Development of a SCION router towards a secure Internet

Bernardo António Santos Silva Lima Conde

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisors: Prof. Fernando Manuel Valente Ramos
Prof. Adrian Perrig

Examination Committee

November 2022

Acknowledgments

Firstly, I would like to thank my advisors Professor Fernando Ramos and Professor Adrian Perrig for their guidance and support throughout this project. I am very grateful to Eduard Marin, and the folks from Telefónica Research Barcelona, for teaching and helping me get this project to completion. In addition, I would like to thank my colleague Francisco Pereira, for being extremely helpful and patient with some of my questions. I would also like to profoundly thank my parents for being there for me and giving me this opportunity. Finally, I would like to thank my friends from Instituto Superior Técnico. Without them, these last 5 years wouldn't have been as fun. Except André Silva, that boy ain't right.

Abstract

Today's Internet architecture, while a marvel of software engineering, is plagued with serious security issues, mostly due to some of its early design decisions. These issues are extremely hard to fix with incremental solutions, prompting the question of whether we should design a new, future Internet architecture, capable of delivering the same service we have grown to love, without all the security problems it entails. One relevant example is SCION, a new Internet architecture with focus on security and availability. SCION's clean-slate approach means that its architecture is not compatible with the current network infrastructure. To gain traction, it is therefore fundamental to address a hard challenge: SCION has to achieve at least the same level of packet processing performance as current Internet routers. As a step in this direction, in this thesis we present an implementation of a terabit speed SCION data plane router. We implement our solution in a state of the art network device, a Programmable Switch, powered by an Intel Tofino Application-Specific Integrated Circuit (ASIC). These devices allow for the definition of their network functions to be done in software, using the P4 programming language. However, in order to achieve line-rate speeds, their computation model is restrictive, exacerbating the challenge. We show that our solution is able to achieve terabit per second line-rate speeds in our target, while performing the required on-demand cryptographic operations, which represents more than two orders of magnitude speedup over the state of the art.

Keywords

SDNs; SCION; EPIC; P4; Programmable Switches;

Resumo

A atual arquitetura da Internet, embora seja uma maravilha da engenharia de software atual, possui sérios problemas de segurança, devido a várias decisões feitas na sua génese. Este problemas são extremamente difíceis de resolver com soluções incrementais, levando-nos a questionar se deveríamos desenhar uma nova arquitetura, capaz de fornecer o mesmo nível de serviço da arquitetura atual, sem todos os problemas de segurança existentes atualmente. Um exemplo relevante é o SCION, uma nova arquitetura de Internet, focada em segurança e disponibilidade de serviço. A abordagem de raiz do SCION faz com que este não seja compatível com a infrastrutura existente. Para que o SCION possa ser adotado, é fundamental resolver este problema: o SCION tem de atingir pelo menos o mesmo nível de performance de processamento de pacotes da Internet atual. Como um passo nesta direção, nesta tese propomos o design e implementação de um router de Internet para o SCION, utilizando um aparelho de rede do estado da arte, um Programmable Switch, que contem um Intel Tofino Application-Specific Integrated Circuit (ASIC). Estes dispositivos permitem definir as suas funções em software, utilizando a linguagem de programação P4. No entanto, para atingirem velociades de line-rate, o seu modelo de computação é limitado, dificultando o problema. Nós mostramos que a nossa solução é capaz de atingir velocidades de terabit neste aparelho, enquanto faz operações criptográficas por pacote necessárias, o que representa um melhoramento superior a duas ordens de magnitude sobre o estado da arte.

Palavras Chave

Redes Programáveis; SCION; EPIC; P4; Switches Programáveis;

Contents

List of Figures

List of Tables

Acronyms

ALU	Arithmetic Logic Unit
AS	Autonomous System
ASIC	Application-Specific Integrated Circuit
BGP	Border Gateway Protocol
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
DoS	Denial of Service
EPIC	Every Packet Is Checked
FPGA	Field Programmable Gate Arrays
FIND	Future INternet Design
NSF	U.S National Science Foundation
GUID	Globally Unique Identifier
HDL	Hardware Description Language
ICMP	Internet Control Message Protocol
IP	Internet Protocol
ISD	Isolation Domain
MAC	Message Authentication Code
NPU	Network Processing Units
PISA	Protocol Independent Switching Architecture
RPKI	Resource Public Key Infrastructure
SCION	Scalability, Control and Isolation On next-generation Networks
SCMP	SCION Control Message Protocol
SEM	Simplified Even-Mansour

1 Introduction

Contents

1.1 Introduction

Today's Internet is an undeniable success, having completely revolutionized global communications and created multiple indispensable services, becoming the backbone of our modern, everyday life. However, the current Internet architecture was conceived 50 years ago, without any expectations that it would achieve the success that it has. As such, some of its early design decisions have caused multiple severe limitations, particularly with respect to security.

To make matters worse, today's Internet architecture suffers from an "ossification" of its network layer, due to the forceful use of the Internet Protocol [6]. Internet Protocol (IP) allows hosts to forward packets to one another. Unfortunately, due to its design, it makes the paths each packet takes completely opaque to its hosts, not allowing any control over the path to use nor to know which path a packet took towards its destination. Another issue is that it forces routers to maintain state, namely forwarding tables, making it more expensive to maintain, evolve, and scale the network infrastructure. Finally, IP packets are not authenticated, allowing a malicious host to lie about its identification, spoofing an IP address of another host. This enables possible Denial of Service (DoS) attacks that have caused serious disruptions [7].

Another "ossified" protocol is the inter-domain control plane protocol of the Internet, the Border Gateway Protocol (BGP) [8]. This protocol is responsible for deciding how packets get routed across the Internet, through the exchange of routing and reachability information. Unfortunately, due to the lack of authentication in this exchange, the protocol is susceptible to many security attacks that, when exploited, can result in interception of users' information or the complete blackout of essential services. While there have been extensions to this protocol [9, 10] that address some of these issues, these solutions also come with problems of their own.

In order to fix these issues, a new secure-by-design Internet architecture, called Scalability, Control and Isolation On next-generation Networks (SCION), has been proposed and created. SCION is a "path-aware Internet architecture, designed to provide complete route control, failure isolation, and explicit trust information for end-to-end communications" [1]. An advantage of SCION is the clean separation between the data plane protocol (the part of a network that carries user traffic) and the control plane protocol (the part of a network that carries signaling traffic and is responsible for routing) that it uses. Because of this separation, it is possible to have different protocols in each plane, while still respecting the SCION architecture. The SCION data plane is currently EPIC [11], a family of protocols focused on increasingly and incrementally adding stronger security guarantees. It has 4 versions, from EPIC L0 (the minimum offer of security) to EPIC L3 (the most secure, offering all the security guarantees from EPIC L0 to L2, plus others). These guarantees will be detailed in Section 2.4.

1.2 Motivation and purpose

SCION is a production-ready Internet architecture, currently offered by 12 SCION-native Internet Service Providers [12]. However, for SCION to be even more broadly adopted, it is necessary for its data plane to deliver similar performance to today's Internet. Recent work [3] has implemented an initial prototype of EPIC L0 in hardware, targeting an FPGA target, achieving 40 Gbps throughput. In this work we move forward in three fronts. First, we will design and implement a SCION-compatible router data plane in a programmable switch, with the goal of increasing the data plane performance by more than two orders of magnitude. Our specific target is the Intel Tofino Application-Specific Integrated Circuit (ASIC), capable of 12.8 Tbps. Second, we will implement both the EPIC L0 and the (more secure) L1 protocol versions. These protocols offer more security guarantees than SCION, fixing possible attack vectors. Our implementation will be done fully on the data-plane of the programmable network devices, allowing it to run at terabit speeds. Finally, we will implement our prototype using P4 [13], a domain-specific language for packet processing that can be compiled to various targets, from SmartNICs to programmable switching ASICs. Programming in P4 will facilitate porting to other targets.

1.3 Contributions

In this thesis we will:

- Design and implement a SCION-compatible router, running the EPIC L1 protocol fully on the data-plane, which offers better security guarantees than regular SCION. We will target a programmable network switch, capable of terabit speeds and we will implement our router using the P4 programming language.
- In order to achieve this, our router will need to perform one cryptographic operation per-packet. Due to limitations of previous work, we will propose and implement a lightweight cryptographic construct, the Simplified Even-Mansour. We will implement this construct for our target in an efficient manner, so our implementation is able to achieve the required throughput
- Finally we will evaluate our implementation against the state of the art and show that, in all cases, we are able to achieve comparable or better performance, in terms of throughput, while providing better security guarantees

1.4 Organization of the Document

This thesis is organized as follows:

- Section 2 presents background on the topics approached by this thesis;
- Section 3 describes the previous state of the art solutions, and their problems;
- Section 4 describes our proposed solution, that aims to extend the state of the art by improving security and performance;
- Section 5 provides the evaluation of our solution against previous work;
- Section 6 presents some conclusions.

2

Background

Contents

The following sections describe the background related to the subject of this thesis. As a starting point, Section 2.1 describes how (un)secure the present Internet Architecture is, exploring the problems with the most dominant protocols used today. Section 2.2 discusses the need for a clean-slate network architecture, presenting the inherent disadvantages of the status-quo. Section 2.3 introduces several proposals in the future network architectures space, their advantages and disadvantages. Section 2.4 details SCION [1], the secure-by-design Internet architecture that is the focus of our work. Finally, Section 2.5 introduces some background on programmable switches, Smart NICs, and the P4 language [13]; the core technologies of our approach.

2.1 Security in today's Internet Architecture

The Internet is a global, decentralized network, comprised of tens of thousands of interconnected networks, or Autonomous Systems (ASs). Each AS is normally under the control of a single administrative entity. Information travels between ASes using one out of many paths, chosen by a routing protocol, which exchanges information about the reachability of every destination.

While the rapid development of Internet services and technologies make it clear that the Internet is progressing quickly, the same is not true of its architecture. While its top (Application and Transport) layers and bottom (Datalink and Physical) layers have evolved significantly, the Internet middle stack has stagnated for decades (shown in Figure 2.1): both the data-plane and the inter-domain routing have been dominated by two protocols, IP [6] and BGP [8], respectively.

2.1.1 Problems with the Internet Protocol

The Internet Protocol, or IP, is one of the fundamental protocols of the current Internet architecture, responsible for the forwarding of packets between hosts. For this, each sender only needs to know the receiver's address, which will then be written on each of the packets' header and disseminated through the network.

Unfortunately, while this approach is simple, it has many drawbacks. One of these drawbacks is the fact that the path for a packet is opaque both for the sender and the receiver. Due to a lack of specification of any path in a packet, neither the sender nor the receiver have any ability to influence the path a specific packet can take, nor the information to know which path a packet took. This makes it impossible to prevent a packet from being routed through non-trusted networks, or for an end-host to choose a more suitable path through a desired metric.

Another serious issue is the fact that routing and forwarding are bound together, since packet forwarding depends on the state of forwarding tables in routers, which can change over time. This means

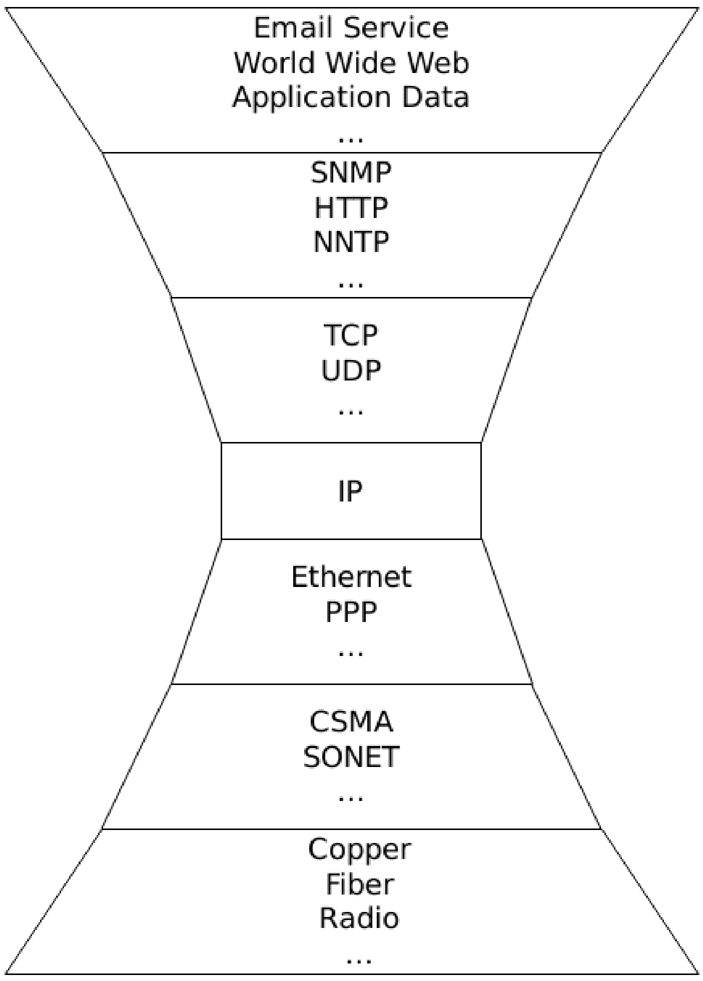


Figure 2.1: The Internet Hourglass Problem: while application-level protocols and physical level protocols can be changed/upgraded very easily, today's Internet architecture has a very hard dependency on its network-level protocol, the Internet Protocol, making it extremely hard to fix its problems or replace it with another, better protocol.

that a working path can change or even break after an update to the forwarding tables' state, which can occur at any point, potentially causing serious connectivity issues.

Additionally, due to each router's necessity to maintain forwarding tables, and to perform a lookup for each packet, each router's hardware must have special memory constructs to deal with these constraints. These components are very expensive and energy-intensive, leading to an increase of hardware's expenses. Plus, various Denial-of-Service attacks can come from attackers that exhaust a router's memory by filling forwarding tables maliciously [14].

Finally, IP lacks any security guarantees over routing and packet deliverance: a packet can pass through any intermediate hop at any time due to its lack of path authentication. This makes enforcing the use of specific paths for certain hosts extremely challenging.

2.1.2 Problems with BGP

Unfortunately, while BGP works well most of the time, one of its main limitations is security [15]. In BGP, the control plane and the data plane are not cleanly separated: each AS advertises its reachability information about its list of IP prefixes as a BGP UPDATE message that is sent using the data plane. This lack of separation often causes packet forwarding to fail when a valid route changes (due to the acceptance of a BGP UPDATE), seriously affecting the underlying traffic [16].

Another issue with BGP is the lack of fault isolation: since it is a completely distributed system with no hierarchy or isolation, any BGP client can affect the whole network, which means that any client can seriously disrupt global connectivity. As an example of this: in 2008, a single wrongly-configured router from Pakistan Telecom made YouTube unreachable to the whole world for hours [17]. Because each update needs to be sent to every AS as well, BGP tends to not scale very well and, due to the time it takes to disseminate a message through the whole network, a consistent global view of all correct and available routes can take several minutes, which can provoke service outages for users (in fact, it has been shown that, for certain situations, BGP may never fully converge [18] or converge non-deterministically [19]).

BGP also only selects a single path, providing no multi-path support. This can lead to bottlenecks when BGP selects a legitimate but inefficient route through a congested link. Because of the inability of the end hosts to choose their own path for their packets to take, they have no choice but to wait until ASes modify policies such that a more appropriate path gets chosen.

Moreover, BGP performs no authentication nor authorization of paths, allowing a malicious AS to perform a hijack attack (as the Pakistan incident described above) or an interception attack, giving a malicious entity the ability to passively monitor and collect traffic, or to create a blackhole.

To address these and other security problems, two mechanisms, the Resource Public Key Infrastructure (RPKI) [9], and BGPsec [10] have been proposed. These standards allow an AS to cryptographically sign a BGP route announcement, thereby making the above attacks impracticable.

2.1.3 Problems with the RPKI and BGPsec

BGPsec is an IETF standard that attempts to assure an AS that the content of a received BGP UPDATE message correctly represents the inter-AS propagation path of the update, from the point of origin to the receiver of the route. This authentication makes it possible to prevent a wide variety of route hijacking attacks against BGP. It relies on the RPKI certificates that attest the allocation of AS number and IP address resources. An AS can associate its AS number with its own public key. When it receives a BGP UPDATE message, it adds its own AS number, the destination's AS number, and a cryptographic proof over this information, generated using its private key. Other ASs that receive this BGP UPDATE message can check all of the signatures on each message and, if all signatures are correct, know that the inter-AS propagation path must be correct.

However, the RPKI and BGPsec do not solve all problems. For instance, while the RPKI offers origin authentication and thus works against the IP hijack attacks, it still allows other, more sophisticated attacks to occur [20]. For example, a malicious AS trying to hijack a particular IP prefix can still send a BGP UPDATE message claiming that it is *directly connected* to its legitimate owner. Recipients of such an announcement would accept it as the legitimate owner of the addresses, as the malicious AS is noted as the last AS in the BGP UPDATE and would then start sending the traffic destined for those IP addresses to the attacker, who can then inspect, reroute, or drop it.

BGPsec addresses this issue by signing the entire path, but attackers are still able to create so called ”wormhole” attacks [21] (two ASes conspire with each other to generate valid BGPsec signatures, in order to convince victim ASes to use them for communication) and cause forwarding loops.

BGPsec has also been reported to work poorly unless all ASes use and enforce BGPsec [22]. When used in a partial deployment scheme (only some ASes enforce BGPsec, while others do not), BGPsec can lead to severe issues like instability. Due to backwards compatibility requirements, BGPsec is also prone to downgrade attacks (where attackers fake not supporting BGPsec in order to more easily attack the network).

Another problem with BGPsec is the existence of circular dependencies [23]. In order to be able to participate in the network securely, one must be able to fetch and exchange cryptographic keys and RPKI certificates. In turn, in order to fetch these, one needs to already be able to participate in the network.

Another problem is the ability of enforcement of network sovereignty, as organizations at the root of the RPKI hierarchy have the power to create or revoke certificates. Depending on the jurisdiction, local courts of some of the countries of origin for these organisations may gain the power to shut down parts of the Internet (with the obvious possibility of abuse).

Finally, BGPsec exacerbates BGP's scalability issues. Under BGP, in order to provide global connectivity, every single global AS in the world needs to know how to reach every other AS. This requires a large number of BGP UPDATE messages, the processing of which requires much more resources in BGPsec, due to the additional cryptographic checks. Furthermore, prefix aggregation, which is used to combine multiple IP prefixes to reduce the number of routes and announcements, no longer works in BGPsec. This is particularly cumbersome as the increasing fragmentation of the IP address space and the trend towards announcing ever smaller IP address ranges have caused a strong growth of the number of paths that internet routers need to store and exchange.

All these problems have hindered the widespread adoption of BGPsec.

2.2 Future Internet: clean-slate or evolution?

Due to the incredible success of today's Internet architecture, does the argument for a new, clean-slate design make sense? After all, the current Internet architecture has been able to scale, provide adequate support to millions of people world-wide, so why would it be necessary to start again from scratch? Why not just fixing the existing architecture?

On one hand, we have "evolutionary designs", consisting of incrementally improving the current architecture. This has many advantages. One is the ability to reuse current infrastructure, taking advantage of the existing knowledge and the multiple optimizations made over the years. Another advantage is that an evolutionary design is easier to be adopted. If the new developments remain backwards-compatible, we would not face the risk of segmenting parts of the global network, due to technologically or economically backed disputes over which Internet architecture improvements to use. But the main reason in favor of this approach is that today's Internet architecture has proved itself in practise, as an enabler for world-scale, distributed, instant communication.

However, some of the most serious problems with the current architecture are deeply rooted in early design decisions that are very hard to change. The main one is security, as is clear from the extremely damaging real-world attacks, like Phishing, Spam, and (Distributed-)Denial-of-Service attacks that plague the Internet. These security issues cannot be easily "patched up", since they are caused by assumptions of the early design, namely related to trust, but also the lack of sophisticated cryptography at the time.

Another problem is that the current Internet architecture cannot handle mobile hosts gracefully. TCP/IP uses *Source* and *Destination* fields in their packets to know where to send and deliver them. It is assumed, however, that the end-hosts do not lose connection to the network (or have an intermittent connection), nor change its physical network connection (also known as roaming). In a world of mobile computing devices, these restrictions are no longer sensible.

Today's Internet architecture also does not provide any guarantee that data is delivered or that delivery meets any quality of service – it only offers best-effort delivery. While the simple nature of this delivery method made it very easy to be implemented and adopted world-wide, it has the unfortunate downside that most protocols and modes of use of today's Internet don't lend themselves very well to this packet delivery method. While one can create more reliable methods on top of this (like TCP), one can argue that forcing these guarantees to be provided at the network-layer instead can provide better efficiency and reliability in the long run.

Finally, because of the lack of separation of control plane and data plane, multiple management systems can only indirectly tune and change the network traffic flow. Today's Internet architecture also forces all intermediate routers to have fast, internal memory in order to be efficient. Due to these requirements, the hardware and infrastructure required to manage a large network is both extremely expensive, and very easy to get wrong.

With all these limitation, the debate for a clean-slate architecture is still worth having [24].

2.3 New Internet Architectures

The design and implementation of new Internet architectures has been a very active area of research [25]. One early example was the creation of the Future INternet Design (FIND) research area, from the U.S National Science Foundation (NSF), focused on "stimulating innovative and creative research to explore, design, and evaluate trustworthy future Internet architectures".

Multiple promising new Internet architectures arose from such efforts. One of the most promising is an Internet architecture called **Named Data Networking** [26]. NDN focused on removing the restriction that packets can only name communication endpoints, changing the semantic of the network service from *delivering a packet to a host* to *fetching data from the network*. While today's Internet has a host-centric architecture, Named Data Networking on the other hand explores a data-centric one. It accomplishes this by generalizing the endpoint information in IP packets: it allows any name (where name in this context is any possible identifier) to be used as a possible endpoint. This architecture continues to allow host-to-host communication (any particular host can be easily identified by a specific name) while also allowing other types of communication, such as service oriented communication (the name can, for example, be an identifier for a specific piece of data; allowing one to just request it from the network, and retrieve it from any source that serves it). These names can also be much more complex: they can have structure to them, allowing one to define specific relations and meta-information to the data queried (for example, a video produced by UCLA may have the name **/ucla/videos/demo.mpg**, where '/' delineates hierarchical name components, similar to URLs). In order to find which hosts/ASes to contact in order to find a specific endpoint/service, these hosts/ASes no longer broadcast just IP Prefix lists, but Name Prefix lists (this forces names associated to globally accessible data to be globally unique, but names used for local communications require only local uniqueness, similar to IP addresses).

An Internet architecture focused on mobile devices is **MobilityFirst** [27], with the objective of solving the connection issues associated with the presence of mobile devices (mainly roaming and intermittent physical connections). It accomplishes this by completely separating names or identifiers from addresses or network locations, creating a logically centralized global name service (like a Domain Name System) at the network-layer. With this, one can significantly enhance the allowed network mobility of all end-hosts. MobilityFirst defines the concept of a *Self-Certifying identifier*, a Globally Unique Identifier (GUID) representing a one-way hash of a public key, allowing authentication between hosts to occur. If host X wants to know if a certain GUID is owned by host Y, it can know this by sending Y a random nonce; then Y responds with a tuple $(PublicKey, Encrypt_{PrivateKey}(nonce))$; which allows X to check if the hash of the $PublicKey$ is equal to the GUID, and whether the decryption of the second tuple member, using the supplied $PublicKey$, equals to the sent nonce; if both are correct, then X can be sure that Y is the owner of the information referenced by that GUID. It exposes two resolution services to the network: one mapping human-readable names to the identifiers described above, and another mapping identifiers to a flexible set of attributes, including but not limited to their network address. When one host changes their physical network connection, by only updating these resolution services, the rest of the traffic can adjust and change, maintaining a viable, network connection between end-hosts.

Another example is the **eXpressive Internet Architecture** [28], or XIA. XIA identifies the host-centrality of today's Internet architecture as a problem, and aims to solve it by allowing and accommodating other types of principals, such as content, services or users, in an extensible way, allowing the architecture to evolve with future use cases. It does this, similarly to Named Data Networking and MobilityFirst, by extending the hosts' identification fields with other identifiers, allowing more meta-information to be communicated in each network connection. XIA starts by describing 4 basic types of identifiers: Host Identifiers, that define who you communicate with; Service Identifiers, that define what the entities in the network do; Content Identifiers, that allow hosts to retrieve content from anywhere in the network; and finally Network Identifiers, that identify a specific AS, allowing one to verify that a connection is being made with the intended network. XIA allows very flexible addressing, by allowing network addresses to be expressed as Directed Acyclic Graphs (DAGs) of identifiers. This allows a user to express, for each packet, other fall-back methods of access, as well as choosing network scoping.

While all these new Internet architectures have their upsides, none of them addresses the serious security issues that plague today's Internet. The main proposal that focuses strongly on these issues is SCION, the focus of this work, which we describe next.

2.4 SCION

SCION [1], short for Scalability, Control, and Isolation On Next-Generation Networks, is a new, clean-slate, path-aware, Internet architecture, aiming at "offering complete route control, failure isolation, and explicit trust information for end-to-end communications". SCION groups existing Autonomous Systems into Isolation Domains (ISDs) that connect with each other to provide global connectivity (Figure 2.2). Each ISD is administered by a subset of its ASes, called the ISD core, which is responsible for setting the ISD policy. These ISD cores establish the available paths for communication inside and outside the ISD (this is called the path-exploration process), which will be disseminated through all of the hosts of each ISD. These hosts will then specify which path to use for each packet, by making them explicit in the packet's header. Every packet sent outbound the ISD must pass through the ISD core, allowing each ISD to set strong forwarding policies and to prevent malicious path creation. It also enables defenses against network attacks: since the full path of a packet is carried in the packet itself, finding out the original sender of a packet is possible, allowing the receiver and the network to deal with them accordingly. SCION also completely separates the control plane from the data plane, ensuring that forwarding cannot retroactively be influenced by control plane operations. For the control plane, SCION uses the SCION Control Message Protocol (SCMP), analogous to the Internet Control Message Protocol (ICMP). It provides functionality for network diagnostics, such as $ping$ and $traceroute$, and error messages that signal packet processing or network-layer problems.

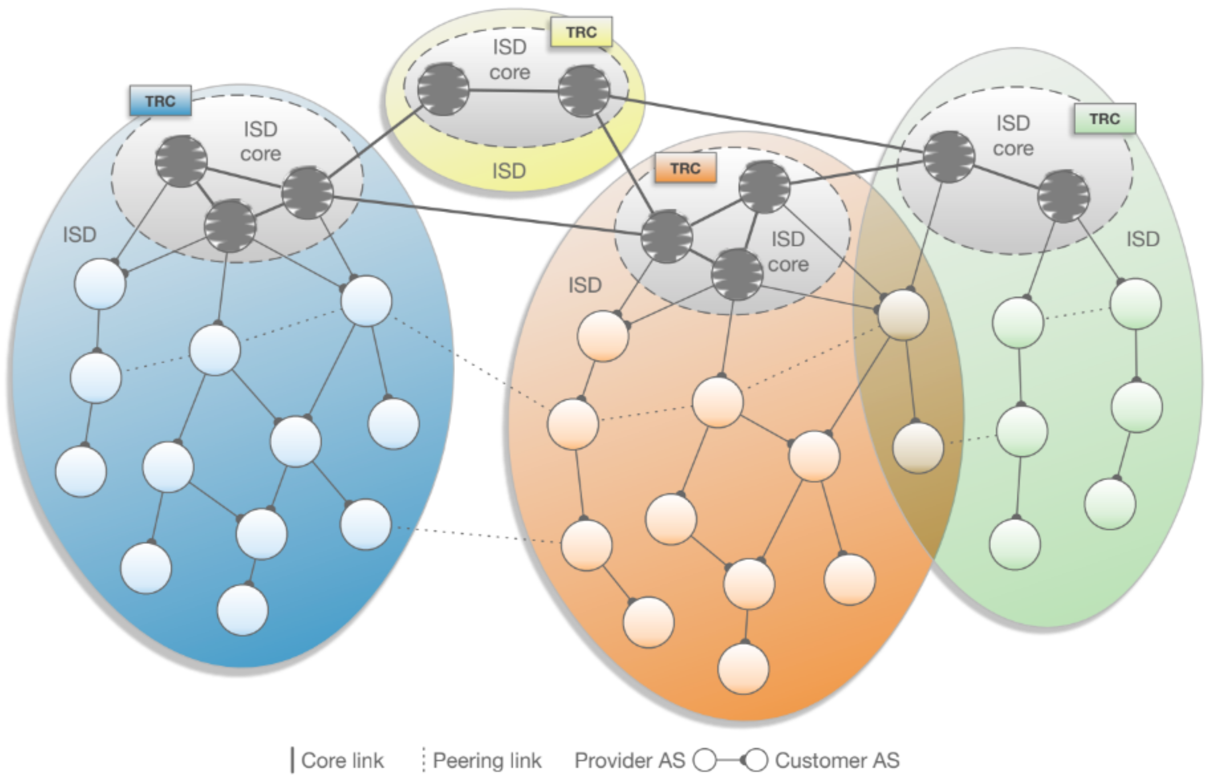


Figure 2.2: SCION ISD model (from [1]): Multiple different ASes that trust each other join together, becoming ISDs. For each ISD, there is a group of special ASes, called the ISD core, responsible for enforcing rules over all the traffic sent to other ISDs. These ISD cores also are responsible for creating and disseminating the authorized paths to each of their own ASes, as the ISD policy seems fit.

2.4.1 EPIC

Every Packet Is Checked (EPIC) [11] is a family of data plane protocols designed to be used in path-aware internet architectures like SCION. It presents 4 different versions, each designed to provide increasingly strong security properties, while allowing: network operators to be able to impose their own policies; end hosts to verify that their forwarding decisions are followed by the network; and intermediate routers and recipients to authenticate the source of packets.

The EPIC family of protocols is made up of four different versions:

- EPIC L0: EPIC L0 is a simplified version of the original data plane protocol used in SCION. During the path-exploration process, each AS generates a Hop Authenticator: a static Message Authentication Code (MAC), using each AS key, over the path's starting timestamp, the hop information, and the previous Hop Authenticator, truncated to len_{ha} bytes. When each source obtains a path, including all Hop Authenticators for that path, it will send their Hop Validation Field as just each Hop Authenticator; every intermediate router checks if the packet came from the correct interface and if the Hop Validation Field corresponds to the correct Hop Authenticator. L0 has a clear problem: if the Hop Authenticator length len_{ha} is not big enough, its security properties can be broken with an online brute-force attack, generating valid Hop Authenticators for invalid paths. Since Hop Authenticators will be used as Hop Validation Fields, and these Hop Authenticators can be reused for different packets, one only needs to do this costly attack once.

- EPIC L1: EPIC L1 solves the brute-force attack problem by making each Hop Validation Field dependent on packet timing information, thereby making sure that they cannot be reused. Each Hop Validation Field is a MAC of the packet timestamp, plus source and the host address, using the Hop Authenticator of each AS as a key.

- EPIC L2: EPIC L2 extends the previous security guarantees by also allowing intermediate routers to authenticate the source of a packet and the destination to authenticate its payload. Each intermediate AS generates a new key, derived from the source host and AS information. The source host will then generate, for each Hop Field Value, a MAC using the same information as L1, but using this new, generated, AS-unique key. For the destination, another new key will also be generated, based on the source and destination's host and AS, which will then be used by the source to create a MAC of the contents of the packet, plus information about the path. Since all the information to derive these keys is public, all keys can be locally generated.

- EPIC L3: Finally, EPIC L3 provides the strongest security properties, adding also the ability for both the source and the destination to perform path validation. For that purpose, each on-path AS overwrites their Hop Validation Field with a proof that they have processed the packet. This proof is the higher part of the non-truncated MAC used to generate each Hop Validation Field. Since, at the start, these fields contain the lower part of the MAC, truncating it so it fits, and since this is information that both the source and the destination already have, both can check that each packet has passed through each AS. The destination gets this updated information right when it receives the packet; in order for the source to get it, the destination sends this information as an EPIC L2 packet, in order to prevent cyclical confirmations.

EPIC Version	Path Authorization	Freshness	Packet/Source Authentication	Path Validation
L0	Yes	No	No	No
L1	Yes	Yes	No	No
L2	Yes	Yes	Yes	No
L3	Yes	Yes	Yes	Yes

Table 2.1: Security Guarantees per EPIC Protocol version

Overall, EPIC is a family of protocols that offers strong security guarantees, while being lightweight enough that we expect it may be possible to run it fully on a fast data plane, such as a modern SmartNIC or a programmable switch.

2.4.1.A EPIC Header Structure

As an high level overview, an EPIC packet header (Figure 2.3i) consists of an arrangement of 5 different header substructures.

The first one is the Common Header (Figure 2.3a). The Common Header contains metadata about the packet. It specifies attributes like the packet's version of SCION (currently only 0 is supported), the type and length of source and destination addresses (they may be different types and have different sizes; for example, source may be a IPv4 address and destination may be a IPv6 one), and the packet's type (whether it contains payload, etc.). It also has information about the overall length of the header and the size of the payload.

The second one is the Address Header (Figure 2.3b). The Address Header has information about the addresses of both source and destination, plus their respective ISDs and ASes. This allows an intermediate hop to check if the packet belongs to its ISD/AS.

The third one is the Path Meta Header. The Path Meta Header is responsible for containing meta information about the path contained in the path header. It has two important fields: $CurrInf$ or CI (field has 2 bits and its value can be at most 2, mandated by the protocol) and $CurrHF$ (fields has 6 bits and its value be at most 64), fields that identify which Info Field Header or Hop Field Header is related to our hop, respectively. If we are an intermediate hop, after processing a packet, we are required to increment one or both values, in order to allow the next hop to correctly process the packet. EPIC L1 needs to have a packet timestamp embedded in the packet headers, in order to guarantee freshness. We decided to add an extra 32-bit field to the Path Meta Header, with this information (see Figures 2.3c and 2.3d).

Additionally, there can be between 1 and 3 Info Field Headers (Figure 2.3e), responsible for containing additional information about segments of the path. These contain a truncated MAC of the previous hop, called segment id, $SegID$. This guarantees chaining of MACs, making reordering the hops of a path impossible.

Finally, there can be between 1 and 64 Hop Field Headers (Figure 2.3f). Each Hop Field Header is responsible for holding the MAC information of each hop, coupled with the identifiers of the specific path on each switch, the $ConsIngress$ and $ConsEgress$.

2.5 Programmable networks

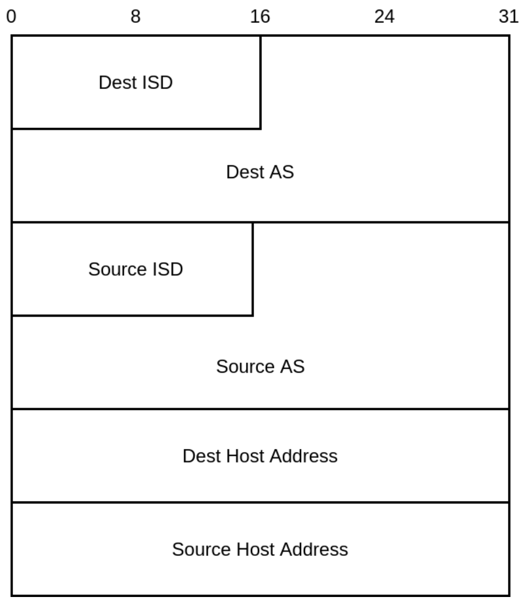
The switches and routers that make up the Internet, and that form the core of Internet service providers or data center networks, scale to the Terabit scales necessary to handle the increasing amount of bandwidth demands. Providers thus always require the fastest, most efficient data plane hardware to meet their user needs. Recently, the need for flexibility has led to the inception of a new kind of network data plane that can be summarised in two words: open and programmable.

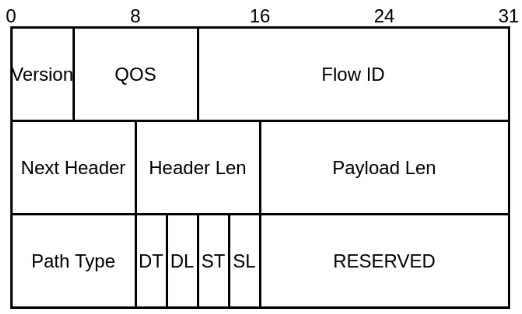
In this section we give some background on two types of fast, programmable data planes – programmable switching ASICs and SmartNICs – and of the P4 language, used to program these types of packet processors.

2.5.1 Programmable switching ASICs

Traditional routers and switches are *fixed-function*: the way they process packets is fixed during the chip design phase. A new class of programmable switching ASICs has recently emerged that allows packet processing to be specified by a program, and to be reconfigured in the field, improving the flexibility of modern networks [29]. The architecture of switches tends to follow a multi-stage pipeline process, where different stages of the pipeline look and act on different header fields of each packet. While this process tends to add end-to-end latency to each packet, it also allows for multiple packets to be processed simultaneously, since different stages of the pipeline can run concurrently in different packets.

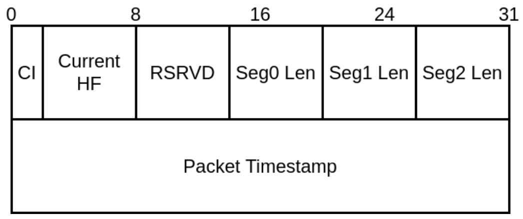
The pipeline of a programmable switching ASIC is often refereed to as the Protocol Independent Switching Architecture (PISA) (see Figure 2.4). This architecture starts by running a programmable parser on each packet, responsible for recognizing the header fields and matching them to later stages. Then, a sequence of Match/Action rules runs over all matched packets, tasked with potentially acting upon one or more of the identified header fields. Finally, there is a deparser, that serializes the packet metadata, obtained from all the in-memory header fields processed by earlier stages, into the packet, in order for it to be transmitted into the output link.

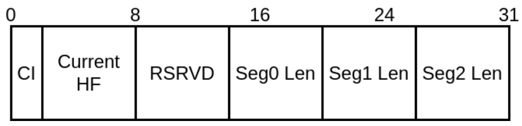




(a) EPIC Common Header

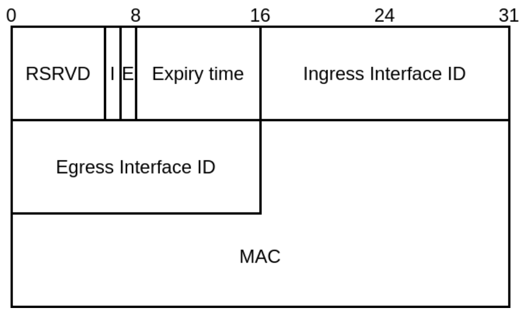
(b) EPIC Address Header

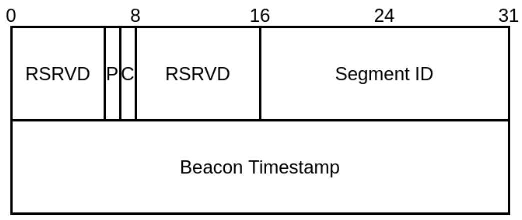




(c) EPIC L0 Path Meta Header

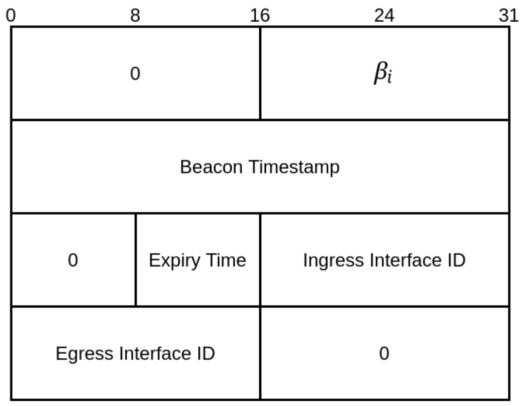
(d) EPIC L1 Path Meta Header

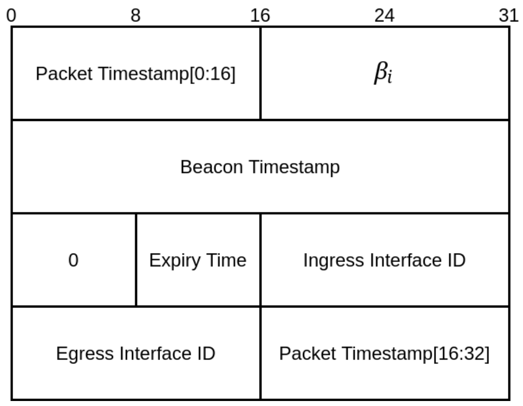




(e) EPIC Info Field Header

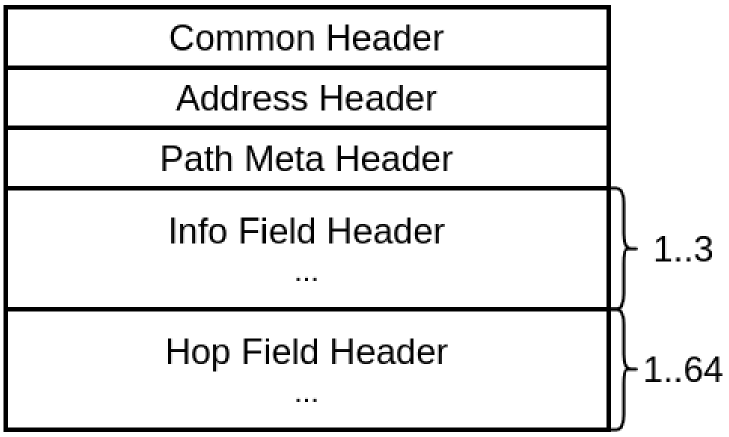
(f) EPIC Hop Field Header





(g) EPIC L0 MAC Input Data

(h) EPIC L1 MAC Input Data



(i) EPIC Packet. An EPIC packet contains: a Common Header (Figure 2.3a), containing header metadata about the packet; an Address Header, containing information about sender and receiver addresses; a Path Meta Header (Figure 2.3c and 2.3d), containing meta information about the SCION path in the packet; multiple Info Field Headers (Figure 2.3e), containing information about parts of the path in the packet; and multiple Hop Field headers (Figure 2.3f), containing the authentication information for each hop of the path.

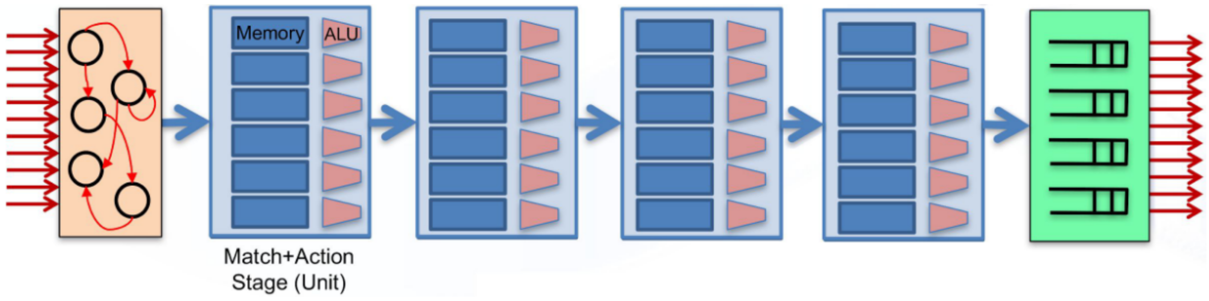


Figure 2.4: The Protocol Independent Switch Architecture (PISA) is made of 3 steps: the parser (red), a pipeline comprised of multiple Match/Action units (blue) and finally a deparser (green). In order to achieve fast speeds, the computations in each Match/Action unit are very constrained.

2.5.2 SmartNICs

Smart Network Interface Controllers, also called SmartNICs, are state of the art devices, capable of having their functions defined by software, while attaining close to hardware-backed speeds. They do this by moving functions that previously were executed by general-purpose processors into programmable hardware, such as Field Programmable Gate Arrays (FPGA) or Network Processing Units (NPU). This programmable hardware can then be controlled by domain-specific, high-level languages, like P4, described next.

2.5.3 P4 Language

In recent years, the Programming Protocol-Independent Packet Processors [13] language, better known as P4, has received a lot of support, becoming the *de-facto* way of programming programmable data planes, including ASICs, FPGAs, SmartNICs, and even x86 software switches.

P4 is a domain-specific language, custom-made for providing:

- Protocol Independence: P4 allows a programmer to define the header formats and field names of any protocol, directly in the P4 program, which will in turn be interpreted and processed by the program and target device.

- Target Independence: P4 is designed to be implementation-independent, allowing the same P4 program to be compiled into multiple different P4 targets, as per above.

- Reconfigurability: Due to it being a high-level language, a P4 target can change the way it processes packets after it is deployed, by simply having it run a different P4 program.

The P4 computation model (see Figure 2.5) is different from that of a general purpose Central Processing Unit (CPU), as it targets specialized packet processors. Its first stage consists of a programmable parser, that parses the header fields to be extracted from each packet. Then, several programmable match/action tables will be applied onto each matching packet. The P4 programmer can specify the type of matches to be done in each field of the packet (exact match, longest prefix match, etc.), and program the sort of actions available (forward packet, change header fields, perform simple Arithmetic Logic Unit (ALU) computations, access stateful memory, etc.).

A P4 program is structured into several different components:

- Headers: The specification of the header formats, as structures with named fields and their lengths.

- Parsers: A finite state machine defining how to parse each header into one of the previously defined structures.

- Tables: Description of multiple match/action rules, and which actions are to be performed for each match.

- Actions: Set of primitive functions to build more complex sequences of actions.

- Control Programs: Specification of the order that match/action rules shall be applied for a packet, defining the control flow of the packet processing.

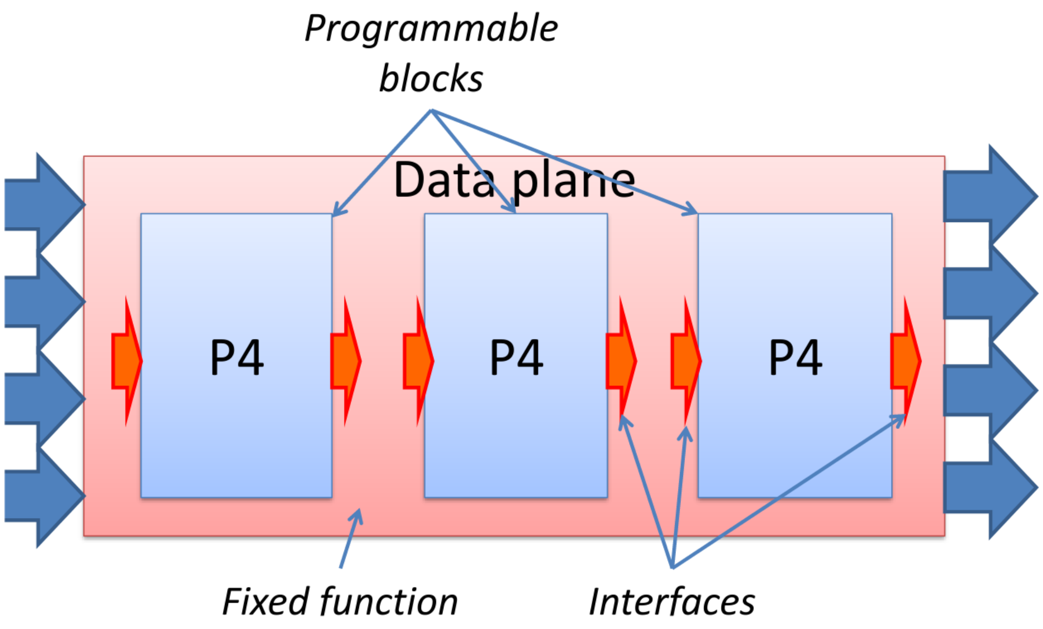


Figure 2.5: P4 Model from [2]

Each packet processing unit starts with the processing of arriving packets by the parser, which extracts each header. These header fields are then sent through a series of match/action tables divided in two parts: the first is the ingress pipeline, that determines the egress port and queue order of the packets, besides other functions; and the egress pipeline, responsible for defining the destination ports and number of instances of the packet to send, besides other programmer-defined actions, again.

The flexibility of P4 targets lends themselves as viable targets to become the infrastructure elements of a new Internet architecture, as it allows the precise definition of the packet processing of the network data plane.

2.6 Summary

This chapter presented the multiple fundamental security problems present in today's Internet architecture, mainly in IP and in BGP, and made the case for the need of a new, clean-slate redesign. We analysed multiple possible new designs including SCION, a new Internet architecture focused on security and efficiency. We also introduced EPIC, a family of increasingly-secure data-plane protocols made to be used in SCION architectures and explained their security properties. Finally, we introduced the concept of programmable data-plane devices and their advantages. We also described the P4 programming language and its computational model.

In subsequent chapters, we analyse previous implementations of SCION routers and present a new implementation that aims to offer increased security guarantees, while maintaining terabit performance.

3 Related Work

Contents

This chapter discusses previous implementations of SCION compatible data planes. To the best of our knowledge, there are currently three implementations of the SCION data plane: the official, software-based, x86_64 implementation [30]; a NetFPGA-based one by Soucková [3]; and one targeting a programmable network switch by Joeri and Schutijser [4]. In this chapter, we will present them one by one, and analyze their advantages and disadvantages. These solutions showed that it is possible to run the SCION data plane at tens of Gbit speeds. The challenge is now how to scale this in order to achieve terabit speeds, while providing better security guarantees. A positive aspect of these implementations was that they led to an in-depth understanding of the limitations of the original SCION data plane specification. Since it was initially designed considering software targets, it made some choices in is design that made it hard – or strictly impossible – to port it directly to hardware.

3.1 x86_64 SCION official implementation

The SCIONLab institution offers a free, open-source, complete implementation of all the SCION protocols [30], including a full specification of designs. Our solution follows this specification, in order to be compatible with existing software. This implementation is written in the Go programming language. It is only a reference implementation, useful for testing purposes. It is not intended for use in large-scale networks, as its performance is limited.

3.2 NetFPGA SUME hardware implementation

The NetFPGA SUME [31] is an FPGA-based PCI Express board with I/O capabilities for 100Gbps operation, capable of running P4-based programs. In 2019, Soucková [3] presented an implementation of the SCION data-plane, very similar to EPIC L0, running on a NetFPGA SUME (see Figure 3.1 for a design overview). By reconfiguring the internal FPGA into an extremely optimized AES module, they showed parts of an implementation processing packets at 40Gbps, in a device with 4 10Gbps ports.

The most important part of this design was the fact that the cryptographic operation was ran *online*: for every packet, a MAC was computed by the FPGA module and checked with the one contained in the packet, for authentication purposes. This work was extremely important in showing that an implementation of EPIC L0 that performs its cryptographic operations per packet is possible.

The main drawback of this design is its throughput: in this day and age, 40Gbps is not enough for a production-ready router. Also, this design implements the original SCION data-plane protocol, that is vulnerable to MAC brute-force attacks. Our objective is to improve the performance of these metrics by at least two orders of magnitude, while making it more secure against this type of attacks.

Worth of note is that the implementation failed to be completed due to latency issues: in order to be

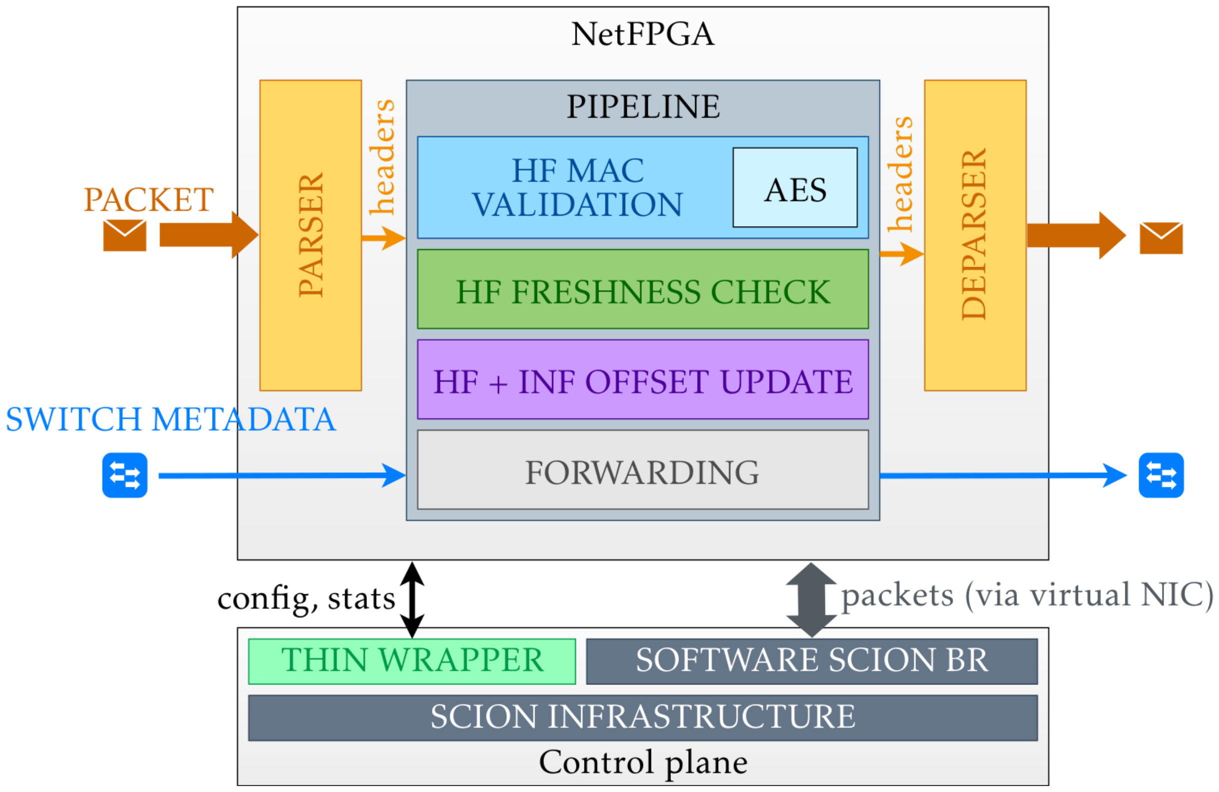


Figure 3.1: Overview of the NetFPGA SUME SCION implementation (from [3]): it computes a AES MAC validation per packet, at 40Gbps.

able to process packets at line-rate, they were required to have the entire processing pipeline fit in a 5ns window; however, their only able to fit their fastest implementation in 5.08ns. However, parts of it were still able to be evaluated independently. We will use these independent results in the evaluation against our solution.

3.3 Intel Tofino implementation of EPIC L0

Recently, Joeri and Schutijser [4] presented an implementation of the SCION data-plane protocol running on an Intel Tofino (see Figure 3.2 for a design overview). In order to be able to run on the Tofino, they were forced to do two things: the first was to pre-compute all cryptographic operations; the second one was to heavily refactor the headers of a SCION packet in order to be easier to parse.

Since AES is too expensive to be done on the data-plane of a Tofino, Joeri and Schutijser had to move all the cryptographic operations to outside the data-plane. Instead, during path discovery, the general CPU on the device is responsible for performing all the cryptographic computations, precomputing a MAC for every possible path and storing them in a table, accessible by the data-plane. Since, on EPIC L0, each MAC only authenticates information related to a path, this allows the cryptographic operation

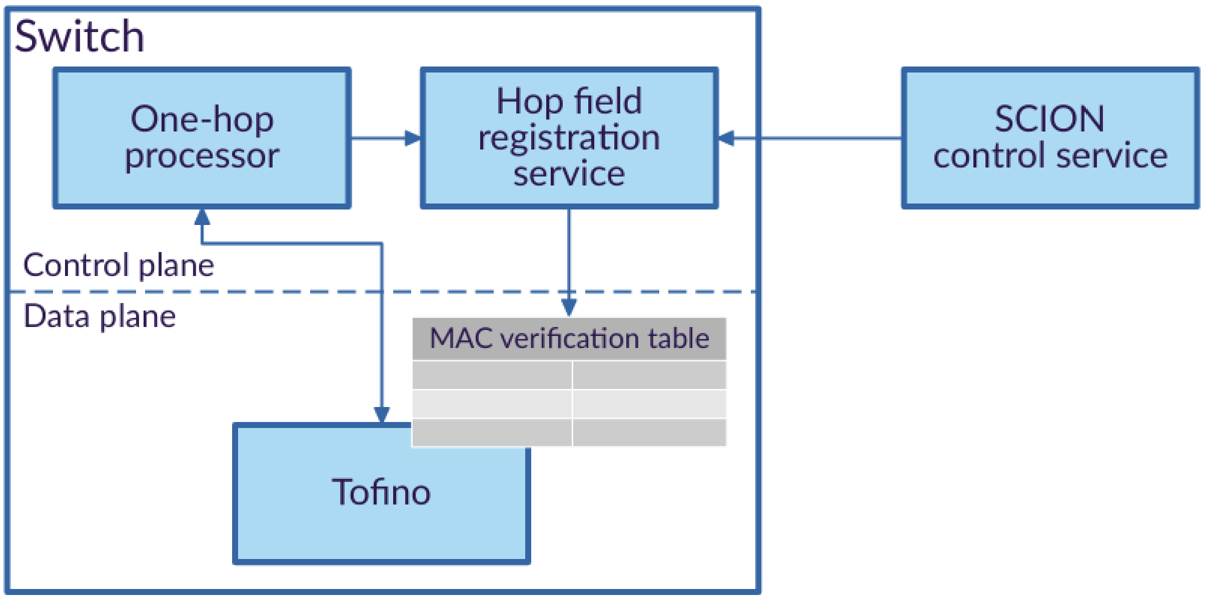


Figure 3.2: Overview of a SCION implementation on an Intel Tofino (from [4]): it pre-computes MACs for all the paths, and stores them in a table. The data-plane can then access this table and check if the packet's MAC is valid.

for each packet to become a simple table read and comparison operation. If, for a specific path, the MAC on the precomputed table is the same as the one in the packet, then the packet is authenticated and can pass through.

The other changes proposed in this work [4] refer to the SCION protocol headers. These changes were important in order to make SCION more easily adapted to hardware targets. These changes were:

- Explicit length for address fields: previously the header only specified the type of address (IPv4 or IPv6), and not its length, which meant that all nodes had to be aware of the possible sizes of each type of address. Only end-nodes have to process this information; intermediate nodes do not need to process these addresses, but they must know their sizes in order to skip over them. By adding a address length field, intermediate nodes can more easily skip this information.

- Flatten all header data structures: previously, SCION used nested lists for forwarding paths (each structure can contain lists of other structures). Hardware implementation have the need of having to statically allocate all resources; making it very complex to implement an efficient parser of nested structures. By separating and flattening these lists, it becomes easier and more efficient to parse all of the header information.

- Change header offsets into indexes: two specific header fields, *CurrINF* and *CurrHF*, that indicate the current information field and the current hop field to be parsed, respectively, were previously encoded as absolute byte offsets into the packet. In specialized hardware, where data is processed as a stream and random access might not be possible, one would need to keep track of the number of bytes processed up until that moment, which proves to be very inefficient. A more hardware-amenable design consists in changing these fields to represent indexes instead.

- Make all data structures have static sizes: some data structures, like Hop Fields, were variable size, making hardware-based implementations of the parser very challenging. By making all structures fixed size, packet parsing becomes easier and makes it possible to jump ahead to the relevant field with the use of an index.

The main advantage of this implementation is its data plane throughput: due to running in a programmable switch, this implementation is capable of running at 12.8Tbps, with each port running at 100Gbps. Another advantage is its portability: as long as a switch implements the same open-source interface as the Tofino, porting to it is a simple matter of compiling the P4 program.

However, there is a drawback: due to the fact that the cryptography is all precomputed *offline*, it is *impossible* to implement the more secure EPIC L1 protocol. For EPIC L1, the MAC not only protects the path but also the timestamp in the packet. While this protects against brute-force attacks, it also means that it is impossible to precompute all possible MAC values.

Our solution runs the SCION data plane in a programmable switch, in order to achieve terabit performance. We also improve its security properties by implementing the EPIC L1 protocol in the switch. The challenge is to perform the required cryptographic operations *online*

3.4 Summary

In this chapter, we present the 3 main implementation of SCION routers: the reference implementation by SCION Labs, a NetFPGA SUME implementation by Soucková and an Intel Tofino implementation by Joeri and Schutijser. We expose their advantages and disadvantages. In the next chapters, we will introduce our solution, that aims to increase security over previous implementations, while maintaining efficiency.

4 Proposed Solution

Contents

The following chapter describes the implementation of a SCION data plane router, running on Tofino-enabled programmable switches, implemented in the P4 programming language. First, we give an outline of the Tofino architecture and the challenges of implementing cryptographic primitives in this architecture. We then present our solution to these challenges, and the resources used by our implementation of these primitives. Finally, we give an in-depth overview of our data plane design and implementation, describing each of its pipeline's functions.

4.1 Analysis of Programmable Switches and the Intel Tofino

Intel Tofino is a programmable ASIC, developed by Barefoot Networks, capable of very fast packet processing. The data-plane functions of these Tofino devices can be reprogrammed in a domain specific language, P4, while maintaining line-rate speeds (the current generation of Tofino devices is able to process packets at 12.8 Tbps [29]).

The Intel Tofino follows the PISA model (Figure 2.4. This model is comprised of three main steps: the parsing step, where the incoming packet headers are parsed into structures accessible throughout the P4 program; a control pipeline step, where we can encode arbitrary constrained logic and execute it against the parsed packet's structures; and finally a deparser that writes the modified P4 structures back into a packet.

Programmable switches normally come with two computing modules: the Tofino ASIC chip, described above, capable of processing packets at terabit speeds; and a general purpose CPU, responsible for the general management of the Tofino chip (it normally offers a full-blown development environment, capable of compiling and loading P4 programs onto the Tofino chip, modifying P4 tables, process control-level packets, etc).

One of the objectives of this project is to implement on-demand cryptography per-packet. One possible way of doing this would be doing all of the cryptography in the general purpose CPU: the Tofino ASIC processes everything except the cryptographic operations, it then sends a request with the necessary data to the general purpose CPU to do a cryptographic MAC, the CPU computes the MAC and finally sends the information back to the Tofino ASIC, so it can complete the processing of a packet. This approach has several advantages: we can use any well-known cryptographic scheme available, since all of them are made to execute in a general purpose CPU; and it is very easy to implement, since most of the complexity is in ensuring that the communication between the Tofino ASIC and the CPU is done well. Unfortunately, it has a fundamental limitation: it is extremely slow. For reference, the Tofino ASIC and the general purpose CPU communicate through a PCIe interface. This interface adds around 900ns of round-trip delay, making our implementation three orders of magnitude slower [32].

Some programmable switches have an FPGA module, capable of being accessed directly through

P4 with low latency. By reconfiguring this module to work as a dedicated cryptographic processor, we can have all of the advantages of the previous solution, while increasing the processing speeds. In fact, the NetFPGA router, described in section 3.2, is able to process packets at 40 Gbps while performing an AES MAC operation by leveraging the included FPGA.

Unfortunately, this solution also comes with its own set of drawbacks. The first is that portability is impacted: not only does the programmable switch need to be compatible with P4, but it also needs to have the same or an equivalent FPGA attached. The second is complexity: FPGAs are programmed in a Hardware Description Language (HDL), which is difficult to program, requiring domain-specific expertise. Finally, the state of the art implementations [33] [34] of cryptographic algorithms for FPGAs can only output a fraction of the speeds a Tofino chip can get: at most hundreds of Gbps, but not Tbps.

Due to these drawbacks, we decided to implement a cryptographic scheme directly on the data-plane, in P4.

4.2 Fast Cryptography in the data-plane

Unfortunately, in order to be able to process packets at 12.8 Tbps, the computational model of P4 programmable switches is extremely limited. In these pipelines, we are only allowed to do a very small number of single operations (very small number of ALU operations and small number of table accesses, typically limited to one read/write operation per stage). It is therefore very challenging to implement on-demand cryptography, namely running one full MAC operation per packet. In order to execute an expensive cryptographic scheme, like AES, in the data-plane of one of these devices, we would be forced to recirculate packets: pass the packet through the pipeline multiple times until the computation is complete. While this would eventually get us a valid MAC, it suffers from a performance penalty as recirculation limits throughout. For example, the fastest current implementation of AES in the data-plane on these devices can only output less than 11 Gbps [35].

Our solution was to implement a different, but more lightweight cryptographic scheme, that can run in one full pass through the pipeline: the Simplified Even-Mansour scheme.

4.2.1 The Simplified Even-Mansour scheme

The Simplified Even-Mansour (SEM) scheme [36], created by Orr Dunkelman, Nathan Keller, and Adi Shamir, is a one round Even-Mansour scheme where both the pre-whitening key (key applied before the permutation) and post-whitening key (key applied after the permutation) are the same:

$$SEM(M) = (P1(M \oplus K) \oplus K) \tag{4.1}$$

where $P1$ is a N-bit permutation.

We chose this as it is the simplest possible construction of a block cipher which has a formal proof of security. The maximum security upper bound it can offer is $\frac{2^n}{D}$, where D is the number of known ciphertexts. While this is less secure than schemes like AES (AES is able to offer

$$\mathcal{O}(2^126.1)$$

in terms of security [37]), with $n = 128$, we can offer up to 2^{80} security if the attacker has at most D = $2^{40} = 1.0995116e + 12$ known ciphertexts. SCION also offers mechanisms of key rotation and path revalidation, which makes it even harder in practice for attacker to break this encryption scheme.

As for the n-bit permutation, we decided to implement a Substitution-Permutation-Substitution layer where each substitution is different depending on the byte position. Each SBOX was generated using the method described in [38].

For a n-bit permutation to be secure, it needs to have two properties. The first one is Confusion, meaning that each bit of the ciphertext should depend on several parts of the key, obscuring the connections between the two. In substitution–permutation networks, confusion is provided by substitution boxes (SBOX). The second one is Diffusion, meaning that if we change a single bit of the plaintext, then about half of the bits in the ciphertext should change, and similarly, if we change one bit of the ciphertext, then about half of the plaintext bits should change. In substitution–permutation networks, diffusion is provided by permutation boxes (PBOX).

In order to offer as much Confusion and Diffusion as possible, while still maintaining the requirements of fitting in one passthrough of the Tofino's pipeline, we implement multiple, non-linear, dynamic S-BOXes, all independent from one another, which are used in different byte positions. This means that small changes in the permutation layer can get much more amplified, as the SBOX that is applied to each input also changes depending on the byte position.

4.2.2 Implementation in P4

We present a stand-alone implementation of the Simplified Even Mansour scheme, described above, for the Tofino (Figure 4.1. Due to the limitations that the target imposes, in order to get it running at line-rate, we had to be very careful on how we implemented each of the scheme's constructs, in order to maximize data and code parallelism. As such, we made two design decisions.

The first was that all operations are done at the byte level. While the P4 Language Standard demands support for bit-slice operations, due to the design of the Tofino's ALU, operations on data with size different than a multiple of 1 byte are extremely costly. Not only this, but operations on bit-slices can introduce data dependencies. These data dependencies can block possible optimizations and in-

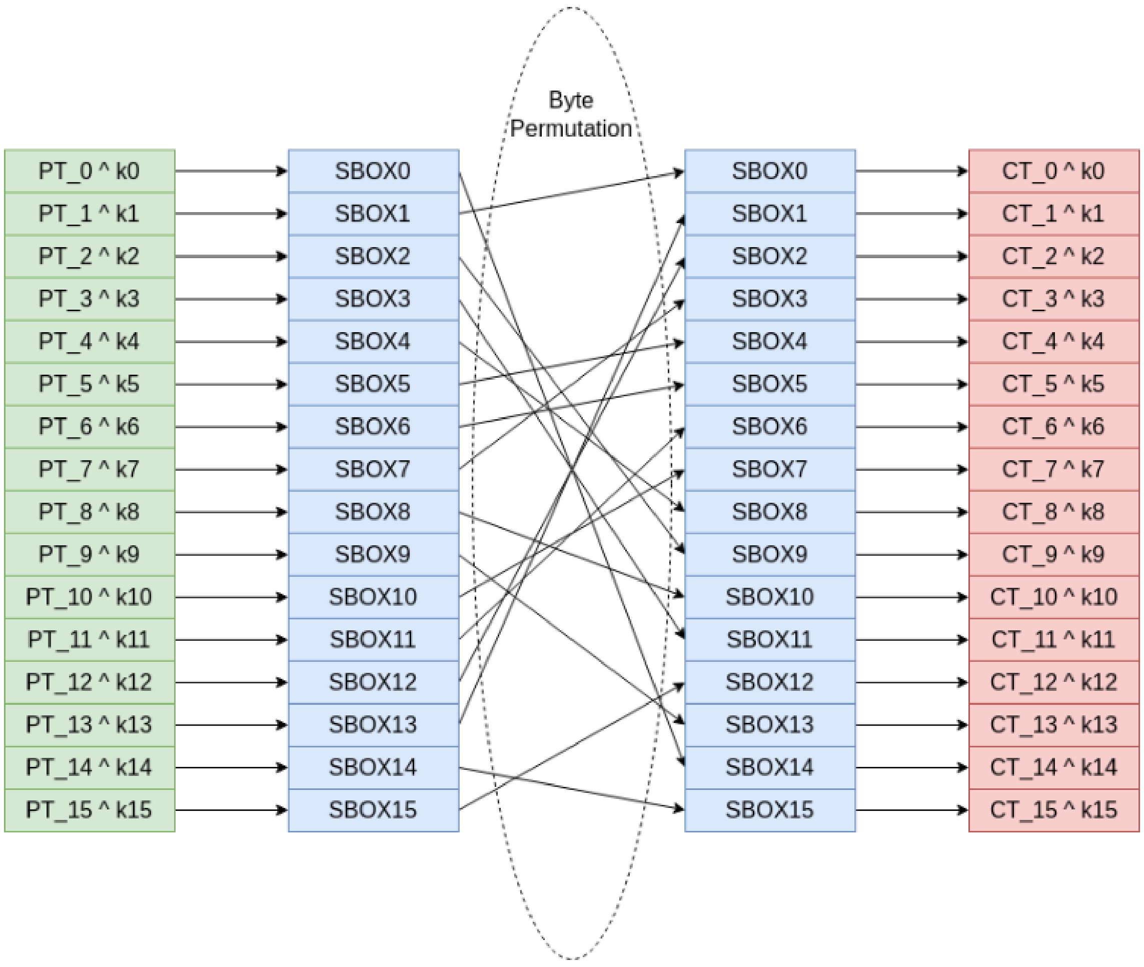


Figure 4.1: The Simplified Even Mansour cipher construct: a SPS layer serves as the n-bit Permutation primitive. We have 16 different SBOXes, one for each byte, applied depending on the byte position. Careful work was put into making sure that each SBOX non-linear (cannot be modeled as a linear function, making them hard to invert), and that no one-hop loops exist $(SBOX(SBOX(x)) \neq x$ for all possible values). The permutation was influenced from the design of the GIFT-128 lightweight cipher [5].

struction reordering from the Tofino compiler, which would lead to an increase on the number of stages necessary due to the need for more computations, resulting in an inefficient implementation. Due to this, we decided to implement all the scheme's operations entirely at the byte level, parsing bigger words into smaller, byte-sized variables. Each SBOX and the permutation layer also act only at the byte-level, allowing the Tofino compiler to further optimize the code at each stage.

The second was that all SBOXes were implemented as statically allocated P4 tables. This allows us to translate this substitution primitive as a main function of the Match/Action units, making it extremely efficient. In order to guarantee even more data-parallelism, we also instantiate multiple copies of each SBOX, so that every table lookup is independent of all other lookups.

With these two methods, we are able to fit this scheme into the full pipe-line of the Tofino programmable switch, and only use 8 stages (of the 12 available in our switch). For a more in-depth view, figure A.2 shows exactly how many resources our scheme uses.

4.3 SCION EPIC L1 entirely in the data-plane

The proposed solution is implemented on top of a programmable switch, with a Tofino ASIC, using the P4 programming language. In order to be fast, the router only deals with well-constructed, valid, packets. Since the main purpose of this work is to show the viability of implementing EPIC L1 in a programmable switch, features like error generation are not implemented: if a packet is invalid, for whatever reason (bad parsing, wrong MAC, invalid path, etc.), the packet is sent directly to the general-purpose CPU where it can then be dealt with accordingly.

The router is divided into 3 parts: the packet parser, which parses the packet (according to Figure 2.3i) into P4 data structures; the MAC generator, that takes the data from the packet, generates a MAC and compares it with the MAC from the packet to authenticate it; and finally the packet header fixer, that updates the packet header values so the next-hop can process the packet correctly, according to the EPIC protocol. These last two are fully implemented in the Ingress part of the Tofino ASIC.

An overview of the protocol implemented is as follows:

1. Parse the incoming packet according to the header structure described in Figure 2.3i. In case of error, send to the CPU.

2. Extract and parse both the current hop field and the previous hop field (Figure 2.3f). If we are the originating hop field only parse this one. Parsing the previous hop field is necessary since its MAC will be used in the data block for our own MAC verification.

3. Verify that the ingress interface in in the packet corresponds to the port it originated from. The path exploration phase records which ASes are connected to each of our ports. If we receive a packet from an AS connected to a different port than recorded, send to the CPU.

4. With all of the information parsed, assemble the data block (Figures 2.3g and 2.3h) and compute the MAC over it. Check if the computed MAC matches the MAC on our packet hop field. If it does not match (meaning that the MAC in the packet is incorrect), send to the CPU

5. If the destination AS is our AS, then forward it correct port, accordingly to intra-AS rules stored in the switch's tables.

6. If the destination AS is not our AS, we are an intermediate hop. We update the packet's Current Info ($CurrInf$) field and Current Hop ($CurrHF$) field in the packet, so the next AS can correctly parse the fields related to it.

7. Forward the updated packet to the next AS, identified through the ingress interface in the packet.

4.3.1 Packet Parser

The packet parser is implemented as a state machine, where each state parses specific header fields. In order to be as compatible as possible with the SCION protocol, we decided to base the header format of our packets on the SCION Header Specification [39], reusing as much as we can of the header structure. Thankfully, a lot of previous work has been done in making sure that the SCION headers can be efficiently parsed by these programmable switches (as mentioned, the work in [4] redesigned the SCION header structure, so it would be possible to implement an efficient parser in P4).

The trickiest part of implementing the parser was parsing the $Info$ and $HopField$ headers, since their number is variable and depends on previous header information. While P4 has support for variable size fields, high speed P4 targets as the Tofino switch do not support them given its complexity, and the requirement of guaranteeing line rate performance. For parsing the $Info$ headers, we used multiple different states in the parser state machine, one for each possible number of fields (3 states in total). However, for the $HopField$ headers, this is not a suitable solution. The maximum number of $HopFields$ in a header is limited to 64. Adding 64 different stages only for parsing these structures would make the parser inefficient and clutter the code. Due to this, we decided to exploit a design decision of SCION. In SCION packets, we only need to access, at most, two $HopField$ headers: the current header, and, if it exists, the last header before the current one. This is because these two headers contain all of the information necessary to perform the MAC computation. While we must include the full hop field information in every packet we process, we don't need to parse them all into individual structures, since we won't be accessing their data. Because of this, we can parse all of the unimportant $HopField$ headers into large-size buffers, and jump over them. We only need to be careful to correctly parse the two above-described headers into their correct structures.

4.3.1.A Incompatibilities with SCION

While, for EPIC L0, no change to the SCION's header structure was necessary, the same is not true for EPIC L1. Since EPIC L1 needs an extra field for its MAC calculation, namely a unique timestamp for each packet, it was necessary to add this extra field. We decided to add it to the $PathMetaHeader$ structure (since we considered a packet timestamp part of a packet's metadata information), as a 32-bit UNIX timestamp. This, unfortunately, makes our version of EPIC L1 incompatible with the SCION protocol and tooling.

4.3.2 MAC Generation and Authentication

After parsing the packet into P4 structures, we do a preliminary check to validate the $ConsIngress$ and the $ConsEgress$ interface IDs. These IDs act as path identifiers, allowing each hop, in the path-exploration process, to identify from which a path originated from and from which port a path continued on. This check is made in order to make sure that the packet was received and will be sent through the port in the switch associated with each ID.

Afterwards, we do the MAC authentication. The MAC authentication consists of generating a full MAC, with the Hop Authenticator as a key, based on a 128-bit block, constructed with data present in the packet. Figures 2.3g and 2.3h show the construction of the input data block to the MAC function, for EPIC L0 and EPIC L1 respectively. Each router computes the MAC for its own hop, and compares with the MAC value present in its Hop Field header. As for the MAC construction, we do a full round of the Simplified Even-Mansour scheme, described in 4.2.1, on a 128-bit block.

The main difference between MAC construction between EPIC L0/previous SCION implementations and EPIC L1 is the presence of a new packet timestamp, generated when this packet is sent. This timestamp, when included in the data block used by the MAC function, acts as a freshness value, assuring that MACs are not only dependent on the path a packet will take but are also dependent on this timestamp. This means that an attacker can not brute force a valid MAC for one specific path, since if the timestamp in the packet is too old, a hop can simply compare the timestamp in the packet with the timestamp of the packet's arrival (present in the Tofino's metadata structures) and simply discard it if it is too old. This forces the attacker to update the timestamp in the packet, which in turn changes the MAC value.

The EPIC paper [11] suggests small MAC sizes, truncating the full computation to 3 bytes per hop, in order to diminish the necessary overhead in packet header sizes. However, in order to have stronger guarantees against brute-force attacks, the original SCION header specification [39] mandates MAC sizes of 6 bytes. In order to be more compatible with the SCION protocol, we only support 6-byte MAC fields.

4.3.3 Update Packet Header fields

After the MAC Authentication procedure, it is necessary to update the packet header values (in specific, both the $CurrInf$ and/or $CurrHF$ fields), so the next switch in the chain can correctly find its own $InfoField$ and $HopField$ header values. We perform this computation last, since in case of an error in the parsing or in the authentication procedure, we want to be able to return the packet as it came through the ingress, before any fix-ups are performed.

4.4 Summary

In this chapter, we have presented our proposed design of a SCION router, implementing EPIC L1 as its data-plane protocol. We analysed the architecture of our target, a programmable switch, and showed that performing cryptographic operations like AES at terabit speeds on these devices is currently not possible. By understanding the strengths and weaknesses of these devices, we identified and implemented a different cryptographic scheme, the Simplified Even-Mansour, which is capable of running one full execution while maintaining the desired speed. We present an implementation of EPIC L1 running entirely on the data-plane of these devices and explain how an EPIC packet is parsed through our solution. In the next chapters, we will evaluate our solution against the state of the art, both in terms of throughput and resource usage.

5

Evaluation

Contents

In this chapter, we present the evaluation of our solution. We compare our own implementation of the EPIC L0 protocol against our implementation of the EPIC L1 protocol, which was the main contribution of this thesis, and show that there are no meaningful differences in performance between the two, and that both are able to process packets at line-rate. In addition we compare our solution to the previous state of the art, and show that we are able to maintain or surpass the throughput of previous solutions, while providing the stronger security guarantees that EPIC L1 offers.

5.1 Comparison between EPIC L0 and EPIC L1

In this section, we compare our implementation of EPIC L0 with our implementation of EPIC L1, both targeting a programmable switch backed by an Intel Tofino. We evaluate them both on resource usage of the Tofino chip (how many stages each use, how many resources they use on each stage through the PISA pipeline), and on throughput. We conclude by showing that resource usage of both implementations is similar, with both are able to achieve terabit speeds on existing hardware.

5.1.1 Resource usage

We start by asking one question: do we need more resources to implement EPIC L1 versus EPIC L0. Due to the additional security guarantees that EPIC L1 offers (freshness), our expectation was that EPIC L1 would consume more resources than EPIC L0. However, this is not the case: our implementation of EPIC L1 has very similar resource usage than EPIC L0 (see Table 5.1). They use the same number of stages (8 out of 12), so they will have the same latency for every processed packet. Difference between implementations in total of resources used is minuscule, making the overall resource usage near indistinguishable.

5.1.2 Throughput

Both implementations were tested on a APS Advanced Programmable Switch BF2556X-1T, powered by a Barefoot BFN-T10-032D-020 Tofino 2.0T chip. The controller was programmed and results were collected using the P4Runtime API, running on the Ubuntu 20.04 operating system.

We test throughput by utilizing the packet generation feature included in the Tofino chip and checking if we are able to process it all. We show results (Figure 5.1b) for packets with different payload sizes and for packets with smaller/bigger number of hop fields in their paths.

The compilation of both EPIC L0/L1 implementations was successful on the Tofino compiler, which means both solutions can run at terabit speeds, with enough servers (64) in our testbed (each server connected to one 100Gbps port). However, we do not have such a setup. As such, we utilized the packet

Resource	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	Total
Action Data Bus Bytes	0.78%	0%	1.56%	2.34%	0%	0%	0.78%	0.78%	0%	0%	0%	0%	0.52%
Exact Match Input Xbar	-0.78%	-0.78%	0.78%	0.78%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Gateway	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Hash Bit	-2.40%	-2.40%	2.40%	2.40%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Logical Table ID	-6.25%	-6.25%	6.25%	6.25%	0%	0%	0%	0%	0%	0%	0%	0%	0%
SRAM	-1.25%	-1.25%	1.25%	1.25%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Stash	-6.25%	-6.25%	6.25%	6.25%	0%	0%	0%	0%	0%	0%	0%	0%	0%
VLIW Instruction	0%	-4.55%	4.55%	6.25%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Exact Match Search Bus	0%	-6.25%	6.25%	6.25%	0%	0%	0%	0%	0%	0%	0%	0%	0.52%
Exact Match Result Bus	-6.25%	-6.25%	6.25%	6.25%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Ternary Result Bus	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

Table 5.1: Relative resource usage difference between EPIC L0 and EPIC L1, in each Stage and in Total. Results taken from Barefoot P4 Insight tool (Figure A.1). Differences between implementations are almost zero, making them virtually identical. Variation in specific stages where the Total becomes 0% can be attributed by small, irrelevant deviations by the compiler.

generation feature (capable of around 50Gbps with small packets) in order to test our implementations. We see that the size of payload and hop fields leads to the same line rate result, which shows that we have indeed achieve line rate.

5.2 EPIC L0/L1 against the state of the art

In this section, we compare our implementations of EPIC L0 and EPIC L1 with the previous hardware-based implementations described in Chapter 4. We will compare them mainly on two metrics: the security guarantees that each solution provides, and the performance, in terms of throughput, of each solution. We will show that our solution is capable of either matching or exceeding the packet processing performance of previous work, while delivering better security guarantees.

5.2.1 Security Guarantees

Both Soucková's [3] and Joeri and Schutijser's [4] solutions implement a version of the EPIC L0 data-plane protocol. The main difference between them is the type of cryptography operations. Soucková's NetFPGA solution performs the cryptographic MAC computations online, once per-packet. By contrast, Joeri and Schutijser's Tofino-based solution precomputes all valid MACs, stores them on a table and,

Number of Hop Fields in path	Payload (bytes)	Gbps
1	0	49.98
1	128	49.74
1	512	49.92
32	0	49.87
32	128	49.73
32	512	49.87
64	0	49.96
64	128	49.99
64	512	49.83

(a) EPIC L0 Throughput Evaluation

Number of Hop Fields in path	Payload (bytes)	Gbps
1	0	49.82
1	128	49.96
1	512	49.98
32	0	49.78
32	128	49.89
32	512	49.74
64	0	49.97
64	128	49.93
64	512	49.91

(b) EPIC L1 Throughput Evaluation

Figure 5.1: EPIC Throughput Evaluation: Packet Generation ran at 50Gbps; we evaluated each P4 implementation multiple times, by adding a P4 packet counter at the end of its processing stage. We show no practical difference between implementations. Relative differences between the theoretical maximum of 50Gbps can be attributed by slight deviations in the user-space clock used to get the measurements.

in order to check the packet MAC's validity, it performs a table lookup to see if the MAC present on the packet is also present in the table.

Both our EPIC L0 and EPIC L1 implementations do online on-demand cryptographic operations per packet. This is specially important for the EPIC L1 implementation: since this protocol includes a freshness guarantee for each packet, by adding a packet's timestamp to each packet's header, it is not compatible with Joeri and Schutijser's table-based approach, since it would not be viable to do all precomputations ahead of time. Table 5.2 summarizes this information.

Implementation	Protocol Implemented	Type of Cryptography implemented	Achieves line-rate	Freshness
Soucková's [3]	EPIC L0	On-demand	No	No
Joeri and Schutijser's [4]	EPIC L0	Table-based	Yes	No
Our EPIC L0 implementation	EPIC L0	On-demand	Yes	No
Our EPIC L1 implementation	EPIC L1	On-demand	Yes	Yes

Table 5.2: Security properties of each implementation

5.2.2 Performance

The main goal of our thesis was to implement the EPIC L1 protocol in a programmable switch, running at line-rate. We have shown that we have achieved this goal in Section 5.1.2. We compare our results with the state of the art of Soucková's [3] and Joeri and Schutijser's [4] works. These results are summarized in Table 5.3.

Soucková's implementation target was an NetFPGA, a device capable of a maximum throughput on all of its ports of 40Gbps (4 ports of 10Gbps). Their implementation was not able to fully fit onto this device, as explained in Section 3.2. However, they were able to perform throughput measurements on a partial design, while accounting for different frame sizes (1500B and 115B frames). We compare our implementation measurements with these.

Joeri and Schutijser's [4] implementation was able to run at line-rate, while targeting a Tofino device, the same type of device as our solution. They performed measurements on a single 100Gbps port of the device, which is sufficient to show that the implementation runs at line-rate (a 128-port Tofino ASIC chip is capable of processing up to 12.8Tbps). We run similar measurements on our implementation as well, in order to allow better comparisons with this solution.

Implementation	Protocol Implemented	Max throughput on target device	Achieves line-rate
Soucková's [3]	EPIC L0	$<$ 40Gbps*	No
Joeri and Schutijser's [4]	EPIC L0	12.8Tbps	Yes
Our EPIC L0 implementation	EPIC L0	12.8Tbps	Yes
Our EPIC L1 implementation	EPIC L1	12.8Tbps	Yes

Table 5.3: Maximum throughput of each implementation. Note*: since Soucková's full implementation was not capable of fitting in the target, this is the advertised theoretical maximum, tested only on their partial design.

Our implementation is able to achieve the same speeds as Joeri and Schutijser's implementation and over two orders of magnitude faster than Soucková's theoretical maximum. However, like Soucková's implementation and unlike Joeri and Schutijser's, we do all of the cryptographic operations per-packet. This means our implementation of EPIC L1 is able to run at terabit speeds, while providing increased security guarantees, in the form of MAC freshness.

5.3 Summary

We evaluated our implementations of EPIC L0 and EPIC L1, both in resource usage in our target, and in performance, by measuring throughput. We showed that both are able to achieve terabit speeds, and that the EPIC L1 implementation, that offers more security guarantees, has similar resource usage and overall latency to our EPIC L0 implementation. We also evaluate our implementation with the state of the art, and show that we are able to either match or exceed them in performance, while still increasing security.

6 Conclusion and Future Work

Contents

6.1 Conclusion

In this thesis, we made the case for the need of a new Internet architecture, that can overcome the security problems present today, while still being able to deliver the level of service we have come to expect. SCION is the first, new, production-ready Internet architecture, with a focus on security while still remaining efficient enough for today's use-cases. We have implemented a SCION-compatible router data plane in a programmable switch, powered by an Intel Tofino ASIC, with increased performance of over two orders of magnitude over the previous state of the art. We have implemented both EPIC L0 and EPIC L1. This last one provides more security guarantees than SCION, by being more resistant against brute force attacks. Both our implementation were written in P4, in a cross-platform manner, targeting any PISA-compatible device.

One of the major challenges in this thesis was the implementation of cryptographic primitives in our target, that were able to run at line-rate. In order to do this, we adapted the Simplified Even-Mansour cryptographic scheme and ported it to our target, since previous implementations of other cryptographic schemes were insufficient to get the throughput that we required. The permutation used on this implementation was based on a Substitution-Permutation-Substitution layer, where all substitution were performed by non-linear SBOXes, mapped to P4 tables for maximum efficiency. The permutation network was adapted from previous lightweight ciphers like GIFT-128 [5]. Overall, we were able to implement a full cryptographically-secure scheme in 8 stages of the device.

Finally, we compared our implementations with the previous state-of-the-art solutions, showing that we were able to get comparable or faster throughput speeds, while providing improved security guarantees.

Our results show that both the EPIC family of protocols, and the SCION Internet architecture are viable solutions at high speeds, since they are suitable for hardware-backed implementations. As such, both can be considered for world wide large-scale deployments. We have also shown that programmable network devices, programmed with P4, can be use to implement relatively complex protocols while still performing at line-rate. This makes these devices extremely useful tools that can simplify the development and implementation of network protocols for a variety of different targets, while still providing enough performance for real-world cases.

6.2 Future Work

Designing and implementing secure and efficient protocols, targeting these new programmable network devices, is a very promising area to explore. This allows us to solve long-standing architectural problems in our current infrastructure, while still being able to process great amounts of traffic, which is extremely important in maintaining the level of service we have come to expect. Some topics that could improve our work presented in this thesis include:

- Implement the EPIC L2 and EPIC L3 protocols in modern programmable network devices: after implementing EPIC L0 and EPIC L1, the next logical step would be to implement the most secure siblings of this family of protocols. Further analysis can be made in order to find out how much each added security property contributes in the overall performance of each implementation.
- Design and implement better and safer cryptographic primitives in the data-plane: while the Simplified Even-Mansour scheme may be considered sufficient for various use cases, very little work has been done on developing cryptographic primitives for PISA devices. Further work can be done either by improving, both in efficiency and security, the current Simplified Even-Mansour primitive, or by creating a new cryptographic primitive, custom-made for this new architecture.

Bibliography

[1] A. Perrig, P. Szalachowski, R. M. Reischuk, and L. Chuat, *SCION: a secure Internet architecture*. Springer, 2017.

[2] M. Budiu and C. Dodd, “The p416 programming language,” *ACM SIGOPS Operating Systems Review*, vol. 51, no. 1, pp. 5–14, 2017.

[3] K. Soucková, “Fpga-based line-rate packet forwarding for the scion future internet architecture,” Master’s thesis, ETH Zurich, 2019.

[4] J. de Ruiter and C. Schutijser, “Next-generation internet at terabit speed: Scion in p4,” in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, 2021, pp. 119–125.

[5] S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, “Gift: a small present,” in *International conference on cryptographic hardware and embedded systems*. Springer, 2017, pp. 321–345.

[6] J. Postel, “Internet protocol,” Internet Requests for Comments, IETF, RFC 791, 9 1981. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc791

[7] D. Wagner, D. Kopp, M. Wichtlhuber, C. Dietzel, O. Hohlfeld, G. Smaragdakis, and A. Feldmann, “United we stand: Collaborative detection and mitigation of amplification ddos attacks at scale,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 970–987.

[8] T. L. Y. Rekhter, “A border gateway protocol 4 (bgp-4),” Internet Requests for Comments, IETF, RFC 1654, 7 1994. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1654

[9] R. A. R. Bush, “The resource public key infrastructure (rpki) to router protocol,” Internet Requests for Comments, IETF, RFC 6810, 1 2013. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6810

[10] K. S. M. Lepinski, "Bgpsec protocol specification," Internet Requests for Comments, IETF, RFC 8205, 9 2017. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8205

[11] M. Legner, T. Klenze, M. Wyss, C. Sprenger, and A. Perrig, "Epic: Every packet is checked in the data plane of a path-aware internet," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 541–558.

[12] A. Davidson, M. Frei, M. Gartner, H. Haddadi, J. S. Nieto, A. Perrig, P. Winter, and F. Wirz, "Tango or square dance? how tightly should we integrate network functionality in browsers?" *arXiv preprint arXiv:2210.04791*, 2022.

[13] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[14] S. M. Bellovin, "Security problems in the tcp/ip protocol suite," *ACM SIGCOMM Computer Communication Review*, vol. 19, no. 2, pp. 32–48, 1989.

[15] K. Butler, T. R. Farley, P. McDaniel, and J. Rexford, "A survey of bgp security issues and solutions," *Proceedings of the IEEE*, vol. 98, no. 1, pp. 100–122, 2009.

[16] N. Kushman, S. Kandula, and D. Katabi, "Can you hear me now?! it must be bgp," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 75–84, 2007.

[17] N. Ripe, "Youtube hijacking a ripe ncc ris case study," *http://www. ripe. net/news/study-youtube-hijacking. html*, 2008.

[18] T. G. Griffin, F. B. Shepherd, and G. Wilfong, "The stable paths problem and interdomain routing," *IEEE/ACM Transactions On Networking*, vol. 10, no. 2, pp. 232–243, 2002.

[19] T. Griffin and G. Huston, "Bgp wedgies," Internet Requests for Comments, IETF, RFC 4264, 2005. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4264

[20] Q. Li, J. Liu, Y.-C. Hu, M. Xu, and J. Wu, "Bgp with bgpsec: Attacks and countermeasures," *IEEE Network*, vol. 33, no. 4, pp. 194–200, 2018.

[21] Q. Li, Y.-C. Hu, and X. Zhang, "Even rockets cannot make pigs fly sustainably: Can bgp be secured with bgpsec," in *Workshop SENT'14, 23 February 2014, San Diego, USA, Copyright 2014 Internet Society: Proceedings*. Internet Society, 2014.

[22] R. Lychev, S. Goldberg, and M. Schapira, "Bgp security in partial deployment," *Is the juice worth the squeeze*, 2013.

[23] D. Cooper, E. Heilman, K. Brogle, L. Reyzin, and S. Goldberg, “On the risk of misbehaving rpki authorities,” in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, 2013, pp. 1–7.

[24] J. Rexford and C. Dovrolis, “Future internet architecture: clean-slate versus evolutionary research,” *Communications of the ACM*, vol. 53, no. 9, pp. 36–40, 2010.

[25] D. Fisher, “A look behind the future internet architectures efforts,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 45–49, 2014.

[26] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, “Named data networking,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[27] A. Venkataramani, J. F. Kurose, D. Raychaudhuri, K. Nagaraja, M. Mao, and S. Banerjee, “Mobilityfirst: a mobility-centric and trustworthy internet architecture,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 74–80, 2014.

[28] D. Naylor, M. K. Mukerjee, P. Agyapong, R. Grandl, R. Kang, M. Machado, S. Brown, C. Doucette, H.-C. Hsiao, D. Han *et al.*, “Xia: Architecting a more trustworthy and evolvable internet,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 50–57, 2014.

[29] C. K. Anurag Agrawal. Intel tofino2 – a 12.9tbps p4-programmable ethernet switch. Intel Corporation. [Online]. Available: https://hc32.hotchips.org/assets/program/conference/day2/HotChips2020_Networking_Tofino.pdf

[30] SCIONLab, “Scion,” https://github.com/scionproto/scion.

[31] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, “Netfpga sume: Toward 100 gbps as research commodity,” *IEEE micro*, vol. 34, no. 5, pp. 32–41, 2014.

[32] R. Neugebauer, G. Antichi, J. F. Zazo, Y. Audzevich, S. López-Buedo, and A. W. Moore, “Understanding pcie performance for end host networking,” in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 327–341.

[33] L. Henzen and W. Fichtner, “Fpga parallel-pipelined aes-gcm core for 100g ethernet applications,” in *2010 Proceedings of ESSCIRC*. IEEE, 2010, pp. 202–205.

[34] S. Ghosh, L. S. Kida, S. J. Desai, and R. Lal, “A¿ 100 gbps inline aes-gcm hardware engine and protected dma transfers between sgx enclave and fpga accelerator device,” *Cryptology ePrint Archive*, 2020.

[35] X. Chen, “Implementing aes encryption on programmable switches via scrambled lookup tables,” in *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, 2020, pp. 8–14.

[36] O. Dunkelman, N. Keller, and A. Shamir, “Minimalism in cryptography: The even-mansour scheme revisited,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 336–354.

[37] A. Bogdanov, D. Khovratovich, and C. Rechberger, “Biclique cryptanalysis of the full aes,” in *International conference on the theory and application of cryptology and information security*. Springer, 2011, pp. 344–371.

[38] H. Zhu, X. Tong, Z. Wang, and J. Ma, “A novel method of dynamic s-box design based on combined chaotic map and fitness function,” *Multimedia tools and applications*, vol. 79, no. 17, pp. 12 329–12 347, 2020.

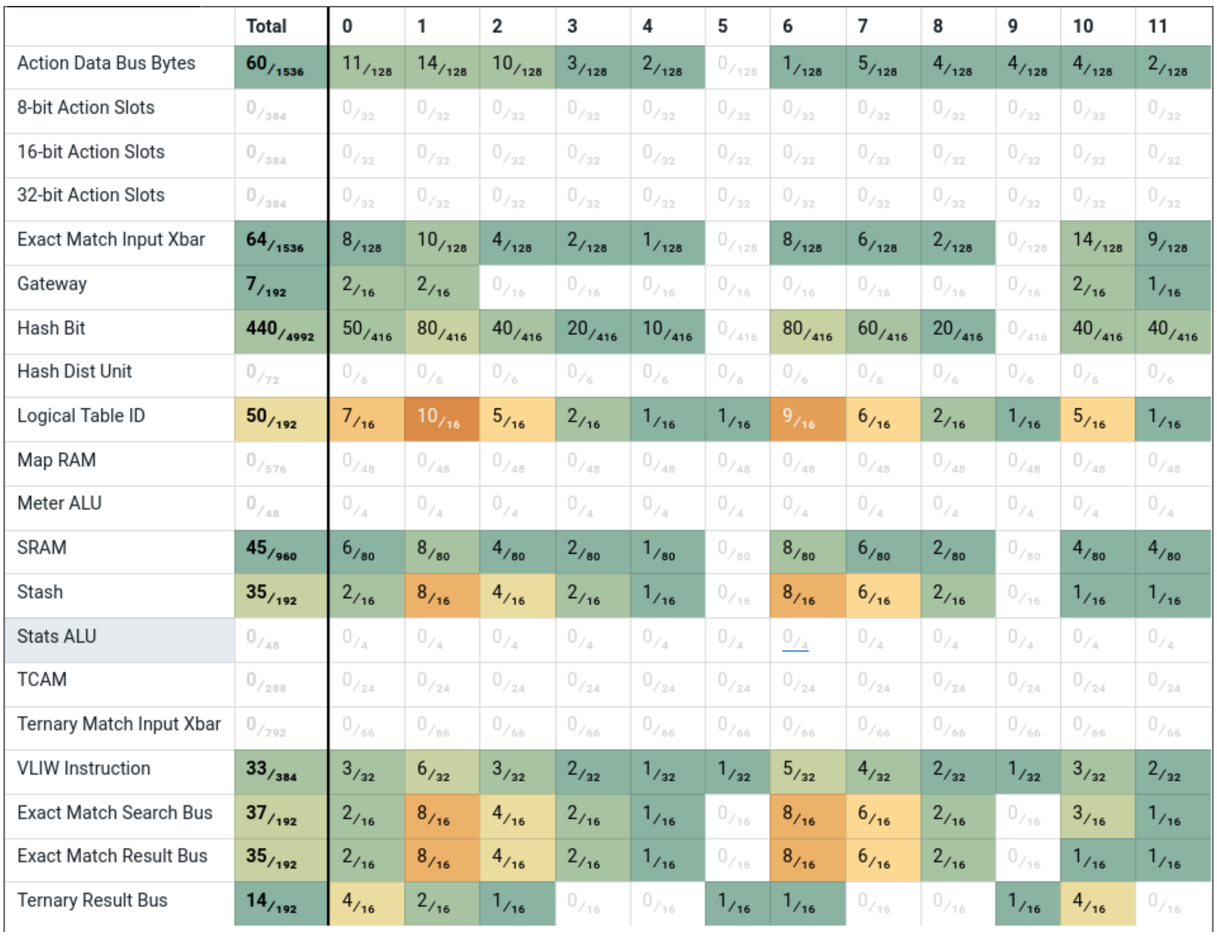
[39] *SCION Header Specification*, Anapaya Systems, 2021.

A

P4 Insight Results

This appendix has the tables concerning the results extracted from the Barefoot P4 Insight tool.

Figure A.1: EPIC L0 and L1 resource consumption. Due to the use of on-demand cryptography in the EPIC L0 implementation, the difference in resource usage between both implementation is negligible (same number of stages, and similar resources used in each stage)

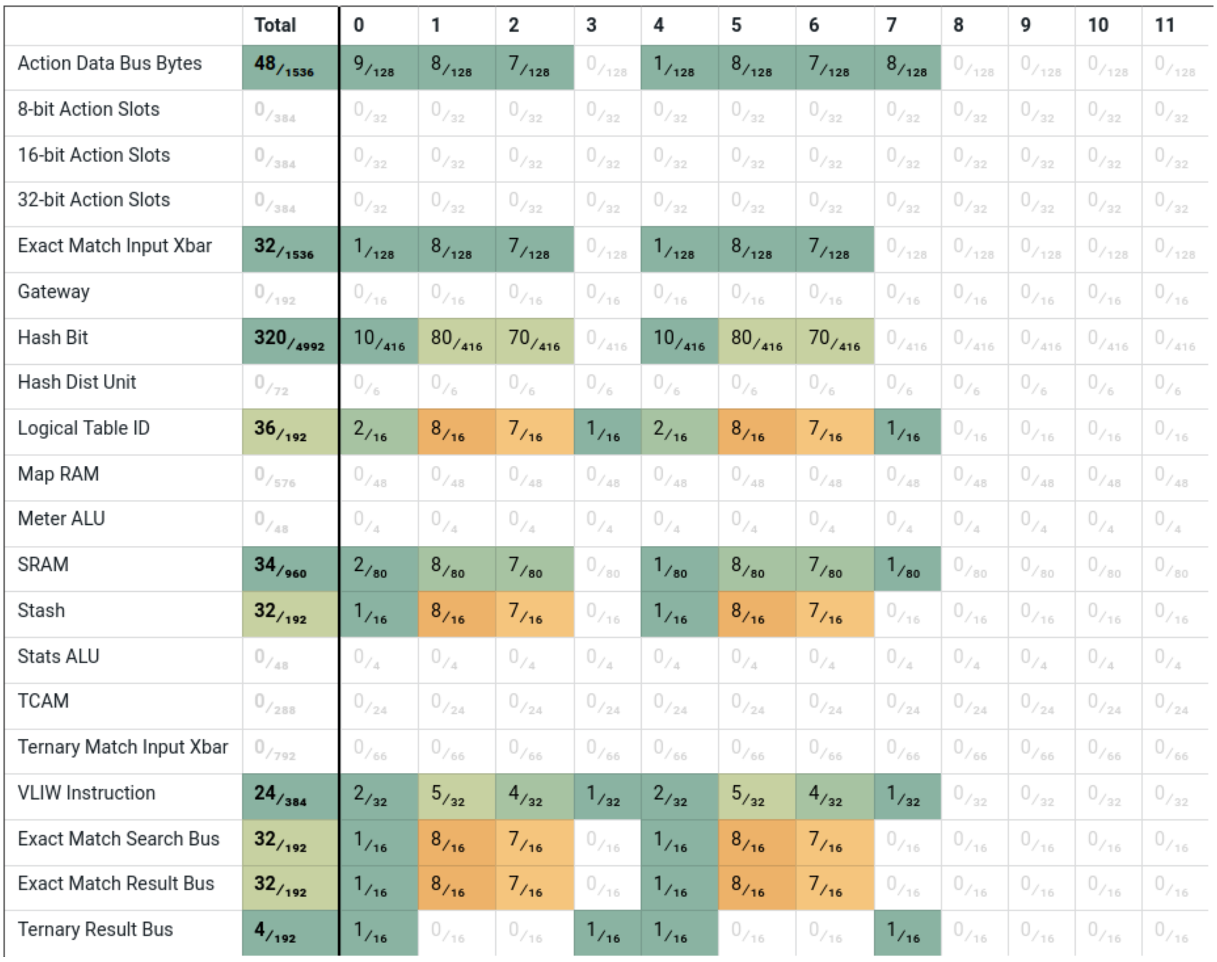


	Total	0	1	2	3	4	5	6	7	8	9	10	11
Action Data Bus Bytes	60/1536	11/128	14/128	10/128	3/128	2/128	0/128	1/128	5/128	4/128	4/128	4/128	2/128
8-bit Action Slots	0/384	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32
16-bit Action Slots	0/384	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32
32-bit Action Slots	0/384	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32
Exact Match Input Xbar	64/1536	8/128	10/128	4/128	2/128	1/128	0/128	8/128	6/128	2/128	0/128	14/128	9/128
Gateway	7/192	2/16	2/16	0/16	0/16	0/16	0/16	0/16	0/16	0/16	0/16	2/16	1/16
Hash Bit	440/4992	50/416	80/416	40/416	20/416	10/416	0/416	80/416	60/416	20/416	0/416	40/416	40/416
Hash Dist Unit	0/72	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6
Logical Table ID	50/192	7/16	10/16	5/16	2/16	1/16	1/16	9/16	6/16	2/16	1/16	5/16	1/16
Map RAM	0/576	0/48	0/48	0/48	0/48	0/48	0/48	0/48	0/48	0/48	0/48	0/48	0/48
Meter ALU	0/48	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4
SRAM	45/960	6/80	8/80	4/80	2/80	1/80	0/80	8/80	6/80	2/80	0/80	4/80	4/80
Stash	35/192	2/16	8/16	4/16	2/16	1/16	0/16	8/16	6/16	2/16	0/16	1/16	1/16
Stats ALU	0/48	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4
TCAM	0/288	0/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24
Ternary Match Input Xbar	0/792	0/66	0/66	0/66	0/66	0/66	0/66	0/66	0/66	0/66	0/66	0/66	0/66
VLIW Instruction	33/384	3/32	6/32	3/32	2/32	1/32	1/32	5/32	4/32	2/32	1/32	3/32	2/32
Exact Match Search Bus	37/192	2/16	8/16	4/16	2/16	1/16	0/16	8/16	6/16	2/16	0/16	3/16	1/16
Exact Match Result Bus	35/192	2/16	8/16	4/16	2/16	1/16	0/16	8/16	6/16	2/16	0/16	1/16	1/16
Ternary Result Bus	14/192	4/16	2/16	1/16	0/16	0/16	1/16	1/16	0/16	0/16	1/16	4/16	0/16

(a) EPIC L0 Resource Consumption

	Total	0	1	2	3	4	5	6	7	8	9	10	11
Action Data Bus Bytes	**68/1536**	12/128	14/128	12/128	6/128	2/128	0/128	2/128	6/128	4/128	4/128	4/128	2/128
8-bit Action Slots	0/384	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32
16-bit Action Slots	0/384	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32
32-bit Action Slots	0/384	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32
Exact Match Input Xbar	**64/1536**	7/128	9/128	5/128	3/128	1/128	0/128	8/128	6/128	2/128	0/128	14/128	9/128
Gateway	**7/192**	2/16	2/16	0/16	0/16	0/16	0/16	0/16	0/16	0/16	0/16	2/16	1/16
Hash Bit	**440/4992**	40/416	70/416	50/416	30/416	10/416	0/416	80/416	60/416	20/416	0/416	40/416	40/416
Hash Dist Unit	0/72	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6
Logical Table ID	**50/192**	6/16	9/16	6/16	3/16	1/16	1/16	9/16	6/16	2/16	1/16	5/16	1/16
Map RAM	0/576	0/48	0/48	0/48	0/48	0/48	0/48	0/48	0/48	0/48	0/48	0/48	0/48
Meter ALU	0/48	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4
SRAM	**45/960**	5/80	7/80	5/80	3/80	1/80	0/80	8/80	6/80	2/80	0/80	4/80	4/80
Stash	**35/192**	1/16	7/16	5/16	3/16	1/16	0/16	8/16	6/16	2/16	0/16	1/16	1/16
Stats ALU	0/48	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4
TCAM	0/288	0/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24
Ternary Match Input Xbar	0/792	0/66	0/66	0/66	0/66	0/66	0/66	0/66	0/66	0/66	0/66	0/66	0/66
VLIW Instruction	**33/384**	3/32	5/32	4/32	2/32	1/32	1/32	5/32	4/32	2/32	1/32	3/32	2/32
Exact Match Search Bus	**38/192**	2/16	7/16	5/16	3/16	1/16	0/16	8/16	6/16	2/16	0/16	3/16	1/16
Exact Match Result Bus	**35/192**	1/16	7/16	5/16	3/16	1/16	0/16	8/16	6/16	2/16	0/16	1/16	1/16
Ternary Result Bus	**14/192**	4/16	2/16	1/16	0/16	0/16	1/16	1/16	0/16	0/16	1/16	4/16	0/16

(b) EPIC L1 Resource Consumption



	Total	0	1	2	3	4	5	6	7	8	9	10	11
Action Data Bus Bytes	**48**/1536	9/128	8/128	7/128	0/128	1/128	8/128	7/128	8/128	0/128	0/128	0/128	0/128
8-bit Action Slots	0/384	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32
16-bit Action Slots	0/384	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32
32-bit Action Slots	0/384	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32	0/32
Exact Match Input Xbar	**32**/1536	1/128	8/128	7/128	0/128	1/128	8/128	7/128	0/128	0/128	0/128	0/128	0/128
Gateway	0/192	0/16	0/16	0/16	0/16	0/16	0/16	0/16	0/16	0/16	0/16	0/16	0/16
Hash Bit	**320**/4992	10/416	80/416	70/416	0/416	10/416	80/416	70/416	0/416	0/416	0/416	0/416	0/416
Hash Dist Unit	0/72	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6
Logical Table ID	**36**/192	2/16	8/16	7/16	1/16	2/16	8/16	7/16	1/16	0/16	0/16	0/16	0/16
Map RAM	0/576	0/48	0/48	0/48	0/48	0/48	0/48	0/48	0/48	0/48	0/48	0/48	0/48
Meter ALU	0/48	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4
SRAM	**34**/960	2/80	8/80	7/80	0/80	1/80	8/80	7/80	1/80	0/80	0/80	0/80	0/80
Stash	**32**/192	1/16	8/16	7/16	0/16	1/16	8/16	7/16	0/16	0/16	0/16	0/16	0/16
Stats ALU	0/48	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4
TCAM	0/288	0/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24
Ternary Match Input Xbar	0/792	0/66	0/66	0/66	0/66	0/66	0/66	0/66	0/66	0/66	0/66	0/66	0/66
VLIW Instruction	**24**/384	2/32	5/32	4/32	1/32	2/32	5/32	4/32	1/32	0/32	0/32	0/32	0/32
Exact Match Search Bus	**32**/192	1/16	8/16	7/16	0/16	1/16	8/16	7/16	0/16	0/16	0/16	0/16	0/16
Exact Match Result Bus	**32**/192	1/16	8/16	7/16	0/16	1/16	8/16	7/16	0/16	0/16	0/16	0/16	0/16
Ternary Result Bus	**4**/192	1/16	0/16	0/16	1/16	1/16	0/16	0/16	1/16	0/16	0/16	0/16	0/16

Figure A.2: Resource utilization of a standalone implementation of the Simplified Even Mansour cipher, given by the P4 Insight tool. Through clever data parsing, knowledge of Tofino's ALU, and multiple table instantion per use, we are able to make it fit onto only 8 stages. There is enough space for other applications