SPARC

ICT-258457

Deliverable 5.1

Description of emulation platform architecture and according implementation plans

Editor:	*Dimitri Staessens, IBBT*
Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	M9
Actual delivery date:	M10
Version:	1.0
Total number of pages:	27
Keywords:	Emulation, testbed, tools

Abstract

This deliverable reports the progress of work on the emulation frameworks to be used for experimentation in the SPARC project. It presents a descriptions of the experimentation effort, focusing on the requirements and architecture of the experimental setup.

Disclaimer

Imprint

[Project title]	***Split Architecture***
[short title]	***SPARC***
[Number and title of work package]	***WP5 – Validation and Performance Evaluations***
[Document title]	***Description of emulation platform architecture and according implementation plans***
[Editor]	***Dimitri Staessens, IBBT***
[Work package leader]	***Dimitri Staessens, IBBT***
[Task leader]	***Dimitri Staessens, IBBT***
[PM (estimated)]	***2.4***
[PM (consumed)]	***2.4***

Copyright notice

Executive summary

This deliverable reports the progress of work on the emulation frameworks to be used for experimentation in the SPARC project. It presents the requirements and architecture of the experimental setup. Some draft experiments are presented, focusing on the requirements they impose on the emulation platform. The main conclusions are that we should support all the software required for running OpenFlow, and also tools to emulate network traffic on a carrier grade level. We detail some tools to be used in the experiments, such as switch emulation softwares (CLICK, XORP), OpenFlow softswitches (reference switch, Open vSwitch), openflow controllers (NOX, FlowVisor) and several tools for general network performance testing such as traffic generators. Also an overview of IBBT's iLab.t virtual wall network experimentation testbed is given. From the experimentation requirements we have identified some potential issues in the testbed. We detail these, and their solutions in the final chapter. The development of the emulation framework for SPARC on IBBT's iLab.t virtual wall network experimentation testbed is progressing according to plan. All contributing partners have made drafts for experiments, and some potential problems were identified early on in the project. The most severe of these, the VLAN incompatibility, was solved with well in advance of the start of the experimentations. Furthermore, most of the software to be used, being the softswitches (Open vSwitch, reference implementation and OpenFlowClick), the NOX controller, FlowVisor and traffic generation and shaping tools were installed and compatibility tested on the virtual wall. For each of the software packages a snapshot image was created, allowing for one-click deployment of emulated network elements on the virtual wall. Collaboration with the OFELIA project towards the shared use of experimentation equipment is ongoing.

List of authors

Organisation/Company	Author
IBBT	Dimitri Staessens
IBBT	Wouter Tavernier
IBBT	Sachin Sharma
DTAG	Andreas Gladisch
DTAG	Fritz-Joachim Westphal
DTAG	Mario Kind
EICT	Andreas Koepsel
ACREO	Pontus Skoldstrom

Table of Contents

1 Introduction

This document presents the requirements and architecture of the experimental setup which will be used in the SPARC project. Some draft experiments are presented, focusing on the requirements they impose on the emulation platform. We then detail some tools to be used in the experiments, such as switch emulation softwares (CLICK, XORP), OpenFlow softswitches (reference switch, Open vSwitch), openflow controllers (NOX, FlowVisor) and several tools for general network performance testing such as traffic generators. Also an overview of IBBT's iLab.t virtual wall network experimentation testbed is given. From the experimentation requirements we have identified some potential issues in the testbed. We detail these, and their solutions in the final chapter.

2 Experiments for testing Split Architecture

This section outlines some potential experiments and related high-level requirements on the evaluation environment. The primary target is not to describe experiments which will be carried out in a later phase of the project, but to show the potential demands on an evalutation platform by these experiments. With respect to the contents of the experiments, the focus is on the extensions developed and under development by WP4 and the use case on access / aggregation networks as described in D2.1. Due to the current status of the discussions in WP3, it is not possible to precisely predict which experiments will be carried out. The presented topics show a diverse area of experimentation and tests and a selection will be made after the architecture and prototyping activities have converged, so the most important experiments will be performed in this work package.

2.1 SPARC control plane enhancements

2.1.1 Hierarchy of controllers for network virtualization and multi-layer

In an OpenFlow environment controllers will become the most important element between the data-plane elements and the application and network service components. Up to now, sophisticated controller architectures fulfilling the different requirements derived in D2.1 are not available. For example, slicing the physical network infrastructure among various operators is a key for the introduction of network virtualization technologies, but today OpenFlow defines only a common hardware abstraction for OpenFlow compliant data-path elements. It currently specifies a data path with a specific port, forwarding and processing abstraction, but neglects any operation in multi provider scenarios. In a virtualized environment, it is important to expose only that information which is needed by the higher layer, e.g. for security reasons. In principle, the core OpenFlow API can be used to create (as a first assumption) and expose arbitrary network topologies towards higher layers by manipulating port and data-path entity information sent to higher layer controllers. For the time being, it is assumed that this would result in additional complexity, especially with respect to management, and should be avoided. Another, potentially more sophisticated, solution is the stacking of controllers. In a stack of controllers the network topology as seen by each of these controllers may vary depending on technical or business constraints, defined by the operator which is providing the lower-layer services.

At the current time, WP3, which focuses on developing the SPARC architecture, is evaluating different virtualization approaches, one of which is the stacking of controllers. Therefore, the experimentation environment should provide a physical topology (defined as the lowest layer or “real” topology) with a number of data path elements and a sufficient degree of meshing for testing network virtualization approaches with stacked controllers. While ascending the controller stack, the level of abstraction increases, while the number of visible hops and links decreases, so the number of nodes and links should be large enough. It can be assumed that a physical host may emulate a larger portion of an OpenFlow enabled network hosting several data path elements. Conducting performance tests apart from pure functional tests and the emulation platform’s ability to provide adequate performance in such an environment is for further study while work in WP5 is ongoing.

For emulating a physical topology, various options for realization may apply, e.g. deploying software switches (Open vSwitch) or emulating OpenFlow enabled networking domains. In the latter case an emulated domain may consist of fully operational data path elements (i.e. with fully functional forwarding functionality) or each data path element may send appropriate events on a data path / controller OpenFlow API compliant interface emulating a real data path.

As a starting point availability of the following number of data path elements can be assumed: provisioning of OpenFlow enabled networking domains in the order of O(10) each hosting a number of nodes at least in the order of O(10), i.e. approximately an emulated networking environment with ~100 physical nodes can be required. The decision on the exact number of nodes required should be based on the work on scalability conducted in WP3. In order to extend the emulated infrastructure, simulation may be added.

Adequate traffic generators are required for emitting flow events in an OpenFlow enabled domain. It is intended to use test patterns based on available measurements from carrier environments to emulate realistic behavior.

A key performance indicator is the delay added by each virtualized network layer to the overall controlling stack which will be seen by a controlling operator and the user controlled by this operator. Special experiments should be made in order to estimate the additional introduced delay in relation to the number of stacked controller layers. Thus, one key result should be the maximum acceptable depth of stacked layers for certain service depending on certain delay requirements.

2.1.2 Dynamic control composition

In D2.1 some preliminary thoughts regarding a more dynamic association between data path elements and controller entities in split architecture have been formulated. In OpenFlow's current architecture the association between data path and controller is a static one, i.e. unless the communication link between data path and controller is broken, a data path does not change its associated controlling entity. However, a more dynamic association might prove useful: this has been named Dynamic Control Composition in D2.1. A more in-depth discussion of this topic will be part of D3.2.
Experiments should concentrate on the functional evaluation of the extensions. For example, tests should verify whether the virtualized namespace is isolated or not. Therefore, the testbed should allow setup an isolated dynamic control network.

2.2 Access/aggregation use case specific experiments

2.2.1 Introduction

This section defines requirements for an experimental environment for an access/aggregation domain based on split architecture principles. In deliverable D2.1 requirements and constraints for a state of the art access / aggregation domains have been discussed including the adoption of MPLS as proposed by the Broadband Forum in TR101bis [1]. The requirements for WP5 concerning testing applicability of split architectures in access/aggregation domains are fourfold:

1. An Ethernet based state of the art access / aggregation domain. As outlined by deliverable D2.1, state of the art access / aggregation domains following the architectural principles of BBF TR101 use the Point-to-Point protocol (PPP) for user traffic encapsulation.
2. Support is needed for an MPLS based access / aggregation domain, as defined by the Broadband Forum in its TR101bis specifications. The MPLS user and control plane within OpenFlow will comprise the implementation done within WP4 including fast restoration support.
3. Support for enhanced network virtualization schemes (use of the same physical hardware infrastructure by several network operators in parallel). The detailed requirements depend on the work conducted in WP3 and WP4.
4. The discussions regarding the service creation approach are ongoing in WP3 and no specific requirements for WP5 could be identified at the current stage of the project. The potential extension will include a selected set of network applications within the OpenFlow control plane to provide core functionality for this access / aggregation domain (= access / aggregation control stack). This includes AAA functionality and support for triple-play service classes. In the following subsections, some initial ideas and related requirements are detailed but they require more discussion and decision on implementation later on.

Primary goal of this prototypical realization of a split architecture based access / aggregation domain in WP4 and WP5 is the implementation of the enhanced OpenFlow control protocol stack, definition of a functional test specification and finally, conducting these functional tests. These functional tests will be enhanced by realistic performance tests based on a sophisticated emulation environment.

A number of extensions are required to OpenFlow's user and control plane. This might be done using simple extensions to existing software implementations of OpenFlow (e.g. Open vSwitch) or by providing a more enhanced architecture with support for generic processing as currently discussed in WP3. Decisions about implementation are postponed until a final architecture is derived in WP3.

In general the test and experiments can be classified as functional and performance tests. The functional tests cover all verifications of protocol specific headers and related behavior. In contrast, performance test are more sophisticated and evaluate the data plane and certain control plane performances like send rate or delay.

Required elements depend on the tests performed. Verification of a single component is a system test, there as investigations in a combination of elements represent network tests. They differ in complexity of the tests and the number of required elements. Typically, the tests could be performed by the help of traffic generators and especially for the performance tests with the help of some synchronization efforts. The traffic generators require some input for the test patterns, which will be based on measurements in carrier networks.

2.2.2 MPLS

MPLS defines a network service framework in combination with a container technology for other network technologies. Therefore, test cases are manifold and it is needed to define areas of scope for testing. Assuming this perspective, there are four areas:

- Network connectivity service tests like point-to-point or multi-point
- MPLS functions (and related headers)
- Intra domain tests (and related protocols)
- Inter domain tests (and related protocols)
    - With MPLS domains
    - With non-MPLS domains

The testbed will need support for testing the distribution of MPLS labels and the interaction of this distribution with routing protocols like BGP and IS-IS.

2.2.3 BRAS

The Broadband Remote Access Service is the service creation approach still used in many network operator networks. It combines data plane functions with control plane functions for providing a carrier-grade operational mode for the network including important network features like AAA, auto-configuration, Ethernet-layer configuration and Quality of Service. This is enabled by using the PPP suite with its various protocols. In the future, PPP will be replaced by alternatives, but a BRAS like function must be supported in order to enable Internet connectivity for users with PPP, as obviously, it is not possible to switch all users at a certain point in time. In contrast to the existing PPP suite, only a small fraction must be implemented. While the architecture is not yet defined in detail during the work on this deliverable, the description of detailed test scenarios will be finalized in a future Milestone and detailed in the final deliverable D5.2.

Preliminary, the following tests should be performed for validation:

- Functional tests will simply verify the behavior of the functions and related protocols
    - Data plane
        - Verification of frame format
        - Verification of protocol specific marking in header, e.g. session number in PPP header
    - Authentication & authorization
        - Test of transmission of credential
        - Verification of sequence of messages and replies
        - Verification of authorization and related limitations like access attempts to restricted resources
        - Verification of profiles in network devices (potentially with network management features) and related alarm signaling in case of violations
        - Verification of data base entries for requests

- Accounting
  - Verification of existence of connectivity
  - Verification of creation of connectivity data records (CDR)
  - Verification of transmission of CDR
- Auto configuration
  - Verification of correct protocol messages (request and reply) and related implementation in devices (e.g. counter, timer, etc)
  - Typical protocols for verification
    - LCP for Ethernet layer configuration
    - IPCP for IP and higher layer configuration (incl. IP address release, sub net mask, gateway, SIP server, etc)
    - Extensions for IPv6 and MPLS
- Quality of service
  - Verification of preference of profiles
  - Mapping of certain special or incorrect markings (see D2.1 section 2.2.2, page 19/20)
- Behavior in failure/restoration scenarios

- Performance tests will correlate with the functional test but will verify the performance of the different functions for different parameters like delay, packet loss, jitter, answer time, acceptance rate
  - General performance of data plane
  - Number of authentication requests per minute (with resulting requests for authorization, accounting, auto-configuration etc)
  - CPU load for different functions in data plane device and controllers, as well as BRAS entity)

In addition, some more specific tests might be performed like behavior in restoration process after failure.

The tests will require a BRAS component, a user device and a simple network in between. Later tests could integrate the MPLS components as well. The BRAS is a software component and could be hosted on the VirtualWall. The user device should be compliant with a typical simple device available in every electronic discounter today operating with the PPP model and should take into account all relevant BBF standards. The network in between should make use of OpenFlow as specified in WP3, but could neglect in a first run the typical topology, resulting in one step or two step access / aggregation.

For the functional tests, one or few (< 10) end user devices will be sufficient. In contrast, the performance tests should make use of a large number of devices. In D2.1 section 2.2.3, a typical access/aggregation area of a Western European carrier network is detailed. It is obvious that it is not possible to emulate the complete access / aggregation network so a meaningful equivalent, such as a combination of emulation and simulation using Mininet (See Section 3.8), or a downscaled experiment with extrapolation of results, will be used.

In addition, test patterns for carrier networks should be used to verify the performance of the protocols and its implementation. These test patterns could be based on available measurements from carrier environments, specific sources have to be analyzed and an appropriate set for the probes selected. A traffic generator is required which could emulate the different test patterns into the network. For the analysis of jitter and delay, a synchronization mechanism between the traffic sources and sinks is required. For certain test cases it might be useful to have a traffic impairment generator included in between the network elements.

2.2.4 Alternative service creation approach

The BRAS model lacks in simplicity in some points and there are different frameworks for improvements under way. The principal models and their related protocols will be detailed in D3.2. In general, the models have to provide the same functional set as the BRAS model with extended functionality. One special feature is for example the support of multicast in the data/control plane. Accordingly, the tests have to adopt in details but not in their principle targets. Preliminary, the following modifications can be expected:

- EAP and associated message sequences for authentication
- Authentication based on profiles distributed via policies
- Accounting will make use of network management information
- DHCP will be used for Layer 3 and higher layer configuration
- Quality of Service will be similar
- Performance tests have to be adapted accordingly

Similar to the tests, the required devices and components will be similar.

2.2.5 Combinations of service creation approaches

The latter two sections have detailed the standard service creation model of today (BRAS) and the upcoming one in the near- to mid-term future (Alternative service creation approach). In the future, both models will have to operate in parallel on one converged infrastructure. Therefore, test will have to adapt to analyze the behavior of parallel operations. The following tests / test areas have to be tackled:

- Functional test
    - Verification of protocol sequences in multi-user environment (especially verification of security issues like restricted access, distribution of broadcast and control messages to appropriate users only
    - Verification of distribution of data plane identifiers
    - Verification of correct policies for traffic forwarding (incl. analysis of traffic forwarded to controller(s))
- Performance tests
    - Typical data plane tests for performance evaluation (similar to latter sections, but with multi-user environments)
    - Tests with different mixes of BRAS and alternative models
    - Load measurements in controller and other entities

Again, the required components are detailed in the previous sections. For the functional tests, one could restrict themselves for a limited number of end user, but should scale to a larger meaningful number for the performance tests.

2.2.6 Generic processing

Work on deliverables D2.1 and D3.1 has revealed that carrier networks would significantly benefit from extending the rather limited processing capabilities of the current OpenFlow specifications with some form of generic processing support. In the access/aggregation use case description in D2.1 Point-to-Point protocol (PPP) has been depicted exemplarily as a core funcitonality for state-of-the-art Ethernet based aggregation domains as defined by the Broadband Forum. The lack of PPP support makes a transition strategy from today's Ethernet based deployments to an OpenFlow enabled aggregation domain impossible. As adding more and more protocols would pollute the OpenFlow base protocol, the architectural discussion in WP3 focuses on integrating support for deploying generic processing of flows either based on dedicated hardware or software defined processing.

Work on generic processing in WP5 should focus on validation of the generic processing approaches developed within WP3. This would boil down to two complementary experiments: 1) verify proper operation of the proposed extensions for generic processing support to the base OF protocol and also the data path elements, and 2) study potential solutions for distributed processing in a chain of OF complient data path elements. The detailed description of potential experiments must be postponed after architecture (WP3) and implementation (WP4) discussions.

3 Existing emulation options for network evaluations

3.1 XORP

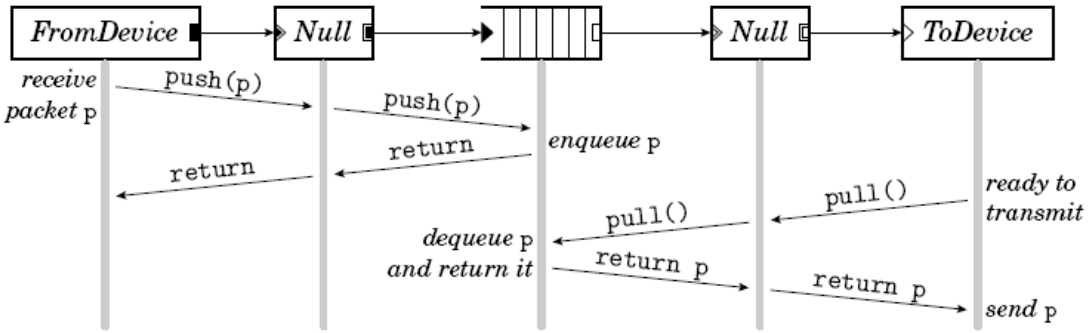
XORP [1] is the industry's only extensible open source routing platform. It is in broad use worldwide, with thousands of downloads by companies and educational institutions and an active international developer community. Designed from the start for extensibility, XORP provides a fully featured platform that implement IPv4 and IPv6 routing protocols (such as BGP and OSPFv2) and a unified platform to configure them. It is the only open source platform to offer integrated multicast capability (such as PIM-SM and MLD). XORP's modular architecture allows rapid introduction of new protocols, features and functionality, including support for custom hardware and software forwarding. XORP is available for free download under a BSD-style license.

3.2 QUAGGA

The Quagga [3] routing suite is an open source fork of the earlier Zebra project, released under the GNU General Public License. It contains daemons for the routing protocols OSPF (version 2 and 3), IS-IS, RIP (version 1, 2, and ng), and finally BGP version 4. It supports most common Unix-like platforms such as Linux, NetBSD and FreeBSD. Architecturally it consists of one daemon, zebra, which acts as an abstraction layer between the underlying operating system and the different protocol daemons (e.g ospfd, bgpd) whom connect to zebra. Zebra provides access to e.g network interfaces while the connected daemons transmit their calculated forwarding tables to zebra for installation into the operating systems network stack. In addition to this Quagga consists of multiple libraries to simplify development of uniformly looking and behaving user interfaces.

3.3 CLICK

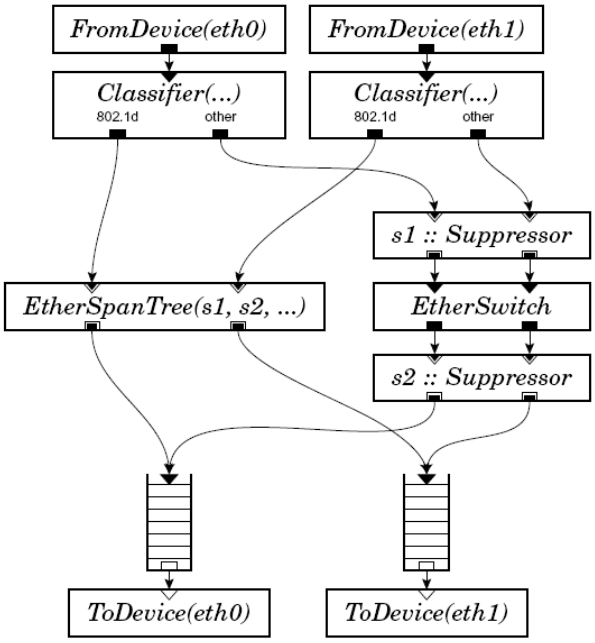
The CLICK Modular Router [4] is an open source emulation platform developed in C++ that allows to let a Linux PC with multiple network interfaces to act as a configurable router or switch. The CLICK architecture is built in a modular way, to allow the configuration of a router or switch to be implemented as the interconnection of a chain of connected modules, or elements which process packets or frames. The interconnection of the elements, which enables packet handoff between the different elements, is defined in what is called a CLICK script. Elements can be of two types: push and pull. In push elements, frames are initiated by the source and pushed downstream, whereas in pull elements the trigger comes from the sink which pulls packets into its direction. Another aspect of elements is that they can have several incoming and outgoing interfaces such as to receive and send frames from and to different other elements. Finally elements can be configured through configuration strings at the moment their interconnection is defined in a CLICK script. Once the CLICK-script has been defined and started, elements can still be accessed or reconfigured through the use of handlers, which are a type of Unix sockets through which certain attributes at C++ class level can be tweaked.



The framework has a lot of pre-built elements, for example for classification, queueing and even elements which perform Ethernet specific functions such as Spanning Tree (STP) control or act as an Ethernet

learning switch. Besides pre-built elements, CLICK can be extended with custom made elements in C++. As the example below will illustrate, special elements called FromDevice and ToDevice are elements which enable to capture and send packets or frames to the standard Ethernet interfaces in Linux.

The configuration of a CLICK router or switch can be represented as a graph. In the next example the basic configuration of a simplified IEEE 802.1d bridge is shown. The typical packet flow, starting at the point of arrival from one of the Ethernet interfaces, is that it first undergoes a classification such as to distinguish between control and data plane frames, the first class is processed by the spanning tree element, the second class goes through a learning switch (Etherswitch). Before and after the learning elements called suppressors are used, such as to force that certain ports are blocked as a result of the spanning tree (this is also the reason why these elements are referenced in the configuration of the STP element).



CLICK has also a proven track record to reach acceptable performance numbers, for example the paper "Measuring CLICK's Forwarding Performance" by Felipe Huici has shown that a PC running CLICK on commodity hardware can forward at rates of hundreds of Mbps even for minimum-sized packets. More specifically, an Opteron 2.0 GHz computer with 2 GB of memory and Intel Pro 1000MT Server adapters with polling drivers can forward at rates of as much as 361 Mbps, largely outperforming Linux native forwarding rates of approximately 265 Mbps under the same setup.

There are currently two drivers that can run CLICK router configurations, a Linux in-kernel driver and a user-level driver that communicates with the network using Berkeley packet filters [McCanne and Jacobson 1993] or a similar packet socket mechanism. The user-level driver is most useful for profiling and debugging, while the in-kernel driver is good for production work.

One of the advantages of the CLICK platform is that click configurations can not only be used for emulation purposes, but also for simulation goals in combination with the ns2 simulator. This setup, typically referred to as "nsclick", can make use of several CLICK nodes in an ns2 simulation test.

3.4 DRAGON

DRAGON [5] is an open source-project which envisages dynamic resource provisioning across heterogeneous network technologies and vendor equipment. The DRAGON software suite was designed to operate within the control plane of GMPLS networks. The control plane architecture consists of four key

elements: the Client System Agent (CSA), Network Aware Resource Broker (NARB), Virtual Label Switch Router (VLSR) and the Application Specific Topology Builder (ASTB).

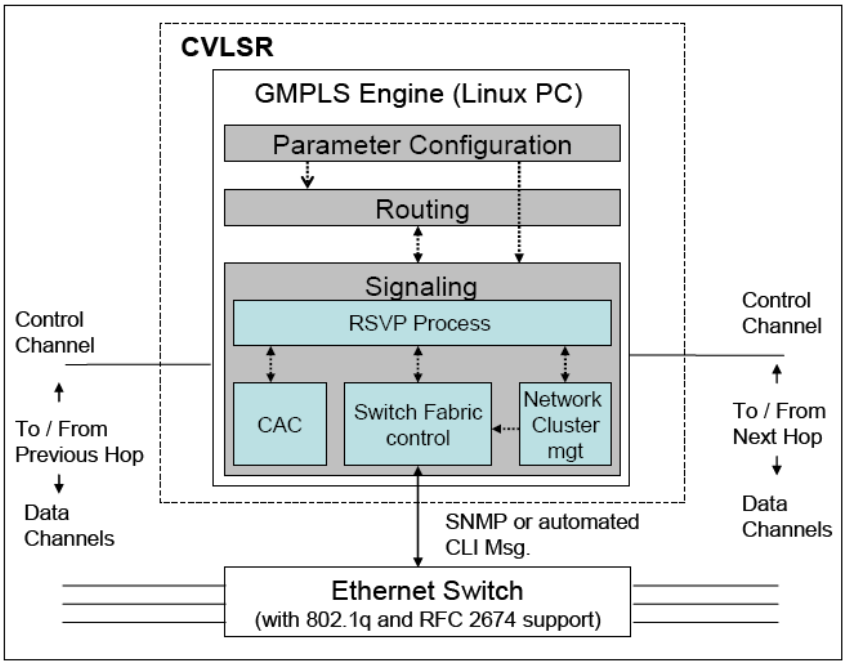


Figure 1: A VLSR to control an Ethernet switch via SNMP in a GMPLS network (diagram from the Cheetah project).

The VLSR has been developed for using vendor equipment in GMPLS networks which are non GMPLS capable. A VLSR is the combination of a PC running the DRAGON software suite and a non GMPLS capable switch, being an Ethernet, TDM or optical switch. What a VLSR does besides participating in the GMPLS protocols is translating GMPLS commands into switch specific commands like SNMP, TL1, CLI, XML, or similar protocol. By the use of these commands a VLSR can control the switch and for example set a switch port in the specific VLAN. To communicate with other VLSR's and CSA's a VLSR uses the routing protocol OSPF-TE and path signaling protocol RSVP-TE.

3.5 Traffic generation and performance tools

3.5.1 Ping

Ping is a network administration utility used to test the reachability of a host on an IP network and to measure the round trip time (RTT) for messages sent from the originating host to a destination computer. Ping operates by sending ICMP *echo request* (type 8) packets to the target host and waiting for an ICMP *echo reply* (type 0) response. In the process it measures the time from transmission to reception *round-trip time* and records any packet loss. The results of the test are printed in form of a statistical summary of the response packets received, including the minimum, maximum, and the mean RTTs.

Ping may be run using various options (command line switches) depending on the implementation that enable special operational modes, such as to specify the packet size used as the probe, automatic repeated operation for sending a specified count of probes, time stamping options, or to perform a ping flood. Using automated repetition, ping can be used as a simple packet generator.

More elaborate flavors have been developed, such as bwping (Bandwidth ping) and nping. bwping also measures the bandwidth between two IP adjacencies, while nping has far more options and is therefore more useful for traffic generation. nping is part of the nmap suite and will be discussed further.

3.5.2 Iperf

Iperf [6] is a cross-platform tool for checking the available IP bandwidth. It can create TCP and UDP data streams and measure the (data) throughput of a network that is carrying them. It can be used for comparison of wired and wireless networking equipment and technologies in an unbiased way. Since it is also open source, the measurement methodology can be scrutinized by the user as well. Iperf output contains a timestamped report of the amount of data transferred and the throughput measured.

3.5.3 Ethernet packet generator

The Ethernet packet generator (packETH) [7] is a Linux GUI packet generator tool for ethernet. It allows you to create and send any possible packet or sequence of packets on an ethernet network. It supports ethernet II, ethernet 802.3, 802.1q, 802.1ad, ARP, IPv4, IPv6, and user defined network layer payload, UDP, TCP, ICMP, IGMP, user defined transport layer payload, RTP (payload with options to send sin wave of any frequency for G.711).

3.5.4 pktGen

PktGen is a linux tool to generate packets, similar to packEth and nping. Because pktgen is “in-kernel,” it can generate high bandwidth and very high packet rates to load routers, bridges, or other network devices. It allows precise control of the send rate on nanosecond timescales. Downside is that PktGen has no TCP support.

3.5.5 Nmap

Nmap [8] is a free and open source utility for network exploration or security auditing. It can also be useful for tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime. Nmap uses raw IP packets to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. It was designed to rapidly scan large networks, but works fine against single hosts. Nmap runs on all major computer operating systems, and official binary packages are avalable for Linux, Windows, and Mac OS X. In addition to the classic command-line Nmap executable, the Nmap suite includes an advanced GUI and results viewer (Zenmap), a flexible data transfer, redirection, and debugging tool (ncat), a utility for comparing scan results (ndiff), and a packet generation and response analysis tool (nping).

nping is an open source tool for network packet generation, response analysis and response time measurement. nping allows to generate network packets of a wide range of protocols, letting users to tune virtually any field of the protocol headers. While nping can be used as a simple ping utility to detect active hosts, it can also be used as a raw packet generator for network stack stress tests, ARP poisoning, Denial of Service attacks, route tracing, etc.

3.5.6 Distributed Internet traffic generator (D-ITG)

D-ITG [9] is a platform capable to produce traffic at packet level accurately replicating appropriate stochastic processes for both IDT (*Inter Departure Time*) and PS (*Packet Size*) random variables (*exponential*, *uniform*, *Cauchy*, *normal*, *Pareto*, ...). D-ITG supports both IPv4 and IPv6 . traffic generation and it is capable to generate traffic at the network, transport, and application layers.

D-ITG is currently available on Linux and Windows. It presents both a multithread and a multitask implementation. The supported protocols are: TCP, UDP, ICMP, DNS, Telnet, VoIP (G.711, G.723, G.729, Voice Activity Detection, Compressed RTP). D-ITG can perform both one-way-delay (OWD) measurements and round-trip-time (RTT) measurements, packet loss evaluations, jitter and throughput measurements. D-ITG is capable to store information about sent and received traffic on both the sender and receiver. Additionally, D-ITG enables the sender and the receiver to delegate the logging operation to a remote log server.

3.5.7 VoIP traffic generation : SIPp

SIPp [10] is a free Open Source test tool / traffic generator for the SIP protocol. It includes a few basic SipStone user agent scenarios (UAC and UAS) and establishes and releases multiple calls with the INVITE and BYE methods. It can also reads custom XML scenario files describing from very simple to complex call flows. It features the dynamic display of statistics about running tests (call rate, round trip delay, and message statistics), periodic CSV statistics dumps, TCP and UDP over multiple sockets or multiplexed with retransmission management and dynamically adjustable call rates. Other advanced features include support of IPv6, TLS, SIP authentication, conditional scenarios, UDP retransmissions, error robustness (call timeout, protocol defense), call specific variable, Posix regular expression to extract and re-inject any protocol fields, custom actions (log, system command exec, call stop) on message receive, field injection from external CSV file to emulate live users. SIPp can also send media (RTP) traffic through RTP echo and RTP / pcap replay. Media can be audio or audio+video. While optimized for traffic, stress and performance testing, SIPp can be used to run one single call and exit, providing a passed/failed verdict.
SIPp can be used to test many real SIP equipements like SIP proxies, B2BUAs, SIP media servers, SIP/x gateways, SIP PBX, ... It is also very useful to emulate thousands of user agents calling a SIP system.

3.5.8 Video traffic generation: VideoLAN

VideoLAN [11] is a complete software solution for video streaming, developed by students of the Ecole Centrale Paris and developers from all over the world, under the GNU General Public License (GPL). VideoLAN is designed to stream MPEG videos on high bandwidth networks. The VideoLAN solution includes VLS (VideoLAN Server), which can stream MPEG-1, MPEG-2 and MPEG-4 files, DVDs, digital satellite channels, digital terrestial television channels and live videos on the network in unicast or multicast and
VLC (initially VideoLAN Client), which can be used as a server to stream MPEG-1, MPEG-2 and MPEG-4 files, DVDs and live videos on the network in unicast or multicast ; or used as a client to receive, decode and display MPEG streams under multiple operating systems
VideoLAN Manager is a small media manager designed to control multiple streams with only one instance of VLC. It allows multiple streaming and video on demand (VoD). This manager being a new feature, can only be controlled by the telnet interface or the http interface.

3.5.9 OpenFlow benchmarking tools

OFlops (OpenFLow Operations Per Second) is a standalone controller that benchmarks various aspects of an OpenFlow switch. Oflops implements a modular framework for adding and running implementation-agnostic tests to quantify an switch's performance.
Cbench (controller benchmarker) is a program for testing OpenFlow controllers by generating packet-in events for new flows. Cbench emulates a bunch of switches which connect to a controller, send packet-in messages, and watch for flow-mods to get pushed down.
Cbench can measure the maximum supported packet_in message generate rate, maximum supported port status messages rate , and also calculates the delay between packet arrival and corresponding packet_in.

3.6 OpenFlow controllers

3.6.1 NOX

NOX [12] is an open-source OpenFlow controller intended to simplify the creation of software for controlling or monitoring networks. Programs written within NOX (using either C++ or Python) have flow-level control of the network. This means that they can determine which flows are allowed on the network and the path they take. In addition, NOX provides programs with access to the network state including the network topology and the location of all detected hosts.
NOX was designed to support both large enterprise networks of hundreds of switches (supporting many thousands of hosts) and home networks with a single switch.

3.6.2 FlowVisor

FlowVisor is a special OpenFlow controller which allows virtualization of the network. Like the virtualization layer on a computer, FlowVisor sits between the underlying physical hardware and the

software that controls it. . It acts as a transparent proxy between multiple controllers (such as NOX) and the OpenFlow switches, where each controller gets its own share of network resources, called a slice. Slices are isolate one from another (both the data path traffic belonging to the slice, and the control of the slice). For a detailed description of FlowVisor, see Section 7.4 in SPARC Deliverable D3.1 Since slicing on VLAN tags is very popular, the testbed should also support the use of VLAN tags in experiments.

3.7 Open Flow capable softswitches

Softswitches are software packages which emulate a switch on a pc with multiple network interfaces. The following softswitches are capable of emulating an OpenFlow switch and are used in the SPARC project.

3.7.1 Stanford reference implementation

The Stanford Reference Implementation is the de facto Openflow software switch. The current stable version is 1.0, implementing the OF 1.0 specification. A reference implementation for 1.1 will be available soon.
The package contains the Linux Software Reference System which adds OpenFlow Switching capability to a Linux PC with multiple NICs. Reference tests provide a regression mechanism for OpenFlow Switching implementations. The OpenFlow Wireshark is a Wireshark plugin which dissects the OpenFlow Switching protocol. Also, a NetFPGA Reference System is available which provides a 4 x 1 GE line-rate OpenFlow Switching implementation for the NetFPGA platform. The reference system is available as a Git repository via native git and as a tar.gz downdoad [13].

3.7.2 Open vSwitch

Open vSwitch [14] is an open-source, production quality, multilayer virtual switch with OpenFlow Switching support. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces (e.g. NetFlow, sFlow, RSPAN, ERSPAN, CLI). In addition, it is designed to support distribution across multiple physical servers similar to VMware's vNetwork distributed vswitch or Cisco's Nexus 1000V.

The current kernel mode release of Open vSwitch is 1.1.0pre2 which supports the OpenFlow 1.0 specification. Open vSwitch can also operate, at a cost in performance, entirely in userspace, without assistance from a kernel module. This userspace implementation should be easier to port than the kernel-based switch, but is considered experimental.

The main components of open vSwitch are

1) ovs-vswitchd, a daemon that implements the switch, along with a companion Linux kernel module for flow-based switching
2) ovsdb-server, a lightweight database server that ovs-vswitch queries to obtain its configuration.
3) ovs-brcompatd, a daemon that allows ovs-vswitchd to act as a drop-in replacement for the Linux bridge in many environments, along with a companion Linux kernel module to intercept bridge ioctls.
4) ovs-dpctl, a tool for configuring the switch kernel module.
5) Scripts and specs for building RPMs that allow Open vSwitch to be installed on a Citrix XenServer host as a drop-in replacement for its switch, with additional functionality.
6) ovs-vsctl, a utility for querying and updating the configuration of ovs-vswitchd.
7) ovs-appctl, a utility that sends commands to running Open vSwitch daemons.
8) ovsdbmonitor, a GUI tool for remotely viewing OVS databases and OpenFlow flow tables.

Open vSwitch also provides an OpenFlow implementation and specific tools for OpenFlow:

9) ovs-openflowd, an alternative to ovs-vswitchd that implements a simple OpenFlow switch, without the special features provided by ovs-vswitchd.
10) ovs-controller, a simple OpenFlow controller.

11) ovs-ofctl, a utility for querying and controlling OpenFlow switches and controllers.

12) ovs-pki, a utility for creating and managing the public-key infrastructure for OpenFlow switches.

3.7.3 OpenFlowCLICK

OpenFlowClick [15] is the Linux reference implementation ported as a CLICK software router element. This OF element can be connected to other CLICK elements and allows integration of OpenFlow functionality in a CLICK software switch.

3.7.4 Beacon

Beacon [16] is a fast, cross-platform, modular, Java-based controller that supports both event-based and threaded operation. The controller is open-source and uses a Java openflow protocol implementation called OpenFlowJ.

3.7.5 Maestro

The Maestro [17] project focuses on building an OpenFlow controller using Maestro. Maestro is a morge general architecture and not only limited to OpenFlow networks. The programming framework of Maestro provides interfaces for:

- Introducing new customized control functions by adding modularized control components.
- Maintaining network state on behalf of the control components.
- Composing control components by specifying the execution sequencing and the shared network state of the components.

By design Maestro is both portable and scalable:

- Developed in Java (both the platform and the components) - Highly portable to various operating systems and architectures.
- Multi-threaded - Takes full advantage of multi-core processors.

Maestro is licensed under the GNU Lesser General Public License version 2.1.

3.8 Mininet

Mininet [18] is a tool for creating scalable (up to hundreds of nodes) software-defined (e.g. OpenFlow) networks on a single PC or Virtual Machine by using Linux processes in network namespaces. It supports research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having a complete experimental network on only one piece of hardware. It includes a basic set of parametrized topologies, but additionally supports arbitrary custom topologies by providing a straightforward and extensible Python API for network creation and experimentation. Mininet provides an easy way to emulate correct system *behavior* and enables simple and quick setup of experiments with topologies in varying scale and complexity. Code developed and tested on Mininet, including OpenFlow controllers, switches, or hosts, can be migrated to real hardware with no changes, facilitating real-world testing, performance evaluation, and deployment.

3.9 RouteFlow

RouteFlow [19] is a commodity routing architecture that combines the line-rate performance of commercial hardware with the flexibility of open-source routing stacks (remotely) running on general purpose computers. The main goal of RouteFlow is enabling remote IP routing services in a centralized way, as a consequence of effectively decoupling the forwarding and control planes. This way, IP networks become more flexible and allow for the addition and customization of protocols and algorithms, paving the way for virtual router and IP network as a Service in the software-defned networking era.

RouteFlow does this by running the control stack in a separate virtual control network (in the RouteFlow Server) which reflects the physical network. This RouteFlow Server communicates to the physical network through an OpenFlow application. RouteFlow is the evolution of partnering Quagga with OpenFlow and works transparently to any Linux-based routing engine (e.g., XORP, BIRD).

3.10 Emulab

Emulab [20] is a network testbed, giving researchers a wide range of environments in which to develop, debug, and evaluate their systems. The name Emulab refers both to a facility and to a software system. The primary Emulab installation is run by the Flux Group, part of the School of Computing at the University of Utah. There are also installations of the Emulab software at more than two dozen sites around the world, ranging from testbeds with a handful of nodes up to testbeds with hundreds of nodes. Emulab is widely used by computer science researchers in the fields of networking and distributed systems.
Emulab is a universally available time- and space-shared network emulator which achieves new levels of ease of use. Several hundred PCs in racks, combined with secure, user-friendly web-based tools, and driven by ns-compatible scripts or a Java GUI, allow you to remotely configure and control machines and links down to the hardware level. Packet loss, latency, bandwidth, queue sizes-all can be user-defined. Even the OS disk contents can be fully and securely replaced with custom images by any experimenter; Emulab can load ten or a hundred disks in less than two minutes. Emulab strives to preserve the control and ease of use of simulation, without sacrificing the realism of emulation and live network experimentation.

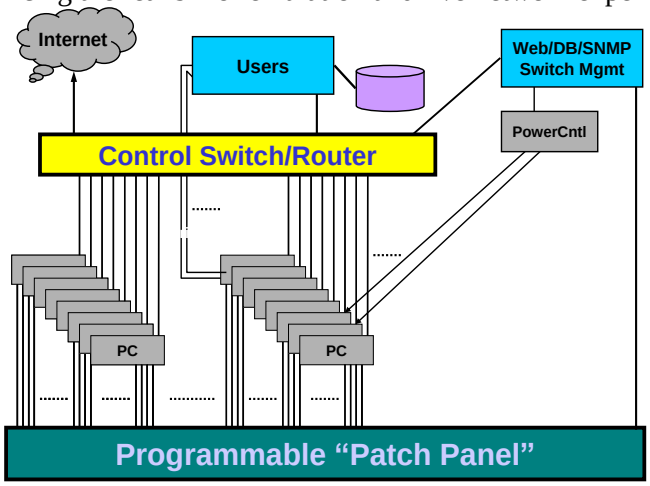


Figure 2: Emulab Architecture

The emulab architecture is depicted in Figure 2. At the heart of the setup is a programmable Patch Panel. In effect this is an Ethernet switch with VLAN (IEEE 802.1q) capability. A large number of PCs are connected to this switch using multiple interfaces per PC in a star topology. The VLAN programmability allows users to specify virtual meshed IP topologies over the physical star topology, using a single distinct VLAN per virtual link. Various disk images are maintained containing different setups for the nodes, allowing fast installation of various operating system and software options. All this functionality is automated by the emulab software. The experiment setups (i.e all topology, node configuration,…) are scripted and maintained on a central file system, in the correct user's directory. The users have direct access to each of the nodes in their experiment via a separate management network through the control switch

3.11 IBBT iLab.t virtual wall

The iLab.t Virtual Wall facility is a generic test environment for advanced network, distributed software and service evaluation, and supports scalability research. The infrastructure is based on emulab.

The virtual wall facilities consist of 100 nodes (dual processor, dual core servers, 6x1 Gb/s interfaces per node) interconnected via a non-blocking 1.5 Tb/s VLAN Ethernet switch, and a display wall (20 monitors) for experiment visualization. Each node is connected with 4 or 6 Gigabit Ethernet links to the switch. The Virtual Wall nodes can be assigned different functionalities ranging from terminal, server, network node, and impairment node. The nodes can be connected to test boxes for wireless terminals, generic test equipment, simulation nodes (for combined emulation and simulation) etc. The Virtual Wall features Full Automatic Install for fast context switching (e.g. 1 week experiments), as well as remote access.

Each node is fully dedicated and under full control of one experiment, so the users can test all kind of operating system images, configurations. Some of the nodes may be used for controlling network parameters (bandwidth, delay, packet loss) to make experiments with data traffic between servers/clients realistic. Some of the nodes can be connected to real network elements such as an OpenFlow enabled Ethernet switch.

4 Required modifications and implementation plan

4.1 Preparing IBBT's virtual wall for SPARC experimentation

4.1.1 Software requirements (OpenFlow and Open vSwitch)

Open vSwitch and NOX have specific requirements on the installed operating system. We found NOX-0.5 (supporting OF 0.8.9) to be incompatible with Debian Lenny, however from NOX-0.6 (supporting OF 1.0) onwards, the software is compatible with both Ubuntu and Debian. For our support of NOX we created new OS images on the wall based on Ubuntu. Open Vswitch is compatible with both our versions of Ubuntu and Debian Lenny. Therefore Ubuntu 10.04 LTS is the preferred OS for SPARC experiments using Open vSwitch and NOX.

For using OpenFlowClick, we need a specific kernel version. This is because the CLICK modular router requires this kernel. The iLab.t virtual wall had an OS image for CLICK based on Debian Sarge. However, since Sarge is not supported anymore a new image based on Debian Lenny and the vanilla 2.6.24.7 kernel was created for using OpenFlowClick within SPARC.

4.1.2 Emulation platform update to support VLAN tags in experiments

A VLAN (Virtual Local Area Network) is a subset of the active topology of a bridged LAN. VLANs facilitate easy administration of logical groups of stations that can communicate as if they were on the same LAN. Traffic between VLANs is restricted. Bridges forward unicast, multicast, and broadcast traffic only on LAN segments that serve the VLAN to which the traffic belongs.
VLAN membership of frames is determined based on a VLAN-ID which is put in a so-called Q-tag of the IEEE 802.1Q frame header (see Figure 3). The VLAN tag allows frames to carry priority information, even if the frame has not been classified belonging to a particular VLAN.

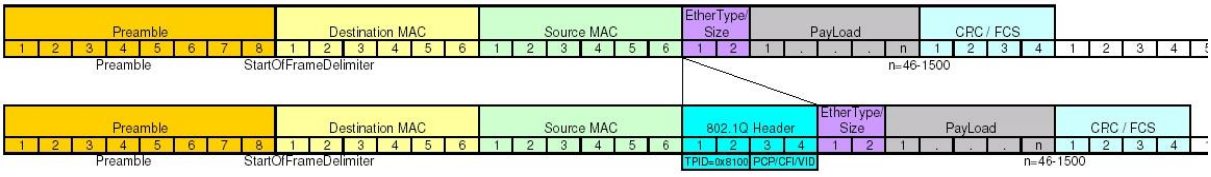


Figure 3 - Ethernet frame headers (above: untagged format, below: tagged format)

The current virtual wall emulab environment of IBBT (ilab.t) does not natively allow transparent use of VLAN tags (IEEE 802.1Q frames with Ethertype 0x8100). This is because experimental topologies on the virtual wall are set up VLAN tags to form virtual topologies between the nodes of the experiment. If VLAN-tagged frames are used in experiments in the above setup, experimental traffic in VLAN-tagged frames are: either i) dropped at the Force 10 switch because the frames arrive at a switch interface which is not configured to accept frames with the given VLAN-tag, or ii) misinterpreted because the Force 10 switch interprets the VLAN-tag as part of the control of the Emulab topology setup.
To allow transparent usage of VLANs in experimental setups, a kernel patch, to be applied on all experimental nodes, is made that:

- ⇨ Translates outgoing frames with Ethertype 0x8100 to 0x9100 (now the virtual wall control switch forwards them transparently)
- ⇨ Translates incoming frames with Ethertype 0x9100 to 0x8100 (such that the processing node has never known the intermediate step of Ethertype translation)

Once the above patch is installed, VLAN-tagged traffic using default Ethertypes can be transparently used in ilab.t experiments.

4.2 Stacked controllers

The development of emulating fully functional data path elements providing the flexibility to create arbitrary network topologies for stimulating stacks of controllers is in progress. For this means, an OpenFlow 1.0

compliant framework written in C++ is being developed. It is used within a self-developed data path and controlling element. This OpenFlow compliant framework will be extended to emulate complex user planes.

4.3 Enhanced traffic generator

The general analysis of potential experiments in section 2 show a demand for appropriate realistic traffic statistics. Based on the available tools mentioned in section 3, traffic patterns will be developed and provided in order to simulate an access / aggregation network. This will be based on anonymized statistics collected in carrier environment of a large Western European country. Detailed information on used protocols, relationship will allow a relative accurate description of today's environment. For the extension e.g. with alternative service creation approaches, an extrapolation will be performed for the related protocols.

It is necessary to align the available format of data with the input required for traffic generator.

The work on the statistics and required data formats has already started. Initial results on the protocols will be available in the beginning of the second half of 2011. Based on the work of WP3, statistics for other protocols will be available in a timely manner.

4.4 Interoperability between Mininet and the iLab.t Virtual Wall

Currently the Mininet framework does not allow direct interconnection with an emulation framework. The interoperability of the CLICK modular router and the ns2/ns3 network simulators has shown that such features are of great benefit for large scale experimentation. There is some work towards the integration of OpenFlow in the ns3 simulator. The ns-3-openflow code allows researchers to experiment with OpenFlow through simulation using the discrete event network simulator, ns-3. However, the Mininet framework has native OpenFlow support and is therefore the preferred simulation option within SPARC. In order to add emulation to Mininet, we leave open the possibility for extending Mininet to allow it to send and receive real packets on a physical interface and in this way have the simulation interact with the emulation in real time.

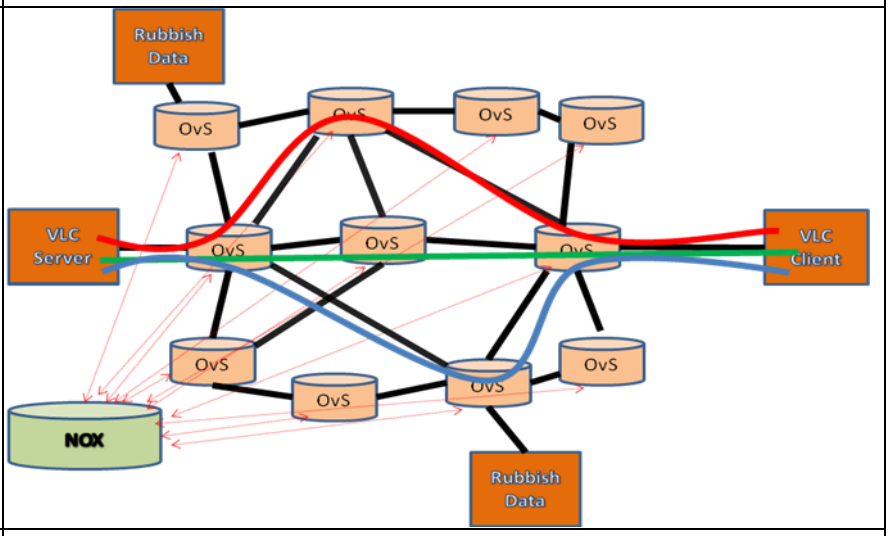
4.5 Collaboration with FP7 OFELIA project

In order to extend the opportunities for experimentation, the SPARC consortium also answered an open call for experimentation on the facilities being developed in the OFELIA project. The goal of the proposal is to be able to do extended experiments using OFELIA hardware switches, NetFPGA's and to experiment further with possible L0/L1 interoperability. This proposal is attached in the annex of this document.

5 Conclusions

The development of the emulation framework for SPARC on IBBT's iLab.t virtual wall is progressing according to plan. All partners have received VPN credentials to be able to conduct experiments and the access has been verified. Based on the architecture outlines and prototyping activity in work packages 3 and 4, some potential problems were identified early on in the project. The most severe of these, the VLAN incompatibility, was solved with well in advance of the start of the validation and performance evaluation experiments in tasks T5.2 and T5.3. Furthermore, most of the software to be used, being the softswitches (Open vSwitch, reference implementation and OpenFlowClick), the NOX controller, FlowVisor and traffic generation and shaping tools, were installed and compatibility tested on the virtual wall. Some worked better on Debian, some better on Ubuntu, but for each of the SPARC-critical packages a snapshot image was created, allowing for one-click deployment. In order to increase the scalability beyond the number of interfaces on the virtual wall, multiple options can be considered. We can try to simulate a large customer base using specific traffic tools developed from traffic measurements in production networks. Another effort is the possible extension of the Mininet simulation environment to allow direct integration with the virtual wall. This will require the translation of real incoming packets from the testbed to virtual packets in the simulator and vice versa. A final effort is the possible collaboration between SPARC and OFELIA. The OFELIA testbed would greatly extend the possible experimentation options for SPARC. Steps towards collaboration have been taken through the submission of a proposed experimental for the OFELIA open call.

Annex

Use Case	
Scenario/use case name	EU FP7 STReP SPARC
Organisation involved	EU FP7 STReP SPARC
Summary scenario (storyline)	Split architectures enable network operators a disjoint evolution of data path and control mechanisms. Based on the OpenFlow approach proposed in the clean slate program at Stanford university, it may pave the way towards more dynamic control of services and connectivity in carrier networks. OpenFlow introduces a fundamental split of control and forwarding plane in data path elements (switches, routers) enabling researchers to study new network protocols and architectures while allowing operators smooth deployment of such new networking paradigms. Carrier grade operation poses additional challenges to the OpenFlow architecture. SPARC studies such challenges and evaluates potential extensions to OpenFlow.
Goal	Initial focus is on reliability and scalability of the OpenFlow architecture. Focus is on how to implement high performance OpenFlow in data-centers and access / aggregation networks.
User community	SPARC project partners
Figure to visualise the use case scenario	
Virtualization layer and involved network/IT resources	Initially Ethernet, but access to L0/L1 virtualization would be of great benefit
Hardware requirements	Up to 30 high end general purpose Linux machines with preferably 1 GbE or 10 GbE links for scalability testing. Servers capable of streaming HD video.

Access method to the infrastructure (OFELIA Islands)	SSH /VPN access to the switches and VLC servers
Access method to the infrastructure slice	Individual access to controller and end nodes
Openflow controller requirements	A custom OpenFlow controller is developed within the project based on NOX, controlling Open vSwitch
Monitoring requirements	Basic linux tools for monitoring / logging will be installed on the switches
Life-cycle of service (phases from start to end)	January 2011-June 2012
Pre-conditions	*<Specific requirements for the slice, if any>*
Post-conditions	<Requirements after the experiment, if any>
Other comments	

References

[1] Broadband Forum Technical Report TR-101 "Migration to Ethernet-Based DSL Aggregation", April 2006, available online. http://www.broadband-forum.org/technical/download/TR-101.pdf

[2] XORP: http://www.xorp.org/

[3] QUAGGA: http://www.quagga.net/

[4] CLICK: http://www.read.cs.ucla.edu/click/

[5] DRAGON: http://dragon.east.isi.edu/

[6] Iperf: http://iperf.sourceforge.net/

[7] packEth: http://packeth.sourceforge.net/

[8] nmap: http://nmap.org/

[9] D-ITG: http://www.grid.unina.it/software/ITG/

[10] SIPp: http://sipp.sourceforge.net/

[11] VideoLAN: http://www.videolan.org/

[12] NOX: http://noxrepo.org/

[13] http://openflowswitch.org/downloads/openflow-1.0.0.tar.gz

[14] Open vSwitch: http://openvswitch.org/

[15] OpenFlowClick: http://www.openflow.org/wk/index.php/OpenFlowClick

[16] BEACON: http://www.beaconcontroller.net

[17] Maestro: http://code.google.com/p/maestro-platform/

[18] Mininet: http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet

[19] RouteFlow: https://sites.google.com/site/routeflow/

[20] Emulab: http://www.emulab.org/