Architecture of a Wireless Transport Network Emulator for SDN Applications Development

Alexandru Stancu(✉), Alexandru Vulpe, Simona Halunga, and Octavian Fratu

University Politehnica of Bucharest, 060042 Bucharest, Romania
{alex.stancu, alex.vulpe}@radio.pub.ro,
{shalunga, ofratu}@elcom.pub.ro

Abstract. Software-Defined Networking is no longer just a paradigm, but a novel architecture that is gaining momentum in the industry, covering all aspects of a network, from campus networks and data-centers, to optical transport and microwave networks. In Wireless Transport networks, the emergence of the Microwave Information Model, ONF TR-532, in late December 2016, is an important milestone in integrating SDN into such networks. Having the right tools for evaluating the model and for developing SDN applications based on it, is very important. This paper proposes an architecture of a Wireless Transport Network Emulator (WTE), based on TR-532, which enables SDN application developers to implement and test such software programs, based on the microwave model, without the need of owning expensive wireless transport devices in the process.

Keywords: Software-Defined Networking · Network Emulator
Wireless transport networks

1 Introduction

With the advancement of technology in aspects such as processing power or increased storage and with the increased multimedia content and streaming on the Internet, traditional networks began to show their limitations. Software-Defined Networking (SDN) is a novel architecture that proposes to mitigate those limitations and thus to advance the networking industry as well, making networks more agile and vendor independent [1].

SDN decouples the networks' data and control planes, moving the control logic inside an external entity, called the SDN Controller, making the network elements simple forwarding devices, which would just move the packets according to what instructions they get from the control plane. This makes the networks programmable through software applications that run on top of the SDN Controller, increasing their agility. In this respect, SDN becomes a key enabler for technologies like Cloud Computing, Internet of Things (IoT), or, more recently, 5G communication networks.

SDN is emerging in all aspects of a network, from campus and data centers to optical transport, wireless transport networks and even Internet exchange points. This momentum is maintained by organizations like the Open Networking Foundation

O. Fratu et al. (Eds.): FABULOUS 2017, LNICST 241, pp. 107–112, 2018.
https://doi.org/10.1007/978-3-319-92213-3_17

(ONF), which militates for the adoption of SDN through the development of open standards and opensource software ecosystems. The Wireless Transport (WT) Project is part of the Open Transport Working Group inside ONF and has been focusing on the development of the Microwave Information Model, a data model that describes in an abstract, vendor independent manner, wireless transport devices. Three Proofs of Concept (PoCs) [2–4]) were successfully conducted inside this project, resulting in the formal release of the Microwave Information Model, as ONF TR-532 [5]. This data model, which refers also another model developed by ONF, the Core Information Model, TR-512, should be implemented by equipment vendors in their devices and can be used by SDN application developers for creating software programs that could ease the management of wireless transport networks. Such applications could be, as demonstrated also in the PoCs: detection of aberrances in the network configuration (comparison between the values configured on the devices and the values from the planning database), performance monitoring, rerouting applications etc.

Since the information model was only recently released, in December 2016, it is not yet available embedded in the wireless transport devices' software. Having the right tools for emulating entire wireless transport networks, that expose this data model, without the need of owning such expensive equipment, is an important advantage for SDN application developers, because they do not need to wait for the model being available in the devices for implementing and testing their software.

The main contribution of this paper is proposing an architecture for a Wireless Transport Network Emulator. Several options are analyzed. The Emulator should have the ability to simulate Network Elements exposing the aforementioned information models, and links between different interface layers of those elements.

The remainder of the paper is organized as follows: Sect. 2 presents similar work that was done in the field of emulators for SDN, Sect. 3 illustrates the proposed architecture of the Wireless Transport Network Emulator, while Sect. 4 concludes the paper.

2 Related Work

Several methods for development and testing of network applications or protocols exist, from experimental testbeds, which are expensive but have the advantage of utilizing the real hardware that the network uses, to simulations or emulation of networks. Simulations are fast and easy to deploy, usually requiring only a laptop or a personal computer, but have an important disadvantage: under the same conditions, the simulation results are not always replicable and consistent. Emulations, on the other hand, are executed in real-time and have replicable results.

Since SDN is not a mature technology yet, only a few options for simulating or emulating software-defined networks exist. The most notorious solution, used on a large scale is Mininet [6, 7]. It is an emulation framework, written in Python, that can create hosts, switches, routers and links between them using a single Linux kernel. Everything is done in software, and because of its flexibility it is easily extendable. The focus of Mininet is on the OpenFlow protocol, which is used between the forwarding devices and the SDN Controller. It is considered an important tool in the context of

SDN, having purposes from testing the OpenFlow protocol to teaching SDN principles to students, as described in [8]. Other tools that can be used in the context of SDN are ns-3 [9], or EstiNet [10]. The former provides support only for the OpenFlow protocol, but it is not developed anymore and it is not compatible with the latest versions, while the latter, although it has an ability to provide performance results for SDN applications in a consistent, replicable and correct manner, it only supports one southbound protocol, the same OpenFlow.

Software tools that provide NETCONF as a southbound protocol do not exist yet, probably because YANG models that describe network devices in an abstract, vendor independent manner, have just started to emerge. The Wireless Transport Emulator, whose architecture is proposed in this paper, is intended to fill in this gap in the research community and provide a simple to use emulator for SDN applications developers, which exposes the recently emerged information models.

3 Architecture of WTE

The architecture proposed for the Wireless Transport Emulator is somewhat similar to Mininet. It was chosen not to extend the existing solution, but to create a new one, for two main reasons: (i) Mininet is based on OpenFlow, and the WTE will be based on the NETCONF protocol, exposing YANG models as defined by TR-532 and TR-512, (ii) Mininet uses a Python API that the user can use for implementing its own network topology, so one would need to know basic Python for creating a custom network; the WTE approach is different, providing a topology configuration file, in JSON format, that describes the Network Elements, different layers of interfaces described by the Core Information Model and links between those interfaces. This provides the ability for the user to create a custom network without the need of having any programming skills, just by describing the topology in a specific format.

The WTE should be able to simulate Network Elements that run a NETCONF server, exposing the aforementioned YANG models through a *management* interface, which should be used by an SDN controller to connect to the simulated device. Also, the emulator should support links between interfaces at different layers, both inside the NE and between different NEs. The hosts emulation is different that Mininet, meaning that they are not separate processes accessible through *ssh*.

The architecture of the WTE is based on two main pillars: a NETCONF server implementation and a Python framework that glues everything together. Each Network Element should be simulated as a docker container, isolated from the rest of the applications of the host that runs the emulator. The docker container should run a Linux image and the NETCONF server inside. With this approach, the NETCONF servers will be separated, not interfering with one another. This offers flexibility and extensibility to the emulator. One can implement its own docker container that runs a NETCONF server of choice, as long as it can be run on Linux. An example of NETCONF server based on TR532 is described in [11]. It is a C implementation that could easily be modified to run inside a docker container, providing the NETCONF functionality that the WTE proposes to offer.

The Python framework should represent the core of the WTE. It should be responsible for interpreting the JSON file that describes the topology of the emulated network, reading any other configuration that the WTE might have and then generating the topology. This generation implies several steps: after the topology JSON file is parsed, the framework is responsible for creating and starting the docker containers associated with the NEs (this implies starting also the NETCONF server inside the container), creating the interfaces described by the user, along with the necessary connections between them, and creating the links between the devices. After everything is set in place, the framework should expose a Command Line Interface (CLI), which is topology aware and provides the user the ability of interacting with both the network and the framework. Also, a cleanup ability is needed in the framework so that everything can be stopped when the emulation has ended, so that the environment can be ready for the next run.

A high-level overview of the WTE components when emulating a simple linear topology containing three devices is illustrated in Fig. 1. The idea behind the emulator is that all traffic that can be passed between the simulated NEs is only Layer 2 traffic (Ethernet), so we should achieve network isolation between the NEs at higher layers. This can be done through docker networks. Each simulated NE, in order to have its own management interface, isolated from the other devices, can have its own docker network. A more detailed overview of a single NE can be seen in Fig. 2.

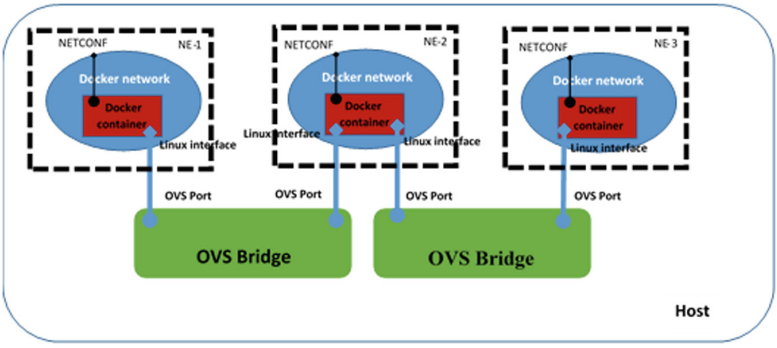

Fig. 1. High-level overview of the WTE components.

There are two proposed solutions for emulating links between two network elements. The first solution implies using a software switch, Open vSwitch (OVS).

Each physical interface defined for a wireless transport device is emulated as a Linux network interface inside its corresponding docker container. For connecting two interfaces from different containers, an OVS bridge can be created and used for connecting the two interfaces, as seen in Fig. 1. Modeling link characteristics, such as bandwidth or latency, can be done using the QoS features provided by OVS. Emulating a single connection between two interfaces implies creating an OVS bridge for each link. If multiple Linux interfaces were to be connected in the same bridge, they would have connectivity between one another and this is not the intention of the emulator.

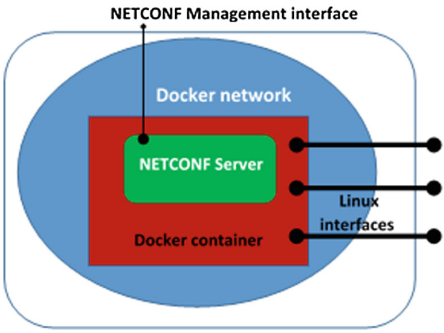

Fig. 2. Overview of the components of a simulated device.

The second proposed solution is more feasible and easier to implement. It consists of replacing the OVS bridge with a *veth* (virtual Ethernet) pair. This feature, which is present in Linux, creates a virtual pipe between the two ends, transferring data between them. The first advantage over the previous proposal is that if an interface will malfunction or set to administratively *down* by the user, the other end will notice that and will go *down*. In the case of an OVS bridge, if one interface connected to the bridge is *down*, the other interface is not affected. The second advantage would be using Linux tool *tc* for emulating the link characteristics, which is easier than using the QoS in OVS.

4 Conclusion

SDN is beginning to have an increased presence in all aspects of the networks nowadays, and through the work done in research communities and in organizations like the ONF, that promotes its adoption through the development of open standards and open-source software ecosystems, it will keep its momentum and will eventually mature as a solution.

Having the right tools that can help both the standardization process and the SDN ecosystem, by aiding software programmers to develop SDN applications from a very early stage, is of vital importance. This paper aims of doing exactly that, by proposing an architecture and the main components of a Wireless Transport Emulator that should aid interested users in simulating wireless transport networks, containing devices that expose the Microwave and Core Information Models.

Acknowledgements. This work was supported by a grant of the Ministry of Innovation and Research, UEFISCDI, project number 5 Sol/2017 within PNCDI III and partially funded by UEFISCDI Romania under grant no. 60BG/2016 Intelligent communications system based on integrated infrastructure, with dynamic display and alerting - SICIAD and by University Politehnica of Bucharest, through the Excellence Research Grants Program, UPB GEX, Identifier: UP - BEXCELENTA2016, project Platform for Studying Security in IoT, contract number 96/2016 (PaSS-IoT).

References

1. Kreutz, D., Ramos, F.M., Verissimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S.: Software-defined networking: a comprehensive survey. Proc. IEEE **103**(1), 14–76 (2015)
2. Wireless Transport SDN Proof of Concept White Paper, September 2015. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/ONF_Microwave_SDN_PoC_White_Paper%20v1.0.pdf
3. Wireless Transport SDN Proof of Concept 2 Detailed Report, June 2016. https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Wireless_Transport_SDN_PoC_White_Paper.pdf
4. Third Wireless Transport SDN Proof of Concept White Paper, December 2016. https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Third-Wireless-Transport-SDN-Proof-of-Concept-White-Paper.pdf
5. ONF TR-532 – "Microwave Information Model", Version 1.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-532-Microwave-Information-Model-V1.pdf
6. Heller, B.: Reproducible network research with high-fidelity emulation. Ph.D. thesis. Stanford University (2013)
7. Lantz, B., O'Connor, B.: A mininet-based virtual testbed for distributed SDN development. ACM SIGCOMM Comput. Commun. Rev. **45**(5), 365–366 (2015)
8. Lisa, Y., McKeown, N.: Learning networking by reproducing research results. ACM SIGCOMM Comput. Commun. Rev. **47**(2), 19–26 (2017)
9. Riley, G.F., Henderson, T.R.: The *ns-3* network simulator. In: Wehrle, K., Güneş, M., Gross, J. (eds.) Modeling and Tools for Network Simulation, pp. 15–34. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12331-3_2
10. Wang, S.-Y., Chou, C.-L., Yang, C.-M.: EstiNet OpenFlow network simulator and emulator. IEEE Commun. Mag. **51**(9), 110–117 (2013)
11. Stancu, A., Vulpe, A., Fratu, O., Halunga, S.: Default values mediator used for a wireless transport SDN Proof of Concept. In: 2016 IEEE Conference on Standards for Communications and Networking (CSCN), Berlin, pp. 1–6 (2016). https://doi.org/10.1109/cscn.2016.7784889