At A Glance: NetIDE
An integrated development environment for portable network applications

Project Coordinator: Elio Salvadori

February 12, 2015

Date
Description
Initial structure of the
April 10, 2015
Carmen Guerrero
April 14, 2015
Second Draft for in-
Carmen Guerrero

First Draft for inter-
Carmen Guerrero

Integration of part-
Carmen Guerrero

Debugging Toolset 1st Release

Document Responsible: Carmen Guerrero (IMDEA)
Document Editor: Carmen Guerrero (IMDEA)
Authors: Carmen Guerrero (IMDEA), Elisa Rojas (TELCA)
Roberto Doriguzzi Corin (CREATE-NET), Alec Leckey (Intel)
Andres Beato (TELCA), Sergio Tamurejo (IMDEA)

Target Dissemination Level: PU
Status of the Document: In progress
Version: 2.0, 16:25

Disclaimer:
This document has been produced in the context of the NetIDE Project. The research leading to these results has received funding from the European Community’s Seventh Framework Programme (FP7-ICT) under grant agreement n° 619543.

All information in this document is provided “as is” and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors view.
At A Glance:

NetIDE

An integrated development environment for portable network applications

Project Coordinator

Elio Salvadori

CREATE-NET

Tel: +0461-408400 (1301)
Fax: +39-0461-421157
Email: elio.salvadori@create-net.org

Project website: http://www.netide.eu

Partners:

CREATE-NET (IT)
CZ.NIC (CZ)
Fujitsu Technology Solutions (DE)
IMDEA Networks (ES)
Intel (IR)
Telefonica I+D (ES)
Thales Communication & Security (FR)
University of Paderborn (DE)

Duration: 01/2014 – 12/2016

Funding scheme: STREP

Total Cost: €4.545m

EC Contribution: €2.844m

Contract Number: CNECT-ICT-619543

Main Objectives

Nowadays, while most of the programmable network apparatus vendors support OpenFlow, a number of fragmented control plane solutions exist for proprietary Software-Defined Networks (SDN). Thus, network applications developers need to re-code their solutions every time they encounter a network infrastructure based on a different controller. Moreover, different network developers adopt different solutions as high-level control plane programming language (e.g., Frenetic, Procera, Maple), which leads to network programs that are neither reusable nor shareable. So despite having OpenFlow as the candidate for a standard interface between the controller and the network infrastructure, interworking between different controllers and network devices is hindered and walled gardens are emerging.

NetIDE will deliver a single Integrated Development Environment to support the whole development lifecycle of network controller programs in a vendor-independent fashion.

The project is industry-driven thanks to the involvement of key European players and market leaders such as TID, Intel, Thales, Fujitsu and the contribution from CZ.NIC, an high-tech SME.

NetIDE will constitute a relevant enabling technology for both equipment vendors and service providers, providing the possibility of much shorter development cycles, supporting flexible deployments on multi-vendor infrastructures and agile testing.

Simplifying the Network Apps development process and ensuring their portability are expected to create commercial potential for European telcos and network solution providers.

NetIDE major components will be released according to an open-source license with the aim of promoting the establishment of an open SDN model based on an Open Community of developers.

NetIDE will result in one-stop solution for the development of SDN applications that covers all the development lifecycle.

<table>
<thead>
<tr>
<th>REVIEW</th>
<th>Main reviewer</th>
<th>Second reviewer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary of suggested changes</td>
<td>• one</td>
<td>• two</td>
</tr>
<tr>
<td>Recommendation</td>
<td>( ) Accepted ( ) Major revision ( ) Minor revision</td>
<td></td>
</tr>
<tr>
<td>Re-submitted for review</td>
<td>day/Month/year</td>
<td></td>
</tr>
<tr>
<td>Final comments</td>
<td>Approved</td>
<td>day/Month/year</td>
</tr>
</tbody>
</table>
Abstract:

This document is the second deliverable produced by WP4 and corresponds to the first release of the NetIDE Debugging Toolset. This document includes a initial description of the Debugging Tools and a proposal for integrating the tools in the NetIDE Architecture, an analysis of the related work on debugging tools for SDN ecosystems and a description of the requirements and functionality of each of the tools (Logger, Resource Manager, Garbage Collector, Model Checker, Profiler, Debugger and Simulator).

Finally, this deliverable includes the software release of the NetIDE Logger and its related documentation.

Keywords:

- software defined networks
- networking
- network application
At A Glance: NetIDE

An integrated development environment for portable network applications

Project Coordinator: Elio Salvadori

CREATE-NET
Tel: +0461-408400 (1301)
Fax: +39-0461-421157
Email: elio.salvadori@create-net.org

Project website: http://www.netide.eu

Partners:
CREATE-NET (IT)
CZ.NIC (CZ)
Fujitsu Technology Solutions (DE)
IMDEA Networks (ES)
Intel (IR)
Telefonica I+D (ES)
Thales Communication & Security (FR)
University of Paderborn (DE)

Duration: 01/2014 – 12/2016
Funding scheme: STREP
Total Cost: €4.545m
EC Contribution: €2.844m
Contract Number: CNECT-ICT-619543

NetIDE aims at delivering a single IDE to support the whole development lifecycle of Software-Defined Network controller programs in a vendor-independent fashion. The project includes partners from Italy, Spain, Germany, France, Ireland, Czech Republic.

Main Objectives:

Nowadays, while most of the programmable network apparatus vendors support OpenFlow, a number of fragmented control plane solutions exist for proprietary Software-Defined Networks (SDN). Thus, network applications developers need to re-code their solutions every time they encounter a network infrastructure based on a different controller. Moreover, different network developers adopt different solutions as high-level control plane programming language (e.g. Frenetic, Procera, Maple), which leads to network programs that are neither reusable nor shareable. So despite having OpenFlow as the candidate for a standard interface between the controller and the network infrastructure, interworking between different controllers and network devices is hindered and walled gardens are emerging.

NetIDE will deliver a single Integrated Development Environment to support the whole development lifecycle of network controller programs in a vendor-independent fashion. The project is industry-driven thanks to the involvement of key European players and market leaders such as TID, Intel, Thales, Fujitsu and the contribution from CZ.NIC, a high-tech SME. NetIDE will constitute a relevant enabling technology for both equipment vendors and service providers, providing the possibility of much shorter development cycles, supporting flexible deployments on multi-vendor infrastructures and agile testing.

Simplifying the Network Apps development process and ensuring their portability are expected to create commercial potential for European telcos and network solution providers.

NetIDE major components will be released according to an open-source license with the aim of promoting the establishment of an open SDN model based on an Open Community of developers.

NetIDE will result in one-stop solution for the development of SDN applications that covers all the development lifecycle.
Contents

List of Figures ... v
List of Acronyms ... viii
List of Corrections ... x

1 Introduction 1

2 Related work 3
2.1 OFTest ... 3
2.2 STS ... 4
2.3 TASTE ... 5
2.4 NetSight ... 5

3 NetIDE Debugging Tools Architecture 9

4 NetIDE Debugging Tools Catalogue 11
4.1 Logger ... 11
4.2 Resource Manager and Garbage Collector ... 13
4.2.1 Resource Manager ... 13
4.2.2 Garbage Collector ... 13
4.3 Model Checker ... 14
4.4 Profiler ... 15
4.5 Debugger ... 15
4.6 Simulator ... 17

5 Conclusions and Future Work 19

A NetIDE Logger 1st Release Installation Guide 21
A.1 Installation ... 21
A.1.1 Install RabbitMQ Server ... 21
A.1.2 Running the logger ... 22
A.2 Next Steps ... 22

B Source code repositories 23

C Bibliography 25
At A Glance:

NetIDE

An integrated development environment for portable network applications

Project Coordinator
Elio Salvadori

CREATE-NET
Tel: +39 0461 408400 (1301)
Fax: +39 0461 421157
Email: elio.salvadori@create-net.org

Project website: http://www.netide.eu

Partners:
CREATE-NET (IT)
CZ.NIC (CZ)
Fujitsu Technology Solutions (DE)
IMDEA Networks (ES)
Intel (IR)
Telefonica I+D (ES)
Thales Communication & Security (FR)
University of Paderborn (DE)

Duration: 01/2014 – 12/2016

Funding scheme: STREP

Total Cost: €4,545m
EC Contribution: €2,844m

Contract Number: CNECT-ICT-619543
List of Figures

2.1 OFTest architecture

2.2 STS runs mock network devices, and interposes on all communication channels

2.3 TASTE architecture

2.4 NetSight architecture

2.5 Processing flow used in NetSight to turn packets into packet histories across multiple servers

3.1 NetIDE Debugging Tools Architecture

4.1 NetIDE Logger Architecture

4.2 Communication between the Logger and the Shim Layer

4.3 Debugger tool in the NetIDE tool architecture

4.4 Components of the debugger tool architecture
At A Glance:

NetIDE

An integrated development environment for portable network applications

Project Coordinator

Elio Salvadori

CREATE-NET

tel: +0461-408400 (1301)
fax: +39-0461-421157
email: elio.salvadori@create-net.org

Project website: http://www.netide.eu

Partners:

CREATE-NET (IT)
CZ.NIC (CZ)
Fujitsu Technology Solutions (DE)
IMDEA Networks (ES)
Intel (IR)
Telefonica I+D (ES)
Thales Communication & Security (FR)
University of Paderborn (DE)

Duration:

01/2014 – 12/2016

Funding scheme:

STREP

Total Cost:

!4.545m

EC Contribution:

!2.844m

Contract Number: CNECT-ICT-619543

An integrated development environment for portable network applications

NetIDE aims at delivering a single IDE to support the whole development lifecycle of Software Defined Network controller programs in a vendor-independent fashion. The project includes partners from Italy, Spain, Germany, France, Ireland, Czech Republic.

Main Objectives

Nowadays, while most of the programmable network apparatus vendors support OpenFlow, a number of fragmented control plane solutions exist for proprietary Software Defined Networks (SDN). Thus, network applications developers need to re-code their solutions every time they encounter a network infrastructure based on a different controller. Moreover, different network developers adopt different solutions as high-level control plane programming language (e.g. Frenetic, Procera, Maple), which leads to network programs that are neither reusable nor shareable. So despite having OpenFlow as the candidate for a standard interface between the controller and the network infrastructure, interworking between different controllers and network devices is hindered and walled gardens are emerging. NetIDE will deliver a single Integrated Development Environment to support the whole development lifecycle of network controller programs in a vendor-independent fashion.

The project is industry-driven thanks to the involvement of key European players and market leaders such as TID, Intel, Thales, Fujitsu and the contribution from CZ.NIC, an high-tech SME. NetIDE will constitute a relevant enabling technology for both equipment vendors and service providers, providing the possibility of much shorter development cycles, supporting flexible deployments on multi-vendor infrastructures and agile testing. Simplifying the Network Apps development process and ensuring their portability are expected to create commercial potential for European telcos and network solution providers.

NetIDE major components will be released according to an open-source license with the aim of promoting the establishment of an open SDN model based on an Open Community of developers.

NetIDE will result in one-stop solution for the development of SDN applications that covers all the development lifecycle.

NETIDE FP7 ICT Objective 1.1 Future Networks
List of Acronyms

AMQP    Advanced Messaging Queueing Protocol
DPRL    Data Path Requirement Language
NetIDE  NetIDE
OF      OpenFlow
SDN     Software Defined Networking
SNMP    Simple Network Management Protocol
WAN     Wide Area Network
At A Glance:
NetIDE
An integrated development environment for portable network applications

Project Coordinator
Elio Salvadori

CREATE-NET
Tel: +0461-408400 (1301)
Fax: +39-0461-421157
Email: elio.salvadori@create-net.org
Project website: http://www.netide.eu

Partners:
CREATE-NET (IT)
CZ.NIC (CZ)
Fujitsu Technology Solutions (DE)
IMDEA Networks (ES)
Intel (IR)
Telefonica I+D (ES)
Thales Communication & Security (FR)
University of Paderborn (DE)

Duration: 01/2014 – 12/2016
Funding scheme: STREP
Total Cost: !4.545m
EC Contribution: !2.844m
Contract Number: CNECT-ICT-619543

An integrated development environment for portable network applications

NetIDE aims at delivering a single IDE to support the whole development lifecycle of Software-Defined Network controller programs in a vendor-independent fashion. The project includes partners from Italy, Spain, Germany, France, Ireland, Czech Republic.

Main Objectives

Nowadays, while most of the programmable network apparatus vendors support OpenFlow, a number of fragmented control plane solutions exist for proprietary Software-Defined Networks (SDN). Thus, network applications developers need to re-code their solutions every time they encounter a network infrastructure based on a different controller. Moreover, different network developers adopt different solutions as high-level control plane programming language (e.g. Frenetic, Procera, Maple), which leads to network programs that are neither reusable nor shareable. So despite having OpenFlow as the candidate for a standard interface between the controller and the network infrastructure, interworking between different controllers and network devices is hindered and walled gardens are emerging. NetIDE will deliver a single Integrated Development Environment to support the whole development lifecycle of network controller programs in a vendor-independent fashion.

The project is industry-driven thanks to the involvement of key European players and market leaders such as TID, Intel, Thales, Fujitsu and the contribution from CZ.NIC, an high-tech SME. NetIDE will constitute a relevant enabling technology for both equipment vendors and service providers, providing the possibility of much shorter development cycles, supporting flexible deployments on multi-vendor infrastructures and agile testing. Simplifying the Network Apps development process and ensuring their portability are expected to create commercial potential for European telcos and network solution providers.

NetIDE major components will be released according to an open-source license with the aim of promoting the establishment of an open SDN model based on an Open Community of developers.

NetIDE will result in one-stop solution for the development of SDN applications that covers all the development lifecycle.
List of Corrections
1 Introduction

The objective of the NetIDE WP4 Debugging Tools is to provide a framework that enables the developer to systematically test, profile, and tune the Network App. The first efforts to achieve this objective in the NetIDE project are focused on identifying the set of tools needed in Software Defined Networking (SDN) environments. Currently, we have identified the following categories: Logger, Simulator, Model Checker, Resource Manager, Garbage Collector and Profiler.

The challenges for developing these tools can be summarized in the following questions that will be tackled during the lifetime of the project:

- **what the specific requirements of each tool in a SDN ecosystem, in general, and in NetIDE project, in particular are:**
- **how each tool is integrated in the overall NetIDE architecture (WP2) and in the IDE (WP3);**
- **how the design and development of each tool will evolve in parallel with the design and development of the NetIDE architecture; and**
- **what the impact in terms of scalability and performance of each tool is**

This deliverable documents the first release of the NetIDE Logger tool. This first tool lets us to prove a preliminary architecture approach for integrating the tools in the NetIDE architecture. In parallel to this first software release, we have started working on other tools and have reached different levels of consolidation. The first step has been to identify the functionality and requirements of each tool. In some of them, we have proposed an architecture for integrating them into the NetIDE architecture.

Although this deliverable is described as a prototype in the DoW, we document in this deliverable the current status of the work conducted on designing and developing the full set of tools in addition the information related to the software release proper.

Next steps are focused on identifying the requirements of each tools in terms of communications capabilities within the NetIDE architecture. A first version of the NetIDE Logger lets us proof the feasibility of the publish-subscribe approach based on message queuing. Specific requirements for runtime tools (as Resource Manager and Garbage Collector) and development tools (as Simulator, Debugger, Profiler or Model Checker) will drive the further design of the communications functionality of the proposed architecture.

This document is organized as follows: Chapter 2 includes an analysis of the current state of the art on debugging tools in SDN ecosystems, Chapter 3 describes a preliminary architecture for integrating the tools in the overall NetIDE architecture, Chapter 4 includes the software architecture of the NetIDE Logger and an initial description of the functionality and requirements of each NetIDE Debugging tool. Conclusions and next steps are provided in Chapter 5. Appendix A includes the installation guide of the NetIDE Logger and Appendix B shows the NetIDE Github repository details for the NetIDE Logger.
At A Glance:
NetIDE
An integrated development environment for portable network applications

Main Objectives

Nowadays, while most of the programmable network apparatus vendors support OpenFlow, a number of fragmented control plane solutions exist for proprietary Software-Defined Networks (SDN). Thus, network applications developers need to re-code their solutions every time they encounter a network infrastructure based on a different controller. Moreover, different network developers adopt different solutions as high-level control plane programming language (e.g. Frenetic, Procera, Maple), which leads to network programs that are neither reusable nor shareable. So despite having OpenFlow as the candidate for a standard interface between the controller and the network infrastructure, interworking between different controllers and network devices is hindered and walled gardens are emerging.

NetIDE will deliver a single Integrated Development Environment to support the whole development lifecycle of network controller programs in a vendor-independent fashion. The project is industry-driven thanks to the involvement of key European players and market leaders such as TID, Intel, Thales, Fujitsu and the contribution from CZ.NIC, an high-tech SME. NetIDE will constitute a relevant enabling technology for both equipment vendors and service providers, providing the possibility of much shorter development cycles, supporting flexible deployments on multi-vendor infrastructures and agile testing. Simplifying the Network Apps development process and ensuring their portability are expected to create commercial potential for European telcos and network solution providers.

NetIDE major components will be released according to an open-source license with the aim of promoting the establishment of an open SDN model based on an Open Community of developers. NetIDE will result in one-stop solution for the development of SDN applications that covers all the development lifecycle.
2 Related work

Debugging and troubleshooting have been important subjects in computing infrastructures, parallel and distributed systems, embedded systems, and desktop applications [5]. The two predominant strategies applied to debug and troubleshoot are runtime debugging (e.g., gdb-like tools) and post-mortem analysis (e.g., tracing, replay and visualization). Despite the constant evolution and the emergence of new techniques to improve debugging and troubleshooting, there are still several open avenues and research questions [6].

Debugging and troubleshooting in networking is at a very primitive stage. In traditional networks, engineers and developers have to use tools such as ping, traceroute, tcpdump, nmap, netflow, and SNMP [7] statistics for debugging and troubleshooting. Debugging a complex network with such primitive tools is very hard. SDN offers some hope in this respect. The hardware-agnostic software-based control capabilities and the use of open standards for control communication can potentially make debug and troubleshoot easier. The flexibility and programmability introduced by SDN is indeed opening new avenues for developing better tools to debug, troubleshoot, verify and test networks.

In this chapter, we briefly describe and analyse four different developments for debugging and troubleshooting in SDN namely OFTest [1], STS [2], TASTE [8] and NetSight [4], which we considered the most relevant for the initial development of the tools for the NetIDE Network App Engine.

2.1 OFTest

OFTest [1] is a framework and test suite to test compliance to the OpenFlow (OF) specification. It tests basic functionality of OF 1.0 and 1.1, and support for 1.2 is currently in development. The system diagram for OFTest is shown in Fig. 2.1. The test fixture (the OFTest server) connects to both the control plane and the data plane of the switch under test. It coordinates OF commands with data plane stimulus and monitoring.

- OFTest starts with the very basics of OF but provides a framework for development of more complicated tests
- It was used as the primary vehicle for validating OF 1.1 (see the oft-1.1 branch in the git repository)
- A prototype implementation of an OF 1.1 switch, OFPS, was implemented in the same framework as OFTest. (Also in the oft-1.1 branch)
- Parts of OFTest are being adapted and used for standards based compliance testing

Summary: OFTest is developed in Python, it focuses on Floodlight [8] controller framework and it supports the following OF versions: 1.0, 1.1 and 1.2. Code available at [9].
2.2 STS

STS [2] is a proposal which shares some authors from an older SDN debugger proposal called OFRend [10]. It discusses how to improve control software troubleshooting by presenting a technique, retrospective causal inference, for automatically identifying a minimal sequence of inputs responsible for triggering a given bug. Retrospective causal inference [2.2] works by iteratively pruning inputs from the history of the execution, and coping with divergent histories by reasoning about the functional equivalence of events. Their technique is applied to five open source SDN control platforms or frameworks, namely Floodlight [8], NOX [11], POX [12], Pyretic [13] and ONOS [14].

The retrospective causal inference technique is supposed to be especially valuable for troubleshooting distributed controllers running complex applications, which are recently becoming available to the public and the broader research community. STS is focused on SDN control software, but it is also intended to be applicable to general distributed systems. Also, neither ndb [2.4] nor OFRend address the problem of diagnostic information overload: with millions of packets on the wire, it can be challenging to pick just the right subset to interactively debug, while
retrospective causal inference is the first system that programmatically provides information about precisely what caused the network to enter an invalid configuration in the first place.

**Summary:** STS is developed in Python, it focuses on the following controller frameworks: Floodlight [8], NOX [11], POX [12], Pyretic [13] and ONOS [14], and it initially supports OF 1.0. Code available at [15].

### 2.3 TASTE

TASTE [3] proposes an adaptation of test-driven software development methodologies to SDN. To support their methodological guidelines, they propose an expressive requirement formalization language. Further, they describe a prototype tool able to check the compliance of an SDN controller with requirements expressed in the proposed language [2,3]. Their evaluation of the prototype shows promising results on the practical viability of their approach.

![TASTE architecture](image)

**Figure 2.3:** TASTE architecture [3]

**Summary:** TASTE uses an extension of the FML language called Data Path Requirement Language (DPRL). Their prototype runs on top of the Mininet platform. Their tool is developed in Python, but it does not focused on any specific SDN controller framework and it does not specify the OF versions supported. Code available at [16].

### 2.4 NetSight

While previous proposal focus on debugging of SDN applications, NetSight [3] is a project which includes different tools for SDN projects: ndb (which can be considered the initial proposal [17].
that later evolved into the whole NetSight project), netwatch, netshark and nproof, which are an interactive network debugger, a live network invariant monitor, a network packet history logger and a hierarchical network profiler, respectively. Their architecture is shown in Fig. 2.4.

![NetSight Architecture](image)

**Figure 2.4: NetSight architecture**

NetSight assembles packet histories using postcards (Cf. Fig. 2.5), event records sent out whenever a packet traverses a switch. This approach decouples the fate of the postcard from the original packet, helping to troubleshoot packets lost down the road, unlike approaches that append to the original packet. Each postcard contains the packet header, switch ID, output port, and current version of the switch state. Combining topology information with the postcards generated by a packet, NetSight can reconstruct the complete packet history: the exact path taken by the packet along with the state and header modifications encountered by it at each hop along the path.

**Summary:** NetSight is developed in C++, it focuses on the following controller frameworks: NOX [11], POX [12] and RipL-POX [18], but it does not specify the OF versions supported. Code is available at [19].
Figure 2.5: Processing flow used in NetSight to turn packets into packet histories across multiple servers [4]
At A Glance: NetIDE

An integrated development environment for portable network applications

Project Coordinator
Elio Salvadori

CREATE-NET
Tel: +0461-408400 (1301)
Fax: +39-0461-421157
Email: elio.salvadori@create-net.org

Project website: http://www.netide.eu

Partners:
CREATE-NET (IT)
CZ.NIC (CZ)
Fujitsu Technology Solutions (DE)
IMDEA Networks (ES)
Intel (IR)
Telefonica I+D (ES)
Thales Communication & Security (FR)
University of Paderborn (DE)

Duration: 01/2014 – 12/2016
Funding scheme: STREP
Total Cost: €4,545,000
EC Contribution: €2,844,000
Contract Number: CNECT-ICT-619543

An integrated development environment for portable network applications

NetIDE aims at delivering a single IDE to support the whole development lifecycle of Software-Defined Network controller programs in a vendor-independent fashion. The project includes partners from Italy, Spain, Germany, France, Ireland, Czech Republic.

Main Objectives

Nowadays, while most of the programmable network apparatus vendors support OpenFlow, a number of fragmented control plane solutions exist for proprietary Software-Defined Networks (SDN). Thus, network applications developers need to re-code their solutions every time they encounter a network infrastructure based on a different controller. Moreover, different network developers adopt different solutions as high-level control plane programming language (e.g. Frenetic, Procera, Maple), which leads to network programs that are neither reusable nor shareable.

So despite having OpenFlow as the candidate for a standard interface between the controller and the network infrastructure, interworking between different controllers and network devices is hindered and walled gardens are emerging. NetIDE will deliver a single Integrated Development Environment to support the whole development lifecycle of network controller programs in a vendor-independent fashion.

The project is industry-driven thanks to the involvement of key European players and market leaders such as TID, Intel, Thales, Fujitsu and the contribution from CZ.NIC, an high-tech SME. NetIDE will constitute a relevant enabling technology for both equipment vendors and service providers, providing the possibility of much shorter development cycles, supporting flexible deployments on multi-vendor infrastructures and agile testing.

Simplifying the Network Apps development process and ensuring their portability are expected to create commercial potential for European telcos and network solution providers.

NetIDE major components will be released according to an open-source license with the aim of promoting the establishment of an open SDN model based on an Open Community of developers.

NetIDE will result in one-stop solution for the development of SDN applications that covers all the development lifecycle.
3 NetIDE Debugging Tools Architecture

Fig. 3.1 illustrates a first version of the architecture of the NetIDE Debugging Tools. The main modules that leverage the communication between the Engine and the Tools are based on a broker approach that implements the message exchange between applications in an open and flexible way.

![NetIDE Debugging Tools Architecture](image)

**Figure 3.1: NetIDE Debugging Tools Architecture**

The main modules of this architecture are the following:

- **LogPub** module integrated in the *shim layer* is the responsible of intercepting all the messages that arrive to the shim layer (i.e. the messages exchanged between the networks elements, NE, and the client controller) in order to replicate and send them to the broker element.

- **Broker Server** module is an intermediate element that capture the message produced by the first module. Each tool will be able to retrieve these messages accessing to this broker. It is composed by two principal actors, the Exchange and the Queue. The Exchange is a very simple element that receives messages from producers (*LogPub*) and then pushes them to queues. This element must know exactly what to do with a received message (append the message to a particular queue, several queues or discard it). A Queue is the element where the messages are stored. Many producers can send messages that go to one queue and many consumers can try to receive data from one queue.

- The **Tool** module is the specific module for each tool and is responsible of retrieving the messages from the queue and execute the action coded on the tool for each message.
The communication between these three modules of this architecture is based on the Advanced Messaging Queueing Protocol (AMQP) implemented using RabbitMQ. AMQP is an open standard application layer protocol for exchanging messages between applications. In our particular case, these applications are the NetIDE Debugging Tools and the shim layer of the Engine. The communication is established across a connection with RabbitMQ server, known as broker. This element, the broker, could be in a local or remote machine. Before sending messages, it is necessary to make sure the recipient queue exists. If we send a message to non-existing location, RabbitMQ will just trash the message. Hence, the second step is to define the exchange element and the queue element where the messages will be stored. Before exiting the program we need to make sure the network buffers were flushed and our message was actually delivered to RabbitMQ. The LogPub module is connected to RabbitMQ Server by defining the Exchange and the Queue elements, making possible that the messages sent exchange are stored in a queue. In order to retrieve the messages, we need to connect the specific tool to RabbitMQ Server.

The next steps for enriching this communication between the server controller framework and the debugging tools are focused on identifying the requirements of each tools in terms of communications capabilities within the NetIDE architecture. A first version of the NetIDE Logger lets us prove the feasibility of the publish-subscribe approach based on message queuing, however this proposed architecture could be limited in the case of other tools. A deep evaluation of the proposed architecture tightly coupled to the Engine need to be conducted in order to evaluate the performance and scalability of the overall solution. Specific requirements for runtime tools (as Resource Manager and Garbage Collector) and development tools (as Simulator, Debugger, Profiler or Model Checker) will drive the further design of the communications functionality in this architecture. Further research is also needed towards analyze the scalability and performance of this proposed architecture.
4 NetIDE Debugging Tools Catalogue

This chapter includes an initial description of the set of NetIDE Debugging Tools to be developed within WP4 during the second and third year of the project. The first release of this tools includes a initial design of the NetIDE Logger as proof-of-concept of the integration of the tools in the new NetIDE architecture produced withing the WP2.

4.1 Logger

In this section we describe the first implemented tool of the toolset proposed in NetIDE for analyzing and diagnosing the network, the NetIDE Logger. It is the first tool developed as a proof-of-concept of the new NetIDE architecture designed in WP2 and the initial version of the proposed architecture for integration of the tools proposed in WP4 and described previously in Chapter 3.

The NetIDE Logger captures the messages exchanged between the data plane (network elements) and the control plane (client controller, where a network application is executed) and print these messages into a screen, allowing to the user to visualize and filtering them accordingly to the type of message. It is based on RabbitMQ (software to queue messages). A complete description of this technology is included in the Deliverable D2.3. The shim layer is modified to include a module that queue the messages required by the tool. In the specific case of the NetIDE Logger, these messages are all the ones that the shim layer intercepts.

The software architecture of the tool and its integration within the overall architecture of NetIDE project are detailed as follows and illustrated in the following Fig. 4.1.

The LogPub module, integrated inside the shim layer, is responsible of intercepting all the messages that arrive to the shim layer (i.e. the messages exchanged between the networks elements, NE, and the client controller) and replicating and sending back to them to the broker element. The second module called Logger will retrieve the messages kept in the broker and will print them in a terminal.

The Fig. 4.2 represents two instances of the second module, Logger (1) and Logger (2), which are consuming the messages kept in the broker and sent by the LogPub. Firstly, the first module establishes a connection with the RabbitMQ server (broker) and captures the messages in order to send to the element X known like Exchange. This new element is in charge of knowing what to do with the messages. In our case, it has been configured for sending the messages to the different queues depending on the type of message that the user wants to visualize. If there is no consumer (an instance of Logger module running) there will not be any queue, hence the Exchange element will drop the messages received.

Finally, the second module (Logger) retrieves the messages of the queues and print them in a terminal. The user has the possibility to choose what kind of message wants to observe.

There are two types of messages, in or out. The former, are messages sent by the Network Elements to the Client Controller and they are represented in green color. The latter, are the messages which do the reverse path (from the controller to the network) and are represented in yellow.
At A Glance:

NetIDE

An integrated development environment for portable network applications

Main Objectives

Nowadays, while most of the programmable network apparatus vendors support OpenFlow, a number of fragmented control plane solutions exist for proprietary Software-Defined Networks (SDN). Thus, network applications developers need to re-code their solutions every time they encounter a network infrastructure based on a different controller. Moreover, different network developers adopt different solutions as high-level control plane programming language (e.g. Frenetic, Procera, Maple, ...), which leads to network programs that are neither reusable nor shareable.

So despite having OpenFlow as the candidate for a standard interface between the controller and the network infrastructure, interworking between different controllers and network devices is hindered and walled gardens are emerging. NetIDE will deliver a single Integrated Development Environment to support the whole development lifecycle of network controller programs in a vendor-independent fashion.

The project is industry-driven thanks to the involvement of key European players and market leaders such as TID, Intel, Thales, Fujitsu and the contribution from CZ.NIC, an high-tech SME.

NetIDE will constitute a relevant enabling technology for both equipment vendors and service providers, providing the possibility of much shorter development cycles, supporting flexible deployments on multi-vendor infrastructures and agile testing. Simplifying the Network Apps development process and ensuring their portability are expected to create commercial potential for European telcos and network solution providers.

NetIDE major components will be released according to an open-source license with the aim of promoting the establishment of an open SDN model based on an Open Community of developers.

NetIDE will result in one-stop solution for the development of SDN applications that covers all the development lifecycle.

---

Figure 4.1: NetIDE Logger Architecture

Figure 4.2: Communication between the Logger and the Shim Layer
4.2 Resource Manager and Garbage Collector

Resource Manager and Garbage Collector are tools for the Network Engine that are both foreseen as external modules for the Shim Layer and that will interact with the network devices with the purpose of optimizing the usage of some of the most critical resources such as: link bandwidth, TCAM memory and energy.

Differently from the other tools like, for instance, Logger and Debugger, Resource Manager and Garbage Collector are not used during the development or testing phases but at running time instead, when the application is already deployed and controlling the network traffic.

Although a preliminary version of Resource Manager and Garbage Collector will be released at month 24 with v0.3 of the NetIDE framework [20], in this section we anticipate the requirements and some of the functionalities that will be implemented within these two components.

4.2.1 Resource Manager

The role of the Resource Manager is to handle and globally optimize the usage of network resources that are under the control of the NetIDE Engine. In this preliminary stage of investigation we foresee two major areas of application for the envisioned operations: enriched feedback to the applications and global network and memory re-optimization.

As just mentioned, we foresee the Resource Manager helping the Network Applications in pursuit high end-to-end network performance. Thus, the goal of the Resource Manager in this case is to collect the network statistics (such as port statistics) or other useful information (such as CPU load and TCAM usage on the switches) with the purpose of providing feedback to the Network Applications, informing it about the multiple options/paths.

An additional feature we envision is the automatic network re-configuration aiming at optimizing parameters such bandwidth utilization or usage of memory. In this case, we foresee for instance a mechanism that not only chooses the links with maximum residual bandwidth, but that could also keep into consideration other parameters, such as the TCAM residual storage space, with the purpose of optimally reducing and distributing the flow entries among the switches through an aggressive use of wildcards (e.g.: flows with "similar" headers should follow the same paths as much as possible).

4.2.2 Garbage Collector

Like for the Resource Manager, the Garbage Collector is foreseen as a component that optimizes the performance of the network resources. In particular, similarly to the analog tool for the Java programming language which automatically restores the memory occupied by objects that are no longer used by a software program, our component removes the unused permanent flow entries (i.e. entries with infinite idle and hard timeouts) from the TCAMs of the switches.

In our case, the Garbage Collector removes all the entries installed by the applications running on top of a specific Client Controller once this controller disconnects from the Server Controller (which is always running).

We are also investigating one possible extension for the Backend component of the Network Engine with the purpose of associating an identifier to each application running on top of a Client Controller. This additional information will allow the Garbage Collector to react and clean the flow tables not only when a Client Controller disconnects but also, in a more efficient way, when a single application is turned off. This can happen with Client Controllers like OpenDaylight which leverage on the OSGi framework that allows for dynamic loading/unloading/starting/stop of Java modules (the Network Applications) without bringing down the controller.
4.3 Model Checker

Testing OpenFlow applications is challenging because the behavior of a program depends on the larger environment. The end-host applications sending and receiving traffic and the switches handling packets, installing rules, and generating events all affect the program running on the controller. The need to consider the larger environment leads to an extremely large state space, which explodes along three dimensions: Large space of switch state, large scale of inputs packets and large scale of events ordering.

While there is recent research in formal verification of SDN networks, model checking the controller behavior as it updates the underlying network has only seen limited application. The objective of the NetIDE Model Checker is to prove the correctness of the network controller program for a given network topology, and an arbitrarily large number of packets. The NetIDE project is exploring the existing approaches that are characterized by limited to verifying the controller for a small number of exchanged packets in the network. In our case, we extend the state of the art by presenting abstractions for model checking controllers for an arbitrarily large number of packets exchanged in the network.

The key challenge in model checking software defined networks is state space explosion resulting from the following factors. (1) There can be a large number of packets alive in the network, resulting in a large buffer state. Further, these packets create interference for each other by sending events to the controller which then trigger updates to the network state. (2) These packets and the corresponding events can have arbitrary interleavings. (3) The switch flow tables store a mapping from the packet header and the input port to the output port, resulting in a large network state modern switches can have tens of thousands of flow table entries. Existing work in the verification of SDNs exploits the fact that the time between updates to the network state by the controller is much larger than the lifetime of a packet through the network. Thus, the state evolution of the network can be viewed as updates from one network configuration to another via intermediate (transient) states, as per the commands from the controller.

Static verification verifies a fixed configuration of the network by using either symbolic simulation or by model checking.

Incremental verification approaches extend the static verification approach and incrementally verify the network for all the network configurations. The property may however be violated in the transient stage.

Safe update builds upon the incremental verification approach by guaranteeing that the property under check holds during the transient stage by using specific update protocols.

Dynamic checking seeks to verify the properties on the network in the presence of arbitrary updates from the controller, even when no specific update protocol has been implemented to ensure the property. This is the space of our work in NetIDE and close related to NICE and FlowLog.

We are going to use a model checking based approach to successfully find important bugs in controller code. Previous works check the controller for a bounded number of exchanged packets. We are going to challenge by extending dynamic checking and scale to an arbitrarily large number of packets.

To address an initial implementation of the NetIDE Model Checker, we are going to base the design on the NICE tool that tests unmodified controller programs by automatically generating carefully-crafted streams of packets under many possible event interleavings. To use NICE, the programmer OpenFlow controller program supplies the controller program, and the specification of a topology with switches and hosts. The programmer can instruct NICE to check for generic correctness properties such as no forwarding loops or no black holes, and optionally write additional, application-specific correctness properties (i.e., Python code snippets that make assertions about

---

1Commodity switches can have in the order of ten megabytes of packet buffering per switch
the global system state). By default, NICE systematically explores the space of possible system behaviors, and checks them against the desired correctness properties. The programmer can also configure the desired search strategy. In the end, NICE outputs property violations along with the traces to deterministically reproduce them. The programmer can also use NICE as a simulator to perform manually-driven, step-by-step system executions or random walks on system states. NICE Tool combines model checking with symbolic execution to identify relevant test inputs for injection into the model checker.

### 4.4 Profiler

The NetIDE Profiler is an specific network analysis tool to profile any collection of network links to understand the traffic characteristics and routing decisions that contribute to link utilization. NetIDE will adopt a first proposal of this tool based on the hierarchical profiler nprof within the NetSight framework. The work during this second year will be to evaluate the main functionalities of NetSight nprof tool and migrate them to the NetIDE architecture. The approach of starting with an initial tool already developed let us to quickly prototype a initial version of this tool and identify the main challenges of it within the NetIDE ecosystem.

The NetIDE Profiler will combine the resulting packet histories with the topology information to provide a live hierarchical profile, showing which switches are sourcing traffic to the link, and how much. The profile tree can be further expanded to show which particular flow entries in those switches are responsible. The NetIDE Profiler can be used to not only identify end hosts (or applications) that are congesting links of interest, but also identify how a subset of traffic is being routed across the network. This information can suggest better ways to distribute traffic in the network, or show packet headers that cause uneven load distributions on routing mechanisms such as equalcost or weighted-cost multi-path.

### 4.5 Debugger

After having studied in detail different debuggers in\(^2\) (namely OFTest, NetSight, TASTE and STS), in this section we describe our debugger tool proposal. This solution is based on the ‘postcard idea’ of NetSight\(^3\), but it has been adapted to the NetIDE tools architecture. The postcard idea is basically converting OF messages into a more legible format for debugging (postcard format). Thus this debugger tool takes care to record all relevant information about the configuration of the network and send it to a centralized module to convert it to legible format and later analysis. In summary, this debugger tool provides a wide visibility of the network\(^4\) in real time and it also provides postmortem analysis.

Referring to above, the requirements of this proposal are the following:

- Having a higher visibility of the network.
- Developing an integrate module into shim layer (considering the specific NetIDE tools architecture).
- The debugger tool needs to receive OF messages.
- The debugger tool needs collect all the amount of information required by the final user to be checked later during debugging time.

\(^2\)By *visibility of the network*, we mean the ability of the person using the tool of being aware of the events being carried out in the network. Thus, we consider this tool will augment that visibility, because users of this tool will be able to follow up more clearly what is actually going on in the network.
Our debugger tool proposal, as we said before, is based on the NetSight project. Indeed the NetSight debugger tool creates the postcards with data plane information and with control plane information, however our solution is thought to work at the control plane level, because from our point of view, the control plane provides all necessary information to debug a network. The proposed architecture is shown in Fig 4.3.

![Figure 4.3: Debugger tool in the NetIDE tool architecture](image)

Seeing the above architecture, the LogPub module (which works integrated into the shim layer) must intercept all OF messages sent between the controller and switches and it must send a duplicate of those OF messages to RabbitMQ. Then the RabbitMQ will send the messages to the debugger tool, where these will be processed as it is explained below. The components of the debugger tool are shown in Fig 4.4.

The **Postcard Generation** module will receive the OF messages and will make the conversion of these messages into a postcard format. The postcard format will contains a copy of the packet’s header and the switch ID. This postcard format will initially be based on the NetSight approach.

The **Topology Generation** module will create the topology of the network based on postcards of each switch. If we have all postcards of a switch, we have all relevant information of that switch configuration, the matches installed, the matches removed, etc. A summary of the process for building the topology is described below:

- Sort postcards by switch ID.
- Build the topology based on the matches tables of each switch.
After postcard creation and topology creation, the idea is that we can check the state of the network from two ways: by using the topology or by looking for a specific postcard, for example the IDE could permit us to make searches about specific postcards.

Considering the future work, first of all, it is necessary to study the code of the NetSight project, because is the starting point to develop this debugger tool. Currently, in the NetSight project there are four applications developed ‘nbdl’, ‘netwatch’, ‘nprof’ and ‘netshark’, but only one of these applications is published online, thus a first step would be contacting the development group of the NetSight project to obtain more information about their solution and their developed application. Therefore, in our proposal it is also necessary to study what possibilities we can provide at the IDE to use this debugger tool, for example, as explained above, to allow searching or filtering the previously saved postcards.

Other future task is looking for other recent debugger tool solutions and keeping this developments tracked, which can be useful to improve this approach, as well as the other tools.

### 4.6 Simulator

The objective of the NetIDE Simulator tool is to provide an environment to emulate network events to allow for quick prototyping and performance prediction. A network may operate efficiently under standard load, but may fail as traffic levels increase due to several reasons. The ability to see these indicators will help the developer to include this in their design. As the developer will require different events to test operation under, the tool should come with some samples included. But to increase the takeup of the tool, the event generation model should be extendable to allow developers to create their own custom events that may need to be catered for in their design. The tool will provide a list of configurable events (eg. standard flow, traffic spike, traffic with certain QoS flags, etc) that can be selected from that will test the Network Application in question. This will allow the developer to make edits to their code to make the network more performance during these events.

The main requirements of this tool are:

- An engine that generates traffic to run through the network
- Rate of traffic through the engine should be configurable to simulate network load or spikes. The different traffic generation models [21] of Greedy Source model, Poisson Traffic model, Long-tail traffic models, Payload data models are examples of what could be selected.

- The engine should be extendable to allow developers to generate custom events.

- An instrumentation module will query performance metrics to see how the network operated during these events.

- A simple user interface or reporting screen to show these indicators.

To support the features the Simulator is required to implement, there are a number of existing open-source solutions which could be leveraged to reduce the development time of this tool:

- Mininet [22] provides an off-the-shelf network emulator. As it supports parameterized topologies, it can be used to create flexible topologies which can be configured based on developer provided parameters.

- MaxiNet [23] extends the Mininet to span the network emulation across several physical machines. This allows it to emulate very large SDN networks and add Wide Area Network (WAN) use cases to the Simulator Tool.

- A traffic generator would be required to send packets through the network. There are a number of opensource tools which could be utilized, eg, ostinato [24], Cat Karat [25], Nemesis [26], or NPing [27].

However, there are modules that would be required to be developed that are custom to this tool:

- An Instrumentation module to query the switches for key performance metrics.

- Depending on application requirements, other tools may be required to extract indicators, such as network sniffers (eg, Wireshark).

- A user interface is required to display views/reports that renders the network performance based on a pre-selected event or defined traffic levels.
5 Conclusions and Future Work

In this Deliverable 4.2 we have presented the implementation details of the first NetIDE Logger and the preliminary description of the functionality and requirements of the rest of the debugging tools to be developed along the second and third year of the project. The level of details of the design and development of the tools is not the same across the toolset because we have focused the effort during the first year to adapt the integration of the tools to the new NetIDE architecture produced in WP2. The prototype of the NetIDE Logger constitutes a proof-of-concept of both the NetIDE architecture and the integration of the debugging tools in the overall architecture. Future work will be focused in (1) the full development and evaluation of the complete toolset with a continuous updating based on the state of the art, (2) analysis of the requirements of each tool on the integrated NetIDE architecture, (3) how the tools impact in the performance of the overall NetIDE architecture, and (4) design of a full integration of the WP4 debugging tools and the WP3 IDE.
At A Glance: NetIDE

An integrated development environment for portable network applications

Project Coordinator
Elio Salvadori

CREATE-NET
Tel: +0461-408400 (1301)
Fax: +39-0461-421157
Email: elio.salvadori@create-net.org

Project website: http://www.netide.eu

Partners:
CREATE-NET (IT)
CZ.NIC (CZ)
Fujitsu Technology Solutions (DE)
IMDEA Networks (ES)
Intel (IR)
Telefonica I+D (ES)
Thales Communication & Security (FR)
University of Paderborn (DE)

Duration: 01/2014 – 12/2016
Funding scheme: STREP
Total Cost: €4.545m
EC Contribution: €2.844m

Contract Number: CNECT-ICT-619543

NetIDE aims at delivering a single IDE to support the whole development lifecycle of Software-Defined Network controller programs in a vendor-independent fashion. The project includes partners from Italy, Spain, Germany, France, Ireland, Czech Republic.

Main Objectives

Nowadays, while most of the programmable network apparatus vendors support OpenFlow, a number of fragmented control plane solutions exist for proprietary Software-Defined Networks (SDN). Thus, network applications developers need to re-code their solutions every time they encounter a network infrastructure based on a different controller. Moreover, different network developers adopt different solutions as high-level control plane programming language (e.g. Frenetic, Procera, Maple, ..), which leads to network programs that are neither reusable nor shareable. So despite having OpenFlow as the candidate for a standard interface between the controller and the network infrastructure, interworking between different controllers and network devices is hindered and walled gardens are emerging. NetIDE will deliver a single Integrated Development Environment to support the whole development lifecycle of network controller programs in a vendor-independent fashion.

The project is industry-driven thanks to the involvement of key European players and market leaders such as TID, Intel, Thales, Fujitsu and the contribution from CZ.NIC, an high-tech SME. NetIDE will constitute a relevant enabling technology for both equipment vendors and service providers, providing the possibility of much shorter development cycles, supporting flexible deployments on multi-vendor infrastructures and agile testing. Simplifying the Network Apps development process and ensuring their portability are expected to create commercial potential for European telcos and network solution providers.

NetIDE major components will be released according to an open-source license with the aim of promoting the establishment of an open SDN model based on an Open Community of developers.

NetIDE will result in one-stop solution for the development of SDN applications that covers all the development lifecycle.
A NetIDE Logger 1st Release Installation Guide

A.1 Installation

The first release of the NetIDE Logger has been developed for Ryu Shim Layer. In this section is explained how to install and run it.

A.1.1 Install RabbitMQ Server

To install RabbitMQ Server on Debian/Ubuntu, a Debian package is provided at [http://www.rabbitmq.com/releases/rabbitmq-server/v3.4.4/rabbitmq-server_3.4.4-1_all.deb](http://www.rabbitmq.com/releases/rabbitmq-server/v3.4.4/rabbitmq-server_3.4.4-1_all.deb)

It can be either downloaded with the link above and installed with dpkg (recommended), or using the apt-get command after including the repository to the repository list as follows:

Add the following line to your `/etc/apt/sources.list` file:

```
deb http://www.rabbitmq.com/debian/ testing main
```

To avoid warnings about unsigned packages, add the public key of the RabbitMQ repository to your trusted key list using `apt-key` with the following commands (Optional):

```
wget https://www.rabbitmq.com/rabbitmq-signing-key-public.asc
sudo apt-key add rabbitmq-signing-key-public.asc
sudo apt-get update
```

Install the package using `apt-get`:

```
sudo apt-get install rabbitmq-server
```

To start RabbitMQ Server:

```
sudo invoke-rc.d rabbitmq-server start
```

To stop RabbitMQ Server:

```
sudo invoke-rc.d rabbitmq-server stop
```

RabbitMQ server uses the [AMQP](http://www.rabbitmq.com/) protocol. Hence, in order to communicate the logger with RabbitMQ server it is necessary to use this library:

```
sudo pip install pika==0.9.8
```

The installation depends on `pip` and `git-core` packages, you may need to install them first.

On Ubuntu:

```
sudo apt-get install python-pip git-core
```

On Debian:

```
sudo apt-get install python-setuptools git-core
```
At A Glance: NetIDE
An integrated development environment for portable network applications

Project Coordinator  Elio Salvadori

NETIDE FP7 ICT Objective 1.1 Future Networks

Ryu controller framework. We plan to develop the same tool for the Opendaylight shim layer.

Currently, the logger provides a console interface only. The next phase of the development will provide a graphical interface integrating the tool within the IDE.

A.2 Next Steps

Note: the logger needs to be started after shim layer and the backend are running.

A.1.2 Running the logger

The logger is started with the command:

```
sudo easy_install pip
```

```
python logger.py <dir>
```

The `<dir>` argument is optional and omitting it will allow you to visualise the messages that are sent to the network (called 'out' messages) and the messages sent to the controller ('in' messages). To sniff 'in' messages only, use the command:

```
python logger.py in
```

And to visualise 'out' messages only:

```
python logger.py out
```

Note: the logger needs to be started after shim layer and the backend are running.

A.2 Next Steps

Currently, the logger provides a console interface only. The next phase of the development will provide a graphical interface integrating the tool within the IDE.

Furthermore, the current version of the logger is written for the shim layer developed for the Ryu controller framework. We plan to develop the same tool for the Opendaylight shim layer.
B Source code repositories

Stable releases of the NetIDE Logger Tool can be downloaded from the "Releases" section of the NetIDE Github repository:

https://github.com/fp7-netide/Tools/tree/development/logger

Alternatively, to get the latest version of the code, clone the "Logger" repository by using the following command:

git clone https://github.com/fp7-netide/Tools/tree/development/logger

The repository is organized as follows:

- logger.py : The Logger module
- logpub.py : The LogPub module within the shim layer
- ryu-shim.py : The Ryu shim layer modified to include the LogPub module that communicates with the Logger
At A Glance: NetIDE
An integrated development environment for portable network applications

Project Coordinator
Elio Salvadori
CREATE-NET
Tel: +0461-408400 (1301)
Fax: +39-0461-421157
Email: elio.salvadori@create-net.org
Project website: http://www.netide.eu

Partners:
- CREATE-NET (IT)
- CZ.NIC (CZ)
- Fujitsu Technology Solutions (DE)
- IMDEA Networks (ES)
- Intel (IR)
- Telefonica I+D (ES)
- Thales Communication & Security (FR)
- University of Paderborn (DE)

Duration:
01/2014 – 12/2016

Funding scheme:
STREP

Total Cost:
€4.545m
EC Contribution:
€2.844m

Contract Number: CNECT-ICT-619543

An integrated development environment for portable network applications

NetIDE aims at delivering a single IDE to support the whole development lifecycle of Software-Defined Network controller programs in a vendor-independent fashion. The project includes partners from Italy, Spain, Germany, France, Ireland, Czech Republic.

Main Objectives
Nowadays, while most of the programmable network apparatus vendors support OpenFlow, a number of fragmented control plane solutions exist for proprietary Software-Defined Networks (SDN). Thus, network applications developers need to re-code their solutions every time they encounter a network infrastructure based on a different controller. Moreover, different network developers adopt different solutions as high-level control plane programming language (e.g. Frenetic, Procera, Maple), which leads to network programs that are neither reusable nor shareable. So despite having OpenFlow as the candidate for a standard interface between the controller and the network infrastructure, interworking between different controllers and network devices is hindered and walled gardens are emerging.

NetIDE will deliver a single Integrated Development Environment to support the whole development lifecycle of network controller programs in a vendor-independent fashion. The project is industry-driven thanks to the involvement of key European players and market leaders such as TID, Intel, Thales, Fujitsu and the contribution from CZ.NIC, a high-tech SME.

NetIDE will constitute a relevant enabling technology for both equipment vendors and service providers, providing the possibility of much shorter development cycles, supporting flexible deployments on multi-vendor infrastructures and agile testing. Simplifying the Network Apps development process and ensuring their portability are expected to create commercial potential for European telcos and network solution providers.

NetIDE major components will be released according to an open-source license with the aim of promoting the establishment of an open SDN model based on an Open Community of developers.

NetIDE will result in one-stop solution for the development of SDN applications that covers all the development lifecycle.
C Bibliography


