Next-generation internet at terabit speed: SCION in P4

Joeri de Ruiter*
SURF
Utrecht, The Netherlands

Caspar Schutijser
SIDN Labs
Arnhem, The Netherlands

ABSTRACT

Regularly, new architectures are proposed to address shortcomings in the current internet. It is not always trivial to evaluate how these proposals would perform in practice. This situation is improved significantly with the introduction of the P4 programming language and programmable network equipment. In this paper we discuss our implementation of one particular future internet architecture, namely SCION. We implemented a SCION router in P4 for switches based on the Intel Tofino ASIC. Having an open source P4 implementation of SCION that runs on high-speed hardware can contribute to its adoption as well as support research in this area. Our work lead to several recommendations for and subsequent changes to the SCION protocol, as well as some generic guidelines when designing protocols. A first analysis of our implementation shows it can process SCION packets at high speeds.

CCS CONCEPTS

• **Networks → Network architectures**.

KEYWORDS

programmable networking, P4, future internet, SCION

ACM Reference Format:
Joeri de Ruiter and Caspar Schutijser. 2021. Next-generation internet at terabit speed: SCION in P4. In *The 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21), December 7–10, 2021, Virtual Event, Germany*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3485983.3494839

1 INTRODUCTION

Though it is hard to imagine a world without internet nowadays, it is not without its flaws. Partly this is due to the fact that it was never designed with the way we currently use it in mind. Most notably, security was not a concern in its initial design and has mainly been bolted onto existing protocols, e.g., through TLS, DNSSEC and RPKI. Regularly there are proposals to fundamentally address issues through new architectures. However, it is not a trivial task to get these ideas from a research environment to real-world deployment. One of the challenges is to evaluate how a new protocol would perform in practice, e.g., when running on hardware. Recent developments in the area of programmable networking offer a solution for this in the form of the domain-specific language P4 and network equipment containing programmable ASICs. In this paper we will discuss how we implemented a new internet architecture that is gaining momentum, namely SCION [18], on programmable network switches. The switches contain Intel Tofino ASICs [12], which we programmed using P4 [4]. This allows us to determine how feasible it is to run the SCION protocol on switch hardware and evaluate its performance. Having an open source P4 implementation that runs on high-speed hardware can contribute to its adoption as well as support research in this area. At the same time, implementing a protocol in P4 for dedicated hardware allows for early feedback to protocol designers, before protocols are widely deployed and it is hard to change them. In this paper we present the design and implementation of a SCION router in P4 and give a first analysis of its performance, indicating how SCION might perform when ran on hardware. Based on our experience we also give several guidelines to take into account when designing a protocol. Our implementation is available as open source on https://github.com/sidn/p4-scion.

*Most of this research was carried out while employed at SIDN Labs.

2 P4

P4 is a domain specific language for packet processing on network devices such as switches and network cards [4] and can be used to add support for new protocols to network devices. As a result it is no longer necessary to wait for vendors to implement new protocols and it allows for easy experimentation with new protocols on hardware. Note that, though P4 is very generic, it depends on the specific hardware target which P4 functionality is available.

For parsing, a state machine is specified, with transitions based on values in the header. Complexity of the parser, measured in the number of transitions and bytes parsed per state, is an important factor in determining potential throughput.

A key concept in P4 is the use of match-action tables. This allows us to perform a certain action based on a value in the headers of the packet that is currently processed. Matching can typically be done at line speed in the data plane. However, adding and removing table entries is slower and is done via the control plane.

3 SCION

Below we give a short introduction of SCION [18] and discuss the aspects that are relevant for our implementation.

Currently the internet consists of interconnected networks, also referred to as *autonomous systems* (ASs). In SCION the internet still consists of autonomous systems. However, an additional layer of hierarchy is added by grouping autonomous systems into so-called *isolation domains* (ISDs). ASs in an ISD might, for example, share a jurisdiction or geographic location. The administration of an ISD is taken care of by the ISD core, a group of autonomous systems referred to as core ASs.

Routing in SCION is based on so-called Packet Carried Forwarding State. That means that every packet contains the path that it needs to travel (at the AS level). This gives senders the capability to determine how their network traffic should travel through the internet. In order to do that, senders need to know which paths to the intended destination AS are available, which is not unlike the way we currently use DNS to lookup IP addresses.

In every ISD, ASs are hierarchically organised based on their connections in a directed acyclic graph with the ISD core at its root. This allows the possible paths to be determined in a straightforward and efficient way. This process, also known as beaconing, is started by the core ASs, which send path segment construction beacons (PCBs) downstream to their neighbouring non-core ASs. When a non-core AS receives a PCB, it adds its own identity to it, as well as some additional information, which will later be used when constructing the paths in the packet headers.

The information that is used to construct the paths is the so-called hop field. In the data plane, these hop fields will be included in packet headers to specify which path the packet should follow. A hop field contains information on how the packet should be forwarded by the corresponding AS, i.e. the expected ingress and desired egress interfaces, and is protected using a cryptographic Message Authentication Code (MAC). This MAC prevents the construction of arbitrary paths and enforces the use of authorised paths, i.e. paths that were constructed through the beaconing process involving the relevant ASs. A hop field also contains an expiration time, after which the hop field is no longer valid.

After adding its information, the AS forwards the PCB to its downstream neighbours, which follow the same procedure. As a result, a PCB contains information about the path it travelled from the ISD core to the current AS. Eventually the PCBs will reach the leaf ASs and, based on the information in the PCBs, all ASs will know at least one path by which the core of the ISD can be reached, namely the path that the beacon followed. As well as forwarding the PCB, every AS stores the path it just learned locally and informs the ISD core about the path(s) over which it prefers to be reached. Finally, the core knows how every AS can be reached and every AS knows how the ISD core can be reached.

The inter-ISD beaconing process, performed between core ASs, also uses the PCBs, but follows a flooding approach similar to BGP.

To bootstrap the beaconing process, a special path type is used: one-hop paths. This path type can be used for communication between direct neighbours when there is no authenticated path established yet. The sender will add a valid hop field in the header for its own AS, but leaves the hop field for the receiving AS empty. Upon receiving the packet, the border router of the receiving network will add a valid hop field before forwarding it to the local network. This way, the recipient can use the hop fields in the packet to construct a return path.

To indicate issues in the network, such as link failures or problems with the verification of cryptographic MACs, SCION uses the SCMP protocol, similar to the ICMP protocol in today's Internet.

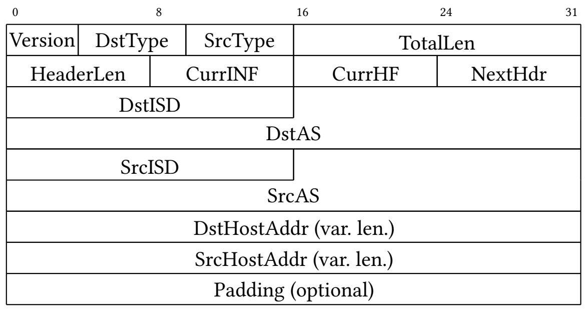
4 PROTOCOL ISSUES AND CHANGES

Originally the SCION protocol was designed mainly with software implementations in mind. As a result we ran into several issues with the header format of the SCION protocol, which made it hard or inefficient to implement on our hardware. Therefore, we proposed several changes to the headers to make them easier and more efficient to implement on hardware. Below we discuss the changes that are included in the new official headers. Previously, we discussed these changes briefly in [7].

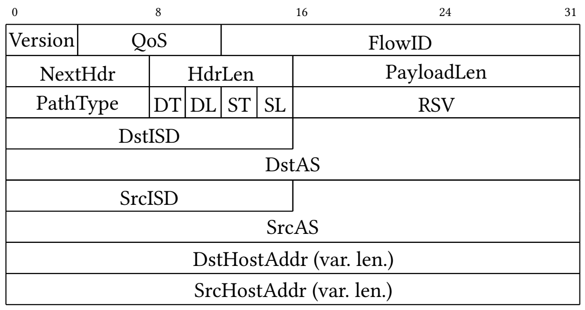
Addresses. The headers contained fields that indicated the types of addresses that were used for the end hosts in the local network (Fig. 1a). Examples of address types are IPv4 and IPv6 addresses. Different address types can have different lengths. The fact that just the type, not the length, of these addresses was included in the packet meant that every node between the end hosts needed to be aware of the address types used. This despite the fact that intermediate nodes do not actually need these addresses for processing and only need to know the lengths to be able to skip over them. We suggested to include the length of those addresses as well, such that intermediate routers do not have to be aware of the various types. In the new format of the common header (Fig. 1b), the lengths are now included explicitly as *DL* and *SL*. This also provides flexibility to experiment with new local address types, as only the sender's and recipient's networks use the address types. In general, including an explicit length instead of this being implicit depending on the value of another header field allows for more flexibility within a protocol.

Forwarding path. A forwarding path can consist of up to three segments, each consisting of a list of hops, which combined form a complete path. In the old headers this was structured as a nested list (Fig. 2a): the first-level lists contained info fields, with generic information on the segment, and each of those fields was followed by another list containing the hop fields. This structure was then repeated for each segment. While the parsing of this structure is fairly straightforward to program in software, it represents a challenge when implemented directly on hardware where all resources need to be statically allocated. Therefore, we suggested to restructure the forwarding path using two simple lists (Fig. 2b): first a list of all info fields, followed by another list containing all hop fields. In general, it is preferable to avoid the use of complex data structures, such as nested lists, in order to provide for easy parsing.

Another change to the forwarding path concerns the *CurrINF* and *CurrHF* fields. These fields are used to point to the current info field and hop field, i.e., the info and hop field the router needs to use when forwarding the packet. In the old version of the protocol (Fig. 1a), those fields pointed to the current info and hop fields using an absolute byte offset into the packet. For generic software implementations that is easy to use by just advancing a pointer based on the offset to get to the desired data. However, on specialised hardware it might not be that easy when incoming data is processed as a stream and no random access to data is provided. In that case it would be required to keep track of the number of bytes processed so far, which might introduce a significant overhead or might not even be possible at all. In the new headers (Fig. 2d), the current hop field is referred to using an index. This is straightforward to implement on hardware as well as in software. To determine the current info field, three new fields are added: *Seg0Len*, *Seg1Len* and *Seg2Len*. These fields contain the number of hop fields in the respective segments. Combining this information with the index in *CurrHF*, we can determine the current info field, thus eliminating the need for a separate *CurrINF* field. These fields also indicate

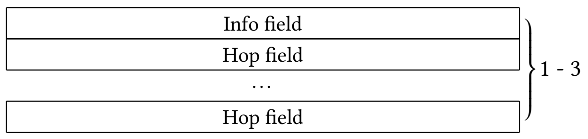


(a) Old format of common header

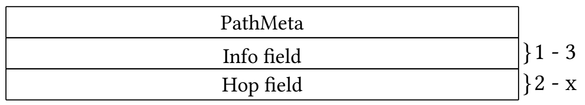


(b) New format of common header

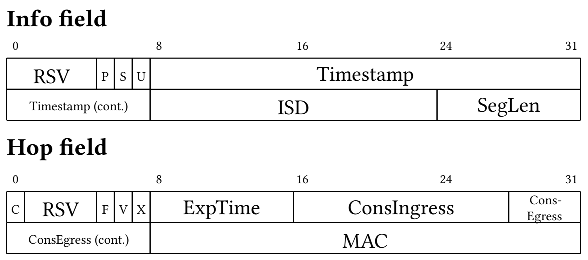
Figure 1: Common header



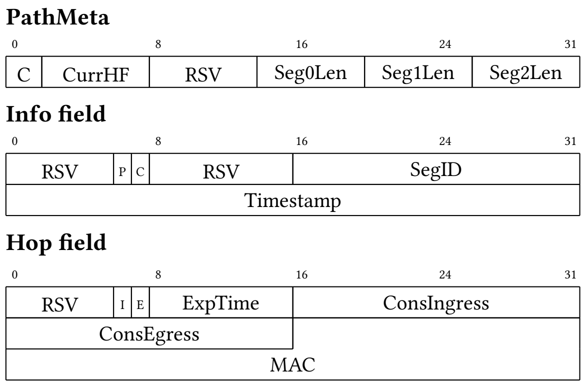
(a) High-level structure of old forwarding path



(b) High-level structure of new forwarding path



(c) Detailed elements of old forwarding path



(d) Detailed elements of new forwarding path

Figure 2: Forwarding path header

how many info fields are included, as an info field is included if the respective number of hop fields is not 0. In general, using indices is preferable over the use of absolute offsets.

This change also solves a couple of other challenges with the old headers, namely that the number of hop fields was implicit, it was not trivial to determine which segment a hop field belonged to and whether a hop field was the last in a segment. Previously, to determine whether the end of the hop fields was reached, the amount of parsed bytes was compared to the contents of the *HeaderLen* field which contains the size of the SCION header in the packet. This led to similar issues as with the absolute offsets. Using the newly added fields we know the total number of hop fields included, making it possible to determine whether the end of the SCION header is reached without knowing the exact current byte offset. At the same time it is now also fairly straightforward to determine to which segment a hop field with a particular index belongs and whether it is the last hop field in a segment. Previously, each first and last hop field in a segment would have a *XOVER* flag set (Fig. 2c). Whether it was the first or the last hop field of a segment depends on whether the flag already occurred earlier in this segment. To determine whether a hop field was the first or last in a segment, one would have to look at the next or previous hop field (which might not exist). With the new design we can determine which segment *CurrHF* belongs to and whether it is the last in a segment, using the fields *Seg0Len*, *Seg1Len* and *Seg2Len*. At the same time, when forwarding a packet it is no longer needed to compute the byte offsets of the next hop field and, optionally, info field but *CurrHF* only needs to be increased by one.

Finally, the size of the hop fields was changed. Previously, hop fields had a variable size, which is challenging when having to statically allocated resources in advance. In the new design the hop fields have a fixed size, easing the parsing of the list of hop fields. This also makes it easy to skip over previous hop fields to the currently relevant hop field using the index *CurrHF*. In general, the use of fixed length fields results in a more efficient implementation.

Combined these changes make it possible to implement the SCION protocol more efficiently for dedicated hardware, such as the Intel Tofino.

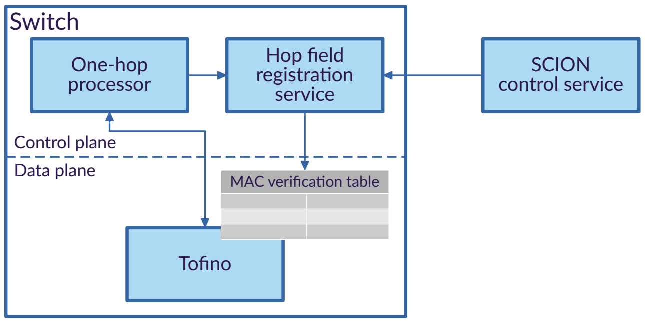


Figure 3: Overview of the different components and their interaction

5 DESIGN AND IMPLEMENTATION

Our implementation consists of multiple components: the P4 implementation and several control plane components (Fig. 3). Below we discuss the different components.

5.1 Dealing with cryptography

When implementing the SCION protocol for the Intel Tofino, the main challenge is the lack of support for cryptographic operations on the Tofino chip. As the hop fields contain a cryptographic MAC which needs to be verified upon receiving a packet, a cryptographic operation would be required for every packet we process. One way to do this would be to forward the packets to the control plane for verification. This not a feasible approach though, as this forwarding is a very slow operation and therefore would severely limit the performance. However, we can make use of the fact that hop fields are typically re-used for multiple packets in the basic SCION protocol. The number of hop fields that are valid at the same time depend on the topology and the configured expiration time. In Section 6.1, we give an idea about the number of hop fields that would need to be stored. This allows us to work around the lack of cryptographic support by using a table that contains all the hop fields that are currently valid, referred to as the *MAC verification table*. Using this table we verify the relevant hop field of incoming packets: if present in the table, it is valid; otherwise it is not and the packet will be dropped. Additionally, we also use a table to verify that a packet is received on a port that matches the ingress interface in the hop field.

In SCION, the beaconing, as well as other administrative tasks within a network, is taken care of by the *SCION control service*. We added functionality to this service to register hop fields at the *Hop field registration service* on the switch, when these are generated in the beaconing process. To support one-hop paths, the receiving network, typically the border router, needs to add a hop field in the packet. As we cannot generate the hop field in the data plane, we forward incoming packets with a one-hop path to the control plane at the switch. There, our *One-hop processor* computes a valid hop field, updates it in the packet and sends it to the *Hop field registration service*. The packet is then processed as usual and forwarded to the intended recipient, which can use the hop fields in the packet to construct a path to return packets. As mentioned before, the method of forwarding the packets to the control plane does not scale well. However, the number of packets with one-hop paths is expected to be very low, as they can only be send by direct neighbours and are typically only used in the beaconing process. Therefore, we do not expect this to be an issue in practice.

As we can not easily check in the data plane whether a hop field expired, we need to make sure that there are only valid hop fields in the table. Unfortunately we cannot use the ageing functionality introduced in P4, as this only takes into account when entries were last used whereas for the hop fields we need to compare a timestamp to the current time. To achieve the desired functionality, the *Hop field registration service* implements an operation that iterates through the *MAC verification table* and removes the expired hop fields. As a result we might be accepting expired hop fields until the next check is executed, so it is important that this process is triggered regularly to keep this to a minimum. An alternative would be to keep a record in the control plane of all hop fields that are present in the table and remove them from the table the moment they expire. This would require the control plane to always have an up to date overview of all hop fields in the table.

5.2 Parsing hop-fields

To implement the parsing of the headers efficiently, the number of state transitions in the parser is ideally as low as possible. The changes of the headers made it possible to implement the parser using less transitions and use the available resources more efficiently. For example, more paths are supported as we now have a maximum for the number of hops in the complete path instead of a maximum per segment. Also, the parsing of the common header and info fields can be done in a straight forward manner. Due to the fixed size of the hop fields and the use of an index in *CurrHF*, we can optimise the parsing of the hop fields. As the number of states that we can introduce is limited, we use the following approach to try and maximise the number of hop fields we can support. As we only need the current hop field for processing, we include long data fields that can contain a different number of hop fields, namely 16, 8, 4, 3, 2 and 1, which are used to skip over the unused hop fields. For each of these lengths we have a dedicated state, to which we can transition based on the value of the corresponding bits in *CurrHF*. This reduces the number of states in the parser and therefore we can support more hop fields. In the current design, we can parse up to 32 hop fields. Longer paths can be used though, as long as the hop field that we need to process is not beyond the first 32 hop fields.

There is one situation in which we need to process two hop fields in a packet, namely when switching between path-segments. In that case, there is one hop field for ingress and one for egress. Instead of processing both hop fields at the same time, we first process the ingress hop field. In case a hop field is valid, we check whether it is the last hop field in the segment. If this is the case, we increase *CurrHF* as usual, but instead of forwarding the packet we recirculate it as we need to completely process the egress hop field as well. As *CurrHF* was just increased, we will then process the next hop field, namely the one for egress, as usual.

5.3 IPv4/IPv6

SCION uses an IP/UDP underlay for direct communication with its neighbours to prevent possible issues with legacy equipment in between. Our implementation has support for both IPv6 and IPv4. However, mixing them in operation, i.e. sending out a packet using IPv6 that was received using IPv4, might not work as expected. As the UDP checksum is mandatory when using IPv6 [11], but not when using IPv4, we cannot expect that the checksum on incoming IPv4 packets is correct. At the same time we cannot compute a complete checksum, as this includes the complete payload and we only have access to the parsed headers.

Enabling both IPv4 and IPv6 at the same time in the implementation also requires more resources, that could otherwise be used to support more entries in the *MAC verification table*.

5.4 Limitations

Currently, we do not yet support all SCION functionality. For example, peering or the generation of SCMP error messages is not implemented yet. Additionally, with the EPIC [15] and COLIBRI [2] protocols that could add additional security and QoS to SCION, the cryptographic MACs in the hop fields will be unique per packet. As a result, for those protocols we would no longer be able to make use of the approach where we store the complete hop fields in a table for verification. If those protocols would be used, depending on what part of the traffic uses them, a solution might be to forward the corresponding packets to the control plane at the switch for verification or to an external node that does provide cryptographic support on hardware. Ideally, this functionality would be included in the data plane on the switch itself, e.g. by adding support for cryptography or allowing to add custom functionality through additional FPGAs.

6 EVALUATION

6.1 Hop fields

If we take the default settings for beaconing, i.e. beaconing at a five-second interval and a validity time of about six hours, we would need to store around 4,250 hop fields per combination of upstream path and downstream interface. With support for both IPv4 and IPv6, we can currently store about 160,000 hop fields and with support for only IPv4 or IPv6, we can store about 200,000 hop fields. In the latter case, we can, for example, support three upstream paths and fifteen downstream interfaces. In practice this will depend on the exact beaconing policy and settings used. Note that there might still be room for improvement, as we have not yet optimised the set-up so as to maximise the *MAC verification table* capacity. Also, the number of supported hop fields might be higher if there is more than one border router in the AS. Also note that we did not take into account the one-hop paths here, but there are unlikely to be many of them, and we could use a relatively short expiration time for their hop fields.

6.2 Test setup

We started by testing the implementation on the Tofino software model to verify its functionality. Once that worked, we knew that the implementation would also run on the actual hardware. To run

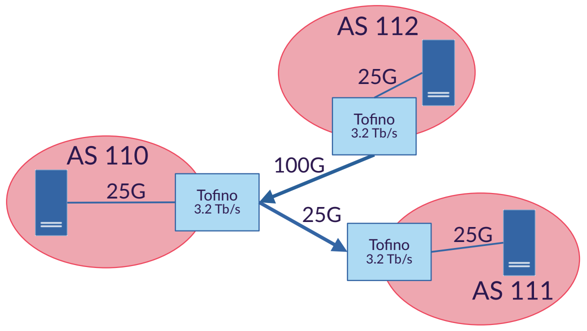


Figure 4: The topology with which we tested our implementation on our testbed. In this topology AS 112 is the core of the isolation domain, which all ASs are part of.

Table 1: Rates observed on the 100G link to AS 112, when traffic is generated and processed in AS 110. The hop field that is used for processing is the last but one in the list.

#hop fields	payload (bytes)	Mpps	Mbps
2	0	90.57	99999.7
	128	46.81	99630.9
	1024	10.73	99817.7
17	0	39.28	99943.3
	128	27.86	99435.6
	1024	9.30	99920.9
33	0	21.43	87434.6
	128	19.57	99903.5
	1024	8.13	99881.4

our implementation in practice, we made use of the 2STiC testbed consisting of P4 programmable equipment at different sites [7]. We used three Edgecore Wedge 100BF-32X switches, which have 32 QSFP28 ports supporting up to 100 Gbit/s per port. In total, each switch has a maximum throughput of 3.2 Tbit/s.

We ran our implementation on a small topology consisting of three SCION ASs at different sites of our testbed (Fig. 4). Every SCION AS consists of one of the programmable switches above to act as the border router and one server connected to provide the other SCION services. With this setup, we showed that the implementation works in a practical scenario and were able to setup communication between all the different ASs.

6.3 Performance

We performed some initial performance tests using the internal traffic generator on the switch located in AS 110. The packets that are generated contain a hop field containing information to forward the packet to AS 112. This way we use the dedicated 100G link between the two ASs and measure at what speed we receive packets at the switch in AS 112, and thus at what speed packets are processed in the P4 implementation. We tested this with only IPv4 support enabled and for different number of hop fields and payload sizes. As can be observed in the Table 1 packets are processed at almost line-rate in most scenarios.

7 RELATED WORK

Součková [20] implemented the old version of the SCION protocol for the NetFPGA, a NIC [3]. As FPGAs offer more flexibility than ASICs in general, they were able to include more functionality in the data plane, such as the verification of the MAC and the expiration time in the hop fields. However, a dedicated ASIC, such as the Tofino, can offer specific functionality with higher performance. In this case that means switching with more ports (up to 64) at higher individual speeds (up to 100G).

Anapaya, an ETH Zurich spin-off company that develops SCION products and operates a production SCION network, developed a proprietary high-speed SCION border router running on commodity x86 hardware [1].

Other new internet architectures which were implemented in P4 are Recursive InterNetwork Architecture (RINA) [16] and Named Data Networking (NDN) [13]. Both architectures have a different focus than SCION, but also include cryptographic operations in the data plane. For RINA, a proof of concept was implemented of an interior router for a so-called software switch [6]. The data plane security components, for which cryptography is used, were considered out of scope. A software switch runs on a commodity server and therefore can be more flexible in functionality compared to an ASIC, but does not achieve similar performance. For instance, there are no hard limits on the number of state transitions in the parser and cryptographic operations can be performed. For NDN, there are several P4 implementations [8, 17, 19], all for a software switch. In [14], a modified NDN architecture is proposed, which can be programmed in P4. To achieve this, a new P4 target architecture is introduced. The target architecture is typically fixed on ASICs, but it can be changed on FPGA-based targets.

In [9], the authors work around the lack of cryptographic operations in the Intel Tofino by sending IPsec packets through the control plane (i.e., the x86 CPU) or to a separate *crypto host*. As a result, AES operations can be performed but not at line-rate. This is a similar approach as we proposed when implementing the EPIC and COLIBRI extensions. In [10], the authors ran into limitations of the NetFPGA when implementing the MACsec protocol in P4.

In [5], an approach is introduced to run AES in the data plane on programmable switches. Such an approach might make it possible in the future to verify hop fields in the data plane on the Tofino. However, at the moment it is not suitable for our use case due to its performance and resource usage.

8 CONCLUSION

With our implementation, we have shown the strength and flexibility of P4 and programmable network devices, which allowed us to run an implementation of a complex protocol such as SCION protocol directly on an ASIC. At the same time, we have shown that we can run SCION on hardware at high speeds, with a first analysis already showing we can almost completely saturate a 100G link. In the process we also established several guidelines that would generally help with making protocols more friendly to be implemented on hardware, namely:

- Use explicit lengths
- Do not use absolute offsets
- Limit the usage of variable length fields
- Do not use complex data structures, such as nested lists

In future work we want to optimise the implementation further and perform a more extensive performance analysis. Also we would like to make our implementation more feature-complete and add support for the EPIC and COLIBRI protocols, which require per-packet cryptographic operations and therefore introduce new challenges.

ACKNOWLEDGMENTS

This work is part of the 2STiC research program (https://www.2stic.nl/).

REFERENCES

[1] [n.d.]. Anapaya CORE. https://www.anapaya.net/anapaya-core
[2] [n.d.]. COLIBRI Service Design. https://scion.docs.anapaya.net/en/latest/ColibriService.html
[3] [n.d.]. NetFPGA. https://netfpga.org/site/#/systems/1netfpga-sume/details/
[4] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (2014), 87–95.
[5] Xiaoqi Chen. 2020. Implementing AES Encryption on Programmable Switches via Scrambled Lookup Tables. In *Proceedings of the Workshop on Secure Programmable Network Infrastructure* (Virtual Event, USA) *(SPIN '20)*. Association for Computing Machinery, New York, NY, USA, 8–14. https://doi.org/10.1145/3405669.3405819
[6] Carolina Fernández, Sergio Giménez, Eduard Grasa, and Steve Bunch. 2020. A P4-Enabled RINA Interior Router for Software-Defined Data Centers. *Computers* 9, 3 (2020). https://doi.org/10.3390/computers9030070
[7] Paola Grosso, Cristian Hesselman, Luuk Hendriks, Joseph Hill, Stavros Konstantaras, Ronald van der Pol, Victor Reijs, Joeri de Ruiter, and Caspar Schutijser. 2021. A National Programmable Infrastructure to Experiment with Next-Generation Networks. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 778–782.
[8] Xingchang Guo, Ningchun Liu, Xindi Hou, Shuai Gao, and Huachun Zhou. 2021. An Efficient NDN Routing Mechanism Design in P4 Environment. In *2021 2nd Information Communication Technologies Conference (ICTC)*. IEEE, 28–33. https://doi.org/10.1109/ICTC51749.2021.9441639
[9] Frederik Hauser, Marco Häberle, Mark Schmidt, and Michael Menth. 2020. P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN. *IEEE Access* 8 (2020), 139567–139586. https://doi.org/10.1109/ACCESS.2020.3012738
[10] Frederik Hauser, Mark Schmidt, Marco Häberle, and Michael Menth. 2020. P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection With MACsec in P4-Based SDN. *IEEE Access* 8 (2020), 58845–58858. https://doi.org/10.1109/ACCESS.2020.2982859
[11] Bob Hinden and Dr. Steve E. Deering. 1998. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460. https://doi.org/10.17487/RFC2460
[12] Intel. [n.d.]. Intel® Tofino™ Series Programmable Ethernet Switch ASIC. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html.
[13] 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* (Rome, Italy) *(CoNEXT '09)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/1658939.1658941
[14] Ouassim Karrakchou, Nancy Samaan, and Ahmed Karmouch. 2020. ENDN: An Enhanced NDN Architecture with a P4-ProgrammabIe Data Plane. In *Proceedings of the 7th ACM Conference on Information-Centric Networking* (Virtual Event, Canada) *(ICN '20)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3405656.3418720
[15] 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)*. USENIX Association, 541–558. https://www.usenix.org/conference/usenixsecurity20/presentation/legner
[16] Vincenzo Maffione, Francesco Salvestrini, Eduard Grasa, Leonardo Bergesio, and Miquel Tarzan. 2016. A software development kit to exploit RINA programmability. In *2016 IEEE International Conference on Communications (ICC)*. IEEE, 1–7. https://doi.org/10.1109/ICC.2016.7510711
[17] Rui Miguel, Salvatore Signorello, and Fernando M. V. Ramos. 2018. Named Data Networking with Programmable Switches. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. 400–405. https://doi.org/10.1109/ICNP.2018.00055

[18] Adrian Perrig, Pawel Szalachowski, Raphael M. Reischuk, and Laurent Chuat. 2017. *SCION: A Secure Internet Architecture*. Springer. https://doi.org/10.1007/978-3-319-67080-5

[19] Salvatore Signorello, Radu State, Jérôme François, and Olivier Festor. 2016. NDN.p4: Programming information-centric data-planes. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. IEEE, 384–389. https://doi.org/10.1109/NETSOFT.2016.7502472

[20] Kamila Součková. 2019. *FPGA-based line-rate packet forwarding for the SCION future Internet architecture.* Master's thesis. ETH Zurich.