Demo: Towards Rapid Prototyping Network-Slicing Solutions in Software-Defined Networks

Fritz Windisch*, Kamyar Abedi*, Giang T. Nguyen†‡, Thorsten Strufe*‡,
*Chair of Privacy and Security, Kastel, KIT, E-mail: {firstname.lastname}@kit.edu,
†Haptic Communication Systems, TU Dresden, E-mail: {firstname.lastname}@tu-dresden.de
‡Centre for Tactile Internet with Human-in-the-Loop (CeTI)

***Abstract*—As part of the most recent mobile broadband standards, 5G and 6G, network slicing solutions will shape the future of networks. By employing network slicing, network operators can protect participants against different attacks over the network by isolating devices and providing resource guarantees. Currently, to evaluate new network-slicing solutions, network researchers or engineers have to either evaluate the solution in a virtual environment that can not accurately scale to the performance of hardware platforms or invest significant portions of time in network configuration on hardware platforms. To quickly prototype network slicing solutions and other solutions depending on software-defined networking (SDN) features, we introduce a hardware abstraction layer featuring a vendor-independent API to manage hardware platforms and a new testbed solution featuring topologies similar to common network evaluation platforms like mininet.**

In this demo, we showcase a network-slicing solution leveraging SDN. The audience can visually observe the performance of the communication channel before and after denial-of-service attacks and with and without the secure network slicing.

***Index Terms*—Network slicing, Prototypes, Network function virtualization, Software-defined networking**

I. Introduction

With the advent of 5G and 6G mobile communication systems, the techniques of network slicing and software-defined networking gained additional traction in the networking community. Network slicing will provide a more secure and robust communication infrastructure for industrial applications and end users. This is achieved by isolating participants from one slice against other traffic and providing bandwidth and availability guarantees within the slice [1]. As modern communication systems are also employed within critical infrastructure, providing a fail-safe state-of-the-art communication infrastructure is paramount. The second cornerstone technology is software-defined networking (SDN). Using SDN, one can program the data plane (the plane forwarding the actual network traffic) from a control plane component using APIs such as OpenFlow or the P4 runtime. The strengths of SDN lie in greater flexibility of network topologies, cost reduction by vendor independence, and the knowledge of a global state so that traffic can be routed more efficiently [2].

In order to develop and evaluate the network-slicing solutions of the future, making our networks a safer place, a researcher or engineer currently has two possible options. The first option is to use a network emulation tool like mininet [3], distrinet [4], or others. The advantage of these frameworks is that prototyping network topologies becomes significantly easier. In the context of SDN, they usually provide utilities to create control and data plane components and feature-rich utilities to create local test setups. However, the problem with these tools regarding network slicing is that they do not support incorporating real-world hardware into their infrastructure. Due to this, evaluation of scenarios where denial-of-service (DoS) is an expected adversary, like in network slicing, can only be performed with limited accuracy due to the limited volume of traffic that can be transmitted in simulated environments. If a local simulated solution is ported to a real-world solution, it can happen that this solution subsequently fails on a larger-scale DoS, thus wasting expensive development time.

To prevent this from happening, one could develop and evaluate directly on hardware from the beginning. The disadvantage is that a lot of development time will be spent creating an individual solution that needs to follow a valid approach from the beginning. Development time is thus lost on solutions that would already have proven ineffective in local environments. However, creating a network testbed solution supporting different hardware from multiple vendors takes time and effort. Furthermore it has a potential risk to result in a convoluted code base with many corner cases being handled in different situations. Our hybrid testbed combines hardware and (virtualized) software support to speed up development. The testbed provides a hardware abstraction layer and a testbed solution to test new network slicing (and other) solutions directly on vendor-independent hardware.

This demo will showcase our solutions and provide an interactive example to demonstrate a network-slicing solution using our abstraction layer and vendor-independent hardware testbed solution. We will first present our network slicing prototyping solution in section II before describing our live demonstration in section III.

II. Technology

This section will describe our network-slicing prototyping solution and its components. To untangle the code base, we thus chose to introduce two abstraction layers: i) the hardware abstraction layer providing unified functionality for one vendor at a time and a network testbed implementation providing the API for the users. The network testbed layer uses the hardware abstraction layer to deploy the desired functionalities. We will describe both parts, beginning with the hardware abstraction layer.

a) Network hardware abstraction layer: We are implementing the hardware abstraction layer to interact with network hardware, like servers and switches from multiple vendors. For each network hardware vendor or series supported by our layer, we provide a program called a ”worker” that interacts with the specific network hardware via different protocols. For example, we use the P4 runtime to interact with SONiC and Stratum. On SONiC, we also use gNMI. For Aruba switches, Cisco switches, and switches running on PicOS, we are currently using a combination of SSH, Netconf, and OpenFlow. To obtain information about switches, we use SNMP. This architecture is also visualized in figure 1.

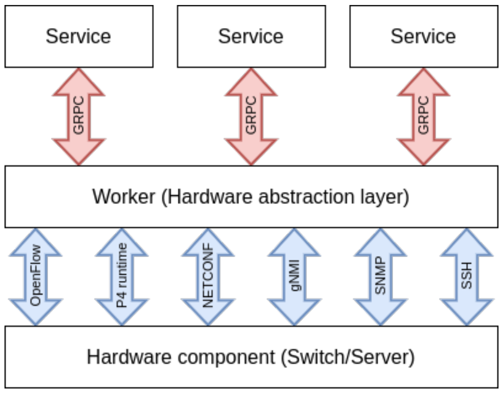
Service
Service
Service
GRPC
GRPC
GRPC
Worker (Hardware abstraction layer)
OpenFlow
P4 runtime
NETCONF
gNMI
SNMP
SSH
Hardware component (Switch/Server)

Fig. 1. Our hardware abstraction layer used to deploy configurations to hardware components like switches or servers. Services (for example our testbed or any other party wanting to configure hardware in a vendor independent fashion) can contact a worker and request or update information, which is then realized by the worker through the use of multiple different protocols. These protocols include OpenFlow, P4 runtime, Netconf, gNMI, SNMP and SSH. The services generally run on another machine. The worker can either be executed on another machine as well, or if the hardware supports it on the managed piece of hardware itself.

The hardware abstraction layer provides a gRPC API for invocations from different services. The API will include management of the general machine configuration, the machine ports, quality-of-service (QoS) configuration, and flow configuration. The general machine configuration will contain names, features, and software version identifiers. The machine port configuration will control the state of specific ports on a machine, including IP address information, VLAN settings, bond settings, and more. The QoS configuration is responsible for creating and managing QoS classes. The flow configuration will realize routing schemes and traffic operations, such as MPLS label management.

The flow configuration must be included in the API, as we currently have a diverse setup of OpenFlow and P4 switches, and full cross-compatibility between the two remains a matter of future work. Even though there is technically openflow.p4, a P4 program able to process OpenFlow instructions written in the P4 language, it is currently unmaintained and not able to handle quality-of-service and other OpenFlow features[1].

Currently, support is planned for SONiC[2], Aruba OS[3], PicOS[4], OpenVSwitch[5], and plain Linux-based servers, as those are currently present in our laboratory environment and already provide great flexibility across vendors. However, we plan to extend our setup in the future to support more options like Cisco IOS and Stratum.

b) Hardware independent network testbed: Our hardware-independent network testbed uses the abstraction layer above to interact with different kinds of hardware. This way, complex topologies can be deployed across vendors. We used a novel approach for our testbed solution called an ”intermediate representation” [5] to create our ecosystem. This term is frequently used in compilers to transform code from source to machine code step by step via multiple intermediate representation languages. In our case, we translate information obtained from user scripts that shape the topology to input for our different tools. We can thus share a common information base among all participating tools that is already prepopulated with all relevant information concerning the topology.

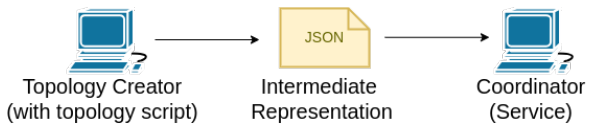

Fig. 2. Architecture of the testbed on mixed infrastructure. The coordinators can run scripts with the intermediate representation as input to deploy topologies to their respective target domain by using the hardware abstraction layer from figure 1. The coordinator from our testbed solution thus acts as a service in our hardware abstraction layer. There can also be multiple coordinators reading only a part of the intermediate representation. Due to this also distributed architectures between multiple different parties/domains are possible (after [5]).

Featuring the capabilities of our intermediate representation, we created multiple tools to generate, modify, and deploy topologies, thus providing an ecosystem of tools around our network testbed solution. Additional advantages of the intermediate representation are the feature to deploy infrastructure from multiple different coordinators (they can share a subset of the same configuration) and the ability to use version control systems like git on the intermediate representation to advance or roll back changes quickly. In figure 2, we outline our typical deployment steps using coordinators from our testbed that deploy functionality through coordinators that then, in turn, use our hardware abstraction layer to deploy the functionality to the servers and switches (see figure 1).

The current tool to generate the intermediate representation uses a topology definition similar to mininet or distrinet to populate information. Tools using the intermediate representation as input currently include command line interfaces, deploy

[1]https://github.com/p4lang/switch
[2]https://sonicfoundation.dev/
[3]https://www.arubanetworks.com/
[4]https://www.pica8.com/picos-software/
[5]https://www.openvswitch.org/

scripts, and a graphical user interface. Figure 3 shows a screenshot of this graphical user interface. Further details on our vendor-independent hardware testbed solution can be found in [5]. It also includes further information about our above claims for the advantages of our intermediate representation and hardware independence.

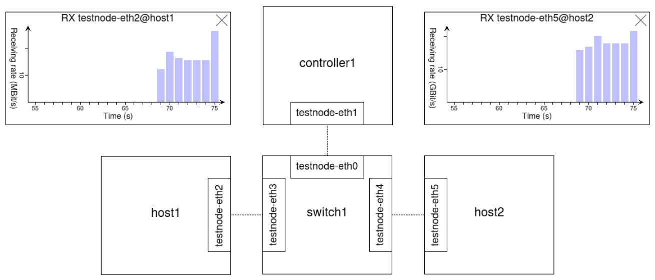

Fig. 3. A screenshot from the graphical user interface of our testbed, featuring a simple SDN topology with one controller, one switch and two hosts connected to the switch. Additionally, two bandwidth measurements are shown for ingress on the two hosts respectively.

III. Demonstration

This section will describe our demonstration at the venue and the key points visitors will take away from it. In a live demonstration, visitors will test a network-slicing solution we built using our hardware abstraction layer and network testbed solution on a hardware switching setup.

Our hardware setup will consist of one switch and two servers out of our test setup shown in figure 4, as well as a laptop to communicate with the servers and manage the network testbed. One switch is enough to establish a communication channel between the laptop and a trusted server, while the other malicious server can attack the connection between them. The visitors can choose between two communication channels to connect the laptop and the trusted server.

First, the visitors can create and attack an unprotected channel. Before attacking it, communication through the channel will be healthy, with good bandwidth and packet loss performance. Our graphical user interface displays both to the visitors on a live updating plot. After beginning a flooding attack with high traffic volume, the visitors will observe the attack impact live via the same plots while being able to adapt the attack to see how various intensities affect the setup. The visitors will observe the degradation of the connection from the live demo.

As a second case, the visitors can then establish a secure network slice via our graphical user interface, which will then be a protected and isolated communication channel. Afterward, the visitors attack this slice again using the same method as previously on the unprotected communication channel. They will observe that the attack does not affect the network slice, as opposed to the previous example. This showcases the effectiveness of network slicing as a defense mechanism against flooding attacks.

The demo thus highlights the importance of network slicing solutions, their successful evaluation by our abstraction layer, and the flexibility of softwarized networks leveraging SDN.

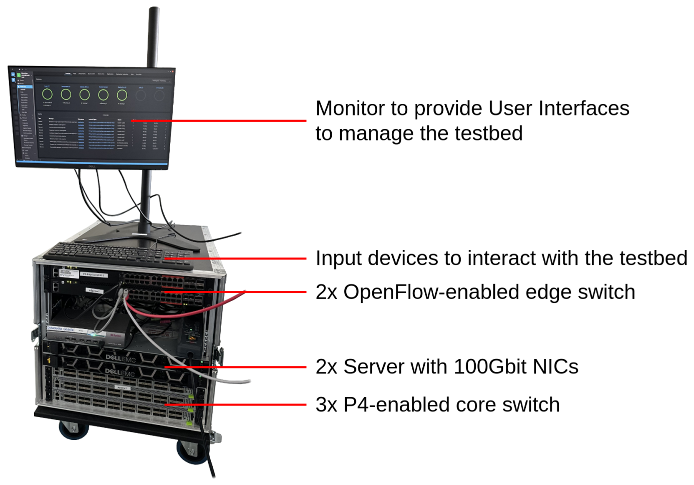
Monitor to provide User Interfaces to manage the testbed
Input devices to interact with the testbed
2x OpenFlow-enabled edge switch
2x Server with 100Gbit NICs
3x P4-enabled core switch

Fig. 4. An image of our network testbed hardware setup, in this image featuring a kubernetes cluster (visualized by OpenLens) and a switching setup using SDN.

Acknowledgment

Funded in part by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) as part of Germany's Excellence Strategy – EXC 2050/1 – Project ID 390696704 – Cluster of Excellence "Centre for Tactile Internet with Human-in-the-Loop" (CeTI) of Technische Universität Dresden, the Federal Ministry of Education and Research of Germany in the programme of "Souverän. Digital. Vernetzt." - Joint project 6G-life - project ID: 16KISK001K and project Open6GHub - project ID: 16KISK010, and the Helmholtz Association through the KASTEL Security Research Labs (HGF Topic 46.23).

References

[1] D. Sattar and A. Matrawy, "Towards secure slicing: Using slice isolation to mitigate ddos attacks on 5g core network slices," in *IEEE Conference on Communications and Network Security (CNS)*, 2019.
[2] M. Karakus and A. Durresi, "Economic viability of software defined networking (sdn)," *Computer Networks*, 2018.
[3] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.
[4] G. Di Lena, A. Tomassilli, D. Saucez, F. Giroire, T. Turletti, and C. Lac, "Distrinet: A mininet implementation for the cloud," *Computer Communication Review*, vol. 51, no. 1, 2021.
[5] F. Windisch, K. Abedi, T. Doan, T. Strufe, and G. T. Nguyen, "Hybrid testbed for security research in software-defined networks," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2023.