NetIDE: removing vendor lock-in in SDN

R. Doriguzzi-Corin[α], E. Salvadori[α], P. A. Aranda Gutiérrez[β], C. Stritzke[γ],
A. Leckey[δ], K. Phemius[ε], E. Rojas[ζ], C. Guerrero[η]
[α]CREATE-NET, [β]Telefónica I+D, [γ]Fraunhofer IPT, [δ]Intel Labs Europe, [ε]Thales, [ζ]Telcaria Ideas S.L., [η]IMDEA Networks

I. Introduction

The Software-Defined Networking (SDN) paradigm allows networking hardware to be made "malleable" and remotely manageable by the so-called SDN controllers. However, the current SDN landscape is extremely fragmented. Different open and closed source controller frameworks such as OpenDaylight [1], Ryu [2], Floodlight [3], etc. exist. Porting SDN applications from one such platform to another is practically impossible and so, SDN users like network operators face a situation where they are either confined to applications working for the platform of their choice, or forced to re-implement their solutions every time they encounter a new platform.

In this companion paper we present a preliminary version of the NetIDE framework[1], whose ultimate goal is to deliver an integrated environment for SDN development that unifies different controller technologies and allows developers to write, test and deploy applications that are independent from the underlying SDN technology. The main building blocks of NetIDE are (i) the Integrated Development Environment (IDE) that supports the design and development of network applications and (ii) the Network Engine, a runtime environment that provides interoperability of network applications written for different platforms.

The paper is structured as follows: Section II provides a global picture of the NetIDE architecture while Section III presents a preliminary prototype of the NetIDE framework and describes how we want to demonstrate some functionalities implemented as proof-of-concept.

II. NetIDE architecture

NetIDE foresees three different environments (sketched in Fig. 1): (i) the different tools, like the topology editor, the code editors, debuggers, etc. in the **Integrated Development Environment (IDE)**, (ii) the **Application Repository** and (iii) the **Network Engine** where the Network Applications are executed.

The IDE includes a set of editors supporting various network programming languages and provides a graphical editor to specify network topologies. The IDE is both a consumer and a producer of Network Applications, since it needs to be able to read and dissect them and send the different components to their associated editors on the one hand, and to assemble Network Applications from the components produced by the different editors. The Application Repository provides a means to store and retrieve Network Applications without any further manipulation. Finally, Network Applications are deployed on the Network Engine, which is therefore a consumer.

[1]This work is partially supported by the EU FP7 NetIDE project [4] under grant agreement 619543.

978-1-4799-7899-1/15/$31.00 ©2015 IEEE

The Network Engine follows the layered SDN controller approach proposed by the Open Networking Foundation [5]. It integrates a client controller layer that executes the modules that compose a Network Application and interfaces with a server SDN controller layer that drives the underlying infrastructure. In addition, it provides a uniform interface to common tools that are intended to allow the inspection/debug of the control channel and the management of the network resources.

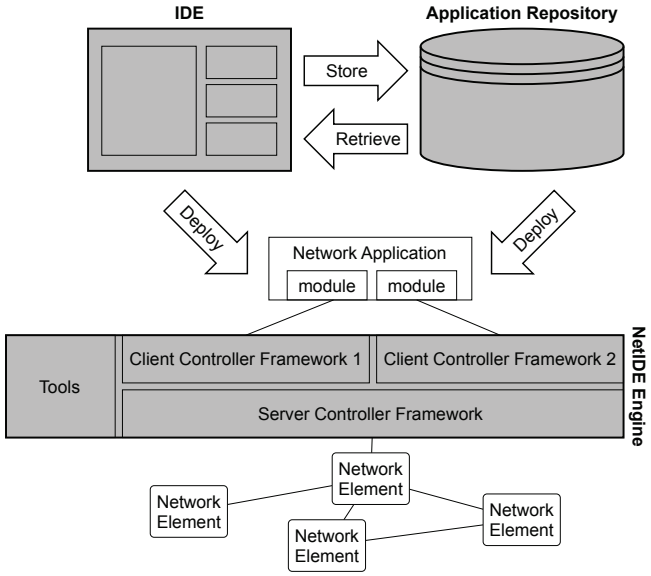

Fig. 1. Workflow between the different NetIDE components.

As shown in Fig.1, the Network Engine architecture foresees applications written for different control frameworks to run simultaneously and controlling/configuring the same physical infrastructure. To this purpose, we are exploring some possible composition mechanisms for the Network Engine that are able to handle the conflicts that may occur with multiple applications running in parallel (a well-know and non-trivial problem already investigated in, e.g., [6], [7] and a few other works).

Finally, the design of the Network Engine is flexible enough to support different control protocols (such as different versions of the OpenFlow specification [8] and OpFlex [9]) and different network management protocols such as Netconf [10] and SNMP [11].

III. Demonstration

The objective of the demonstration is to show two of the NetIDE framework benefits: (i) an **Integrated Development Environment**: one single tool to manage the whole life-cycle

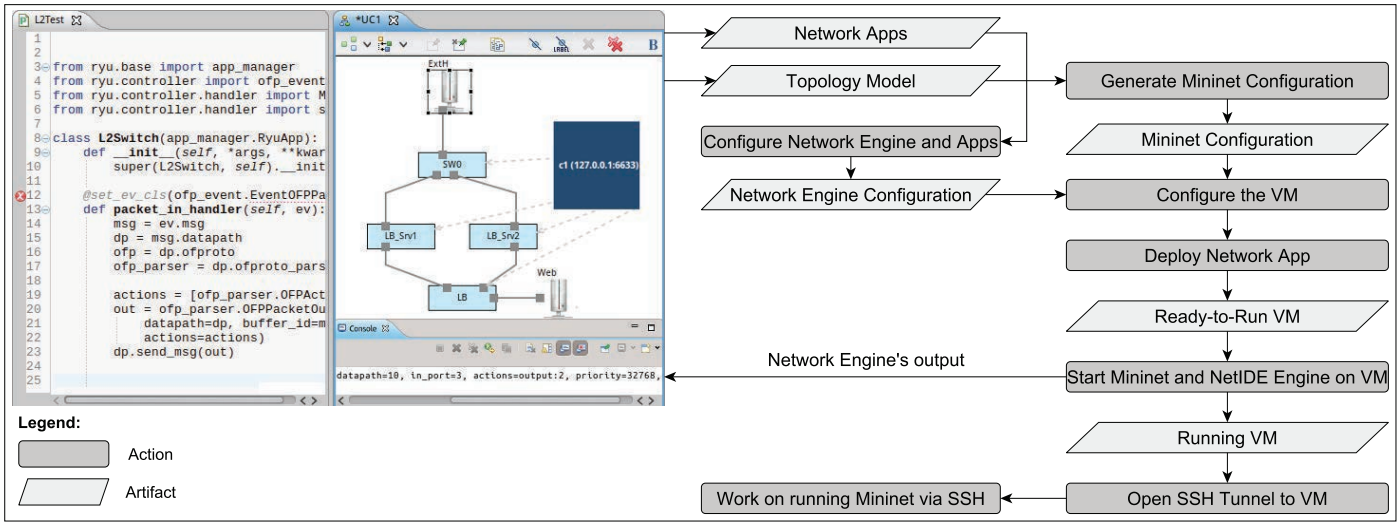

Fig. 2. Workflow of the demonstration.

of a Network Application: from the design, to the implementation, deployment and testing; (ii) **Network Application *re-usability* and *portability***: Network Applications written for many different controller frameworks, e.g. implemented in the past for different environments/needs, can be re-used and executed on top of the controller framework that is currently managing a given network infrastructure or, on the other way around, a well-tested Network Application can be ported and executed "as is" on a second network controlled by another controller framework.
The demo can be broken down into the following steps:

1. We will first use the NetIDE Development Environment to **design** the network topology and to **implement** Network Applications for one or more controller frameworks.
2. Then, we will **deploy** Network Applications and topology to a Virtual Machine (VM) where the Network Engine and a network simulator (Mininet [12]) are installed.
3. We will demonstrate the Network Application re-usability and portability concepts by **running** applications written for a specific controller framework on top of a different framework (e.g., a Floodlight application running on top of Ryu).
4. Besides, we will show **debug** messages coming from the Network Engine on the IDE console.

The workflow of the demonstration is shown in Fig. 2. Essentially, the demonstration starts from the Eclipse-based Development Environment (on the leftmost side of the figure) which is used by developers to write the application's code and that provides a graphical editor for network topologies. Developers can use it to create virtual switches and hosts, assign ports to them, connect the ports and assign controllers to virtual switches.
Another graphical user interface (not shown in the figure) enables developers to link the Network Applications to the Network Engine. In this interface, developers have a list of controllers from which they select the client and the server controller frameworks that will compose the Network Engine (e.g. Ryu, Floodlight or OpenDaylight) and another list of application executables which will run on top of the Network Engine (e.g. Python module, Jar File).

The next step is the generation, through the IDE, of the network simulator configuration (a Python-based Mininet configuration) from the topology model plus the Network Engine configuration.
At this point, we will have a fully configured and running VM where the Network Engine as well as the network simulator are launched automatically via Secure Shell commands from an Eclipse console. Additionally, the Mininet prompt and the console output of the Engine will be accessible from the IDE. Using the prompt, we will generate some traffic on the simulated network, while the console will show the control messages exchanged between the Network Application and the Mininet's switches and that go through the Network Engine.

REFERENCES

[1] "OpenDaylight - A Linux Foundation Collaborative Project." [Online]. Available: http://www.opendaylight.org

[2] "Ryu SDN framework." [Online]. Available: http://osrg.github.com/ryu/

[3] "Floodlight OpenFlow Controller." [Online]. Available: http://www.projectfloodlight.org/floodlight/

[4] "NetIDE website." [Online]. Available: http://www.netide.eu/

[5] "Open Networking Foundation." [Online]. Available: https://www.opennetworking.org/

[6] Jin, Xin et al., "CoVisor: A Compositional Hypervisor for Software-Defined Networks," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015.

[7] Mogul, Jeffrey C. et al., "Corybantic: Towards the Modular Composition of SDN Control Programs," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, 2013.

[8] "OpenFlow Switch Specification version 1.5.0." [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf

[9] M. Smith et al., "OpFlex Control Protocol," IETF, Apr. 2014. [Online]. Available: https://tools.ietf.org/html/draft-smith-opflex-00

[10] R. Enns et al., "Network Configuration Protocol (NETCONF)," RFC 6241 (Proposed Standard), IETF, Jun. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6241.txt

[11] J.D. Case et al., "Simple Network Management Protocol (SNMP)," RFC 1157 (Historic), IETF, May 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1157.txt

[12] "Mininet," http://mininet.org, 2014.