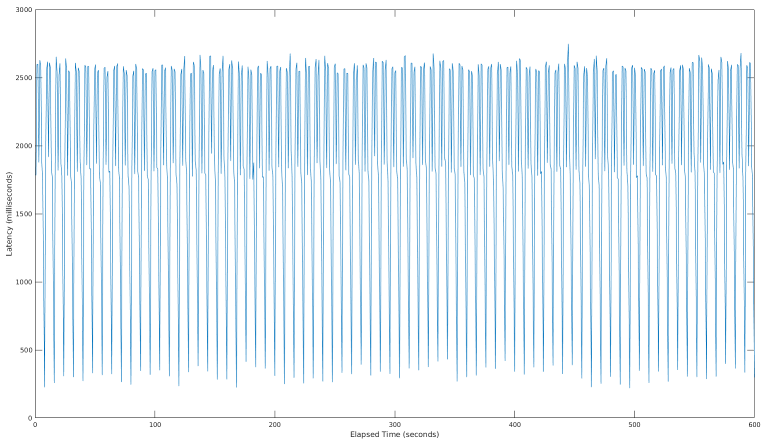


Figure 6.9: Interprocess latency between hosts c2 and b2 using a time interval between each request of 0.8s.

For the test shown in Fig.6.9, performed with a time interval of 0.8s, the minimum, average and maximum latency values are respectively 219.867, 1897.027 and 2748.755 ms, while the mean deviation is 713.837 ms.

6.2.2.4 Overall Considerations

The interesting result of the latency tests is not the latency values measured, which usually depend a lot on the network topology and the performance of the links and devices used. Having used the not-performing `pcapy` *Python* library to sniff the SIG's incoming legacy IP traffic, we actually measured quite long latency values – considering we are working on localhost.

The interesting part of the latency tests is that, differently from any other communication, if the encapsulation process is involved, the sender's transmission rate influences a lot the latency of the communication. Comparing the average latency of Case A) and Case C) with the one of Case B), it can be seen that the average latency decreases a lot, increasing the sender's transmission rate used by the sender, for Cases A) and C), while it does not for Case B) (Fig.6.10). The same trend is followed also by the mean deviation among the

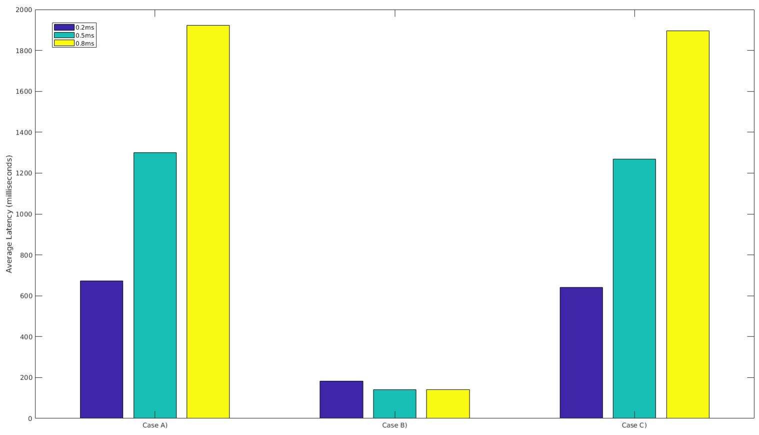


Figure 6.10: Average latency for Case A), Case B) and Case C). The different colors distinguish the sender's transmission time interval.

different measurements of latency (Fig.6.11).

The reason behind this behaviour is that, in our implementation of the SIG ser-

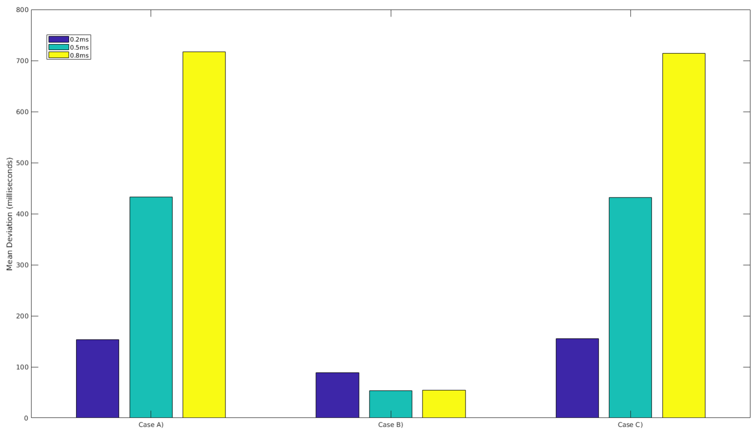


Figure 6.11: Mean deviation of latency measurements for Case A), Case B) and Case C). The different colors distinguish the sender's transmission time interval.

vice every SCION packet has a fixed size (i.e., 128 bytes), equal to the SCION MTU of the path between a1 and b1 (in Listing 5.1 is shown the MTU of the link between the BR in AS11 and the BR in AS12), and the sending SIG needs to receive at least 128 bytes of information before building and forwarding a SCION packet. Naturally, if the same information arrives with a slower frequency (e.g., 0.8s instead of 0.2s) the sending SIG will take more time to receive slots of 128 bytes of information to be forwarded in the form of SCION packets. Thus, the interprocess-communication average latency is shorter with a higher sender's transmission frequency.
Furthermore, also the mean deviation of the latency measurements is smaller with a higher sender's transmission frequency. It has to be considered that not always the same amount of legacy IP packets received by the SIG provides 128 bytes of information and sometimes more packets are required (e.g., 4 packets instead of 2). The result is that, using a low sender's transmission frequency, when the SIG has to wait only 2 packets instead of 4, the overall interprocess-latency will be considerably shorter. In the case the sender's transmission rate is high instead, waiting only 2 packets instead of 4 is not a big time difference, so the final latency will not be affected as much as in the case of a lower transmission rate, resulting in a lower mean deviation.

Analyzing the graphs of Fig.6.10 and Fig.6.11 it can be understood that the latency lowers exponentially lowering the sender's transmission interval, until it reaches its minimum value when the transmission interval is enough short to cause the the SIG to have always at least X bytes (in our case $X = 128$ bytes) of information in its `ip_buf` (i.e., to be always capable to build and send a SCION packet). The guess then is that, since the size of the SCION packets is correlated to the SCION MTU of the path used, the smaller the SCION MTU is, the faster the latency will reach its minimum value. This is because it will take less time for the SIG to have always X bytes of information in its buffer. Measuring the coefficient of this proportion would have required to perform more latency tests, but we did not do it because lack of time.

6.2.3 Goodput Aspects

One of the most important tests that can be performed to evaluate the communication performance is measuring the goodput (application level throughput) of a TCP session between two hosts. Our goal was to evaluate the goodput for Case A), Case B), and Case C).

Unfortunately, due to some issues with the external libraries used and lack of time we could not perform such tests, but in the following we describe anyway its setup. The technical reasons that impeded to run the tests are related to the external *Python* library used by our `gw.py` script to alter and send IP packets (`scapy`). The method `sendp()` provided by this library to send layer-2 packets through an interface can not handle packets of large size and throws the exception `OSError: [Errno 90] Message too long`. We realized this weakness of the library very late and we did not implement a proper IP packets fragmentation mechanism due to lack of time. The workaround we tried to use in numerous attempts to test the goodput has been: setting the system's MTU of the involved machines to a small value (100, 400, 512 and 1024 bytes respectively), setting the **Maximum Segment Size (MSS)** of the TCP testing application in use to the same small value and disabling the **TCP Segmentation Offload (TSO)**. After these arrangements, though, the size of the transmitted packets (sniffed using the `tcpdump` tool) was not respecting the imposed maximum thresholds on MTU and MSS and it was always bigger, causing the `scapy` library to throw the `OSError: [Errno 90] Message too long` exception. The commands used to change the system's MTU and to disble the TSO are shown in listing 6.11, while the traffic analyzing tools used, for which we tried to set the MSS to the values specified above, are `iperf`, `iperf3`, `netperf` and `flowgrind`.

```
sudo ifconfig enp0s8 mtu 1500 up
sudo ethtool -K enp0s8 tso off
```

Listing 6.11: Command to set a new system's MTU on the interface `enp0s8`.

The goodput test should have been performed for Case A), Case B) and Case C) simply running one of the automated tests provided by one of the aforementioned tools. In the following we describe only how we tried to perform the tests using `iperf3`, the other tools work similarly. `iperf3` is a client-server application for traffic analyzing which can perform some tests between two machines. To run a test a server instance must be running on one host (listings 6.12 shows the command used to run it), and a client one must be running on the second host.

```
iperf3 -s
```

Listing 6.12: Command used to run the `iperf3` server instance.

In listings 6.13 there are some of the tests we tried to perform from the client instance. The option `-c` is used to set the server's address, `-M` is used to set up the TCP MSS, `-u` is used to perform a UDP analysis instead of the default TCP one and `-l` is used to set up the size of UDP packets.

```
iperf3 -c 169.254.4.1
iperf3 -c 169.254.4.1 -M 512
iperf3 -c 169.254.4.1 -u
iperf3 -c 169.254.4.1 -u -l 400
```

Listing 6.13: Command used to run some `iperf3` client instances with different options.

Simply running the `iperf3` client and server instances with the commands described above between the hosts a2↔b2 (Case A)), a2↔c2 (Case B)) and b2↔c2 (Case C)) should return the expected goodput value in Mbits/sec. We would have then compared and further analyzed the results of the different case scenarios.

6.2.4 Response to SCION Network Losses

With this test we want to evaluate how a reliable layer-4 protocol as TCP, that is encapsulated by the SIG service and transported over SCION, reacts in case of SCION packet loss. The metric we used to measure this behaviour is again the goodput of the interprocess communication, as in Section 6.2.3, hence the test methodology – and the problems that arose – does not change from the previous section.

The tool `iperf3` should be used in this test following the same methodology of Section 6.2.3, the only difference is that, this time, the sending SIG service emulates a network packet loss by dropping some random outgoing SCION packets that should be forwarded to the receiving SIG service. The dropping process on the sending SIG is based on a simple function, called for each outgoing SCION packet, which returns 0 or 1 with a different probability. If it returns 0 the packet is dropped, otherwise forwarded. If, for example, a packet loss of 1% wants to be introduced on the SCION path between a1 and b1, then the above function has to return 0 with a probability of 1%.

The only meaningful communication to be tested this time would be between the hosts a2↔b2 (Case A)) and between the hosts b2↔c2 (Case C)), since the connection a2↔c2 (Case B)) does not involve the SCION network. To properly analyze the behaviour of the communication, different tests should be performed for each scenario using different packet loss probabilities, which should go from 1% to 5%.

6.3 Scalability Issues

Due to the limitation described in Section 5.5 (specifically the lack of SIG SVC and keep-alive messages implementations, among the SCION tools available), the IAC upgrade mechanism was not implemented in this thesis work, therefore measuring the scalability issues for the SIG service infrastructure was not possible. However, we describe in the following how the discovery time of a new SCION AS that joins the SCION network, or of an AS which leaves it, should be evaluated.

In the following we assume that the IAC polling mechanism discussed in Section 4.2.2.1 is implemented and working.

6.3.1 Discovery Time for New ASes

In this test we extends the topology of listing 6.2 and Fig.5.1 adding a new ISD, namely ISD2, which contains two ASes, namely AS21 and AS22, where AS21 is a Core AS and AS22 is a non-Core AS that deploys the SIG service named d1. AS11 and AS12 already know AS21, but AS22 is not yet enabled.

Once AS22 is activated, d1 pushes its IAC table version to the local CS that, in turn, pushes it to the AS21's CS. The first push is done within 500ms from AS22's activation. At the time of the AS22's activation we also start a timer on our local-host. The SIG service a1 polls one Core AS for each ISD available (in our case only AS21 in ISD2) every 5 hours, so sooner or later it will realize the presence of the new IAC table version in AS21's CS. In that precise moment, a1 queries AS22 to fetch its IAC. Once the IAC is fully received by a1, we stop the timer on our localhost and we check the time expired. Such test should be computed more times and with more ISDs and ASes that are enabled at random times, in order to have a good estimation of the average discovery time.

6.3.2 Discovery Time for ASes that leave the SCION network

The setup for this tests is the same as to the one described in Section 6.3.1 with ISD2, AS21, AS22 and the SIG service d1 within AS22.

This time we start from the state where AS22 is already activated and a1 and b1 already fetched AS22's IAC table. At a random time we deactivate AS22 and we start a timer on local-host. The local CS and AS21's CS will discard AS22's (ISD-AS, IAC ver) entry after 2000ms. The SIG services a1 and b1 poll one Core AS for each ISD available (in our case only AS21 in ISD2) every 5 hours, so sooner or later they will realize the absence of the IAC table version for AS21 and they will decide to update their local mappings removing the entries associated with AS22. In that precise moment we stop the timer on our local-host and we check the time expired. Such test should be computed more times and with more ISDs and ASes that are disabled at random times, in order to have a good estimation of the average discovery time.

6.4 DoS and DDoS Mitigation

As discussed in Section 4.6, the SIG service provides to the legacy encapsulated traffic DoS and DDoS mitigation thanks to the authenticated traffic between SIG services and the traffic filtering performed by the sending SIG. Due to the limitation described in Section 5.5 (specifically the absence of the implementation of an authentication mechanism between two SIG services) we had to run our tests under the big assumption that all the SCION packets sent from the sending SIG could be authenticated by the receiving SIG service.

6.4.1 Traffic Filtering on Sending SIG

During our first test, we checked the traffic filtering mechanism of the sending SIG service trying to perform a **Smurf attack** [17] against the host b2. During a Smurf attack, a large number of ICMP packets with the intended victim's spoofed source IP are broadcast to a network using an IP broadcast address. Most devices in that network will respond to this by sending a reply to the source IP address. If the number of machines that respond is very large, the victim's computer will be flooded with traffic. We set up and ran different new hosts within AS12, namely b3, b4, b5 and b6, all with an IP address belonging to `169.254.2.0/24` and we ran an infinite *ping test* using a spoofed source address from the host a2. The spoofed address used was the one of b2 (`169.254.2.2`) and the destination address used was the broadcast IP address for the network `169.254.2.0/24`. Our goal was to set up an amplification DDoS attack against b2 [18]. We first performed such attack disabling any defence mechanism on the sending SIG service a1 and the attack took only 80 seconds to completely make b2 machine unresponsive. Then we activated the traffic filtering mechanism on the sending SIG and performed again the same attack. The SIG service a1 immediately realized that the address `169.254.2.2` was not belonging to the local network `169.254.1.0/24` and started to drop all the packets coming from this host then preventing the DDoS attack against b2.

Notice that, our traffic filtering protection can defend the system also from different attacks other that the Smurf one.

6.4.2 Origin Tracking

Our second test was executed under the big assumption above described that each SCION packet coming from the sending SIG could be authenticated by the receiving SIG. Whit this test we showed how we could stop a DDoS attack, which generated legacy traffic was encapsulated and sent through SCION, right on the sending SIG service (i.e., on the originating AS).

Under this condition, we set up and ran different new hosts within AS11, namely a3, a4, a5 and a6, all with an IP address belonging to `169.254.1.0/24`. From each one of them we performed a simple **SYN flood** attack [80] targeting host b2. The SYN flood attack is a DoS method that expoits the TCP three-way-handshake for establishing connections by exhausting the server's allocated states for a specific listening application, preventing legitimate connections from being established. Since none of the attackers' addresses were spoofed, but belonging to the local network, all their traffic was encapsulated by the SIG a1, decapsulated by b1 and, finally, forwarded to b2. The attack was performed in parallel by a3, a4, a5 and a6, using the packet generator and analyzer `hping3` through the command shown in listing 6.14.

```
sudo hping3 -c 10000 -d 120 -S -w 64 -p 21 --flood 169.254.2.2
```

Listing 6.14: Command ran on attackers' hosts to set up the SYN flood DDoS attack of our test.

The command's option `-c` indicates the number of packets to send, the option `-d` their size, `-S` indicates that only TCP SYN packets are sent, `-w` indicates the TCP window size, `-p` is the targeted port and finally, `-flood` indicates to send the packets as fast as possible.
During this test, we continuously check b2's connectivity using a new legacy host, namely b3, within AS12. It took only 104 seconds to completely make b2 machine unresponsive.

The previous DDoS attack has been successfully performed due to the lack protection mechanisms on b2. Therefore, we added the `iptables` rule of listing 6.15 to host b2. The rule allows all the incoming connection till the limit is reached. `-limit-burst` specifies the number of matches that are allowed before the `-limit` of 1 per second is applied for the incoming TCP SYN requests.

```
sudo iptables -A INPUT -p tcp --syn -m limit --limit 1/s --limit-
    burst 3 -j RETURN
```

Listing 6.15: Command ran on the victim's host to limit the incoming TCP SYN requests.

With such rule the attack explained above was prevented on the host b2, but we could still realize from b2's log files that a SYN flood attack was targeting the host. The consequence was a considerably high bandwidth wastage on the Internet and SCION networks between the SIG a1 and the legacy IP host b2.

Our goal was to stop the attack right on the origin AS (AS11) to prevent the bandwidth wastage described. We pretended that b2 (or b2's system admin) reported to b1 (or b1's system admin) the attackers' addresses through an automatic or non-automatic tool and that b1, in turn, reported these ones to a1. What we did in practice has been simply to add some `iptables` rules on a1 which started to drop the packets coming from the malicious addresses. We could then check, using the `tcpdump` tool, that no traffic was reaching b1's SCION interface anymore, but due to lack of time we could not estimate properly the savings in term of bandwidth on Internet and SCION links between a1 and b2.

The simple DDoS attack we decided to perform was the SYN flood attack, but many different others (as UDP flood, TCP HTTP ACK flood, HTTP slow POST, etc.) may be very effective when their packets are encapsulated and transported through two communicating SIG services. These attacks may even require more complex protection mechanisms on the destination side, as some sort of IPS system. However, independently from the kind of attack, every time the IPS detects it, the SIG architecture described can always stop it on its source AS with great savings in term of bandwidth.

7 Conclusions

This thesis project extensively highlights several of today's Internet architecture problems, like the ones associated with the IP and BGP protocols, the lack of authentication and trust, some attacks against which Internet has little or no protection and the difficulties encountered in the transition to new architectures. This thesis then introduces the main goals that a secure new Internet architecture, as SCION, should achieve, why they are important and how they can be obtained. Such goals are: availability in presence of adversaries, transparency and control, efficiency, scalability, extensibility, support for global but heterogeneous trust and deployability.

The problems faced during such a big transition towards the desired properties of a new architecture, as SCION, are then illustrated. Such transition can not happen overnight, but, according to the ETH Zürich Network Security Group, it is imperative in the long run. Consequently, SCION needs to provide a set of models and tools that can favor it. As a response to this, this thesis presents the SCION-IP Gateway (SIG), which is the fundamental building block for a full interconnection between SCION and Internet networks.

An exhaustive description of the challenges introduced by our proposal is illustrated (ensuring maximum interoperability with the current Internet through minimal infrastructure changes, enabling transparent operations for legacy end-hosts in the IP domain, preventing downgrade attacks to the legacy Internet), followed by an extensive design description of the SIG service, its iterations with the SCION and Internet elements and the explanation of how it addresses the presented challenges. The security benefits introduced by our solution, even for legacy-to-legacy traffic, are then described, together with all the possible real use case scenarios for achieving maximum interoperability. Furthermore, the DoS and DDoS mitigation mechanism introduced by the SIG for legacy traffic is discussed.

In accordance with the design specifications, a prototype implementation was developed, and it is working to the point where it can be called a minimal viable product. The prototype fulfills many of the requirements specified in Section 4.2, apart from the ones that were constrained by the technological limitations described in Section 5.5.

The developed prototype was tested under many aspects, like interprocess-communication performance, scalability issues and benefits introduced by DoS and DDoS mitigation for legacy traffic. An interesting proportion between sender's transmission rate and interprocess latency of a communication which involves encapsulated traffic, was found. A proportion between the mean deviation of such latency measurements and the sender's transmission rate was also found. The DoS and DDoS protection mechanism showed great savings in terms of bandwidth for the SCION and Internet networks.

7.1 Future Work

Some limitation expressed in Section 5.5 impeded the development of some design blocks described in Section 4.2, in fact, some external SCION architecture modules necessary for the SIG prototype are still under development by the ETH NETSEC team. The very first improvement for this dissertation work would then be the implementation of the missing features of Section 4.2, in particular the IAC upgrading system, the SIG negotiation mechanisms and the protocol negotiation technique.

We want to point out that developing a performant software was out of the scope of this thesis, so many parts of the developed one may be subject to optimization. A first example would be the addition of an IP fragmentation mechanism to the `IP_Sender` thread, in such a way to allow a better path MTU discovery between the legacy hosts within a SCION AS and the local SIG service. This would also allow to overcame the issues encountered during our goodput tests, as explained in Section 6.2.3. The SIG prototype is currently written using the *Python* programming language – and makes use of non-performant libraries as `scapy` and `pcapy` –, while an implemented `go` version (as different modules in the SCION *GitHub* repository) would be more performant.

Our tests were conducted on local-host, while the best way to evaluate the interprocess-communication performance would be on a real physical network comprising different hosts.

Another interesting aspect to further analyze would be related to the policies that system administrators should respect on creating and updating the Incoming and Outgoing Associations Config tables (Section 4.2.1) and the local IAC tables (Section 4.2.2.2).

Bibliography

[1] T. Anderson, K. Birman, R. Broberg, M. Caesar, D. Comer, C. Cotton, M. J. Freedman, A. Haeberlen, Z. G. Ives, A. Krishnamurthy, W. Lehr, B. T. Loo, D. Mazières, A. Nicolosi, J. M. Smith, I. Stoica, R. V. Renesse, M. Walfish, H. Weatherspoon and C. S. Yoo.

[2] X. Yang, D. Clark and A. W. Berger. Nira: A new inter-domain routing architecture. *IEEE/ACM Transactions on Networking*, 15(4):755–788, 2007.

[3] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan and P. Steenkiste. Xia: Efficient support for evolvable internetworking. *Proceedings of the 9th USENIX Conference on Networked Systems Designand Implementation*, pages 23–23, 2012.

[4] X. Zhang, H. Hsiao, G. Hasker, H. Chan, A. Perrig, D. G. Andersen. Scion: Scalability, control, and isolation on next-generation networks. *Proceedings of the IEEE Symposium on Security and Privacy*, 2011.

[5] D. Barrera, L. Chuat, A. Perrig, R. M. Reischuk and P. Szalachowski. The scion internet architecture: An internet architecture for the 21st century. *Communications of the ACM*, 60(6):56–65, 2017.

[6] D. Barrera, R. M. Reischuk, P. Szalachowski and A. Perrig. Scion five years later: Revisiting scalability, control, and isolation on next-generation networks. *arXiv/1508.01651 (under review for Communications of the ACM)*, 2015.

[7] A. Perrig, P. Szalachowski, R. M. Reischuk and L. Chuat. *SCION: A*

Secure Internet Architecture. Springer, Berlin, Heidelberg, To be published in 2017.

[8] S. Akhshabi and C. Dovrolis. The evolution of layered protocol stacks leads to an hourglass-shaped architecture. In F. Peruani N. Ganguly B. Mitra A. Mukherjee, M. Choudhury, editor, *Dynamics On and Of Complex Networks*, volume 2 of *Modeling and Simulation in Science, Engineering and Technology*, pages 55–88. Springer New York, 2013.

[9] S.E. Deering and R. M. Hinden. Internet protocol, version 6 (ipv6) specification. RFC 2460, RFC Editor, 1998.

[10] M. Schuchard, E. Y. Vasserman, A. Mohaisen, D. F. Kune, N. Hopper and Yongdae Kim. Losing control of the internet: Using the data plane to attack the control plane. *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, 2011.

[11] M. Caesar and J. Rexford. Bgp routing policies in isp networks. *IEEE network*, 19(6):5–11, 2005.

[12] C. Labovitz, A. Ahuja, A. Bose and F. Jahanian. Delayed internet routing convergence. *ACM SIGCOMM Computer Communication Review*, 30(4):175–187, 2000.

[13] S. Misel. Wow, as7007! *Merit NANOG Archive, apr*, 199(7):1997–04, 1997.

[14] D. Wendlandt, DG. Andersen and A. Perrig. Perspectives: Improving ssh-style host authentication with multi-path probing. In *USENIX Annual Technical Conference*, volume 200, 2008.

[15] Hijacking, YouTube. A RIPE NCC RIS case study, 2008.

[16] J. Cowie. The new threat: Targeted internet traffic misdirection. *Popular Mechanics, http://www. renesys. com/2013/11/mitm-internethijacking*, 2013.

[17] S. Kumar. Smurf-based distributed denial of service (ddos) attack amplification in internet. In *Internet Monitoring and Protection, 2007. ICIMP 2007. Second International Conference on*, pages 25–25. IEEE, 2007.

[18] M. D. Donno, N. Dragoni, A. Giaretta and A. Spognardi. A taxonomy of distributed denial of service attacks. *To appear in Proceedings of the International Conference on Information Society (i-Society)*, 2017.

[19] F. Gont. Icmp attacks against tcp. 2010.

[20] S. Herzog. Revisiting the estonian cyber attacks: Digital threats and multinational responses. *Browser Download This Paper*, 2011.

[21] B. Krebs. Ddos on dyn impacts twitter, spotify, reddit. *KrebsOnSecurity (October 16, 2016),.'l. ttps: t'krebsonsecurit). COitJ*, 20(16):1, 2016.

[22] N. Perlroth. Hackers used new weapons to disrupt major websites across us. *New York Times*, 21, 2016.

[23] E. Galperin, S. Schoen, P. Eckersley. A post mortem on the iranian diginotar attack. *EFF Blog, September*, 2011.

[24] T. Sterling. Second firm warns of concern after dutch hack, 2011.

[25] D. Andersen, H. Balakrishnan, F. Kaashoek and R. Morris. *Resilient overlay networks*, volume 35. ACM, 2001.

[26] I. Mironov T. Wobber M. Abadi, A. Birrell and Y. Xie. Global authentication in an untrustworthy world. In *HotOS*, 2013.

[27] K. Akdemir, M. Dixon, W. Feghali, P. Fay, V. Gopal, J. Guilford, E. Ozturk, G. Wolrich and R. Zohar. Breakthrough aes performance with intel aes new instructions. *White paper, June*, 2010.

[28] J. B. Postel. Internetwork protocol approaches. *IEEE Transactions on Communications*, 28(4):604–611, 1980.

[29] D. Boggs, J. Shoch, E. Taft, R. Metcalfe. Pup: An internetwork architecture. *IEEE Transactions on Communications*, 28(4):612–624, 1980.

[30] GENBAND. What is a Media Gateway? `https://www.genband.com/company/glossary/media-gateway`, 2017. [Online; accessed 12-June-2017].

[31] Real Time Automation (RTA). Connect Modbus TCP/IP Devices to a Modbus RTU Controller. `http://www.rtaautomation.com/product/460mcmrs/`, 2014. [Online; accessed 12-June-2017].

[32] V. DiCiccio, C. A. Sunshine, J. A. Field, E. G. Manning. Alternatives for interconnection of public packet switching data networks. *Proceedings of the Sixth Symposium on Data Communications*, pages 120–125, 1979.

[33] J. Knopp. X.25 within the payment card industry. Technical report, MasterCard, 2012.

[34] Recommendation, CCITT. Recommendation x.25/interface between data terminal equipment (dte) and data circuit-terminating equipment (dec) for terminals operating in the packet mode on public data networks. *Orange Book*, 2, 1977.

[35] Recommendation, CCITT. Proposal for provisional recommendation x.75 on international interworking between packet switched data networks. *CCITT Study Group VII Contribution*, (207), 1978.

[36] J. Postel. Dod standard internet protocol. RFC 760, RFC Editor, January 1980.

[37] J. Postel. Dod standard transmission control protocol. RFC 761, RFC Editor, January 1980.

[38] J. McQuillan, I. Richer and E.Rosen. The new routing algorithm for the arpanet. *IEEE Transactions on Communications*, 28(5):711–719, 1980.

[39] H. Chen, X. Jia and H. Li. A brief introduction to iot gateway. In *Communication Technology and Application (ICCTA 2011), IET International Conference on*, pages 610–613. IET, 2011.

[40] S. Guoqiang, C. Yanming, Z. Chao and Z. Yanxu. Design and implementation of a smart iot gateway. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 720–723. IEEE, 2013.

[41] K. Romer and F. Mattern. The design space of wireless sensor networks. *IEEE wireless communications*, 11(6):54–61, 2004.

[42] L. D. T. Steenkamp, S. Kaplan and R.H. Wilkinson. Wireless sensor network gateway. In *AFRICON, 2009. AFRICON'09.*, pages 1–6. IEEE, 2009.

[43] L. Wu, J. Riihijarvi and P. Mahonen. A modular wireless sensor network gateway design. In *Communications and Networking in China, 2007. CHINACOM'07. Second International Conference on*, pages 882–886. IEEE, 2007.

[44] S. Matsumoto, R. M Reischuk, P. Szalachowski, T. H. J. Kim, and A. Perrig. Designing a global authentication infrastructure. *arXiv preprint arXiv:1506.03392*, 2015.

[45] M. Lepinski and S. Turner. An overview of bgpsec. IETF Draft, 2012.

[46] T. H. J. Kim, C. Basescu, L. Jia, S. B. Lee, Y. C. Hu, and A. Perrig. Lightweight source authentication and path validation. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 271–282. ACM, 2014.

[47] B. Trammell. Rains (another internet naming service) protocol specification. Internet-draft, 2016.

[48] O. Honda, H. Ohsaki, M. Imase, M. Ishizuka and J. Murayama. Understanding tcp over tcp: effects of tcp tunneling on end-to-end throughput and latency. In M. Atiquzzaman and S. I. Balandin, editors, *Performance, Quality of Service, and Control of Next-Generation Communication and Sensor Networks III*, volume 6011 of , pages 138–146, October 2005.

[49] O. Titz. Why TCP Over TCP Is A Bad Idea. `http://sites.inka.de/bigred/devel/tcp-tcp.html`, 2001. [Online; accessed 18-June-2017].

[50] Y. Rekhter,R. G. Moskowitz, D. Karrenberg, G. J. de Groot and E. Lear. Address allocation for private internets. BCP 5, RFC Editor, 1996.

[51] Juniper Networks. Understanding Origin Validation for BGP. `https://www.juniper.net/documentation/en_US/junos/topics/concept/bgp-origin-as-validation.html`, 2015. [Online; accessed 22-June-2017].

[52] R. Bush. Origin validation operation based on the resource public key infrastructure (rpki). BCP 185, RFC Editor, 2014.

[53] M. Lepinski and S. Kent. An infrastructure to support secure internet routing. RFC 6480, RFC Editor, 2012.

[54] CIDR Report. CIDR REPORT for 20 Jun 17. `http://www.cidr-report.org/as2.0/#Aggs`, 2017. [Online; accessed 20-June-2017].

[55] N. Hilliard. ASN to IP Mapping. `http://seclists.org/nanog/2015/Mar/219`, 2015. [Online; accessed 28-July-2017].

[56] A. Khan, H. Kim and T. Kwon. How complete and accurate is the internet routing registry (irr)? *4th CAIDA-WIDE-CASFI Joint Measurement Workshop*, 2011.

[57] A. Toonk. How Accurate are the Internet Route Registries (RIR). `https://bgpmon.net/how-accurate-are-the-internet-route-registries-irr/`, 2009. [Online; accessed 23-June-2017].

[58] E. Zmijewski. Route Hygiene: The Dirt on the Internet. `http://dyn.com/blog/compliance-scoring-by-country/`, 2009. [Online; accessed 23-June-2017].

[59] C. Alaettinoglu and P. Design. Ripe/ris project bgp analysis cidr at work. *NANOG Oct. 2001, IETF August 2001*, 2001.

[60] RIPE NCC. Routing Information Service (RIS). `https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/`

routing-information-service-ris, 2016. [Online; accessed 23-June-2017].

[61] L. Daigle. Whois protocol specification. RFC 3912, RFC Editor, September 2004.

[62] T. Kernen. traceroute.org. http://www.traceroute.org/#Looking%20Glass, 2011. [Online; accessed 23-June-2017].

[63] M. Przybylski, B. Belter and A. Binczewski. Shall we worry about packet reordering. In *Computational Methods in Science and Technology*, pages 141–146, 2005.

[64] J. C. R. Bennett, C. Partridge and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Trans. Netw.*, 7(6):789–798, December 1999.

[65] V. Paxson, M. Allman, J. Chu and M. Sargent. Computing tcp's retransmission timer. RFC 6298, RFC Editor, June 2011. http://www.rfc-editor.org/rfc/rfc6298.txt.

[66] C. Abad. Ip checksum covert channels and selected hash collision. 2001.

[67] J. Preshing. Hash Collision Probabilities. http://preshing.com/20110504/hash-collision-probabilities/, 2011. [Online; accessed 27-June-2017].

[68] P. Syverson. A taxonomy of replay attacks [cryptographic protocols]. *Proceedings of the Computer Security Foundations Workshop VII*, 7(6):187–191, December 1994.

[69] D. L. Mills. Network time protocol (ntp). RFC 958, RFC Editor, September 1985. http://www.rfc-editor.org/rfc/rfc958.txt.

[70] S. P. Miller, B. C. Neuman, J. I. Schiller and J. H. Saltzer. Kerberos authentication and authorization system. In *In Project Athena Technical Plan*. Citeseer, 1987.

[71] S. M. Bellovin and M. Merritt. Limitations of the kerberos authentication system. *ACM SIGCOMM Computer Communication Review*, 20(5):119–132, 1990.

[72] D. Mazières. Replay Prevention in Authentication Systems. http://web.mit.edu/6.033/1998/www/reports/r05-dm.html, 1998. [Online; accessed 28-June-2017].

[73] J. Mirkovic and P. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, April 2004.

[74] Q. H. Dang. The keyed-hash message authentication code (hmac). *Federal Inf. Process. Stds. (NIST FIPS) - 198-1*, 2008.

[75] R. K. Dahal, J. Bhatta, T. N. Dhamala. Performance analysis of sha-2 and sha-3 finalists. *The International Journal on Cryptography and Information Security*, 3(3):1–10, 2013.

[76] J. Pavlik, A. Komarek, V. Sobeslav and J. Horalek. Gateway redundancy protocols. In *Computational Intelligence and Informatics (CINTI), 2014 IEEE 15th International Symposium on*, pages 459–464. IEEE, 2014.

[77] P. Dubey, S. Sharma and A. Sachdev. Review of first hop redundancy protocol and their functionalities. *International Journal of Engineering Trends and Technology (IJETT)*, 4(5), 2013.

[78] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. RFC 2267, RFC Editor, January 1998.

[79] M. Muuss. The story of the ping program. *Retrieved March*, 30:2010, 1983.

[80] W. M. Eddy. Syn flood attack. In *Encyclopedia of Cryptography and Security*, pages 1273–1274. Springer, 2011.

Abbreviations

AES Advanced Encryption Standard

AS Autonomous System

BGP Border Gateway Protocol

BGPSec BGP Security Extension

BR Boarder Router

BS Beacon server

CA Certification Authority

CS Certificate server

DDoS Distributed Denial of Service

DNS Domain Name System

DoS Denial of Service

DRKey Dynamically Re-creatable Key

ICMP Internet Control Message Protocol

IoT Internet of Things

IP Internet Protocol

IPS Intrusion Prevention System

ISD Isolation Domain

ISP Internet Service Provider

MAC Message Authentication Code

MTU Maximum Transmission Unit

OSI Open Systems Interconnection

PCB Path Construction Beacon

PCFS Packet-Carried Forwarding State

PS Path server

QoS Quality of Service

RAINS Another Internet Naming Service

RPKI Resource Public Key Infrastructure

RTT Round-Trip Time

SCMP SCION Control Message Protocol

SIBRA Scalable Internet Bandwidth Reservation Architecture

SIG SCION-IP Gateway

SSP SCION Stream Protocol

SVC Service

TCP Transport Control Protocol

TLS Transport Layer Security

TRC Trust Root Configuration

TTL Time To Live

UDP User Datagram Protocol

VM Virtual Machine

WAN Wide Area Network

WSN Wireless Sensor Network