A National Programmable Infrastructure to Experiment with Next-Generation Networks

Paola Grosso
University of Amsterdam
Amsterdam, The Netherlands
p.grosso@uva.nl

Cristian Hesselman
SIDN Labs and University of Twente
Arnhem, The Netherlands
cristian.hesselman@sidn.nl

Luuk Hendriks
University of Twente
Enschede, The Netherlands
luuk.hendriks@utwente.nl

Joseph Hill
University of Amsterdam
Amsterdam, The Netherlands
j.d.hill@uva.nl

Stavros Konstantaras
AMS-IX
Amsterdam, The Netherlands
stavros.konstantaras@ams-ix.net

Ronald van der Pol
SURF
Utrecht, The Netherlands
ronald.vanderpol@surf.nl

Victor Reijs
SIDN Labs
Arnhem, The Netherlands
victor.reijs@sidn.nl

Joeri de Ruiter
SIDN Labs
Arnhem, The Netherlands
joeri.deruiter@sidn.nl

Caspar Schutijser
SIDN Labs
Arnhem, The Netherlands
caspar.schutijser@sidn.nl

***Abstract*—We have set up the first multi-domain P4-programmable network in the Netherlands, consisting of six different sites interconnected by SURF's optical network. The purpose of the network is to experiment with novel inter-domain network functions and emerging internet architectures, in particular to improve the transparency, accountability, and controllability of the Internet and future networks. We provide an overview of the testbed's set-up, examples of how we have used it, and our lessons learned.**

***Index Terms*—Programmable networks, testbed, SCION, INT, P4**

I. Introduction

Internet connectivity has become pervasive in the last few decades and has revolutionized our societies. However, it was not designed with the requirements of current applications in mind. This is even more the case for emerging safety-critical applications such as smart energy grids and cooperative driving, which are more stringent in terms of security, stability, transparency, and control over data paths. Furthermore, policy makers, companies, citizens, and other stakeholders around the world increasingly worry about their declining "digital sovereignty" [1], [2]: they more and more depend on systems and services that are operated or manufactured elsewhere (e.g., services in the DNS or 5G network equipment) [1]–[3], but they do not have any insight in or control over how they depend on them.

We set up a testbed to iteratively develop, deploy, and experiment with new network-level systems and protocols that increase the trust and autonomy (sovereignty) of users of next-generation networks, such as a future Internet or networks based on SCION [4], RINA [5], or the "responsible Internet" paradigm [6]. The testbed is part of the 2STiC (Security, Stability, and Transparency for inter-network Communications) research program and currently consists of six sites with P4 programmable equipment [7], a domain-specific language that allows network developers to build data planes of packet forwarding devices. It is the first nation-wide multi-domain P4-programmable network in the Netherlands.

The 2STiC testbed meets our requirements because it constitutes a realistic network with six different operators, three universities and three companies. In addition, the P4 equipment enables us to quickly experiment with new protocols and systems in hardware in a realistic setting, allowing for realistic line-speed forwarding. Traditional network equipment would have been insufficient for our purpose because it comes with vendor-specific software that can be reconfigured but cannot be reprogrammed.

We identified three other testbeds that focus on enhancing network security and/or programmable networks. The *i-P4EN* initiative [8] interconnects P4-programmable networks in other countries, such as Canada, Taiwan and the USA. *FABRIC* [9] calls itself an everywhere programmable nationwide instrument. It features programmable networking technologies such as P4, but OpenFlow as well. *SCIONLab* [10] is a testbed that facilitates experiments with the SCION internet architecture.

In the rest of this paper, we discuss the setup of the 2STiC testbed in more detail. Next, we discuss various use cases that we implemented for the testbed. We conclude with our lessons learned and future work.

II. Testbed Set-up

We use a star-shaped network for the six different sites of the 2STiC testbed, which are distributed across the Netherlands. The network is centered at the SURF site in Amsterdam (SURF is the Dutch National Research and Education Network

operator). The central switch at SURF can be programmed to configure different topologies for the network. Each site contains at least one switch with a P4 programmable Intel Tofino ASIC. We can complement the switches with additional hardware, from a simple server with a programmable Netronome SmartNIC to a complete research group testbed.

Initially, we built the 2STiC testbed based on VLANs over SURF's production infrastructure. Each link between two P4 switches used a dedicated VLAN ID for that link. On SURF's WAN backbone these consisted of 100 Gbit/s Ethernet-based circuits (PBB-TE). On the university campuses, the connection between the P4 switches and the WAN Ethernet circuits needed to be separated from the production traffic. We tried using Q-in-Q (IEEE 802.1ad) [11] for that purpose, but we discovered that some production switches did not properly support it.

To become less dependent on lower layer transport protocols, we decided to separate the 2STiC testbed traffic from the production traffic by using direct links to SURF's optical network. The WAN links of the testbed are now dedicated 200 Gbit/s optical wavelengths. The ports of the P4 switches have a dedicated fiber to the optical transponder equipment. An additional benefit is that the P4 switch ports now transparently connect to each other. An Ethernet frame that is sent by a switch is received by its peer unchanged, as if the two switches are directly connected by a fiber.

P4 enables us to implement any (internetworking) protocol on the hardware in the 2STiC testbed, as long as it aligns with the physical layer of the Ethernet frame. That makes it very suitable for research on new Internet functions and (non-IP) protocols, such as SCION, RINA, and "responsible" networks. The usage of programmable hardware switches and interfaces augments research using network simulators and software switches, because it offers the possibility of verifying results on hardware, for example by showing that a protocol can run on physical switches.

Besides the dedicated optical connections between the testbed nodes, all nodes also have traditional IP connectivity to connect to a management network. All testbed equipment is connected via the Internet to a central system called the jump host, which the members of the 2STiC consortium have access to. From the jump host, it is possible to log in to the rest of the equipment of the testbed. In practical terms, all equipment will be connected to the jump host via a VPN using WireGuard [12]. To ease user administration, we manage authentication centrally through Kerberos, which makes it straightforward to add new users and equipment to the testbed.

III. 2STiC Use Cases

Below we discuss several use cases that we implemented in P4 to run on the testbed.

A. SCION

One of our use cases is a P4 implementation of the SCION "clean slate" internet architecture [4] for the Intel Tofino ASIC. SCION is intended to deliver more stable and secure internet connectivity. At the same time, SCION users will benefit from more control over and insight into the inter-domain routes their traffic takes. Though our implementation is work in progress, it already led to several improvements of the SCION protocol headers, which make them easier and more efficient to implement in hardware and which we fed back to the SCION team. Below we discuss our suggestions that are included in the latest version of the SCION headers.

The first improvement concerns the *forwarding path* in the headers, which captures the path that a packet will traverse (at the level of links interconnecting ASs). From sender to receiver, such a path consists of up to three segments. For each segment as well as each hop that is traversed, some information is embedded in the packet (in the *info fields* and *hop fields*, respectively), such that routers know how to process the packet and where to forward it to. The headers indicate which info field and hop field should be used for processing.

The forwarding path was originally structured as a nested list in the headers: first, the info field for the first path segment, followed by the hop fields of that segment. If present, this was repeated for other segments as well. This is a complex structure, which we discovered can be inefficient to use in hardware where memory is allocated statically. We therefore proposed to flatten the structure and, instead of the nested structure, to use two lists, one for the info fields and one for the hop fields.

In addition, the current info field and the current hop field were referred to using an absolute offset in the packet. In hardware it can be difficult or inefficient to keep track of the exact position in the packet of the byte that is currently being processed. To address this, we proposed to include the indices for these fields. Combined with the new structure of the info and hop fields, this makes it possible to implement a packet parser that is more efficient on certain architectures.

Another factor that complicated packet parsing was the fact that the number of info and hop fields was implicit. This meant, for instance, that determining whether another path segment occurs in a packet was done by comparing the number of processed bytes with the expected number of bytes. This again can be inefficient to implement for certain types of parsers if the position of the byte that is currently processed is not automatically kept track of; therefore, we also proposed to make the number of info and hop fields explicit.

The second improvement pertains to SCION's *address format*. Different types of end-host addresses (IPv4, IPv6, and others) are supported, which can be different in length. The length of an address was implicit in the packet header: only the type of the address was included in the packet. In order to determine the length of an address, the implementation must be aware of that specific type of address to know the corresponding length. However, the intermediate ASs do not process these addresses and therefore only need to know their length. We therefore proposed to include this length explicitly.

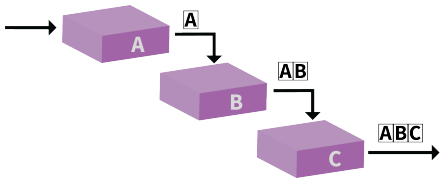

Fig. 1. Adding transited node information to the IPv6 extension header.

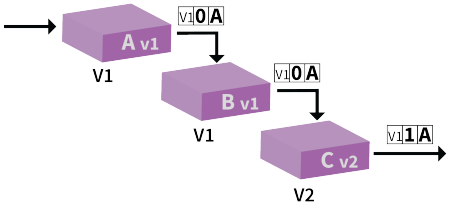

Fig. 2. Adding routing table state to the IPv6 extension header

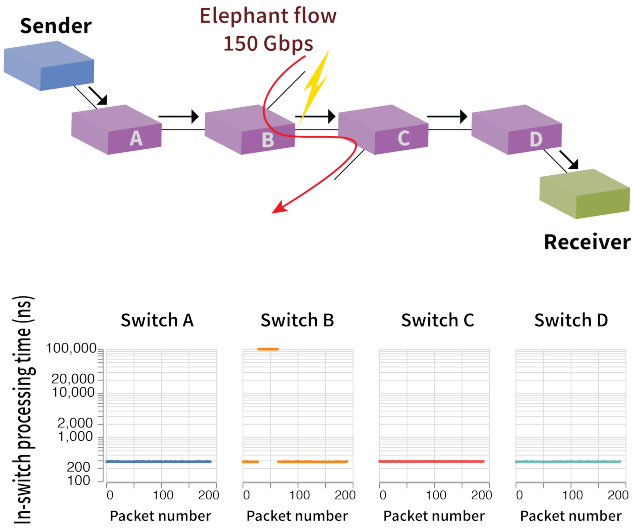

Fig. 3. Latency measurements through information in IPv6 extension headers.

B. IPv6 Path Tracking

Our second use case was to implement path tracking in IPv6, a function that the network utilizes to determine how traffic flowed through the network. We used a software switch with BMv2, INT (In-band Network Telemetry) and the Netronome SmartNICs for our implementation and verified whether the traffic was routed as expected/required or took an unintended path.

At each node, we append a Node Identifier in an IPv6 extension header and extract the complete path a packet took from the extension header at the last node of the path. Fig. 1 illustrates this process, where the data in the IPv6 extension header at the last node contains the three Node Identifiers: A, B and C.

The second chunk of information we add at each node is the routing table version (used to make the forwarding decision), which we also include in the IPv6 extension header. Fig. 2 shows how this works: a Node Identifier field identifies the entry point in the network ("A" in the figure) and a Version field identifies the routing table version at the entry point in the network (which is "V1" in the figure). The second field flags routing table version changes along the path, which is shown by the "1" at the exiting node C, as the routing table changes from V1 to V2.

C. IPv6 In-switch Latency Measurements

For this use case, each P4 switch adds the in-switch latency information to an IPv6 hop-by-hop extension header for each packet in a flow. The switches record timestamps (*In-switch processing time*) when a packet enters the switch and when it leaves the switch. This kind of information is important for operators to understand the latency and jitter added by a network node, as that could influence the overall performance of a network or an application.

Fig. 3 shows our network configuration. Each individual switch adds its *in-switch processing time* to the packet using INT methodology. Using the information recorded by the switches, we can calculate the *latency* for each packet associated with all switches (excluding link latency). Fig. 3 also shows that switch B handles an "elephant flow" going through switches B and C. As a result, switch B's latency increases as the standing queue of the outgoing interface has increased (during packet numbers 25 to 70). This is shown in the bottom graph of Fig. 3. We have not yet assessed the accuracy of the *in-switch procession time* in detail.

D. In-Switch Telemetry Collection with P4 and RDMA

Network telemetry data can take many forms. It may have been inserted into packets using a framework such as Inband Network Telemetry (INT) [13] or by capturing attributes from the packets themselves. Additional data may also be provided by the switch such as ingress port and timestamps.

In our fourth use case, we employed P4 to craft a telemetry message for each packet that passes through the data plane of a device, which typically takes place at high packet per second rates (line speed). By creating a telemetry message for each individual packet, the amount of state the switch needs to track is reduced. However, processing many small packets is more CPU intensive than processing larger packets at an equivalent bit rate.

We used RDMA (Remote Direct Memory Access) to reduce the workload of the system receiving the telemetry messages. With RDMA, a NIC can write data directly to memory without involving the CPU. A program on the collector initializes the RDMA session and provides the switch with the parameters needed to craft RDMA over Converged Ethernet (RoCE) packets, which is a protocol that implements RDMA. The P4 switch subsequently extracts telemetry data from the network traffic and sends it to the collector encapsulated in a RoCE packet, which monitors the buffer and writes the data to disk.

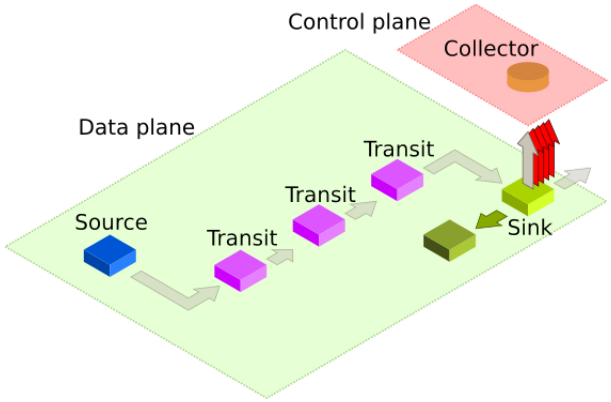

Fig. 4. Two methods of collecting monitoring data: through the control plane or through the data plane.

E. Aggregate In-band Monitoring of Active Flows

Our final use case revolves around collecting in-band monitoring data across routers using P4, enabling path tracking and switch latency measurements. The in-band data is transported in IPv6 extension headers, though similar protocol fields on layer 2 or 3 could be used.

We evaluated two methodologies for gathering/storing such measurement data points (see Fig. 4). Firstly, via the control plane (the red arrows to the orange collector). This typically introduces slow-path performance issues within the switches. Secondly, through the data plane: the INT information is extracted from the traffic and forwarded via the data plane to an external machine (the dark-green collector). The orange and dark-green collectors run dedicated software to process the incoming INT payloads.

Using the 2STiC testbed, we evaluated how those methods perform in different scenarios by varying the nature and volume of traffic, as well as the number and types of INT data points inserted. We found the data plane-oriented data collection to be faster, and closer to the concept of INT.

IV. Lessons Learned

We believe open source-based programmable networks are an important enabler to experiment with and develop mechanisms that improve the security and stability of networks and that give users more insight in and control over their data while "in transit" on the network. However, we learned that open programmable networks also introduce many practical challenges for network designers, programmers, and operators and that they need to go through a steep learning curve.

The first implementation of the 2STiC testbed was based on Ethernet tunnels over the IP layer of the SURFnet network. The P4 switches and the servers connected to them were connected to the campus infrastructure. We used dedicated VLANs to connect pairs of P4 switches and used IEEE 802.1AD (Q-in-Q) to separate the 2STiC traffic from the campus production traffic. However, we found that not all campus switches supported Q-in-Q. We also discovered that trouble-shooting VLANs over multiple domains is difficult and time consuming.

In the second and current implementation of the testbed we connect the P4 switches directly to the optical network of SURF. Pairs of P4 switches are connected via a dedicated 200 Gbit/s wavelength. On the P4 switches these are directly connected to two 100 Gbit/s interfaces. This requires dedicated fiber patches between the P4 switches and SURF's optical equipment. But this was not a problem because universities have enough spare fiber infrastructure on their campuses. This turned out to be a much easier design to implement. It also has the advantage that the P4 switches are transparently interconnected which makes it easier to experiment with non-IP protocols.

Another challenge was the shared nature of the 2STiC testbed. This required us to decide on a secure model for users to access the switches and servers and define procedures to reserve "time slots" on the testbed.

In terms of P4 programming, we learned that implementing novel network technologies in hardware is not a trivial task because protocols such as SCION's are often not designed with high-speed hardware implementations in mind. Implementing new protocols for programmable network equipment allows us to assess their suitability for hardware implementations in an early stage, during which it is still easy to make changes to the protocol. Taking a software-centric approach when designing a new protocol might actually result in a complex hardware implementation. Based on our SCION implementation in P4, we recommend that protocol designers (1) do not use implicit lengths but include them in packet headers; (2) keep their data structures as simple as possible; and (3) avoid the use of absolute offsets, but rather use indices.

In terms of network telemetry, we have greatly increased our understanding of how INT can be implemented. We confirmed our initial expectation that data plane-oriented data collection is fastest and also closest to the in-band concept of INT and we fed these findings back to the OPSAWG/IPPM groups and header extensions discussions in the IETF.

V. Future Work

We are investigating how to extend the interconnection with i-P4EN, FABRIC, and SCIONLab. In this case, our testbed will become part of worldwide initiatives to support distributed network research, providing a development environment for advanced empirical experiments at a global scale.

We will also continue to experiment with new use cases and internet architectures on the testbed to assess how they contribute to secure, stable and transparent inter-network communications.

Acknowledgment

We thank SURF for providing the physical infrastructure of the 2STiC testbed. This work is part of the 2STiC research program (https://www.2stic.nl/), a collaboration of AMS-IX, Delft University of Technology, NDIX, NLnet Labs, SIDN Labs, SURF, University of Amsterdam, and University of Twente. We thank Marijke Kaat for helping with Fig. 1-3.

REFERENCES

[1] EIT Digital, *European Digital Infrastructure and Data Sovereignty – A Policy Perspective*. EIT Digital, June 2020.

[2] ENISA, "Consultation paper – EU ICT industrial policy: Breaking the cycle of failure," July 2019.

[3] Australian Government - Department of Home Affairs, "Protecting critical infrastructure and systems of national significance," August 2020.

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

[5] V. Maffione, F. Salvestrini, E. Grasa, L. Bergesio, and M. Tarzan, "A software development kit to exploit RINA programmability," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–7.

[6] C. Hesselman, P. Grosso, R. Holz, F. Kuipers, J. H. Xue, M. Jonker, J. de Ruiter, A. Sperotto, R. van Rijswijk-Deij, G. C. Moura, A. Pras, and C. de Laat, "A responsible internet to increase trust in the digital world," *Journal of Network and Systems Management*, vol. 28, no. 4, pp. 882–922, 2020.

[7] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[8] J. Mambretti, J. Chen, F. Yeh, and S. Y. Yu, "International P4 networking testbed," SC19 Network Research Exhibition, 2019.

[9] FABRIC testbed. [Online]. Available: https://fabric-testbed.net/

[10] J. Kwon, J. A. García-Pardo, M. Legner, F. Wirz, M. Frei, D. Hausheer, and A. Perrig, "SCIONLab: A next-generation internet testbed," in *Proceedings of the 28th IEEE International Conference on Network Protocols (ICNP)*. IEEE, 2020, pp. 1–12.

[11] "IEEE 802.1ad – IEEE standard for local and metropolitan area networks – virtual bridged local area networks."

[12] J. A. Donenfeld, "WireGuard: Next generation kernel network tunnel," in *24th Annual Network and Distributed System Security Symposium, NDSS*, 2017.

[13] The P4.org Applications Working Group. (2020) In-band network telemetry (INT) dataplane specification – version 2.1. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf