Hybrid Testbed for Security Research in Software-Defined Networks

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

***Abstract*—Tele-operations require secure end-to-end Network Slicing leveraging Software-Defined Networking to meet the diverse requirements of multi-modal data streams. Research on network slicing needs tools to develop prototypes quickly that work on emulation and practical deployment. However, state-of-the-art tools focus only on emulation, needing more support for a mixed testbed, including hardware devices. We decouple the topology generating from the actual deployment on destination domains and apply a divide-and-conquer approach. The master coordinator generates an Intermediate Representation (IR) layer, a serialization of the topology. Via a toolchain, the worker coordinators at autonomous systems convert the IR into full or partial deployment scripts. The testbed introduces a marginal overhead by design, allowing for flexible deployment of complex topologies to study secure end-to-end Network Slicing.**

***Index Terms*—Secure Network Slicing, Software-Defined Networks, OpenFlow, Network Function Virtualization, LXC**

I. Introduction

In teleoperation, exchanging commands and multi-modal feedback is a critical process [1]. This communication occurs in video, audio, and haptic signals, each with unique requirements regarding latency, packet loss, and bandwidth. Specific Quality-of-Service (QoS) levels must be met to ensure the highest quality of connection. However, the traditional approach of overprovisioning often needs to be revised in this regard, particularly when the connection spans multiple telecom providers. A more intelligent approach and greater coordination between providers is required, with technology playing a crucial role in this effort. Coordination can be achieved by implementing flexible, reliable solutions requiring secured end-to-end network slicing [2].

Softwarization technologies such as network function virtualization (NFV) [3] and software-defined networking (SDN) [4] are key enablers that introduce computing into the network. This revolution represents a significant paradigm shift away from agnostic data delivery, especially for mobile communication networks. One of the critical benefits of NFV/SDN is the virtualization of the network, which makes it easier to test and deploy new services and applications. This is because the boundary between the test and production environments becomes less noticeable. As a result, network operators can quickly and efficiently test new services and applications before rolling them out to their customers, ensuring a seamless experience for end-users.

In data center testing, network emulations are crucial in creating virtual environments with realistic properties for OpenFlow [5]. However, tools like Mininet [6] must be improved when emulating large networks with high link bandwidths and traffic volume. One of the issues with existing emulation tools is that they consider the testbed homogeneous and under complete control. This, however, is far from the practical conditions where connections pass through autonomous systems that make decisions on their own. As such, having a mix of virtualized and practical testbeds can be challenging, and coordination in such situations can be complex. Existing solutions in this field are often limited by design, so partially accurate simulations (simulations with real-world hardware) are only sometimes supported. This limitation can significantly impact the confidence of actual network performance in a simulation. This also jeopardizes evaluating security and resilience in mission-critical contexts, where real-world hardware would achieve different results. For example, denial-of-service scenarios can only be evaluated in a similar setup that is used for the final deployment of a network topology.

We identify the root cause of the existing limitation on the tight coupling of the topology deployment and the destination deployment infrastructure, which are mainly computing machines. Subsequently, we decouple the topology script from the actual deployment scripts on destination domains and divide-and-conquer approach. The master coordinator generates an intermediate representation (IR) layer, a serialization of the topology. The worker coordinators at autonomous systems convert the IR into full or partial deployment scripts. The resulting testbed allows for flexible deployment of complex topologies with low overhead by design. Furthermore, it allows for the integration of real-world hardware and, thus, the evaluation of the security and resilience of topologies, which is currently not supported by other network testbed solutions.

The rest of the paper is structured as follows. Section II discusses the work on emulation tools in networking research. After describing the design of our testbed in Section III, we elaborate the implementation of the testbed in Section IV and discuss the implications of the testbed in Section V. Finally, Section VI concludes this paper.

II. Related work

This section describes the requirements of a testbed on mixed infrastructure and discusses the most commonly used emulators for SDN-capable networks.

A. General requirements

For our use case, we want to simultaneously deploy a simulated network to multiple virtual and real-world devices. This is required because we want to measure the network's performance in a real-world scenario using real-world network links and devices to use their corresponding physical properties. It is equally essential to manage multiple network parts from multiple locations. Thus, our goal is to create a testbed that can automatically create a simulated network on multiple virtual and real-world devices, supporting a decentralized architecture and multiple link types.

- R1 ***Decentralization of coordination:*** We require decentralization of our coordination to be able to simulate networks with partial trust accurately. This way topologies can be deployed by multiple coordinators that do not need to trust each other. We want to be able to boostrap testbeds across multiple of such domains.
- R2 ***Hybrid***: We further want to integrate real-world hardware in our setup to acquire accurate performance and Quality-of-Service tests, which are essential to evaluate attack scenarios on topologies in security research.
- R3 ***Performance***: We want our testbed abstraction layer to have a footprint that is as small as possible on our network throughput and latency.
- R4 ***Scalability***: Finally, we require scalability, so that our topology can contain an arbitrary number of components and be executed on any network of machines.

Subsequent sections discuss the pros and cons of state-of-the-art SDN-capable emulation tools regarding the above requirements. We categorize related work into two groups: single-host and multi-host emulators.

B. Single-host emulators

Mininet [6] is a network testbed that simulates different SDN components on a single workstation. It uses processes and network namespaces of the Linux network implementation to simulate an entire virtual network. Simulated network layouts are named *topologies*. A Python script typically defines the network topology and can be interacted with by using the mininet command-line interface. As mininet is also distributed as a VM, the image can be shared with others to test on the same network. Additionally, mininet supports in-built network test commands, such as pinging and sending floods via iperf.

It is important to note that mininet cannot distribute trust due to its single-machine or virtual machine (VM) operation. This means that the central coordinator, the machine's administrator, has complete access and control over all testbed components. As a result, it is impossible to simulate the autonomy of multiple domains. However, while mininet cannot integrate real-world hardware, it can still connect different testbed components using virtual Ethernet devices without adding significant overhead on the network side, meeting the necessary performance requirements. It is worth noting that mininet has limitations in terms of scalability, as it is not capable of scaling to multiple machines.

Extending mininet, ComNetsEmu [7] provides an SDN and NFV network emulation environment based on community-embraced open-source packages and enables replicable research and development on limited resource commodity hardware. ComNetsEmu is a network testbed extending mininet by supporting emulation in nested docker containers on an arbitrary docker host as a form of lightweight virtualization. Due to only extending mininet and not changing the possibility to include multiple machines in the emulation, the exact requirements are violated as with mininet. Due to this, it is also unsuitable for our goals. The authors successfully used the software to provide hands-on courses and tutorials on the subject and are looking forward to seeing what the community will be capable of achieving with it.

C. Multiple-host emulators

Many emulators, such as Maxinet [8] and Distrinet [9], [10] address single-host emulations' performance and scalability bottleneck.

Maxinet is a framework that emulates large software-defined data center networks using just a few physical machines. It is an extension of Mininet, enabling the span of an emulated network over several machines. The paper showcases this by emulating a data center with 3200 hosts. Maxinet uses a maxinet frontend server to coordinate multiple maxinet workers. These workers are then used to bootstrap the machines they run on. Thus requirement R1 remains violated because the frontend server is used centrally for coordination and maintains the global state. Integrating real-world hardware will be challenging because maxinet workers only run on Linux machines, thus practically rendering meeting requirement R2 impossible, especially on systems that only support configuration and not code execution. Apart from that, Maxinet uses Generic Routing Encapsulation (GRE) tap tunnels to interconnect different machines to build a mesh of nodes. Using this will reduce the network Maximum Transmission Unit (MTU) and, thus, the bandwidth, apart from firewall issues not accepting GRE packets. This violates our requirement R3. Maxinet does, however, perform well in scale, according to their paper.

Distrinet is a valuable tool for distributing experiments across multiple hosts, preventing the overloading of a single node in critical Mininet scenarios. Distrinet allows for deploying Mininet topologies on multiple nodes, enabling components to communicate across nodes and open up possibilities for more extensive network topologies. To deploy components on worker nodes via SSH, Distrinet uses a master node and a stateful command-line interface that monitors remote services and enables interactions. To establish a Mininet topology, mapping components to executing nodes is necessary. Links between components use VXLAN [11] to create a dynamic overlay network. Testing functions are consistent with those

in Mininet. Distrinet uses the same API as Mininet to run experiments on Linux clusters or Amazon EC2 cloud[1]. It minimizes the number of hosts required for testing and allows for single-host experimentation. It is great for quick prototyping and allows direct running of Mininet scripts in a distributed environment. However, Distrinet's stateful design as a central command-line interface and coordinator makes decentralized and stateless network management impossible, violating requirement R1. Additionally, the VXLAN-only implementation of Distrinet makes it challenging to integrate real-world devices into the network (requirement R2) and creates network overhead in the form of the VXLAN header (requirement R3). Furthermore, the maintenance of Distrinet's codebase has been dormant, making it hard to seek technical support when needed.

In summary, we need different emulations to set up a testbed on a mix of virtualized and hardware infrastructure.

III. Testbed Design

Now we will describe the design of our testbed to meet our previously mentioned use case. We will introduce our intermediate representation as a concept and further describe our individual components. Afterwards, we specify our interaction and monitoring capabilities.

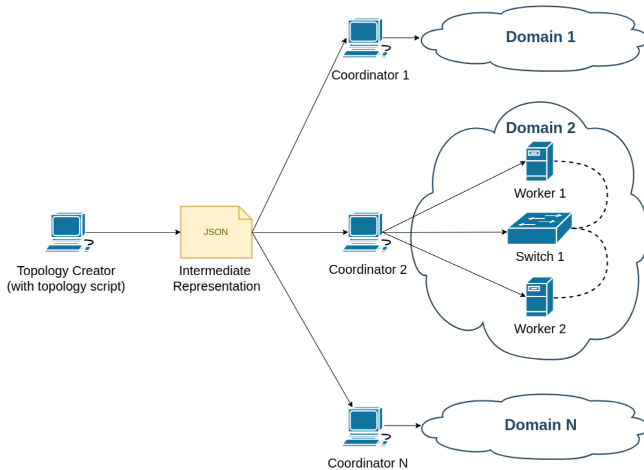

Fig. 1. Architecture of the testbed on mixed infrastructure.

Our approach is to decouple the topology script from the actual deployment scripts on destination infrastructures. There are three novel concepts: 1) The intermediate Representation (IR) layer, 2) Exporter, and 3) Extensions. In the subsequent sections, we elaborate on fundamental abstractions as building blocks for complex networks.

A. Component Abstractions

a) Nodes: : A node is a computing machine or a network switch where the emulator is deployed in part or whole. A node can also be a physical SDN-capable switch in a practical setup as part of the hybrid testbed. The list of all nodes provides a bootstrap list for the emulator for initialization. Nodes represent the underlying infrastructure that an emulated network is overlayed on top of. Figure 1 illustrates a mixed infrastructure consisting of two worker nodes (Worker 1 and Worker 2) and the SDN-capable Switch 1.

b) Hosts: A host is an essential device for computing or networking deployed on a node. Whenever the node is a physical switch, the host assumes the role of a networking device, and its configuration remains consistent across both virtual and real-world networks. In the case of acting as a computing device, multiple hosts can run on a single node; however, running each host in a container or virtual machine is highly recommended to ensure isolation. Moreover, setting CPU and memory limits would be a wise move to restrict the resources.

c) Interfaces: are either real-world or virtual communication endpoints on nodes and hosts, such as linux network devices or switch ports.

d) Links: A link connects nodes or hosts through their interfaces. The emulator offers various link types, both virtual and real-world, to create intricate networks. Each link type has its pros and cons, and users need to be able to configure them accordingly. For example, some links like VxLAN or MAC Virtual LAN (MacVLAN) can make container interfaces accessible to the outside world, while others may have drawbacks such as reduced MTU due to added headers (as with VxLAN) or limited exclusivity (as with binding a host directly to a real-world interface). However, exclusivity can also be an advantage because it provides full access to the device. Routing is accomplished by finding the shortest paths on the simulated network and configuring the routing tables of all hosts accordingly. Additionally, users can apply various Quality-of-Service parameters, such as latency and network jitter, to links.

e) Extensions: An Extension expands a host by commonly used functionality, like add-ons. This is, for example, used for WireGuard [12] tunnels and other network components (apart from links) that can commonly be reused for a whole group of hosts, such as VPN setups. Depending on the underlying environment, an extension does not have to be available in all hosts.

B. Intermediate representation

The purpose of the Intermediate representation (IR) is to allow sharing of topologies among various coordinators and tools in a toolchain, allowing multiple coordinators to initialize network settings from pre-shared configurations. Using an IR, we can create helpful network tools around our topology, such as topology editors, bootstrappers and deployment scripts, Graphical User Interface (GUI), monitoring scripts, and more, without putting all the logic in one program. Without an IR, using only a Python topology script, it is impossible to achieve this because configuration data generated dynamically on other domains might not match our local configuration data (requirement R1), breaking corresponding parts of the simulated

[1]https://aws.amazon.com/ec2/

network. The main objective is to be able to use our topologies in various contexts, whether physical or virtual devices. Therefore, the testbed includes the simulated topology with populated data such as IP and MAC addresses, WireGuard keys, and other dynamic configurations, allowing for sharing across multiple coordinators and tools. Additionally, it should contain all the domain data the coordinator is responsible for and necessary information about external services. The type of external data required depends on the use case and security considerations.

The IR layer provides information to other tools and keeps topology information constant, allowing other states to be automatically determined. The IR aims to ensure that all coordinators have matching views of the global topology when combining their individual views. It includes all topology and dynamic configuration data, such as IP and MAC addresses and WireGuard tunnel keys. These data are sufficient and inclusive to boot up a local part of a testbed. The IEEE 1516 standard uses basic object models to achieve interoperability between actors in simulations, and the IR is similar to these constructs, allowing for decentralized management of the testbed from multiple locations and easy integration of real-world hardware. It is important to note that the IR is simply a serialized topology in JSON format generated from subnets, MAC addresses, and WireGuard key generators. The topology script automatically fills in the topology and uses the IR as input for other tools. The IR is always with the coordinators at the orchestrator PC, and nodes only see SSH commands or shell scripts.

C. Bootstrap, Interaction and Monitoring

The testbed, similar to mininet and distrinet, allows for the generation of topologies using a local script. The resulting representation can then be deployed to multiple nodes using an exporter or bootstrapper. The coordinator manually maps the hosts to nodes based on the available real-world topology. Once deployed, the testbed can be interacted with using monitoring tools that utilize the generated representation as a source of information. The testbed automatically detects available nodes and hosts by polling them through SSH or other protocols. It is up to the user to distribute the load and specify which host goes where. The testbed supports dynamic starting and stopping of nodes and hosts and standard test features like sending traffic, such as ping and flood, and monitoring the results. Alternatively, deployment through bash scripts without network connectivity is also possible.

IV. IMPLEMENTATION

This section describes the technical implementation of the above requirements and design. We begin by describing our used software components and technologies and then give an overview of the technical realization of our testbed solution. The testbed itself has been implemented in *python*, as python is commonly used in research today and is a solid choice for bootstrapping a network due to its script-like nature and easy readability. The emulator's source code is available on Github[2].

A. Software platform

The target of our testbed will be Unix-based systems, even though integration of other hardware should be possible. The coordination of the testbed will always be performed from a Unix-based host to have a standard set of utilities available. This will also make it possible to run the testbed from Windows through Windows Subsystem for Linux (WSL) or MacOSX, even though these possibilities will not be actively tested or maintained. Common Unix command line tools used contain *bash* and *nettools*, which provide a set of networking utilities such as netroute-ip, ifstat, and ping.

a) Hosts: We use the Linux container framework *LXC*[3] to build host containers due to its distinctive advantages. LXC allows for easy management and network orchestration of containers and offers the CPU and memory restriction parameters we require. However, our emulator is independent of any particular virtualization technology. It can also leverage an alternative deployment form with minimum adaptation efforts, such as containing hosts in Linux network namespaces.

b) Links: To build links between containers, we will be using *virtual ethernet pairs* (veth) and standard Linux bridges. We will use *MacVLAN*, *VXLAN*, or direct links to build links between containers and external services. Macvlan links bind on a node interface and receive their IP directly from the real-world network (e.g., via DHCP or static configuration) while allowing multiple MacVLAN connections to share one node interface. VXLAN links bind to a node interface and send/receive data using VXLAN headers. Direct links will bind on a host interface of the corresponding node and use a real-world link completely. All link types offer restricting traffic and limiting QOS parameters by using *qdiscs*, a tool to modify network traffic in real time, which is part of the Unix net-tools mentioned above.

c) Control plane: Further building blocks contain widely-used software tools. The SDN controller uses *RYU* [13], while the SDN switch exploits *Open vSwitch*, and a link encryption utility leverages *wireguard*. For stress testing networks, we use *iperf3*. A control network to interact with the testbed further can be established using MacVLAN devices implemented as a service extension. This will also optionally set up an ssh server on the corresponding service to provide the ability to interact with the service accordingly.

B. Workflow to deploy testbed

To provide decentralization and a stateless testbed as opposed to mininet or distrinet, we will use a different approach when building and deploying network topologies. In this section, we will explain our topology definition and deployment strategies.

[2]https://github.com/FriwiDev
[3]https://github.com/lxc/lxc/

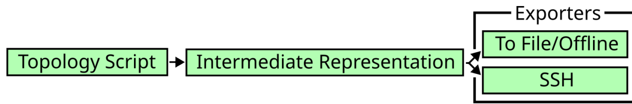

Fig. 2. Basic workflow to deploy a topology script using our solution.

1) Topology Script: We implemented the testbed and the topologies in Python. The user can similarly submit topology definitions as in mininet or distrinet. This will help users migrate their existing networks to our testbed solution. A topology script defines a set of nodes, services, and their corresponding links. The testbed executes this script to obtain the final list of components.

2) Intermediate Representation: Once the topology script completes building the list of components, the topology will be populated (e.g., addresses assigned) and exported to an intermediate representation. This representation contains a full or partial view of the testbed in JSON format that contains static information on the whereabouts of testbed components.

```
# Generate an intermediate representation from our topology script
cd testbed
./generate_topology.sh <topology_script>.py
```

3) Exporters: After successfully generating our intermediate representation, we can deploy the compiled topology through various methods. Two primary options are currently available: *i)* exporting the testbed to a file and *ii)* deploying it live via SSH. Each option offers distinct advantages and limitations, which we will explore in greater depth in the following sections. Ultimately, the deployment method choice will depend on our project's specific needs and goals.

a) To file/Offline: When exporting the testbed to a file, scripts, and resources will be created for each node to deploy the testbed offline. Testbed users can copy the files corresponding to each node onto the respective node and deploy the components using the start and stop scripts. This option is useful when deploying parts of the testbed on a host where SSH is unavailable. It is possible to deploy parts of the testbed offline due to the static design of the intermediate representation. All local and remote components are known, static, and can be targeted statically. However, this is different for MacVLAN links in DHCP-based networks, so additional configuration on the DHCP server and the topology may be required.

```
# Export our topology to file structures that are then used for deployment
cd testbed
./export_topology.sh
# Deploy using our generated start script
cd export/<node_name>
./start.sh
```

b) SSH: We will use SSH as a remote access solution to configure foreign nodes because it has powerful capabilities and is available on almost every system. It is also possible to configure real-world hardware in the testbed automatically by specifying different instructions for these devices. Through the SSH exporter, selected parts of the testbed and the whole testbed can be set up and taken down by issuing remote commands to the respective nodes. However, direct access to the corresponding nodes from the deploying host is required. It is important to note that the testbed will not set up access via SSH on the remote nodes, as this is still up to the user.

```
# Deploy all components via SSH using CLI
cd testbed
./remote_topology.sh start_all
```

A graphical user interface has been created to display the current state of the testbed, which is made possible by connecting to remote nodes, as illustrated in Fig. 3. This interface enables users to start, stop, and delete components and run tests on the testbed. Powerful command-line utilities with similar capabilities and a Python API are also available.

```
# Launch the GUI
cd testbed
./gui.sh
```

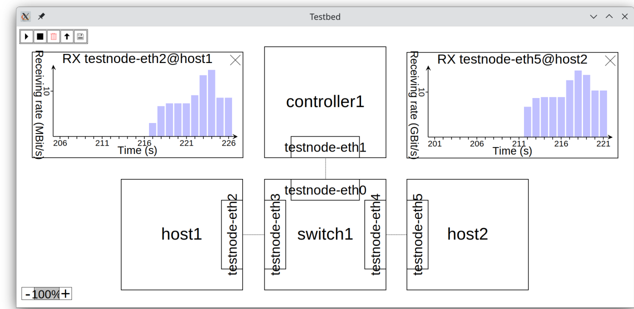

Fig. 3. Graphical user interface featuring an example topology.

C. Monitoring

In order to keep track of the performance and essential data on a testbed in real time, one can use SSH commands to monitor the components on the network. This can be achieved through command-line scripts that provide updates on the status, traffic, and accessibility of the components. There are multiple options to show the statistics, such as via a command-line interface, graphical user interface, or Python API, as illustrated in Fig. 4. Monitoring all nodes is only possible via SSH access, already available through the SSH exporter. It may be possible to extend the monitoring to include other protocols in the future.

V. Discussion

This section discusses the implications of the testbed's design and implementation regarding the design requirements.

A. Decentralization of coordination

Our idea revolves around utilizing an intermediate representation with pre-determined information about the domains involved, such as IP addresses, MAC addresses, and other

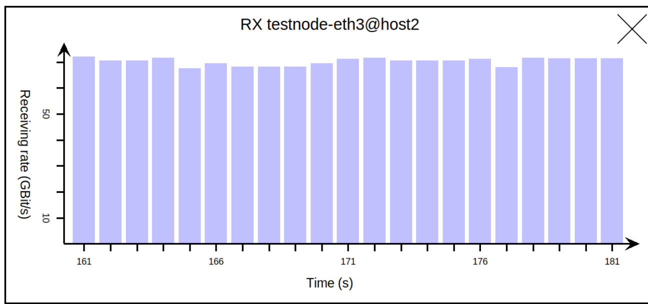

Fig. 4. Screenshot of a network benchmark at a network node.

configuration data. This enables a network administrator, acting as a coordinator, to deploy individual components without understanding the entire topology. It is crucial to consider what information coordinators require to perform their duties securely. Nonetheless, automatic deployment can be accomplished if they have partial access to information from other sources, consequently fulfilling requirements.

B. Hybrid

Our approach to designing hosts and nodes is object-oriented, which enables us to apply abstract configurations to multiple target platforms. Currently, our implementation supports Linux-based platforms, but we can quickly expand it to include other platforms in the future. The testbed supports various link types, including direct links that allow hosts to communicate with other network devices without limitations by binding them to actual interfaces. Additionally, we can reach real-world hardware, which fulfills the requirement of setting up and accessing real-world hardware for the rest of the simulated network. This will also enable us to do security research closer to reality by leveraging the performance of real-world devices.

C. Performance and Scalability

It is essential to consider various deployment types when testing our implementation. We want to ensure we meet the requirement without adding extra cost compared to a baseline setup. There are a few options that we can consider. Firstly, we can use a real-world host. This setup provides equal performance as setting things up manually since the configuration is the same for the host and the node it runs on. Another option is to use a Linux node that runs multiple hosts. This solution is unique because we can run hosts in Linux network namespaces, eliminating container cost penalties. By directly linking the hosts to a real-world interface, we remove any abstraction cost from the network layer and achieve maximum performance. Since both of these scenarios meet the requirement of minimal network performance impact, we can confidently check off this requirement.

Due to our design, we can scale the testbed to multiple machines, thus effortlessly passing the requirement for scalability vertically and horizontally. Subsequently, the testbed's design and implementation meet all four requirements.

VI. Conclusion

We address the challenge of orchestrating testbeds with mixed infrastructure, including virtual and hardware nodes. The testbed decouples the topology-generating process from deployment one by introducing an Intermediate Representation. The testbed seamlessly integrates hardware and virtualized nodes, offering cutting-edge deployment, test, and evaluation mechanics with live monitoring, thus making it ideal for security and network research. We offer a command-line interface, python API, and user-friendly graphical interface, all designed for easy extension to accommodate future distributed network architectures.

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] F. Fitzek, S. Li, S. Speidel, T. Strufe, M. Simsek, and M. Reisslein, *Tactile Internet: With Human-in-the-Loop*. Elsevier, Jan. 2021, publisher Copyright: © 2021 Elsevier Inc.

[2] D. Sattar and A. Matrawy, "Towards secure slicing: Using slice isolation to mitigate ddos attacks on 5g core network slices," in *2019 IEEE Conference on Communications and Network Security (CNS)*, 2019, pp. 82–90.

[3] J. Gil Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[4] M.-K. Shin, K.-H. Nam, and H.-J. Kim, "Software-defined networking (sdn): A reference architecture and open apis," in *2012 International Conference on ICT Convergence (ICTC)*, 2012, pp. 360–361.

[5] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using openflow: A survey," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 493–512, 2014.

[6] 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*, ser. Hotnets-IX. New York, NY, USA: Association for Computing Machinery, 2010.

[7] Z. Xiang, S. Pandi, J. Cabrera, F. Granelli, P. Seeling, and F. H. P. Fitzek, "An open source testbed for virtualized communication networks," *IEEE Communications Magazine*, vol. 59, no. 2, pp. 77–83, 2021.

[8] P. Wette, M. Dräxler, A. Schwabe, F. Wallaschek, M. H. Zahraee, and H. Karl, "Maxinet: Distributed emulation of software-defined networks," in *2014 IFIP Networking Conference*, 2014, pp. 1–9.

[9] G. Di Lena, A. Tomassilli, D. Saucez, F. Giroire, T. Turletti, and C. Lac, "Distrinet: A mininet implementation for the cloud," vol. 51, no. 1, 2021.

[10] G. D. Lena, A. Tomassilli, D. Saucez, F. Giroire, T. Turletti, and C. Lac, "Mininet on steroids: exploiting the cloud for mininet performance," in *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, 2019, pp. 1–3.

[11] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, U. Sridhar, and M. Bursell, "Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks," in *RFC 7348*, August 2014.

[12] J. A. Donenfeld, "Wireguard: next generation kernel network tunnel." in *NDSS*, 2017, pp. 1–12.

[13] N. TELEGRAPH and T. CORPORATION, "Ryu: A component-based software-defined networking framework," in *OSDI '14: 11th USENIX Symposium on Operating Systems Design and Implementation*. Broomfield, CO: USENIX Association, October 2014.