

Myriarch: enabling the coexistence of a myriad of Internets

GUILHERME RIBEIRO, Instituto Superior Técnico, Portugal

The Internet's success has highlighted its intrinsic issues, such as a lack of security and limited extensibility, making innovation difficult. Over the past two decades, researchers have proposed clean-slate Internet designs to address these problems. However, these designs have not been globally deployed and tested, making their efficacy uncertain, and a complete infrastructure replacement seems unlikely.

An alternative to clean-slate designs is developing Inter-Architecture Frameworks that allow multiple architectures to coexist with the Internet. The two main approaches are tunneling (or overlaying) and translation. Tunneling adds a new layer (e.g., L3.5) over current L3 protocols but inherits underlay limitations and introduces overhead. In contrast, translation enables direct interaction between different L3 architectures without needing an underlay. In this work, we adopt the translation approach.

A practical limitation of existing Inter-Architecture frameworks is the fact that their prototypes are software-based and, thus, don't match the Internet's performance. Recently emerged programmable switches, capable of Tb/s throughput and programmable features, could however change this.

This work presents Myriarch, an Internet Architecture translator running on a programmable switch, translating between IP and SCION—a security-centric clean-slate architecture. The main challenges involve managing the memory and computational constraints of the switch architectures. Our system is the first to translate between these architectures effectively in a Terabit-scale network switch.

Additional Key Words and Phrases: Internet Architecture; Programmable switches; P4; SCION

. 1, 1 (June 2024), 11 pages.

1 INTRODUCTION

The Internet, despite its global success, faces intrinsic issues like vulnerability to attacks, lack of QoS guarantees, and not inherently supporting modern use cases, such as content retrieval or mobile devices. Additionally, its design hinders incremental improvements, leading to "ossification."

In response, the research community has proposed clean-slate designs like ICN/NDN[Zhang et al. 2014], MobilityFirst[Raychaudhuri et al. 2012] and SCION [Barrera et al. 2017], each addressing specific issues but lacking a universal solution. A promising approach is inter-architecture frameworks, which support multiple architectures simultaneously, either through a new layer 3.5 or through direct translation between architectures. The translation-based approach is preferred for promoting diversity without inheriting the underlay's issues.

Performance is critical for new architectures to be deployable. Existing implementations, mainly software-based, offer limited throughput. Using programmable switches, which can handle Tb/s throughput, we developed a high-performance IP-SCION translator, facing challenges from the differing architectures and switch limitations.

The solution is part of the Myriarch project, with SCION chosen for its production readiness, security focus, and compatibility with

existing Internet structures. The translator was developed using P4 on an Intel Tofino switch, capable of 12 Tb/s packet processing. The main challenge was dealing with the switch's limited computational resources and memory.

2 RELATED WORK

The Internet's success as a global communication infrastructure highlights both its strengths and its inherent flaws. Over time, numerous problems within the Internet architecture have been identified, primarily falling into two categories:

- **Intrinsic Problems**, such as:
 - Lack of Traffic Control: End hosts lack control over traffic routes, leading to inefficiencies and security risks.
 - Outdated Design: The assumptions of the original design do not account for mobile devices and modern content retrieval for example, requiring inefficient workarounds.
 - Security Issues: The Internet's initial design overlooked security, making it vulnerable to attacks like DDoS and route hijacking.
- **Adaptation Challenges** in which we include:
 - Coupled Components: Deep interconnection of Internet components complicates implementing solutions.
 - Lack of Modularity: The Internet's rigid structure limits flexibility and innovation compared to software development.

Software-Defined Networks [Sof [n. d.]], initially supported by OpenFlow [McKeown et al. 2008], are emergent network designs with the goal of separating the Internet's control and data planes as a response to their deep coupling, ultimately allowing both to be programmable. SDNs achieve this by resorting to successive abstraction layers, with programmable switches at the bottom, whose functioning is dynamic rather than attached to a fixed protocol. The RMT switch (the basis of production switches as the Intel Tofino) consists of a multiple stage pipeline capable of processing several packets simultaneously, and, for the first time, allowing for a new range of programmable actions, while achieving Terabit performance as the fixed-function switch alternatives. The RMT model is composed of an ingress and an egress pipeline that share the same resources; a programmable parser, that identifies header fields to be matched; a match-action stage pipeline where fields are matched and an action is performed based on the match value (e.g., forward to a specific port, drop packet, modify header fields, etc.); and then a deparser where the packets are reassembled and prepared to be queued or sent out some port(s).

Programmable switches enable, for instance, header fields to be added or modified and new actions to be performed, thus allowing these features to be reconfigurable. It is important to notice, though, that the matches and actions are limited in number and complexity in order to fit the line rate performance requirements. In order to program such switches, the imperative language P4[Bosshart et al.

Author's Contact Information: Guilherme Ribeiro, guilherme.miguel.r@tecnico.ulisboa.pt, Instituto Superior Técnico, Lisbon, Portugal.

2014] was created. P4 has a limited set of actions, with simple primitives that can then be used to build functions with more complex actions. The programmability of these network data planes is an important enabler to materialize some of the clean-slate Internet Architectures we discuss next.

2.1 Clean-Slate Internet Architectures

Motivated by solving Internet’s problems, several large-scale research projects came up with “clean-slate” Internet designs, designs made “from scratch” [Barrera et al. 2017; Han et al. 2012; Jacobson et al. 2009; Koponen et al. 2011; Krähenbühl et al. 2021; Raychaudhuri et al. 2012; Wang et al. 2013; Zhang et al. 2014]. The idea was to project a new architecture free from the constraints of the existing infrastructure. These solutions envision an Internet that goes beyond traditional host-to-host TCP/IP communication, supporting functionalities typically managed by complex applications on top of the network layer, which often face additional security and performance issues. Unlike the current Internet, where security was an afterthought, some of these architectures are built with intrinsic security, particularly improving availability. Prototype evaluations show that these architectures can perform comparably to IP, and in some cases, such as CCN [Jacobson et al. 2009], real-world performance is expected to improve due to static content caches.

Projects like MobilityFirst [Raychaudhuri et al. 2012] and Content Centric Network [Jacobson et al. 2009] focus on specific principles—mobility and content, respectively—moving away from host-to-host communication. Conversely, RINA [Wang et al. 2013] and XIA [Han et al. 2012] provide platforms for various functionalities to be developed, but with distinct approaches. RINA uses layered sets of IPC (Inter Process Communication) processes defined by variable policies, while XIA supports functionalities through expressive intents without limiting development to a set of rules.

MF and CCN propose a relatively static approach to the future Internet based on current trends, while RINA and XIA advocate for a more evolvable Internet, capable of adding new functionalities over time. This evolvability is also a feature of the architectures in the next section and our project. However, RINA, XIA, and FII [Koponen et al. 2011] develop specific frameworks for architecture evolution, raising questions about their suitability and backwards compatibility with the current Internet. In contrast, our proposed inter-architecture framework aims to enable the coexistence of different architectures, making changes only at the interface between architectures and simplifying the incorporation of new architectures alongside the existing Internet.

SCION represents a unique approach by reusing substantial parts of the current Internet, focusing on inter-AS communication and incorporating strong security measures to ensure reliable communication between Autonomous Systems, such as preventing DDoS attacks by design [Basescu et al. 2016]. A relevant component of SCION for our work is EPIC (Every Packet is Checked in the [Legner et al. 2020], a set of data plane protocols for SCION. The goal of these protocols is to verify if the behaviour expressed at the control plane is being correctly applied. There are 4 EPIC protocols with incremental security measures. For this project, we only use EPIC L0, the one offering the lowest level of security.

2.2 Inter-Architecture Frameworks

A complete transition to a new architecture is unrealistic. Any solution must allow for gradual deployment and provide backwards compatibility to enable coexistence with the current Internet. This section presents solutions addressing this need.

In an attempt to allow multiple architectures to co-exist, a few projects proposed Inter-Architecture Frameworks. Recognizing the difficulty – or straight impossibility – of coming up with *the* definitive Internet architecture, these frameworks follow a different approach from clean-slate designs. Their goal is to inter-connect different Internets. We categorize inter-architecture proposals into two groups: overlay/tunneling-based and translation-based.

The first group, including Trotsky [McCauley et al. 2019] and EI [Balakrishnan et al. 2021] creates an overlay above IP, leveraging IP as the base for innovation. This approach assumes changing IP is impractical due to its deep integration with current hardware and applications, hence adopting a “if you can’t beat them, join them” strategy. The SIG(SCION-IP Gateway) [Perrig et al. 2017] also represents an overlay approach, though with SCION as an underlay for IP.

The second group, including Plutarch [Crowcroft et al. 2003], PEP-DNA [Ciko et al. 2021], and COIN [Jahanian et al. 2020], advocates for translation mechanisms directly at layer 3, bypassing IP. They argue that IP imposes limitations on new functionalities and that overlays introduce performance and coordination issues. Additionally, IP’s security problems, such as DDoS and route hijacking, cannot be fully mitigated by upper layers.

COIN and PEP-DNA suggest common guidelines for architectures to follow, while Plutarch offers a more flexible approach, assuming a host is tied to a particular context but can communicate across contexts.

Both approaches aim for an extensible, innovative Internet through modularity and abstraction. Overlay-based frameworks may be easier to deploy but don’t address intrinsic Internet problems. Direct translations could offer better performance and functionality, but developing accurate translators is challenging due to the diversity of architectures. However, translations avoid inheriting underlay problems, potentially maximizing the benefits of new architectures.

A challenge in Plutarch’s approach is the increasing number of translators required as architectures proliferate. COIN proposes using a “canonical” form to perform translations, scaling the number of translators linearly with architectures. The expectation is that the number of viable architectures will converge over time.

Despite Trotsky’s critique of Plutarch not adhering to the “end-to-end principle,” advancements in SDNs could mitigate this by allowing endpoints to control network translators. While we prioritize translation mechanisms, we do not discard tunneling, as some features may not be directly translatable and tunneling may preserve certain security properties.

However, from the frameworks with prototypes, all of them are developed only in software and do not perform at the line rate speeds of the current Internet. Our project aims to bridge this gap by developing a SCION-IP translator that operates at line rate in a programmable switch.

3 DESIGN

In this section, we present the design of our project. We start by establishing a comparison between SCION and IP data planes, also describing the translation methods used in our system. After that, we present the complete design with all the different pieces combined.

3.1 IP vs SCION Data Plane Comparison

In order to build our IP-SCION translator, we started by comparing the fields in both types of packets, represented in Figure 1 and Figure 2.

Version	HdrLen	Type of Service	Total Length	
Identification		Flags	Fragment Offset	
Time to Live	Protocol		HdrChecksum	
Source IP Address				
Destination IP Address				
Options				
Data				

Fig. 1. IP packet

Version	TrafficClass				FlowId			
NextHdr	HdrLen				PayloadLen			
PathType	DT	DL	ST	SL	RSV			
Addresses (12-40 bytes)								
Forwarding Path (variable length)								
Extension Header(s)								
Layer-4 protocol and data								

Fig. 2. SCION packet

We also defined the scenario of translation, specifying the kind of IP packets we intended to translate in this first design. We opted for the simplest case, in which we only consider IPv4 packets that are not fragmented, do not specify any kind of quality of service, and have no options. This way, the “Version” field is set to “IPv4” and we ignore the content of the fields “Type of Service”, “Identification”, “Flags”, and “Fragment Offset”. Moreover, we only consider packets with no field “Options”. On the SCION side, we only generate packets without any Quality of Service set, only consider the existence of one packet per flow and do not create any additional headers. Therefore, the field “TrafficClass” is set to 0, “FlowId” is set to any value and no “Extension Header(s)” are added.

Since the kind of packet we intend to generate is the SCION one, we now analyse how to obtain its remaining fields.

3.1.1 Common Header. To begin with, at the time of design, only SCION “Version” 0 is supported.

Since we do not generate Extension Headers, the value of “NextHdr” specifies the Layer 4 protocol that succeeds the SCION header. For widely used protocols, such as UDP and TCP, this value is equivalent to the field “Protocol” of the IPv4 packet.

The value of the field “HdrLen” depends on the size of the variable-length fields “Addresses” and “Forwarding Path”. In this case, since we are only handling IPv4 packets, the length of the field “Addresses” is fixed. On the other hand, the size of the “Forwarding Path” is unknown at the start and cannot be deduced from any field present in the original IPv4 packet.

The value of “PayloadLen” can be calculated using the field “TotalLen” of the IPv4 packet.

For our project, the “Path Type” to be set is the type “SCION”, as we only deal with the EPIC L0 protocol, leaving EPIC L1-L3 for future work. Furthermore, we do not consider special cases in which the path is empty for example.

The fields “DT/DL/ST/SL” are fixed for our project, because, once again, we only consider IPv4 addresses, which have fixed length.

To finalize the common header field analysis, the field “RSV” is set to 0.

3.1.2 Address Header. For our scenario, in the address header, source and destination addresses are represented by the tuple (ISD, AS, IPv4 address). We assume that IPv4 addresses inside ASes remain globally unique in the IP world. This way, a host present in the IP world is able to use the real IP address of a SCION Host, without need for a translation service like NAT. With this scenario, the tuple’s IPv4 addresses can be directly copied from the IPv4 packet. Because, from the SCION network’s perspective, the packet originated from the switch, the source ISD and AS are defined by the switch. However, obtaining ISD and AS Identifiers requires the participation of SCION services.

3.1.3 Forwarding Path Header. One of the main differences between an IPv4 packet and a SCION packet is that, whereas the former depends solely on the destination address to reach the target, the latter requires the full AS Forwarding Path along the network (Packet Carried Forwarding State, as denominated by SCION authors). Furthermore, as previously explained, to obtain a “Forwarding Path” to a certain destination, several steps need to be taken. One needs to perform 3 main actions: Path Lookup, where hosts ask Path Servers for Path-Segments to reach the destination; Path-Segments Verification, in which hosts use public key cryptography to verify the authenticity of the received Path-Segments; and Path Combination, in which hosts choose Path-Segments among those received by the Path Servers and combine them to produce a final Forwarding Path.

3.2 IP-SCION Translation

Having this analysis done, we can divide SCION packet fields in 3 groups according to the kind of operations needed to obtain them:

- The constant fields, whose value is the same for all SCION Packets in our scenario: “Version”, “Path Type”, “DT/DL/ST/SL” and “RSV”;
- The fields obtained by applying simple in-switch operations: “NextHdr”, “HdrLen”, “PayloadLen”, “SrcHostAddr” and “DstHostAddr”;
- The fields that require contacting SCION services: The “ISD”, “AS” identifiers and the “Forwarding Path”.

As the switch is not able to craft the fields from the third group, we rely on the control plane to get and insert these fields on the switch tables. This means the first packet of a flow needs intervention from

Version = 0	TrafficClass -> Ignored	FlowId -> Ignored	
NextHdr = IPv4.Protocol	HdrLen = (12 + 24 + Path_size) / 4		PayloadLen = IPv4.TotalLen - 20
PathType = SCION	DT = IPv4	DL = 4 B	ST = IPv4
	SL = 4 B	RSV = 0	
DstISD = ?		DstAS = ?	
SrcISD -> Defined by us		SrcAS -> Defined by us	
DstHostAddr = IPv4.DstAddress		SrcHostAddr = IPv4.SrcAddress	
Forwarding Path = ?			
Layer-4 protocol and data			

Fig. 3. Representation of the way we obtain SCION packet fields

the control plane, and thus experience an initial setup delay, but not the other packets from a flow, as the rules are already installed in the switch data plane. There are 3 important properties of the SCION network that enable scalable translation in the switch data plane:

- The first is that Forwarding Paths carried by SCION packets are destined to an AS, not to an address, meaning that one single Forwarding Path can be used to reach several hosts inside an AS;
- Secondly, as mentioned in [Barrera et al. 2017], current paths along the Internet traverse an average of less than 4 ASes. Despite considering that this number tends to increase for a SCION network, Forwarding Paths are still expected to be small, with a 16-hop Forwarding Path already being considered an extreme case;
- Finally, for the standard SCION protocols (EPIC L0), Forwarding Paths are not one time usage pieces of information. Once a host has constructed a Forwarding Path, it is usually valid for several hours and can be used multiple times.

3.2.1 Obtaining SCION Addresses and Forwarding Paths. We now describe how to obtain SCION Addresses and Forwarding Paths. For simplicity, we refer to the set of ISD and AS identifiers as just the AS identifier, as it uniquely identifies an AS.

The Myriarch translator’s control plane acts like a regular SCION host since it enters the SCION network. It must contact specific SCION services to obtain the necessary address and path fields. Typically, a host starts with Name Resolution, providing the service name to Name Servers, which reply with the corresponding SCION address. However, since the IP packet only has the destination’s IP address, we need an address resolution service to map this IP address to a SCION AS identifier.

Currently, there is no direct SCION service for this, so we use the SCION-IP Gateway (SIG) mentioned in 2.2. SIGs allow IP hosts to communicate via the SCION network by encapsulating IP packets into SCION packets. To allow this behaviour, each section of SCION network traffic needs a SIG at both ends: one to encapsulate and another to decapsulate the traffic. SIGs use static mappings between IP prefixes and AS identifiers, which currently must be manually configured.

As should be clear, SIGs provide similar functionality to our intended translator, with some relevant differences. They both generate SCION packets from IP packets, forwarding them to the SCION network. The crucial difference, however, is that a SIG transmits IP packets by *encapsulating* them in SCION packets. SCION thus serves as an underlay for IP traffic. Myriarch effectively performs a translation (as shown in Figure 4), it is not an overlay approach. Our method would have the advantage of enabling an IP host to communicate with another host inside a non-IP SCION AS, but would have the disadvantage of losing information in the process.

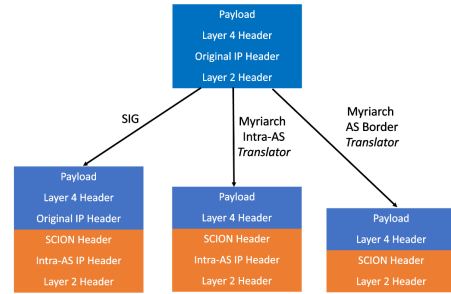


Fig. 4. Layout comparison of packets generated by SIG, on the left, by an Intra-AS translator in the middle, and by a translator situated in an AS border, on the right.

Since our translation requires the same kind of address mapping and Forwarding Path services the SIG uses, we will use the SIG as a proxy to obtain this information. However, they are not the only services useful to us as SIGs also use the regular SCION services for building Forwarding Paths. Our plan is then to take advantage of the entire set of services enjoyed by the SIG to create a Forwarding Path, by making them available for the switch’s control plane.

3.3 Putting it all together

We now present, in Figure 5, the flow of our translation mechanism.

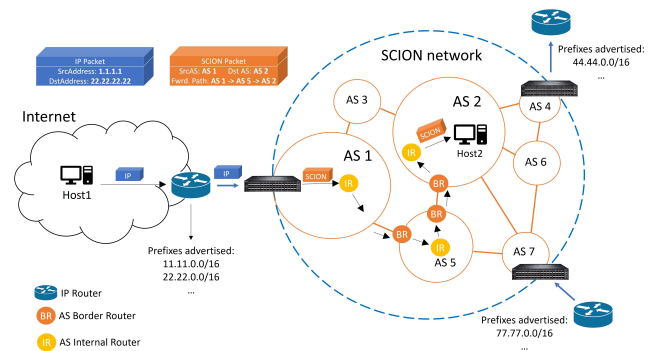


Fig. 5. Representation of an entire packet flow starting at IP Host 1 and ending at a SCION Host 2

We envision programmable switches in the borders between the Internet and the SCION network translating packets between the

two architectures. Viewing from the SCION side, switches act as hosts that belong to a certain AS.

As shown in the figure, the flow is initiated by an IP host (Host 1), in the Internet, which creates a packet destined to a server running in a SCION AS (Host 2). The packet is routed according to its destination IP address to one of the SCION network interfaces, represented by the routers connected to the switches. In this case, the packet reaches the router that advertises the prefixes 22.22.0.0/16, where Host 2's IP address is included. The router provides the packet to the switch, which performs the translation by generating a new SCION packet, represented in high-level at the top-left corner of the figure. From this point on, the packet is forwarded as a regular SCION packet according to its Forwarding Path. After going through the different ASes's Internal and Border Routers, it reaches Host 2 at AS 2.

Now we take a closer look into what happens at the translator switch. When the packet arrives at one of its ports, the first assertion checked is whether the Forwarding Path matching the packet's IP destination address is present in the switch's tables. Depending on the result of this assertion, the switch takes one of two possible actions:

- If the Forwarding Path is not present, the packet is sent to the control plane. The control plane extracts the destination IP address from the packet and sends a request to the SIG services for the respective destination SCION address and SCION Forwarding Path. Once received, these fields are installed in the switch Match-Action tables as the matching translation rules for a certain IP prefix. The packet is now returned to the switch where the initial assertion has become true. This process is illustrated in Figure 7.
- If the Forwarding Path is present, the switch generates an inter-AS SCION header and appropriate intra-AS headers. In the case of an IP AS, these intra-AS headers consist of an IPv4 header destined to one of the AS's Border Router and a UDP header, with the "Destination Port" set to the port on which the Border Router is listening. The inter-AS SCION header is generated directly in the data plane, as the translation rules were installed previously (process illustrated in Figure 6).

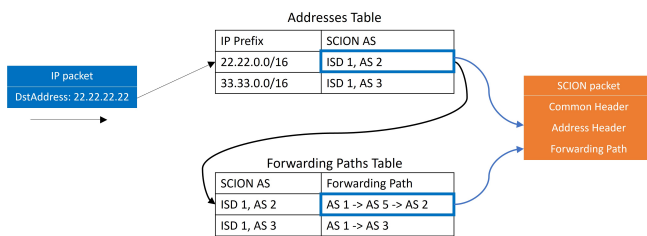


Fig. 6. Representation of the destination SCION Address and the SCION Forwarding Path extractions from the switch's Action-Match Tables

The first packet of a flow goes to the control plane adding an initial set-up delay. However, all subsequent packets remain entirely in the data plane, so it runs at line rate.

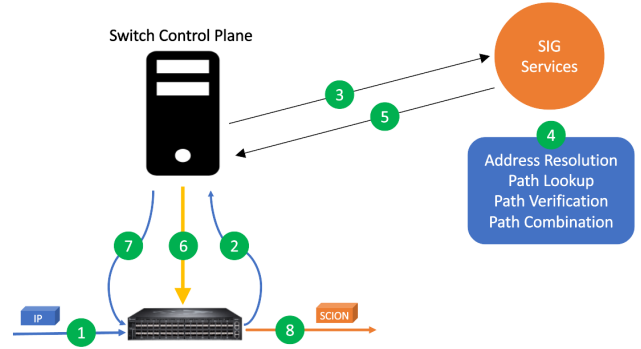


Fig. 7. Program flow in case of miss in the switch tables: 1) Arrival of an IP packet; 2) There is a table miss for the destination IP address, so the packet is sent to the switch control plane; 3) The control plane asks the SIG Services for a SCION address and Forwarding Path; 4) The SIG Services perform the required steps to create a Forwarding Path; 5) The address and path are sent back to the control plane; 6) The control plane installs the required translation rules in the switch tables; 7) The original IP packet is returned to the switch, which is now able to translate it; 8) The final SCION packet leaves the switch

4 IMPLEMENTATION

In this section, we describe the implementation of our solution. We start by defining the scenario of translation, giving a practical example. We continue by describing the data plane and the main challenges encountered during its development, followed by a description of the control plane component.

4.1 Translation Scenario

We considered 3 options for the location/configuration of our Myriarch translator. These are represented in Figure 8.

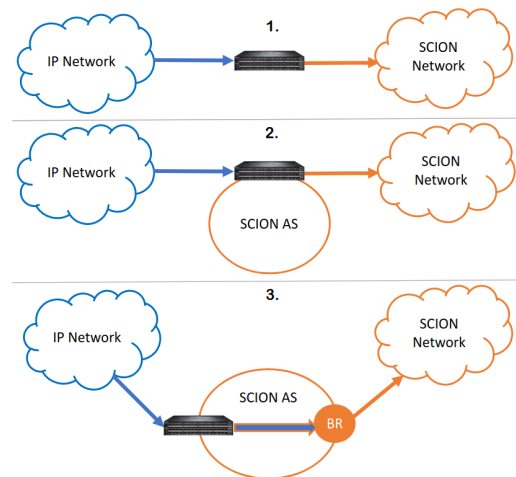


Fig. 8. Possible locations/configurations of the IP-SCION Translator

The first option is for the switch to receive packets from a native IP link and send traffic to a native SCION link. In this scenario,

the switch is not part of a SCION AS, meaning it does not have a SCION Address attributed to it. This complicates the generation of SCION packets, since they need a source SCION Address and the Forwarding Path must have a starting AS. These features are especially useful for SCION hosts to know where and how to send the response, if desired. An alternative, represented in case 2, is for the switch to be part of a SCION AS, replacing one of its border routers. This means the switch would need to perform both the tasks of translator and border router. Although a SCION border router has already been implemented in P4 for the t2na model, namely in [de Ruiter and Schutijser 2021], combining the two implementations implies sharing the switch’s resources, which increases the complexity of the problem. This option would also not be in line with modularity and abstraction principles.

Finally, in the solution illustrated at the bottom, the switch belongs to a SCION AS without replacing one of its border routers, thereby sending packets to an internal AS link. In this scenario, the switch can have an appropriate SCION Address without having to deal with the additional tasks of a border router. The drawback is that the generated packets must have an underlay for internal AS communication (consisting, for instance, of an IP layer and a UDP layer), so they can be correctly forwarded to a border router. We opted for this solution, since apart from length fields and checksums, which can be handled by in-switch operations, these underlays can be fixed for all packets. Figure 9 illustrates the layers of the received IP and the translated SCION packets and how they relate to each other.

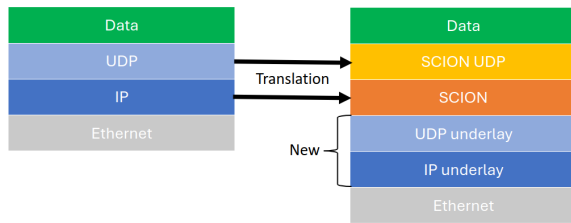


Fig. 9. How headers in the translated SCION packet relate to the headers in the original IP packet

4.2 Data Plane

The data plane was initially implemented using P4 bmv2, a generic software switch for developing P4 programs and control planes. This tool does not impose the memory and action constraints of physical programmable switches, allowing for initial validation of the program’s logic.

Subsequently, the program was developed for t2na, designed for the second-generation Intel Tofino switch, delivering up to 12 Tb/s throughput. Though t2na P4 programs have stricter syntax and memory limits, the core logic remained similar. The t2na model posed more challenges, so only this implementation is described.

A P4 program consists of four main parts:

Header Definitions Define packet headers such as Ethernet, IP, UDP, TCP, SCION, and SCION UDP.

Parser Extracts headers from incoming IPv4 packets (Ethernet, IPv4, UDP/TCP). Layer 4 headers are extracted for computing checksums for SCION packets.

Match-Action Pipeline Performs core translation by creating new IPv4 and UDP headers, filling fields based on packet and SCION Forwarding Path length, and computing checksums in the deparser.

Deparser Reassembles packets for transmission.

To obtain SCION Addresses and Forwarding Paths, several tables are defined for retrieval, depending on a match with the IPv4 packet’s destination address. If no match is found in the first table, the packet is sent to the control plane; if matched, four fields are retrieved: the SCION destination address, the Path Meta Header, the Forwarding Path size, and the packet’s egress port.

The remaining fields of the Forwarding Path (Info Fields and Hop Fields) are not retrieved initially due to their uncertain number, which can range from 1 to 3 Info Fields and 0 to 64 Hop Fields. The P4 language cannot handle arbitrary-length action arguments, so defining a maximum Forwarding Path length and removing unused parts would waste memory. Instead, we use P4’s ability to choose which headers to include in the final packet. We define a maximum size for the Forwarding Path in the headers and the space for it is automatically allocated.

The t2na compiler allows retrieval of up to 8 hops in a single action. Thus, we divided the Forwarding Path retrieval across multiple tables, each retrieving a power-of-2 number of fields (1, 2, 4, or 8). For example, a 6-hop path fills a 4-hop and a 2-hop entry. Figure 10 pictures yet another example.

The data plane searches all tables and retrieves fields from matched tables, which are then added to the final packet. This power-of-2 approach efficiently covers all possible Forwarding Path sizes, minimizing wasted space. Knowing SCION traffic patterns allows for adjusting table sizes at compilation to accommodate more common path lengths. For example, if 3, 4, and 5-hop paths are more frequent, tables storing 1, 2, and 4-hop entries would be larger than those for 8-hop entries.

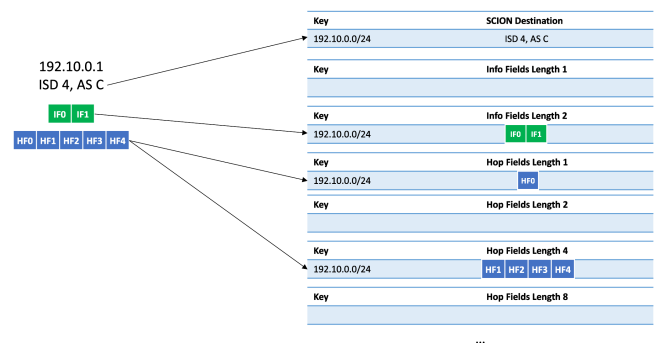


Fig. 10. Representation of the way the Control Plane distributes a SCION Address and its respective Forwarding Path for the several matching tables.

Finally, in the deparser, we compute three checksums: IPv4, UDP, and SCION UDP. The t2na model provides an extern function for checksum calculation, requiring us to specify the fields involved.

The IPv4 checksum is straightforward, excluding only the checksum field itself. The UDP checksum includes UDP header fields, IPv4 addresses, IPv4 protocol field, UDP length (twice), and the payload. Since the switch can't access the payload, this checksum is computed indirectly using the original packet's UDP checksum. The t2na model supports this with appropriate externs.

In the parser, fields affecting the UDP checksum that change during processing are subtracted from the original checksum. In the deparser, updated fields and new fields added to the UDP payload are included. Both the IPv4 addresses and UDP header change, and we also add the full SCION and SCION UDP headers. The SCION UDP checksum differs from the regular UDP checksum by including SCION addresses instead of IPv4 addresses.

Initially, we planned to compute one checksum before the other, but checksum fields aren't immediately updated by the switch. Instead, we used a property of one's complement arithmetic. The UDP checksum is calculated by dividing fields into 16-bit chunks, summing them using one's complement arithmetic, and then one's complementing the result. The sum of the fields and the checksum should result in a value with all bits set to 1 (representing 0 in one's complement arithmetic).

Listed in Figure 11 are the fields used to calculate the UDP checksum side by side with the ones used to calculate the SCION UDP checksum. If all the fields included in the SCION UDP checksum were also included in the UDP checksum, then we would only need to exclude them along with the SCION UDP checksum field from the UDP checksum. This is because, as we have explained before, their one's complement sum would originate a value with all bits set to 1. Therefore, adding this group of fields to the UDP checksum computation would be the same as adding a value with all bits set to 1, which in one's complement arithmetic is the same as adding a zero value, or in other words, not adding anything.

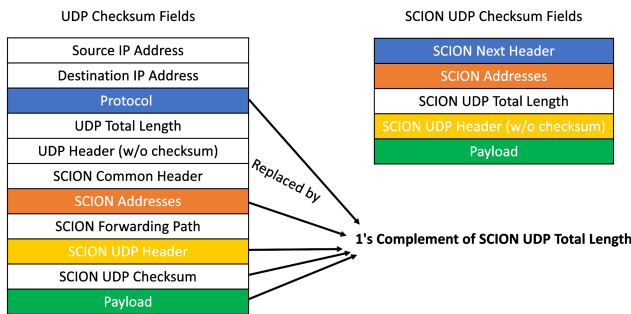


Fig. 11. Comparison between the fields included in the UDP checksum and in the SCION UDP checksum. The coloured fields are common to both checksums.

However, not all fields present in the SCION UDP checksum are included in the UDP checksum. The fields included in both are the SCION Addresses, the SCION UDP fields (excluding the checksum field) and the Payload. In our test case, since we only use SCION UDP for the Layer 4 protocol, the IPv4 protocol and the SCION Next Header fields have the same value (which is 17, the code that represents the UDP protocol), therefore they can also be considered

as being the same. However, the field SCION UDP length is only present in the SCION UDP checksum calculation. This means that, if we exclude the previously mentioned fields and the SCION UDP checksum field from the UDP checksum calculation, we would not be excluding a total value with all bits set to 1, but a value that if summed with the SCION UDP length would have all bits set to 1. If this value is represented by v and the symbol $+$ represents the one's complement sum, we have the following equation:

$$v + \text{ScionUdpLength} = 1111111111111111 \quad (1)$$

We can deduce that the value v is the one's complement of the field SCION UDP length, since a binary number summed with its one's complement is equal to a value with all bits set to 1. We recall that our goal is to find a way to indirectly include the SCION UDP checksum field in the UDP checksum calculation since we do not know its value. Having calculated the value v , we can then replace the aggregate of SCION Addresses, IPv4 Protocol, SCION UDP header (including the checksum field) and Payload for the one's complement of the SCION UDP length, which we can obtain with a simple operation in the match-action pipeline. Figure 11 pictures the fields included in both checksum computations, signaling which ones are replaced in our P4 program. We described the process for the case SCION UDP is used, however considering the similarities with other checksums, the same logic can be applied for other SCION Layer 4 protocols, by excluding all fields present in both checksums (from the UDP checksum) and adding the one's complement of the fields only present in the SCION Transport Layer checksum.

4.3 Control Plane

The Control Plane has the task of acquiring the fields the switch is not able to calculate: the destination AS Address and the Forwarding Path. As we mentioned in the design section, there is a set of services we need to contact to obtain these fields: the Address Resolution service used by the SIG to get the AS Address and the Path Servers to obtain a Forwarding Path. The SCION network used to evaluate our solution was made available by the SCION Lab. The SCION Lab is a research platform used for testing the SCION network, having several Autonomous Systems deployed worldwide. Although SCION's goal is to run as the native network layer in the links between ASes, the SCIONLab still uses overlay links over the public Internet to connect some of the ASes. This happens in order to simplify the management and joining of new ASes. The general structure of the SCIONLab is represented in Figure 12.

The SCIONLab allows us to create our own SCION AS and connect it to the SCION network via an overlay link. We opted for creating two ASes, not directly connected. Our intention was evaluating our solutions by sending packets from one to the other. In a production environment, each AS service would run in a separate server, but for our prototype we installed all the SCION services required for each AS in a single machine. The first AS (AS 1) is installed in a machine running Ubuntu 22.04 LTS and the second AS (AS 2) is installed in a virtual machine running Ubuntu 18.04 LTS hosted in the same physical machine (the different Ubuntu versions have to do with different installation methods). We chose a virtual machine for AS 2, since we needed fewer resources for it. Both ASes belong

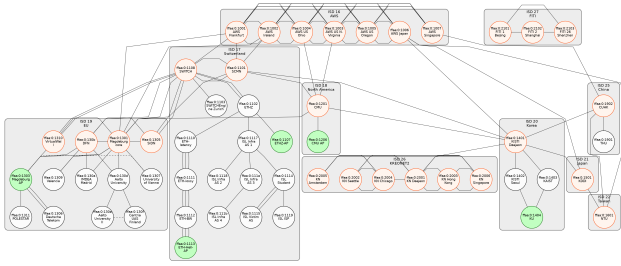


Fig. 12. The topology of the SCIONLab infrastructure ASes and the ISD boundaries. Core ASes are red. SCIONLab attachment points are green.

to ISD 19 and, since we do not have access to a native SCION link, both are connected to the SCION network through a VPN. These ASes use IP as their internal Layer 3 protocol, thus IP Addresses are attributed to internal hosts.

We also installed the SIG application in both ASes setting up a static IP route between them. Once the full setup is ready, the SIGs are able to connect two IP networks, one in each AS, providing a virtual network between IP hosts present in both SCION ASes. Particularly, we intend to establish communication between 2 chosen IP hosts, one with the Address 172.16.11.1 belonging to the network 172.16.11.0/24, internal to AS 1 and one with the Address 172.16.12.1 belonging to the network 172.16.12.0/24, internal to AS 2. With this application, an IP message destined to the Address 172.16.12.1 goes through the SIG and is sent to its destination via the SCION network, as exemplified in Figure 13. In this case, an IP host (Host that is not aware of SCION or that simply intends to use IP) in AS A desires to communicate via IP with another IP host in AS B.

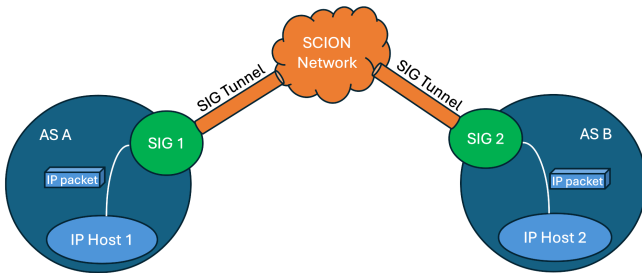


Fig. 13. Simplistic representation of how the SIG tunnels IP packets across the SCION Network

We could have more ASes and more hosts in each AS, but this scenario is sufficient to evaluate our implementation. To recall, we require the Address mapping system used by the SIG to provide us with the appropriate SCION Address matching a given IP Address. For instance, in the case of the IP Address 172.16.12.1, we require the response "ISD 19, AS 2, 172.16.12.1". Since this Address mapping system is intrinsically tied to the SIG and currently there is no API to communicate with it, we considered two different ways of obtaining this translated Address. The first one was to design an API for it, which would imply changing the SCION base code for our AS to add this new feature. The second way was to make the SIG send

a packet to our intended destination, then capture this packet and finally extract from it the complete SCION Address. We can do this since, in our testing scenario, all the SCION services, including the SIG, are running in the same machine.

Although the first option is cleaner, we decided not to change the SCION code, primarily for time reasons. The second option is simple, helping us validate our hypothesis, but less efficient. It allows us not only to obtain the SCION Address but also the Forwarding Path to the destination, as it too can be extracted from the packet.

Procedure. We developed a simple python program, running in AS 1, that receives an IP Address and uses Ping to send an ICMP packet destined to that Address. If this Address references a host inside the SCION network and it is reachable via the SIG, then the packet is routed to the SIG. There, the SIG encapsulates it in a SCION packet, setting the adequate fields in the SCION header for it to get to the correct SCION AS. Our program then captures this SIG packet with Pyshark, a python wrapper for Tshark. Tshark is a network protocol analyzer that enables us to capture packets and read their contents. We used a SCION protocol dissector for Tshark (a plugin developed by the SCION team) to more easily identify and extract from the packet the SCION Address and Forwarding Path fields.

The Control Plane is thus composed of two python programs. The main one and the capture one. The main program is essentially responsible for writing into the switch tables the SCION Addresses and Forwarding Paths necessary for the translations. The program flow starts when an IP packet is received by the switch. The next actions are reading the IP Address the packet is destined to and sending a request to the capture program asking for the corresponding SCION fields. After a response is received, the Forwarding Path is broken up into several segments according to its size, as described in Section 4. The control plane then installs these translation flows across the various tables of the switch ASIC.

5 EVALUATION

In this section, we evaluate mainly the correctness of the translation functionality, as well as the throughput and latency achieved by our solution. We performed experiments both using the Virtual Switch (SDE) running our P4 program and the SIG, the latter serving as our baseline.

5.1 Setup and Configuration

In this section, we explain the setup that allows us to validate our solution. Namely, we describe how IP packets are directed to the switch, and how they reach the SCION network after translation in the Tofino switch.

Our idea was to turn the switch into a multi-homed entity placed at the border of the IP and SCION networks, acting as an IP router for the IP network and as a special border router for the SCION network. We say special, since a regular border router connects two SCION ASes. In this case, it connects two distinct network architectures.

We ran our prototype in a virtual environment, using the Intel P4 SDE. If a program correctly runs in the Intel SDE it is ensured to also run on a physical switch at Terabit speeds. Therefore, before running on a physical switch, a P4 programmer firstly develops

and tests his programs on the SDE, taking advantage of its developing, debugging and optimization tools, as well as its portability and simplicity to set up. Running this kind of program on a physical switch though, requires several lengthy and error-prone configuration tasks including setting up the switch and establishing all the necessary physical connections for it to communicate with the control plane, the IP network and the SCION network. Due to lack of time we took the decision to leave them for future work. As such, in this thesis, we mainly access the correct functioning of the P4 program in software.

In section 3, we mentioned that our solution would involve fully replacing the IP header for a SCION one. This is our final goal, and we developed a solution likewise. However, since we have not yet built a prototype for the inverse translation (SCION to IP), we cannot directly evaluate this solution. We can test a scenario where we send messages from AS 1 to AS 2 and check if they arrive correctly to their destination, but getting a reply would require an appropriate inverse translator. Therefore, for testing purposes, we developed a solution where our switch replaces the SIG of AS 1. Not fully, since the SIG cannot be currently developed in a programmable switch (due to the use of stateful memory, for example, merge several IP packets into one) but using its main features and adapting them to our needs. This implies encapsulating the original IP packet, as illustrated in figure Figure 4, and adding an adequate SIG header. Building packets and identifying them as coming from the SIG allows the receiving server at AS 2 to also reply via the SIG tunnel, namely sending a packet through its respective SIG, which we refer to as SIG 2. Note that although this implies an encapsulation of the original packet, *our prototype translates the IP packet to a SCION packet effectively, the core component of a translator-based solution.* The need for an encapsulation is a mere artifact of our limited experimental set up and implementation.

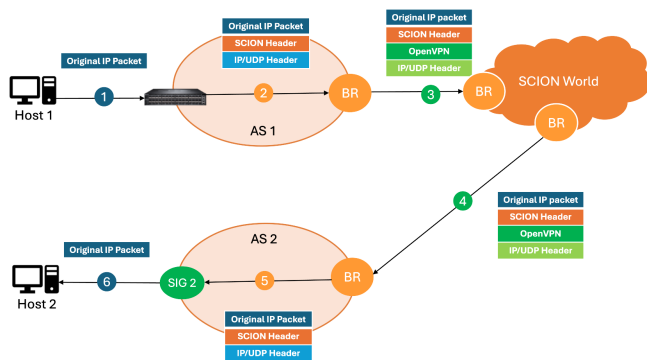


Fig. 14. Setup used for evaluation

5.1.1 Evaluation Scenario. The scenario used to evaluate our implementation is displayed in Figure 14. In the figure, we represent 2 hosts. Host 1 acts as the client and Host 2 as the server, for instance from an iPerf connection. Host 1 directly sends IP packets to the programmable switch, which translates them into SCION packets. As explained in the previous section, the original IP packet is encapsulated in a SCION packet due to not yet existing an inverse translator.

These SCION packets are forwarded to the Border Router and from there to the next AS on the SCION Path. Since our custom ASes do not have access to a native SCION link, the Border Routers use a VPN to connect to the SCION network. Inside the SCION network, the packet traverses intra and inter-AS links, until it reaches the last AS before AS 2. Here, a VPN layer is added again for communication between Border Routers. After this, the packet is forwarded to SIG 2 which, in turn, decapsulates the original IP packet and sends it to Host 2. As baseline for comparison, we performed the same evaluation procedure but with the SIG in the place of the switch.

5.2 Results

5.2.1 Translation Functionality Correctness. First of all, we analyze the correctness of the SCION packets generated by our translator. As we have mentioned in section 3.2, we focused on the translation of the most important fields in a SCION packet. These include length fields, the SCION Addresses and the SCION Forwarding Path. To other fields we mostly applied static rules. These static rules, however, may not be the most adequate and make the translation process less flexible, since they cannot be changed by the control plane. This is something to be improved in a future version.

The crucial fields for allowing a working translation are successfully calculated or obtained by our program. We verified this by direct observation, first, but our experiments in Sections 5.2.3 and 5.2.4 also demonstrate the correctness of our translation mechanisms: the tools used for communication across the SCION network (ping and iperf) have functioned correctly. Nevertheless, one limitation of our approach is that we assume some fields have constant size or are non-existent. The consequence is that the length fields do not consider, for example, the size of SCION Extension Headers. Regarding SCION Addresses, the only aspect to note is that solely addresses of IPv4 ASes are available in this first prototype.

5.2.2 Length of the Forwarding Path. Due to switch memory limitations, namely of the Packet Header Vector, our solution allows SCION Forwarding Paths with at most 20 hops. Besides creating a table to store 1-hop segments, one to store 2-hop segments, one for 4-hop segments and one for 8-hop segments, as described in Section 4, we were also able to add a 5-hop table. Considering that the average AS Path Length in the Internet is currently around 4 and a 16-hop path is already considered an extreme case by SCION authors, 20 seems enough to cover most cases.

5.2.3 Translation Throughput in the Hardware Switch. Our program compiles in the Intel Tofino SDE, which guarantees Tb/s throughput when running in a physical Tofino Switch. As we did not actually run the program in a physical switch, we evaluated the performance of our software program when compared to the SIG. For evaluating the performance, we used ICMP (by sending ping commands) to measure latency, and iPerf3, a more recent version of iPerf, to measure mainly throughput. The results presented in the next 2 sections were obtained using Ubuntu 22.04.3 LTS with an Intel® Core™ i7-5557U CPU @ 3.10GHz × 4 and 16GB RAM.

5.2.4 Translation Latency in the Software Testbed. We sent 100 pings from Host 1 to Host 2 using each of the devices under evaluation. The cumulative distribution function is represented in Figure 15. The

latency values obtained average around 100 ms for both solutions, which is expected as the communication traverses multiple countries (from Portugal to Germany and back). The average latency of our switch solution is higher about 7ms on average. We conjecture this small difference may be related to slower packet processing in the Tofino SDE and higher CPU usage than the SIG. We return to these points next.

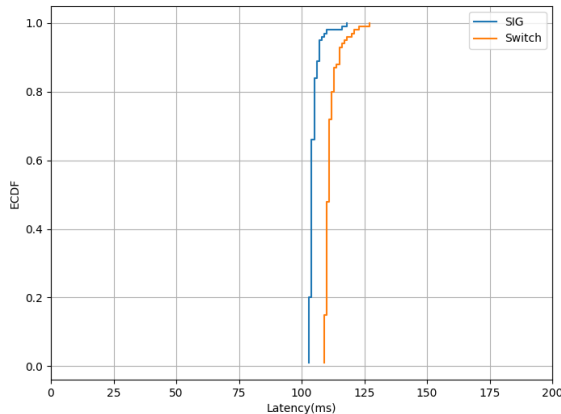


Fig. 15. Latency cumulative distribution function of 100 ICMP requests made from Host 1 to Host 2 using the SIG and the switch

5.2.5 Translation Throughput in the Software Testbed. For the iPerf experiments, we started the server at Host 2 and the client at Host 1. We gathered the results from our baseline, setting up a SIG in both ASes, and from our program, using the switch in AS 1 and the SIG in AS 2. At the client side, we defined the kind of communication to be established. We used UDP since it provides a more constant rate of information transfer, enabling us to measure with more accuracy the performance limit of both devices. We started with a bandwidth of 1 Mbits/s and progressively increased it. We noticed that when the bandwidth surpassed 8 Mbits/s, the packet loss started to be significant. This indicated a possible bottleneck in the SCION network links. As such, we decided to set the bandwidth to 8 Mbits/s for the experiments. We measured the average throughput for 100 iPerf runs, lasting 10 seconds each. The results are shown in Figure 16.

We noticed the throughput achieved by iPerf connections when using the switch was relatively low when compared to the one achieved by the SIG. After considering several possible explanations, we concluded that this was not due to network limitations but to the use of the SDE, which does not seem optimized for performance, but mainly for developing and debugging P4 programs.

5.3 Summary

The fact that the program compiles and that iPerf works correctly guarantees functioning at Tb/s in a programmable Tofino switch which was our ultimate goal.

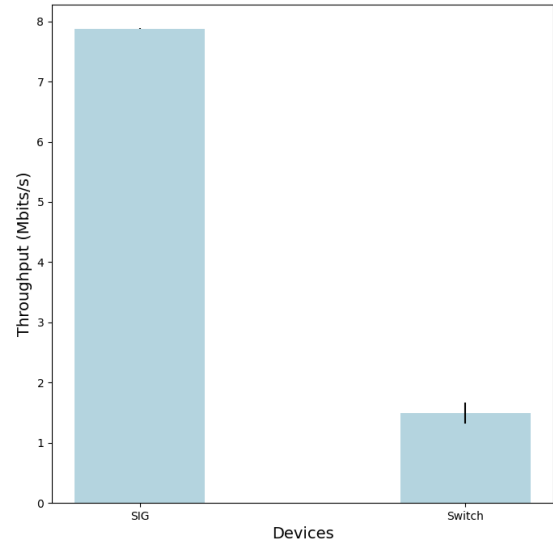


Fig. 16. Throughput results obtained when running iPerf3 between the client Host 1 and the server Host 2 100 times, having established a limit of 8 Mbits/s to the maximum sending rate of the client

6 CONCLUSIONS

In this work, we introduced Myriarch, a project aimed at enabling the coexistence of multiple network paradigms through translation mechanisms. Inspired by the Plutarch framework, Myriarch focuses on translating IP, the most widely used network protocol, into SCION, a security-centric Internet Architecture. Leveraging programmable switches and the P4 language, we developed our solution for a Tofino Switch. Although not tested on a physical switch, running on the Intel Tofino SDE ensures Tb/s throughput.

SCION separates the control and data planes, unlike IP, which integrates them. Despite differences in addressing and routing, SCION's data plane shares similarities with IP's. SCION uses a tuple (Isolation Domain ID, AS ID, internal AS Address) for addressing, and packets are forwarded along pre-defined AS-level paths. To translate IP to SCION packets within the constraints of a programmable switch and P4, we used three methods: setting fields as constants, using information from the IP packet, and querying the SCION control plane for the SCION Address and Forwarding Path. While the initial packet experiences a delay, subsequent packets benefit from cached values.

Our prototype demonstrates that translation can facilitate the integration of new Internet Architectures, allowing them to leverage their capabilities without relying on the restrictive and insecure IP underlay. This project highlights the potential of programmable switches to create flexible networks that adhere to principles of abstraction and modularity.

7 ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor, who was always understanding and pushed me positively to do the best work I could, while giving useful advice and helping me critically think about problems. Secondly, a thanks to my teachers that have been present in all meetings with whom I learned and who helped me create a valid and valuable thesis.

I would like to thank my parents who supported me during bad times and gave me all the love and all the conditions for me to finish this thesis. Couldn't have done it without them. I would also like to thank my sister, who always listened and tried to help me as best as she could. I can also include my parents here. Thanks to my grandparents, my uncles and my cousins for your love and for that extra strength to continue.

Finally, I would like to thank all my friends for giving me wise advice, supporting me, for the fun moments and for making me laugh, really helping me during this journey.

REFERENCES

- [n.d.]. Software-Defined Networks: A Systems Approach — Software-Defined Networks: A Systems Approach Version 2.1-Dev Documentation. <https://sdn.systemsapproach.org/>.
- Hari Balakrishnan, Sujata Banerjee, Israel Cidon, David Culler, Deborah Estrin, Ethan Katz-Bassett, Arvind Krishnamurthy, Murphy McCauley, Nick McKeown, and Aurojit Panda. 2021. Revitalizing the Public Internet by Making It Extensible. , 18–24 pages.
- David Barrera, Laurent Chuat, Adrian Perrig, Raphael M. Reischuk, and Pawel Szalachowski. 2017. The Scion Internet Architecture. *Commun. ACM* 60, 6 (2017), 56–65.
- Cristina Basescu, Raphael M. Reischuk, Pawel Szalachowski, Adrian Perrig, Yao Zhang, Hsu-Chun Hsiao, Ayumu Kubota, and Jumpei Urakawa. 2016. SIBRA: Scalable Internet Bandwidth Reservation Architecture. In *Proceedings 2016 Network and Distributed System Security Symposium*. <https://doi.org/10.14722/ndss.2016.23132arXiv:1510.02696> [cs]
- Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, and George Varghese. 2014. P4: Programming Protocol-Independent Packet Processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.
- Kristjon Ciko, Michael Welzl, and Peyman Teymouri. 2021. PEP-DNA: A Performance Enhancing Proxy for Deploying Network Architectures. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, 1–6.
- Jon Crowcroft, Steven Hand, Richard Mortier, Timothy Roscoe, and Andrew Warfield. 2003. Plutarch: An Argument for Network Pluralism. *ACM SIGCOMM Computer Communication Review* 33, 4 (2003), 258–266.
- Joeri de Ruiter and Caspar Schutijser. 2021. Next-generation internet at terabit speed: SCION in P4. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies (New York, NY, USA, 2021-12-03) (CoNEXT '21)*. Association for Computing Machinery, 119–125. <https://doi.org/10.1145/3485983.3494839>
- Dongsu Han, Ashok Anand, Fahad Dogar, Boyan Li, Hyeontaek Lim, Michel Machado, Arvind Mukundan, Wenfei Wu, Aditya Akella, and David G. Andersen. 2012. XIA: Efficient Support for Evolvable Internetworking. In *Proceedings of NSDI*.
- Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. 2009. Networking Named Content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*. 1–12.
- Mohammad Jahanian, Jiachen Chen, and K. K. Ramakrishnan. 2020. Managing the Evolution to Future Internet Architectures and Seamless Interoperation. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 1–11.
- Teemu Koponen, Scott Shenker, Hari Balakrishnan, Nick Feamster, Igor Ganichev, Ali Ghodsi, P. Brighten Godfrey, Nick McKeown, Guru Parulkar, and Barath Raghavan. 2011. Architecting for Innovation. *ACM SIGCOMM Computer Communication Review* 41, 3 (2011), 24–36.
- Cyrrill Krähenbühl, Seyedali Tabaeiaghdaei, Christelle Gloor, Jonghoon Kwon, Adrian Perrig, David Hausheer, and Dominik Roos. 2021. Deployment and Scalability of an Inter-Domain Multi-Path Routing Infrastructure. In *Proceedings of the 17th International Conference on Emerging Networking Experiments and Technologies*. 126–140.
- Markus Legner, Tobias Klenze, Marc Wyss, Christoph Sprenger, and Adrian Perrig. 2020. EPIC: Every Packet Is Checked in the Data Plane of a Path-Aware Internet. In *29th USENIX Security Symposium (USENIX Security 20)*. 541–558.
- James McCauley, Yotam Harchol, Aurojit Panda, Barath Raghavan, and Scott Shenker. 2019. Enabling a Permanent Revolution in Internet Architecture. In *Proceedings of the ACM Special Interest Group on Data Communication*. 1–14.
- Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review* 38, 2 (March 2008), 69–74. <https://doi.org/10.1145/1355734.1355746>
- Adrian Perrig, Pawel Szalachowski, Raphael M. Reischuk, and Laurent Chuat. 2017. *SCION: A Secure Internet Architecture*. Springer International Publishing, Cham. <https://doi.org/10.1007/978-3-319-67080-5>
- Dipankar Raychaudhuri, Kiran Nagaraja, and Arun Venkataramani. 2012. Mobilityfirst: A Robust and Trustworthy Mobility-Centric Architecture for the Future Internet. *ACM SIGMOBILE Mobile Computing and Communications Review* 16, 3 (2012), 2–13.
- Yuefeng Wang, Flavio Esposito, Ibrahim Matta, and John Day. 2013. *RINA: An Architecture for Policy-Based Dynamic Service Management*. Technical Report BUCS-TR-2013-014. Computer Science Department, Boston University.
- Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, K. C. Claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. 2014. Named Data Networking. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 66–73.