NetIDE: All-in-one framework for next generation, composed SDN applications

P. A. Aranda Gutiérrez$^{\alpha}$, E. Rojas$^{\beta}$, A. Schwabe$^{\gamma}$, C. Stritzke$^{\delta}$, R. Doriguzzi-Corin$^{\varepsilon}$,
A. Leckey$^{\zeta}$, G. Petralia$^{\zeta}$, A. Marsico$^{\varepsilon}$, K. Phemius$^{\eta}$, S. Tamurejo$^{\theta}$
$^{\alpha}$Telefónica I+D, $^{\beta}$Telcaria Ideas S.L., $^{\gamma}$Paderborn University, $^{\delta}$Fraunhofer IEM, $^{\varepsilon}$CREATE-NET, $^{\zeta}$Intel Labs Europe, $^{\eta}$Thales, $^{\theta}$IMDEA Networks

I. Introduction

Software-Defined Networking (SDN) is bringing DevOps [1] capabilities to current networks, reducing the time-to-market for new services and thereby providing a strong incentive for adoption to Service Providers and Network Operators. However, the current SDN landscape is extremely fragmented, so that different open and closed source controller frameworks such as OpenDaylight [2], Ryu [3], Floodlight [4] or ONOS [5] exist. This jeopardises the gains of introducing SDN, since porting SDN applications from one platform to another is time consuming and requires high effort. As a consequence, SDN users (e.g. network operators) face the danger of *vendor* (or *platform*) *lock-in*: they are confined to applications working for the platform of their choice, or forced to re-implement their solutions when they choose a new platform.

In this companion paper[1], we present the consolidated NetIDE framework [6], which delivers an integrated environment for SDN development that unifies different SDN frameworks and allows developers to write, test and deploy applications that are independent from the underlying SDN technology. The project has successfully produced and released to the Free and Open Source Software (FOSS) community (i) an Eclipse-based Integrated Development Environment (IDE) [7], which supports the design and development of network applications and (ii) the Network Engine [8], which allows the composition of network applications written for different platforms into new applications.

In this work, we propose realistic scenarios where the NetIDE framework is not only used to design, implement and test network applications, but also to support the network operator in providing new functions to operational SDN networks without any relevant service disruption.

II. NetIDE overview

NetIDE provides a framework that is composed of: (i) an Eclipse-based IDE called **NetIDE Development Environment**: one single tool to manage the whole life-cycle of a network application, from the design, to the implementation, deployment and testing, and (ii) the **NetIDE Network Engine**, which fosters network application *portability* and *composition*: network applications written for different controller frameworks can be re-used and executed on top of the controller framework that is currently driving a given network infrastructure.

The NetIDE Development Environment provides editors that support network programming languages, a graphical editor to specify network topologies and the interfaces for tools to debug network applications and monitor the network.

The Network Engine (Fig. 1) follows the layered SDN controller approach proposed by the Open Networking Foundation [9]. It integrates a *Client Controller* layer that executes one or more modules of the global network application and a *Server Controller* layer that drives the underlying network infrastructure. In between, the so-called *Core* layer hosts all logic and data structures that are independent from client and server controllers. It also provides a uniform interface to tools to inspect or debug the control channel and manage the network resources.

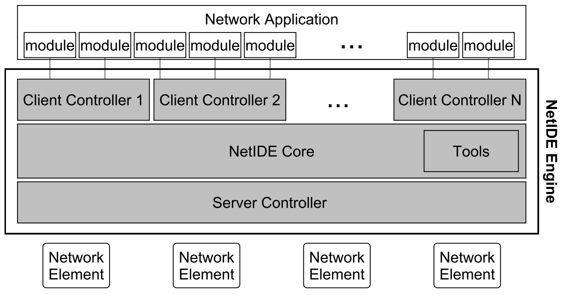

Fig. 1. The NetIDE Engine components.

As shown in Fig. 1, the Network Engine architecture foresees application modules written for different client controllers frameworks. They run simultaneously and are orchestrated by the NetIDE Core, resulting in a global network application that controls the physical infrastructure. To this purpose, the Core implements a composition mechanism that is able to handle the conflicts that may occur between multiple modules running in parallel (a well-known and non-trivial problem already investigated in, e.g., [10], [11] and a few other works). In addition, the design of the Network Engine is flexible enough to support different control protocols such as different versions of the OpenFlow specification [12] and OpFlex [13], and different network management protocols such as Netconf [14] and SNMP [15].

III. Demonstration

The most relevant aspects of the demonstration are represented in Fig. 2 and can be summarized as follows:

1. We first use the NetIDE Development Environment to design the network topology and to build a network application by means of the composition of two or more modules.
2. Then, we start the deployment phase, where the Network Engine is configured to meet the network application requirements in terms of client controller frameworks and

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

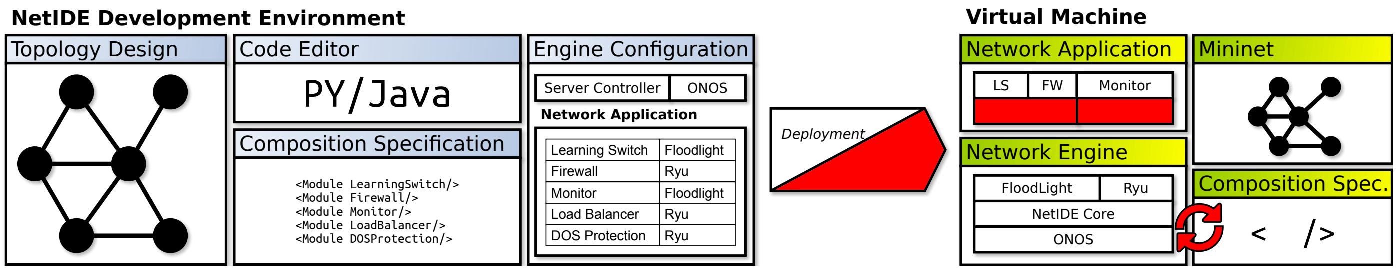

Fig. 2. Graphical representation of the two main phases of the demonstration: design/deployment (white elements) and runtime (red elements).

composition specification, and Mininet is configured to emulate the designed topology.

3. We show how different modules, written for different controller frameworks, cooperate as a single network application on controlling the network.
4. We tune the network application at runtime by stopping/starting/adding modules and reloading the composition specification.

Demo scenario: To demonstrate the concepts listed above, we set up a scenario where a Local Area Network (LAN) is controlled by a Layer 2 forwarding module and protected against unauthorized access by a network element acting as a Firewall. Traffic injected into the LAN is monitored by an SDN application which intercepts relevant control messages and displays them on a terminal.

Design and Development: We start the demonstration by preparing the aforementioned scenario from the NetIDE Development Environment (white elements on left side of Fig. 2), where we design the topology, select/modify a set of SDN application modules (*Learning Switch*, *Firewall* and *Monitor* in the Figure), and finally we prepare the configuration for the Network Engine. Application modules are written either in Java or in Python programming languages and implemented for different controller platforms (in this demo we use Floodlight and Ryu as client controllers). Such modules will cooperate on controlling the network as a single network application thanks to the *Composition Specification*, a set of XML-encoded policies that determine how the NetIDE Core coordinates the modules so that they can operate simultaneously on controlling the same traffic without any conflict.
The output of this phase is a set of artefacts consisting of: (i) a configuration file for Mininet, a configuration file for the Network Engine (the Composition Specification) and the network application as a set of modules written for Ryu and Floodlight.

Deployment: The output of the previous phase is deployed onto a pre-configured Virtual Machine where all the required components are configured and started. As represented by white elements on the right side of Fig. 2, Mininet loads the topology model produced by the NetIDE Development Environment, and the Network Engine is assembled with the required controller frameworks (for the demonstration we will use ONOS as a server controller, although OpenDaylight and Ryu are also supported). As soon as the NetIDE Core loads the Composition Specification, the Network Engine starts controlling the network through the network application based on the policies defined in the Specification.
We demonstrate that the scenario is running as expected by injecting some traffic into the network. Thus, we show that hosts inside the LAN can communicate with each other, the LAN is protected from undesired traffic and it is also monitored to discover possible security breaches coming from trusted users.

Runtime: As a next step, we show how the NetIDE framework can be used to activate new network services at runtime to improve, for instance, performance and security of a production network. Referring to red elements in Fig. 2, we show how the NetIDE Development Environment can be used to tune the Network Engine which is currently controlling the network to satisfy new requirements with minimal service disruption (specifically, without the need of restarting the Network Engine). We add a *Load Balancer* module to spread the users' requests over different web-servers and we install another module that cooperates with the Firewall on protecting the network from Denial of Service (DoS) attacks.

IV. Conclusion

We implemented the NetIDE framework, a tool to manage the whole life-cycle of a network application: from the design, to the implementation, deployment and testing. In this work, we demonstrated how such a framework is also effective in production networks, where SDN functionalities can be dynamically activated or tuned with minimal impact on control and data planes.

References

[1] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, jun 2015.

[2] "OpenDaylight," http://www.opendaylight.org, 2016.

[3] "Ryu SDN framework," http://osrg.github.com/ryu/, 2016.

[4] "Floodlight," http://www.projectfloodlight.org/floodlight, 2016.

[5] P. Berde et al., "Onos: Towards an open, distributed sdn os," in *HotSDN 2014*, Chicago, IL, August 2014.

[6] "NetIDE website," http://www.netide.eu, 2016.

[7] "NetIDE Plugin," http://marketplace.eclipse.org/content/netide, 2016.

[8] "NetIDE Network Engine," https://github.com/fp7-netide/Engine, 2016.

[9] "ONF," https://www.opennetworking.org, 2016.

[10] Jin, Xin et al., "CoVisor: A Compositional Hypervisor for Software-Defined Networks," in *12th USENIX NSDI*, 2015.

[11] Mogul, Jeffrey C. et al., "Corybantic: Towards the Modular Composition of SDN Control Programs," in *12th HotNets*, 2013.

[12] "OpenFlow Switch Specification version 1.5.1," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf.

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

[14] R. Enns et al., "Network Configuration Protocol (NETCONF)," http://www.ietf.org/rfc/rfc6241.txt, IETF, Jun. 2011.

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