MASTER THESIS

TITLE: Testing NEPI usability and features

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Jordi Pérez Rueda

DIRECTOR: Roc Messeguer Pallarès

DATE: June, 28th 2012
Overview

Nowadays before using any new tool on a real scenario it is necessary to conduct some tests in order to know how it will work. When we talk about networks, we need to perform tests in almost-real scenarios trying to cover the maximum number of possibilities. At present, there are applications that make emulations or simulations.

The goal of this project is to evaluate the features, performance and possibilities of NEPI. NEPI is an application that mixes multiple testbed platforms under a single programming language. To achieve our goal, we have built a scenario that mixes the three platforms that are currently supported and we have executed a P2P application to register what happens to NEPI behaviour when some changes affect the network.

In the first chapter we explain the work environment, describing the different kinds of platforms used by NEPI. Furthermore, we outline some of the tools that we have used along the project.

The next chapter studies these platforms and the NEPI application in depth, describing its installation and the design of the first basic scenarios using each one of the platforms. It also explains how to run an external application in a testbed.

By contrast, the third chapter features the final scenario which we have evaluated. It details the experiment configuration, platform by platform, the IP assignment used and the P2P application settings. The objective of this chapter is to allow anyone to reproduce the very same scenario.

The fourth chapter analyses the results and outputs that we have obtained, evaluating two scenario possibilities: one involving no changes at all and one were changes are applied during running time.

The fifth chapter is an essential part of this project, as it gathers and describes all the problems and limitations we have found during this project. Its contents help us to get a final conclusion for the project and achieve our initial goal.

The last chapter explores possible future developments on this project and puts an end to our work by drawing some conclusions from the evaluation of the NEPI application we performed for some weeks.
4.1. Test Scenario .................................................................................................................. 39
4.2. Outputs with nodes on .................................................................................................. 40
4.3. Outputs with nodes off ............................................................................................... 41

CHAPTER 5. PROBLEMS & LIMITATIONS ................................................................. 44
5.1. Problems with NEPI ................................................................................................... 44
5.2. PlanetLab Limitations ............................................................................................... 47

CHAPTER 6. FUTURE LINES & CONCLUSIONS ......................................................... 48
6.1. Future work .................................................................................................................. 48
6.2. Conclusions ............................................................................................................... 48

BIBLIOGRAPHY ............................................................................................................... 50
FIGURES INDEX

Fig. 1.1 Different stages of the network experiment life-cycle ............................................. 4
Fig. 1.2 Example of Boxes and Connectors design............................................................. 5
Fig. 1.3 PlanetLab node map .............................................................................................. 7
Fig. 1.4 PlanetLab Europe node map.................................................................................... 8
Fig. 1.5 Design View example in NEF application .............................................................. 8

Fig. 2.1 First scenario - Topology View ............................................................................... 12
Fig. 2.2 First scenario - Design View .................................................................................. 12
Fig. 2.3 First scenario settings ............................................................................................. 13
Fig. 2.4 Issuing a ping command in the First scenario ......................................................... 13
Fig. 2.5 Use of tunnels in a simple scenario ......................................................................... 13
Fig. 2.6 Topology View when using tunnels ......................................................................... 14
Fig. 2.7 Third scenario - Topology View .............................................................................. 14
Fig. 2.8 Third scenario - Design View.................................................................................. 15
Fig. 2.9 Running the third scenario ..................................................................................... 15
Fig. 2.10 Four Clients and 4 Servers scenario (Design View) ............................................... 16
Fig. 2.11 Four Clients and 4 Servers scenario (Topology View) ........................................... 16
Fig. 2.12 Example of the Mobility View .............................................................................. 17
Fig. 2.13 Simple PlanetLab scenario ................................................................................... 18
Fig. 2.14 PlanetLab nodes configuration .............................................................................. 18
Fig. 2.15 PlanetLab simple scenario - Topology View ......................................................... 19
Fig. 2.16 Tags needed to use PlanetLab and NETNS ........................................................... 19
Fig. 2.17 PlanetLab and NETNS - Design View .................................................................. 20
Fig. 2.18 PlanetLab and NETNS - Topology View ............................................................... 20
Fig. 2.19 Issuing a ping command to PlanetLab node .......................................................... 21

Fig. 3.1 NETNS Design View of the final Scenario ............................................................. 26
Fig. 3.2 NS3 Design View of the final Scenario ................................................................. 27
Fig. 3.3 Connections between NETNS and NS3 ................................................................. 27
Fig. 3.4 First connections between PlanetLab and NETNS ................................................. 28
Fig. 3.5 Second idea between PlanetLab and NETNS connections .................................... 29
Fig. 3.6 Final Scenario - Design View ............................................................................... 29
Fig. 3.7 Final Scenario - Topology View ............................................................................. 30
Fig. 3.8 IP addresses configuration ..................................................................................... 31
Fig. 3.9 Networks used ........................................................................................................ 31
Fig. 3.10 Routes added in PlanetLab nodes ......................................................................... 32
Fig. 3.11 Routes added in NETNS nodes (top) and NS3 (bottom) ......................................... 32
Fig. 3.12 Example typing "status" command ...................................................................... 34
Fig. 3.13 Example when adding variables in the network ................................................... 34
Fig. 3.14 PlanetLab Testbed configuration .......................................................................... 35
Fig. 3.15 NETNS (Left) and NS3 (Right) Testbed configuration .......................................... 36
Fig. 3.16 Adding traces to the scenario ................................................................................. 36
Fig. 3.17 Configuration processes while running the program ............................................. 37
Fig. 3.18 Final Scenario - Topology View .......................................................................... 37
Fig. 3.19 Traces View tab after stopping the test ................................................................. 38
Fig. 4.1 Killing the application ................................................................. 39
Fig. 4.2 Variables and Values table .......................................................... 40
Fig. 4.3 Example adding values ................................................................. 40
Fig. 4.4 Wireshark capture when no fails in the pcap file ....................... 41
Fig. 4.5 Some values cannot be get ............................................................ 42
Fig. 4.6 Empty values when nodes turn on again .................................... 42
Fig. 4.7 Wireshark capture when there are some fails in the pcap file ....... 43
INTRODUCTION

In order to evaluate new network protocols, applications and architectures, we have to use many different experiment environments such as simulators, emulators, testbeds, and, sometimes, a combination of them all.

As network features vary according to time and space, we cannot rely on a single application to perform all the experiments. At present there are different applications to conduct this sort of experiment environments such as PlanetLab, Emulab, ORBIT or ModelNet. Each one has its own features: some are computer networks spread over the world, others are simulators and others simple emulators.

Unfortunately, the cost of implementing the right process to test a new protocol or application is often too high due to the need to use multiple tools to investigate different experimental conditions. The highest cost comes from having to learn new programming languages and new authentication and authorization mechanisms to use each testbed. Moreover, we are unable to keep track of all the experimentation details over many months to ensure that experimentation conditions can be accurately reproduced later. In short, it is very difficult to maintain a detailed view of the experiment when its network topology and its applications setup descriptions are split among many separated files written using different languages.

All these difficulties gave birth to the idea behind NEPI: enabling users that test different experimentation environments to change amongst them easily. By using always the same programming language and the same application we can reduce the cost of learning new languages. In order to reach this goal, NEPI creates a new application where we can build experiments in a simulator, emulators or testbeds as PlanetLab.

There is not a final NEPI version yet. Although we can use it and build experiments, it is still in an “unstable” state and some functions still have to be implemented. This was the fundamental motivation to conduct this project, willing both to help the creators from a user’s point of view, and to know how advanced NEPI is. In this project, we tried to evaluate the application in order to know what kind of experiments or scenarios we can perform at present, and to know how it could be in a future.
CHAPTER 1. PROJECT BACKGROUND

The aim of this chapter is to make a short introduction about the work tools we will use during the project. In order to understand better the environment and the tools that NEPI uses, we will start by defining some basic concepts. After that, we will focus on what NEPI is and the different platforms that NEPI is built upon. As NEPI can use an infinite number of applications, the final point of the chapter explains which one is used in our testbed scenario, so we can get a more accurate idea about what this application has to offer.

1.1. Emulation, Simulation and Distributed Testbed

1.1.1. Network Emulation

Network emulation is a technique where the properties of an existing, planned and/or non-ideal network are simulated in order to assess performance, predict the impact of change, or otherwise optimize technology decision-making.

Network emulation can be accomplished by introducing a device on the LAN that alters packet flow in a way that imitates the behaviour of application traffic in the environment being emulated. This device may be either a general-purpose computer running software to perform the network emulation or a dedicated emulation device. The device incorporates a variety of network attributes into its emulation model – including the round-trip time across the network (latency), the amount of available bandwidth, a given degree of packet loss, duplication of packets, reordering packets, and/or the severity of network jitter. Desktop PCs can be connected to the emulated environment, so that users can experience the performance and behaviour of applications in that environment first-hand. ([1] Network Emulation Definition, Online)

1.1.2. Network Simulation

In communication and computer network research, network simulation is a technique where a program models the behaviour of a network either by calculating the interaction between the different network entities (hosts/routers, data links, packets, etc.) using mathematical formulae, or actually capturing and playing back observations from a production network. The behaviour of the network and the various applications and services it supports can then be observed in a test lab; various attributes of the environment can also be modified in a controlled manner to assess how the network would behave under different conditions. ([2] Network Simulation Definition, Online)
1.1.3. Emulation vs. Simulation

Emulation differs from simulation in that a network emulator appears to be a network; end-systems such as computers can be attached to the emulator and will behave as if they are attached to a network. Network simulators are typically programs which run on a single computer, take an abstract description of the network traffic (such as a flow arrival process) and yield performance statistics (such as buffer occupancy as a function of time).

1.1.4. Distributed Testbed

A testbed is a platform for experimentation of large development projects. Testbeds allow for rigorous, transparent, and replicable testing of scientific theories, computational tools, and new technologies.

The term is used across many disciplines to describe a development environment that is shielded from the hazards of testing in a live or production environment. It is a method of testing a particular module (function, class, or library) in an isolated fashion. A testbed is used as a proof of concept or when a new module is tested apart from the program/system it will later be added to. A skeleton framework is implemented around the module so that the module behaves as if it was already a part of the larger program.

A typical testbed could include software, hardware, and networking components. In software development, the specified hardware and software environment can be set up as a testbed for the application under test. In this context, a testbed is also known as the test environment. ([3] Distributed Testbed Definition, Online)

1.2. What is NEPI?

NEPI, the Network Experimentation Programming Interface, is a life-cycle management tool for network experiments. The idea behind NEPI is to provide a single tool to design, deploy, and control network experiments, and gather the experiment results. Going further, NEPI was specially conceived to function with arbitrary experimentation platforms, so researchers could use a single tool to work with network simulators, emulators, or physical testbeds, or even a mixture of them.

NEPI supports conducting hybrid-experiments, by deploying overlay topologies across many testbeds and communicating them through special tunnelling components. To accomplish this, NEPI provides a high-level interface to describe experiments that is independent from any experimentation platform, but is able to capture platform specific configurations. Experiment definitions can be stored in XML format to be later reproduced, and modified according to experimentation needs.
Currently NEPI provides support for three different experimentation platforms: the ns-3 network simulator, the netns emulator, and the Planet Lab distributed testbed.

NEPI is licensed under GPLv2 and is implemented as a Python library, for those who prefer scripting, but it also provides a Graphical User Interface called NEF. ([4] NEPI Project Homepage, Online)

**1.2.1. Experiment life-cycle support**

NEPI is a life-cycle management tool for network experiments. It covers the design, deployment, control and result gathering stages of the network experiment life-cycle. It is shown in Fig. 1.1.

- **Design**: During the design stage, the user constructs the experiment description using interconnected box components. A box component is defined by a type and by the experimentation platform (or testbed) it is associated to. The user can alter the experiment configuration by setting values on the box components attributes. NEPI will automatically validate connections between boxes and boxes attributes. Box components have a list of traces, which represent result files that can be activated to be generated during experiment execution. The experiment description can be persisted to XML format. This description will be NEPI's input to perform the deployment of the experiment.

- **Deployment**: During deployment, NEPI uses the information on the XML description to instantiate, configure, and connect experiment components. An ExperimentController instance is responsible for processing the XML description, instantiating a specific TestbedController for each testbed instance present in the experiment description, and issuing the right commands to each of the
**TestbedControllers** so they can create the necessary experiment components.

The experiment deployment consists of well-defined steps, that resolve concrete operations. Globally, these steps are:

1. Testbed set-up and configuration
2. Component instantiation
3. Component configuration
4. Connection of components inside a testbed
5. Connection of components from different testbeds
6. Launch of applications

- **Control**: The Control stage occurs after deployment, when the experiment is running. During this stage, the user can interact with the ExperimentController and modify experiment parameters in real-time. The Experiment and Testbed controllers are able to execute in remote locations and communicate via special messages.

- **Results Gathering**: Results can be retrieved, from any remote controller, in a centralized way, at any moment from the moment the experiment starts running.

### 1.2.2. Experiment design

NEPI uses a Boxes and Connectors modelling abstraction to construct the experiment design. Each supported experimentation platform defines a set of box types, which represent the conceptual constructive blocks of an experiment. These boxes can be associated through named ports called connectors. Each different connector in a box has a different function. The boxes present in the experiment definition and the connections between those boxes will define the experiment topology, both at a physical and application level. Boxes also have a set of attributes that allow defining the experiment configuration, and traces that allow defining experiment results to be collected. It is shown in Fig. 1.2.

![Fig. 1.2 Example of Boxes and Connectors design](image)
1.2.2.1. Object model

The main classes that participate in describing an experiment are:

- **ExperimentDescription**: Groups the description of the different parts of the experiment that might be executed in different testbed instances.

- **TestbedDescription**: Describes the topology, applications, and configuration of the part of the experiment to be executed in a particular testbed instance.

- **FactoriesProvider**: Provides the box classes definitions for a concrete testbed type. (Ex: ns-3, PlanetLab).

- **Boxes**: Functional units that describe an experiment. (Ex: Node, Interface, Application, Channel).

1.3. NETNS

NETNS emulator is a project to create an independent library that is able to create network name spaces, configure virtual links with emulated delay, loss and throughput, and to run programs inside those. The API is still under development, so it could change. The important elements of the API are:

- The Node class, representing a separate name space. From it, interfaces are created, routes added, and applications launched.
- Interface class.
- Link class, emulating any communication channel and implemented as a bridge plus tc qdiscs applied to connected devices.
- Application class, not fully defined yet, will provide a popen-like interface to run applications inside a Node.

Architecturally, it is very simple. The process using the library becomes the master; each separate name space is handled by a slave process that is forked from the master. One important design decision is that as it has to interact with a live system, the state will constantly vary, so most operations will invoke system commands to retrieve the current state. ([4] NEPI Project Homepage, Online)

1.4. NS-3

The ns-3 simulator is a discrete-event network simulator targeted primarily for research and educational use. The ns-3 project, started in 2006, is an open-source project developing ns-3. ([5] NS-3, Online)

A few key points are worth noting at the onset:
• NS-3 is not an extension of NS-2; it is a new simulator. The two simulators are both written in C++ but NS-3 is a new simulator that does not support the NS-2 APIs.

• NS-3 is open-source, and the project strives to maintain an open environment for researchers to contribute and share their software.

1.5. **PlanetLab**

PlanetLab is a global research network that supports the development of new network services. Since the beginning of 2003, more than 1,000 researchers at top academic institutions and industrial research labs have used PlanetLab to develop new technologies for distributed storage, network mapping, peer-to-peer systems, distributed hash tables, and query processing. ([6] PlanetLab, Online)

PlanetLab currently consists of 1089 nodes at 532 sites as shown in Fig. 1.3.

![PlanetLab node map](image)

**Fig. 1.3** PlanetLab node map

1.5.1. **PlanetLab Europe**

PlanetLab Europe is the European portion of the publicly available PlanetLab testbed and is a part of the OneLab experimental facility.

PlanetLab, established in 2002, is a global network of computers available as a testbed for computer networking and distributed systems research. Each research project runs a "slice" that gives experimenters access to a virtual machine on each node attached to that slice. See Fig. 1.4 for more details.
Accounts are available to persons affiliated with corporations and universities that host PlanetLab nodes. Those who join PlanetLab Europe have access to the entire system. They also participate in the initiatives built around PlanetLab in Europe.

PlanetLab Europe operates under the direction of Timur Friedman of UPMC Sorbonne Universités, working in collaboration with the Institut National de Recherche en Informatique et en Automatique (INRIA). ([7] PlanetLab Europe, Online)

1.6. NEF

NEF stands for Network Experimentation Frontend, and it is the NEPI GUI (Graphical User Interface). ([4] NEPI Project Homepage, Online)
1.7. **Overlay Weaver**

Overlay Weaver is an overlay construction toolkit, which supports overlay algorithm designers in addition to application developers.

The toolkit provides multiple routing algorithms, Chord, Kademlia, Koorde, Pastry, Tapestry and FRT-Chord. Routing layer under the higher-level services has been decomposed into multiple components, routing driver, routing algorithm and messaging service. The decomposition also facilitates implementation of a new algorithm. A newly implemented algorithm can be tested, evaluated and compared on an emulator, which can host hundreds of thousands of virtual nodes. It enables large-scale emulation and fair comparison between algorithms.

The toolkit consists of runtime and the following tools. Each word between parentheses is a name of a command located under bin directory.

Distributed Environment Emulator (owemu)
- Emulation Scenario Generator (owscenariogen)
- Overlay Visualizer (owviz)
- Message Counter (owmsgcounter)

The toolkit contains the following sample applications.
- DHT shell (owdhtshell)
- Mcast shell (owmcastshell)
- Application-level IPv4 multicast router (owmrouted)

On the routing layer, higher-level services are implemented. Applications usually rely on them. The toolkit provides multicast (Mcast) in addition to distributed hash table (DHT) as higher-level services. The Mcast multicasts over an overlay. It allows a user to join and leave a group specified by an ID, and to multicast messages to the group. It can also notify an application of topology of a spanning tree on which a multicast message is transferred.

DHT shell and Mcast shell are sample applications implemented on corresponding services. They read commands from a character terminal or network and control the underlying services directly. They can be used with the emulator together to test and compare overlay algorithms. ([8] Overlay Weaver, Online).
CHAPTER 2. STARTING WITH NEPI

Once the basic concepts and all NEPI components have been explained, we will focus on the first steps using NEPI. We will explain how to create some testbeds according to the different platforms that NEPI offers in order to differentiate between an emulator, a simulator and a distributed testbed.

All the scenarios explained in this chapter are basic scenarios and that is the reason why we did not use any additional application. However, in the final section of the chapter, we explain how to install Overlay Weaver as this is the application we will use in our final scenario. We decided to include it here because in this chapter we explain the configuration and installation environment.

2.1. First Steps

2.1.1. Work Environment

2.1.1.1. Operating System

First, you must have a computer with a Linux Operating System. NEPI website suggests several distributions, but Fedora 15 ([9] Fedora Project Homepage, Online) is recommended as it is the version they have been testing.

At the beginning of the project there was no installation package for any distribution so I used the version recommended to avoid further problems.

2.1.1.2. NEPI Installation

Nowadays, there are two ways of installing NEPI and related projects:

- Using a python install script
- Installing individual repository packages

As I have mentioned in the previous section, the NEPI project was at its very beginning when I started it, so I had to install it manually and take into account all the several dependencies that the system must met prior to installing NEPI and its modules.

To reproduce any experiment the following dependencies are required:

```
NEPI requires:
* python >= 2.6
* ipaddr >= 2.1.5
```
Starting with NEPI

NETNS requires:
* Linux kernel >= 2.6.36
* bridge-utils
* iproute

NEF requires:
* libqt4 >= 4.6.3
* python-qt4 >= 4.7.3

ns-3 requires:
* python-dev

Once all dependencies are installed, the installation can be tested by running NEF, NEPI's GUI, and start designing your experiment.

2.1.1.3. Using PlanetLab

To make use of PlanetLab, it is necessary to register with it. Once an account in PlanetLab is created, the user needs to upload the SSH key to access the nodes.

Remote access to PlanetLab nodes is restricted to SSH login using RSA authentication. RSA authentication is based on public-key cryptography. Encryption and decryption are performed with separate keys, and it is not possible to derive the decryption key from the encryption key.

To generate an SSH key pair, use the ssh-keygen program on a secure UNIX system:

```bash
ssh-keygen -t rsa -f ~/.ssh/id_rsa
```

This key must be in OpenSSH format. If the system is running a commercial UNIX and the first line of your .pub file does not look like:

```bash
ssh-rsa AAAAB3Nza...
```

Ssh-keygen will generate two files: a private key file named id_rsa and a public key file named id_rsa.pub. Store the private key file id_rsa in a safe place, such as on a removable USB flash memory device. Upload the public key file id_rsa.pub to the PlanetLab website using the Manage My Keys page.

Finally, a slice must be registered. For any experiment with PlanetLab it is mandatory to have access to a PlanetLab slice. The name of your slice will be prefixed with an abbreviated version of your site name and an underscore. Once your slice has been created and you have been associated with it, you may assign nodes to the slice by using the Manage Nodes form.

Once you have been associated with a slice, it may take up to an hour for your slice to be created on all nodes and for your SSH public key to propagate to all of the nodes in your slice. Obviously, you will not be able to login to nodes that are down when your slice is created, and it may take a few minutes after a down node comes back up for your slice to be created on it.
2.2. Provided Platforms

2.2.1. NETNS

In this project, we start to develop a very simple scenario in NETNS emulator using a graphical user interface called NEF.

An interface in NETNS belongs to exactly one namespace. This presents a problem, since we want virtual nodes to share the node's public IP interface.

The first scenario has the topology shown in Fig. 2.1:

![Fig. 2.1 First scenario - Topology View](image)

To design it, we added a test bed instance of NETNS where we added two different nodes. Each node has an application box and a Node Interface box. To connect these two nodes we use a switch. This design is shown in Fig. 2.2:

![Fig. 2.2 First scenario - Design View](image)

The settings in this first scenario were very simple, so we only needed to configure 2 boxes. On the one hand, the application box is used to prompt which command we want to run. In our case, in order to test the connection between these two nodes, we use "xterm"; once the experiment is running we can use ping to check the connection. On the other hand, the node interface box has to be linked up and the IP address has to be edited. These configurations have to be set twice, once in each of our nodes. (See Fig. 2.3).
Starting with NEPI

When the scenario is running the result is shown in Fig. 2.4.

When you run this scenario, a problem may arise due to the kernel version.

Therefore, another scenario is proposed as simple as the previous one, but this time with two test bed instances. To join the nodes a tunnel has been used. You can see the design view in Fig. 2.5.

It works in the same way as it runs a terminal at each node. We can run the ping command to verify the connection. The only difference is related to the topology, as it is shown in Fig. 2.6:
2.2.2. NS-3

If we want to develop complex scenarios we will need an NS simulator that is also included in the graphical user interface (NEF).

In our first scenario, we try to do something easy and try to simulate the same scenario that in NETNS but using both. The topology is as follows in Fig. 2.7:

In this case, when making the design view of the test bed we can see two instances. One that belongs to NETNS, and another including NS3. The NETNS instance has already been explained above, two nodes with applications and their interfaces.

However, in NS3 we have a node with two interfaces, one for each NETNS node. For the experiment to work properly on the nodes of NS3 we must configure the node protocols. In this case 5 protocols are configured. Icmp, ip, tcp, udp and arp. The topology is as follows in Fig. 2.8:
Starting with NEPI

Fig. 2.8 Third scenario - Design View

In this example, after checking the connectivity, it has declined to a client and a streaming video server. To do this, in one of the nodes of the emulator a client was configured with the command "vlc rtp: // ip:port" ([12] VLC command-line, Online) and on the other node a server was configured with "vlc -i dummy /video_path --sout '#rtp{dst=client_ip, port=client_port}'. Moreover, the NS3 node interfaces had to be configured with an ip.

The outcome of this scenario is the visualization of a video on the client node. It is shown in Fig. 2.9.

Fig. 2.9 Running the third scenario

The network topology of the second experiment consists of two parts. In the netns emulator part we have 5 emulated nodes, with one network interface each. 4 of these nodes run VLC servers streaming RTP video to the remaining node, which hosts 4 VLC clients, one for each stream. But the emulated nodes
are not directly connected to the same sub-network, instead they are interconnected through an ns-3 simulated 802.11 wireless network. In the ns-3 part of the experiment we have 4 wireless stations, each connected to one emulated node hosting a VLC server, and 1 AP connected to the emulated node hosting the 4 VLC clients. The AP is in a fixed position, while the 4 wireless stations move away at a speed of 1m/s, with an initial distance of 10m, and a final distance of 100m from the AP.

The following diagram (Fig. 2.10) shows a conceptual representation of the experiment, including the detail of the interconnection between the emulated and the simulated nodes:

![Fig. 2.10 Four Clients and 4 Servers scenario (Design View)](image)

The scenario topology designed by NEPI is as follows in Fig. 2.11:

![Fig. 2.11 Four Clients and 4 Servers scenario (Topology View)](image)
To configure this scenario, we mainly followed the settings of the previous scenarios. In the case of NETNS there is no change. It has 4 separate servers, and 4 client nodes receiving the video.

In the NS3 section, a node is configured using the same elements as described in previous examples. However, a wireless interface and a mobility model have been added to each node.

In the server nodes, a mobility model has been added. In this case the "WaypointMobilityModel" box has been used and set to indicate the initial position of the nodes in the experiment, and the positions the nodes will have 90 seconds later. These nodes are configured as wifi interface stations indicating that they are connected to the same access point. Moreover a rate control mechanism has been added.

The client node instead uses a different mobility model, called "ConstantPositionMobilityModel" which stands still for 90 seconds. This node is configured as an Access Point wifi interface which connects the different stations amongst them. Furthermore a rate control mechanism has been added.

A physical model and a typical channel of NS3, YansWifi, has been added and configured in all 5 nodes. This PHY implements a model of 802.11a. The model implemented here is based on the model described in "Yet Another Network Simulator" web page of NS3. And this PHY model depends on a loss and delay channel model as provided by the PropagationLossModel and PropagationDelayModel classes, both of which are Members of the YansWifiChannel class.

Once the experiment is running, it is possible to see the set up nodes movement. In these two pictures (Fig. 2.12) we can compare the position of the nodes at the very beginning of the test and at the middle of the test.

![Fig. 2.12 Example of the Mobility View](image)
2.2.3. **PlanetLab**

This is the third and the last platform supported by NEPI, so to test it a very simple scenario has been created. The starting point is very similar to NETNS and NS3. In order to verify the interface access to PlanetLab and the accurate configuration this testbed has been created.

First of all, it is necessary to register with PlanetLab. Once a username is created, the user needs to configure an SSH key to access the nodes. Finally, a slice must be created. See 2.1.1.3 to do that.

The design view of this scenario is shown in Fig. 2.13:

![Simple PlanetLab scenario](image1)

As we can see in the picture, there are two different nodes whose interfaces are connected to an internet box. To configure which node we want to connect, we have to configure the hostname attribute in the node configuration as is shown in Fig. 2.14. Also, each node has an "xterm" application configured.

![PlanetLab nodes configuration](image2)

When we run the test, we cannot see if xterm is executing or not, because it does not show up in our computer and it is only showed in the remote machine. But once the nodes are ready we can connect to them through ssh to check the experiment or perform additional tasks. In this experiment we will need to work
very quickly, because the ping command will not run long, and the experiment will be over very soon.

In this scenario, we can check that no errors come up in the configuration because it compiles right. Moreover the access to PlanetLab and to the slices has been verified. Furthermore, the topology view is enabled as we can see in Fig. 2.15.

![Fig. 2.15 PlanetLab simple scenario - Topology View](image)

As in the other two cases, we have complicated the experiment scenario joining two platforms. So in the next step, we will connect a NETNS emulation, running in our local machine, to a PlanetLab node, running in the distributed PlanetLab environment. When the experiment is deployed, an xterm console attached to the NETNS emulation will pop-up, and we will be able to ping the remote PlanetLab node through the virtual overlay interfaces.

To run this experiment we need to add some tags to our slice. To add them a PlanetLab user with "admin" role is needed. In UPC no one has such a role, so we had to request Planet Lab Europe support team to add these tags on our behalf.

The tags our slices need are in Fig. 2.16:

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
<th>NODE</th>
<th>NODEGROUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>vsys</td>
<td>ipfw-be</td>
<td>ALL</td>
<td>n/a</td>
</tr>
<tr>
<td>vsys</td>
<td>vroute</td>
<td>ALL</td>
<td>n/a</td>
</tr>
<tr>
<td>vsys</td>
<td>vif_up</td>
<td>ALL</td>
<td>n/a</td>
</tr>
<tr>
<td>vsys</td>
<td>vif_down</td>
<td>ALL</td>
<td>n/a</td>
</tr>
<tr>
<td>vsys</td>
<td>fd_tuntap</td>
<td>ALL</td>
<td>n/a</td>
</tr>
<tr>
<td>vsys_vnet</td>
<td>{private network address}/{net prefix}</td>
<td>ALL</td>
<td>n/a</td>
</tr>
</tbody>
</table>

- The **vsys ipfw-be** tag enables PlanetLab nodes in a slice to use dummynet for emulating queue and bandwidth limitations, delays, packet losses, and multipath effects.
- The **vsys vroute** tag enables the use of vroute script to manipulate routing tables in a secure manner without interfering with other slices.
• The `vsys vif_up / vsys vif_down / vsys fd_tuntap` tags enable the use of special scripts for tunnel creation using TAP/TUN devices.

• The `vsys_vnet` tag associates a slice to an administrator-approved subnet segment, so addresses in that segment can be assigned to the virtual interfaces in an experiment.

Once the slide meets the requirements, we design our scenario as shown in Fig. 2.17:

![Fig. 2.17 PlanetLab and NETNS - Design View](image)

First of all, when you describe your experiment you should be careful to choose IP addresses within the allocated subnetwork for the virtual `TapInterface` and `TunInterface` PlanetLab boxes. Take into account that PlanetLab does not take care of ensuring that no other slice has the same network segment reserved.

As we can see in the picture, all the configurable elements have been explained in other experiments we tested before, so in this case we do not explain how to configure them.

When we run the experiment we have to wait until the PlanetLab node is configured and, when it is ready, the topology of our scenario pops up (see Fig. 2.18).

![Fig. 2.18 PlanetLab and NETNS - Topology View](image)
In this case, as we explained, an xterm console attached to the NETNS emulation pops up, and we are able to ping the remote PlanetLab node. First we check our IP address and then we ping the remote IP address as is shown in Fig. 2.19.

![Fig. 2.19 Issuing a ping command to PlanetLab node](image)

Up to now, we have implemented some simple experiments in each of the three platforms, and we have seen how NEPI works and how to configure them. From now on, we will build and implement more complex scenarios in order to take the most out of the application.

### 2.3. Building Overlay Weaver

Overlay Weaver runs on all platforms which have Java Platform, Standard Edition (Java SE) 5 or newer. It has been confirmed that Overlay Weaver works on Linux and Windows 7/Vista/XP.

In our case we use Linux. Overlay Weaver requires some software that we have to install in advance.

- Java Platform, Standard Edition (Java SE) 5 or later ([10] Java, Online)
- Ant only required to build Overlay Weaver ([11] Ant, Online)

After installing all these prerequisites, we can download Overlay Weaver and build it.

### 2.3.1. Installing Java Platform in PlanetLab

If we want to install Java in PlanetLab nodes, we can choose amongst the three options offered in the PlanetLab web page. If we follow the third option to install
Java Platform, we may be faced to some problems in some PlanetLab nodes. Therefore, the best way to install Java in PlanetLab nodes is installing open source java.

The first step is to log into our slice, and once we are logged type these two commands:

```
sudo yum -y remove java-1.5.0-gcj
```

This command removes the old Java version that is installed in most of the PlanetLab nodes. In our case the version installed is 1.5.0-gcj and if we install Overlay Weaver with this version, the program does not work despite the Overlay Weaver web page confirming that it works with Java version 5 or newer.

Once the old Java version is removed, we need to install a package with the new version. In our case we install 1.6 with the next line-command.

```
sudo yum -y install java-1.6.0-openjdk
```

If we install this version, we will not have any problem when running Overlay Weaver in PlanetLab.

These operations have to be applied to our PC and to all the PlanetLab nodes we want to use in our scenario.

**2.3.2. Installing Ant**

We need to install Ant only in our PC because it is only needed to build Overlay Weaver. Afterwards, once the program is built, we can copy it to any PlanetLab machine.

To get up and running with the binary edition we need to make sure we have a Java environment installed. Subsequently, we need to download Ant, which is available at http://ant.apache.org/. When it is downloaded, the file has to be uncompressed into a directory.

Before being able to run Ant, some additional set up operations have to be performed:

Set environmental variables JAVA_HOME to our Java environment, ANT_HOME to the directory where Ant was uncompressed to, and add ${ANT_HOME}/bin (Unix) or %ANT_HOME%/bin (Windows) to our PATH.

Assuming Ant is installed in /usr/local/ant, the following sets up the environment:

```
export ANT_HOME=/usr/local/ant
export JAVA_HOME=/usr/local/jdk-1.5.0.05
```
export PATH=PATH:$ANT_HOME/bin

Finally, we can check the basic installation by opening a new shell and typing ant. The following message should appear:

Buildfile: build.xml does not exist!
Build failed

It confirms Ant works. The message indicates we need to write an individual build file for our project. Using the ant -version command, we should get an output like

Apache Ant version 1.7.1 compiled on June 27 2008

2.3.3. Building Overlay Weaver

Overlay Weaver web page offers source distribution, binary distribution or check out source code from CVS. When getting the source distribution of Overlay Weaver, it has to be unpacked to prepare the source tree.

Once the source tree has been built, we have to change into the installation directory.

cd overlayweaver

We build Overlay Weaver with Ant.

ant

If the build succeeds, the following message will appear and compiled binaries will be in the target directory. This directory did not exist before this build process.

...  
BUILD SUCCESSFUL  
Total time: XX seconds

Now we are ready to execute the commands provided by the toolkit under bin directory.

2.3.4. Overlay Weaver in PlanetLab

We use SCP to copy files from one machine to another. SCP replaces rcp and should be used instead of ftp. It also has more flexibility than ftp and can be used to copy directories instead of just files. The general form of SCP is:

scp [[user@]host1:]filename1 [[user@]host2:]filename2
Where filename1 and filename2 can be file or directory names. If your user name is the same on both the local and the remote machines, then you do not have to provide the user@.

If you are copying from your local machine, you do not have to provide the name of host1. For example, to copy a file called temp.ps from a local machine to agave.tamu.edu, the following command can be used:

```bash
scp temp.ps agave.tamu.edu:temp.ps
```

The file temp.ps will be copied to the home directory on agave. If the user name on agave is different, then it has to be specified as shown below.

```bash
scp temp.ps remote-user-name@agave.tamu.edu:temp.ps
```

The “-r” flag allows to recursively copy subdirectories:

```bash
scp -r temp.ps remote-user-name@agave.tamu.edu:temp.ps
```

So in our case, we go to the directory where our file is located and type:

```bash
scp overlayweaver.tar.gz upcple_confine@planetlab1.s3.kth.se:overlayweaver.tar.gz
```

Where upcple_confine is the slice name and planetlab1.s3.kth.se the PlanetLab node where the file will be copied.
CHAPTER 3. TESTING NEPI

Now that we know how to create basic scenarios in NEPI, we need to study the application in depth and investigate what we can do in NEPI. It is necessary to know its real development stage, the functions that it has implemented and the ones that are still missing, as well as to know what it could eventually be used for. For that reason, in this chapter we are going to build a scenario that mixes the different platforms we analyzed in the last chapter.

3.1. Test Scenario

In the scenario we are building, we are going to create a P2P network with four nodes connected. Once the scenario is build some tests will be performed in order to evaluate the network behaviour. If any node fails, we will mainly focus on the network and the NEPI application stability.

3.1.1. Setting up NEPI

The scenario used in this chapter includes many details which have to be set. For this reason, we will consider the scenario to be like three smaller and simpler scenarios, one for each testbed used.

3.1.1.1. NETNS

As in other chapters, we will start explaining how to configure and set up the NETNS scenario.

First of all, when we work with NETNS in such a scenario we usually want to run an application remotely from another machine but displaying it at our local computer.

To do that it is necessary to export the display to our local machine by typing the following command:

```bash
export DISPLAY=:0.0
```

The NETNS scenario consists of two different nodes each one connected by a tunnel to another node of Planet Lab. On the other hand, we have an interface that connects the two NETNS nodes through a scenario in NS3. To build up such a scenario we have proceeded as follows.

We build a testbed instance, where we insert two node devices and configure them.
Afterwards, we need to draw an application box and connect it to both nodes one per node. As it has been explained in Chapter 2 we need to configure the user and the command we want to run. In our case, an `xterm` command is set up in both application boxes.

To interconnect the nodes to another node of NS3 it is necessary to add an interface device. The graphical interface provides five types of devices, and we chose to create a Tap node interface. A network tap simulates a link layer device and it operates with layer 2 packets such as Ethernet frames. The only setting that has to be modified in these devices is the up attribute to indicate at the beginning of the test if the interface is up or down. In our case its value is True in both cases.

The following step is to interconnect the nodes to some PlanetLab nodes. To get it done, we need another interface device. In this case we only explain one of the three connections we have established in order to simplify the explanation. In this case the graphical interface provides five types of devices, and we chose to create a Tun node interface. Tunnel interfaces are used to connect things that may otherwise not be connected. It is a virtual interface, but you need to be really careful when placing them in your network so it is very important to know the correct IP address. As we explain in 2.2.3 the `vsys_vnet` tag is very important when making tunnels. The configuration of the Tun node interface is as simple as in tap node interfaces, we only need to set the attribute up and change it to true.

But when a tunnel is created, a channel for this tunnel has to be added in NETN. That is why a TunChannel box has been added. Moreover, it has to be connected to the Tun node Interface box. In our case, no extra configuration is needed.

If we try to run this scenario, it would not work because it requires completing the tunnels that connect to PlanetLab devices and tap devices with some ns3 nodes.

The design view of this scenario is shown in Fig. 3.1:

![Fig. 3.1 NETNS Design View of the final Scenario](image-url)
3.1.1.2. NS3

The NS3 scenario is the simplest one. There is only one node with two interfaces that interconnect the two netns nodes explained in the previous section. To create this scenario we build an NS3 testbed instance and add a node in the testbed. There is one configuration option we can use in this node box: changing the label name.

In NS3 we must define which protocol is being used and configured; thus, we need to add some different protocol boxes connecting them to the previous node.

In our scenario we add the following protocols: Icmpv4L4Protocol, Ipv4L3Protocol, TcpL4Protocol, UdpL4Protocol and ArpL3Protocol. None of the boxes are configured.

Finally, we create two FdNetDevice boxes and connect them to the NS3 node on the one side and to the NETNS node on the other. We do not need to add any extra configuration. In Fig. 3.2 we can see the design view of ns3 while in Fig. 3.3 we can see the connection of NS3 and NETNS testbeds.

![Fig. 3.2 NS3 Design View of the final Scenario](image1)

![Fig. 3.3 Connections between NETNS and NS3](image2)
3.1.1.3. *PlanetLab*

The PlanetLab scenario is the one which took more time and efforts to build up. The reason is that when we try to configure and test the nodes, many problems and errors arise.

The initial idea when we started to build the testbed was to connect a NETNS node to a PlanetLab node, and connect the latter to the Internet. In so doing, supposedly the rest of all PlanetLab nodes involved in the test would be connected through internet.

As we can see, the initial idea was implemented as shown in Fig. 3.4:

![Fig. 3.4 First connections between PlanetLab and NETNS](image)

After setting all the nodes, some tests were executed to validate the connection between all the nodes. We intended to proof that all NETNS nodes were aware of the whole scenario. However, it was found that the ping command did not reach any PlanetLab node except for the one that was directly connected to the NETNS node.

As a result of this, we had to check all the IP tables to understand what the problem was. After some tests, we could prove that some IP routes were not created. Consequently, we had to get into contact with the NEPI team to make us sure we had found the problem and to look for a solution.

Therefore, the solution was to create more tunnels, one to each PlanetLab node. So we dismissed the initial idea and used a completely new design to build the testbed: we connected the NETNS node to each PlanetLab node using a tunnel as it is shown in Fig. 3.5. However, this solution was not available because an error arose right before running the experiment. Due to this error, we could assume that there were too many devices connected to the same node.
The third and last idea was to create two tunnels from two PlanetLab nodes to the same NETNS node and another tunnel from one PlanetLab node to the other NETNS node. A picture of the final design view of PlanetLab is as follows:

As you can see in Fig. 3.6, the PlanetLab testbed consists of only 3 tunnels, so in order to simplify the description of this scenario configuration, we will only explain how to build one tunnel in PlanetLab.

To create the scenario we build a PlanetLab testbed instance where we add a node. The only configuration option allowed in this node box is to set the
hostname. As we have just explained, our scenario has 3 nodes, and the hostnames used in our case are:

- onelab3.info.ucl.ac.be (Université Catholique de Louvain)
- planetlab4.hiiit.fi (KTH, Royal Institute of Technology, EE)
- planetlab1.s3.kth.se (Helsinki Institute for Information Technology)

Once the node box is created, we have to add an Application box and connect it to the appropriate node. The command that the application box runs is explained in section 2.2.3. In order to complete the tunnel explained in NETNS section, another device had to be added. Thus, a TunInterface box is added in our testbed to have the tunnel completed. As in other cases, we only have to configure the "up" attribute.

Finally another interface is created in order to reach Internet, so a NodeInterface device is added. We do not need to configure anything in this device, but we need to connect one side to the node and the other to a channel box called Internet.

Once the procedure to configure a node has been explained, we only have to repeat the process twice again. The final design view of the entire scenario is shown in Fig. 3.7:

3.1.1.4. IP addresses & routes

Before running the experiment and finishing the NEPI configuration we need to add an IP address to all the interfaces, as well as to add routes to all the nodes.

In Fig. 3.8 we detail all the IP addresses used in our test scenario and explain why we are using them. Moreover, we explain the IP route that must be edited in each node in order to have all nodes connected.
As it has been mentioned, our PlanetLab slice uses network 192.168.10.0/24, so most of the networks used in the scenario will be sub netting thereof.

This is so because we need to do tunnelling, and if we need to use more than one network coming from the 192.168.0.1/24 network we must do IP/27 networks. So in this case, we will have 3 digits to indicate the different subnets. If we use IP/27 some subnets will be left over but if we use IP/26 we will not have enough subnets.

Out of all the networks we get after doing subnetting, we use the ones that are noted in Fig. 3.9.

Apart from these networks, we need to access a particular IP, which specifies the PlanetLab node we have chosen, so we must add three more IP’s to our scenario.

- onelab3.info.ucl.ac.be: 130.104.72.213
- planetlab4.hiiit.fi: 141.20.103.211
- planetlab1.s3.kth.se: 193.167.187.186
Once we have analysed the networks used, we must explain the routes that each node must have in order to be able to establish a connection between all the nodes and thus be able to ping end to end.

The route that must be added to the PlanetLab nodes is very simple, since each node only has to add the path of the tunnel. The routes of our three PlanetLab nodes are as shown in Fig. 3.10:

![Fig. 3.10 Routes added in PlanetLab nodes](image)

The routes of the three nodes that we still have not explained are more complicated. In this case, apart from adding the three IP’s of the PlanetLab nodes, we will need to add all the networks that do not directly connect to each node.

Nodes are configured as shown in Fig. 3.11:

![Fig. 3.11 Routes added in NETNS nodes (top) and NS3 (bottom)](image)
3.1.2. Application setup

In this section we are going to explain the settings of the command attribute of all the application boxes in our scenario.

In Overlay Weaver multiple instances of DHT shell are invoked and DHT is used via these shells. The minimum required number of computers is one, but in our scenario we use a higher number of nodes.

The following instructions assume that the toolkit is installed (extracted and built), and the PATH environment variable is set to execute the toolkit commands in bin directory. These commands can also be executed without setting the PATH by simply preceding the commands with the path of the bin directory. All these instructions are explained in SECTION 2.3.4.

To construct an overlay, we need to invoke the first DHT shell. Due to this in one of our PlanetLab nodes, the attribute command has to execute this first DHT shell this way:

```
/home/upcple_confine/overlayweaver/bin/owdhtshell -n
```

In fact, to execute a DHT shell we only need to type "owdhtshell", but we need to specify the path where the program is installed. Moreover, we must add an "n" command-line argument so as to avoid reading instructions from standard input. This option is useful to invoke this tool via a job management system.

We need to type the same command for the other two PlanetLab nodes, because if we want to construct an overlay we need to establish contact. So the command attribute for the rest of the PlanetLab nodes is "owdhtshell <hostname or IP address of the computer>". In our case:

```
/home/upcple_confine/overlayweaver/bin/owdhtshell -n 130.104.72.213:3997
```

We still have two more ApplicationBox without any command set. At the very beginning of the test, we set them as an "xterm" to ping the nodes and make certain all nodes are connected. At this point, the application box of node2 is not used, so we can delete it from the scenario or leave it there without any command. However, the command xterm is useful in the application box because when we run the experiment, and the shell comes up, we can manually type the same command as in PlanetLab nodes.

In this case, the shell prompts us to give a command by showing "Ready."
Now we can type any shell command. For example if we type the "status" command we can see a routing table of the node like the following shown in Fig. 3.12:
As we can see in the picture, all nodes are connected and the overlay is constructed.

When the overlay is constructed we need to enter a value in DHT. That is why we do all the process manually in the NETNS node, because now we have to interact with all nodes.

To put and get values, when running the experiment, we need to enter the following commands in our shell.

We can type the following command to put a value to DHT.

```
put variable value
```

We can use the status command to see the result of routing when using the put command.

You can see in Fig. 3.13 that the key "variable" is hashed to b46d017243d6895dac7544b9daccbb87b361e9f and the route length was 2.

---

**Fig. 3.12** Example typing "status" command

**Fig. 3.13** Example when adding variables in the network
Type the following command to get the value you have put.

```bash
get variable
```

You see the value "value" you have put. 
key: b46d0172433dd6895dac7544b9dacbb87b361e9f
value: value

The number 10638 is the TTL. It is the remaining lifetime of the variable-value pair.

Finally, we must take into account that the default routing algorithm and routing style are Chord and iterative routing. We can change them by specifying options to the DHT shell.

### 3.1.3. Final Configurations

Before running the experiment, we need to finish configuring all testbeds. When we add a testbed in NEF, we can configure some attributes.

Some errors will come up in PlanetLab if we do not set this configuration. This is because the testbed needs some parameters to access PlanetLab and work properly. In NS3 and NETNS, however, the experiment works even if we do not configure the testbed and use all the default values. Nonetheless, it may work in an undesirable way.

As we mentioned before, the main configuration options we have to set in PlanetLab testbed are used to indicate our UserName, UserPassword, SliceSSHKey and Slice Name. We preferred to change the LogLevel and pILogLevel values. Thus, we will be outlining in more detail how to configure the node. Therefore, if a failure occurs we can easily see the failure point.

As we can see in Fig. 3.14, this is our PlanetLab testbed configuration. Both PlanetLab testbeds in our scenario have the same configuration so we need to do it twice.

![Fig. 3.14 PlanetLab Testbed configuration](image)
In NETNS and NS3 the testbed configuration is the default one even though we can change the LogLevel value or the Home Directory where the experiment saves temporary files.

Our configuration is as shown in Fig. 3.15:

![Fig. 3.15 NETNS (Left) and NS3 (Right) Testbed configuration](image)

Finally and before running the experiment, we need to add some traces in order to study the network. In our case we need some "pcap" files to study it.

To add these traces in PlanetLab we select all the Tunnels Interfaces and select Add Traces. A new menu appears where we can select packets or pcap (Fig. 3.16). In NETNS, however, this option only appears in the Nodes boxes. In the same way, we choose both scenario nodes and select pcap traces. To finish up the configuration, we get to NS3. In NS3 pcap traces are neither in the nodes nor in the tunnels, but in the FdNet devices. Therefore, we select all devices and add the traces.

![Fig. 3.16 Adding traces to the scenario](image)

### 3.1.4. Running the experiment

Once all devices and testbeds are configured, we can execute the experiment. Before running it, we recommend to visit the PlanetLab webpage and log on it. It is advisable to visit our slice and check if all the nodes used in the scenario
are or not available. When we sometimes configure a scenario all nodes seem to be available, but some days later, they happen to be disabled with a fail boot, or are in maintenance.

Furthermore, the experiment will take its time as all PlanetLab configurations and some dependencies must be established to create the tunnels. For all these reasons, we need to wait between 5 and 10 minutes to finish running the experiment.

While the program is running, we can see all the configuration processes in the shell as in Fig. 3.17. It may sometimes seem stopped, but in fact it is not. We only need to be patient.

![Configuration processes while running the program](image)

Fig. 3.17 Configuration processes while running the program

Once the experiment is running, a topology view will activate and a new command shell will appear, as we configured it in NETNS. The topology view of our scenario is as shown in Fig. 3.18:

![Final Scenario - Topology View](image)

Fig. 3.18 Final Scenario - Topology View

So, in the new shell, we type a command line with the path were Overlay Weaver is installed, and then execute it with the command -s as follows:
cd

cd overlayweaver/bin

./owdhtshell -s 192.168.10.130 130.104.72.213:3997

We need to wait until we see:

DHT configuration:

- **hostname:** 192.168.10.130
- **port:** 3997
- **transport type:** UDP
- **routing algorithm:** Chord
- **routing style:** Iterative
- **directory type:** VolatileMap
- **working directory:** .
- **initial contact:** 130.104.72.213:3997

A DHT started.

Ready.

Once we are connected, we can see the nodes that are connected typing `status`, or we may put and get values to study the network as has been explained in section 3.1.2.

If we want to stop the scenario, we just need to click on Execute and later in the Stop button. The Traces View windows will automatically appear and we can choose which pcap file we want to download as it is shown in Fig. 3.19. After downloading all files by selecting them one by one, we can return to the Execute menu and click on Shut down to have the test finished.

![Fig. 3.19 Traces View tab after stopping the test](image-url)
CHAPTER 4. TESTING & OUTPUTS

Up to this point, we only need to check if NEPI outputs are the ones we expected. Moreover, we need to evaluate how flexible NEPI is when trying to test "life networks". In this chapter we are going to make some tests in two different scenarios. One with fixed nodes which are always ON, and another one where the nodes have some deliberately caused fails and generate some OFF-ON behaviour.

4.1. Test Scenario

To study NEPI feasibility and to bring it closer to a real situation we need to be able to turn nodes on and off. To enjoy such a possibility, the GUI offers an UP attribute in all the interfaces, so we can change this value during runtime. However, this function is not supported right now. Some solutions were implemented to try to solve this problem, but we finally decided, as a limited solution, not to set the link down, but rather prevent Overlay Weaver traffic closing the application.

The application has to be closed manually, so we must first connect to the node through SSH. To kill the application the PID number is required, so we type the TOP command line and search our application. The application we want to kill is JAVA, so as we can see in the picture, we kill PID 5640 (in the picture case). To quit the top command we only have to type "q". After knowing the pid, we can proceed to kill the application as it is shown in Fig. 4.1.

To open the application again we type a command line with the path were Overlay Weaver is installed, and execute it with the command -s as follows:

```
/home/upcple_confine/overlayweaver/bin/owdhtshell -n 130.104.72.213:3997
```
4.2. Outputs with nodes on

To evaluate our scenario when all nodes are connected (we can check it by issuing a "status" command) we need to add some values using the put command. The values we want to add to our scenario are shown in Fig. 4.2:

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Jordi</td>
</tr>
<tr>
<td>Surname1</td>
<td>Perez</td>
</tr>
<tr>
<td>Surname2</td>
<td>Rueda</td>
</tr>
<tr>
<td>Age</td>
<td>27</td>
</tr>
<tr>
<td>Country</td>
<td>Spain</td>
</tr>
<tr>
<td>City</td>
<td>Barcelona</td>
</tr>
<tr>
<td>DNI Number</td>
<td>12345678K</td>
</tr>
<tr>
<td>Gender</td>
<td>Male</td>
</tr>
</tbody>
</table>

Fig. 4.2 Variables and Values table

We add all these values to make sure that all nodes have at least one value saved.

Afterwards we can retrieve all the variable values using the get command, as it has already been explained in section 3.1.2. As we can see in Fig. 4.3, we can retrieve all the added values.

![Example adding values](image)

Fig. 4.3 Example adding values

So, in this case, the scenario is working properly and all the data values can be retrieved without any father problem. As we can see in the pcap file (Fig. 4.4), all transmissions use UDP protocol, and we observe different IP source and destination nodes.
If we study the different downloaded files in a network analyzer, all the files seem to have the same content. The only difference is the number of packets studied because the nodes have a different volume of traffic.

4.3. Outputs with nodes off

To evaluate the network with some nodes off, we need to do some adjustments to the scenario. The first settings are similar to those in the last section of 3.1.3. Nonetheless, some variables and some values have to be added to check what happens if a node fails.

Once the values are added, we have to add the following settings to make a node fail. The only two nodes we can set to fail during running time are the NETNS nodes. However, being the ones with one or more tunnels connected, if they fail we will not be able to access PlanetLab.

If we change the Up attribute in one of these nodes, and we run the Overlay Weaver application, an error comes up.

For this reason we decide to set a PlanetLab node to fail. In our case, we choose node 41 that has the hostname, planetlab4.hiit.fi. To make it fail, we close the OverlayWeaver application of that node, and right when the application gets closed, the same error we found before comes up at our shell.

Now we proceed to get the variable values to check if there was any failure and if so, how many.
In this case, we can retrieve all the values except for three of them: Surname2, Country and Age.

![Fig. 4.5 Some values cannot be get](image)

As we can see in Fig. 4.5, when we try to find and retrieve these variables an error appears. However, if we can find and retrieve properly the rest of values.

At this point, we add the same node again, and evaluate what happens. If we open the Overlay Weaver application in the remote node, the errors disappear. Now, when we try to find the Surname2, Country or Age variables, the result is that they are empty and have been erased as it is shown in Fig. 4.6.

![Fig. 4.6 Empty values when nodes turn on again](image)

If we want to retrieve them, we just need to add these three values again.

On the other hand, if we analyse the pcap files using a traffic network analyzer, in this case we find there are some different packets in our capture. These different packets appear when we close the Overlay Weaver application and type the get command.
As we can see in Fig. 4.7 when the scenario is trying to find the destination port, it is unreachable, because the application that opens this port is closed. For that reason, some ICMP packets appear in the capture in the middle of our experiment. However, when we finally open the application again, these packets disappear the rest of the capture.

By means of conclusion, we can assert that NEPI works properly and can provide different ways to study the traffic or analyse the network. However, it is still under development as there are many commands and functions that are provisional or still not working. But, summing up, the results obtained in this chapter are the ones we expected, so we may say the program works properly.
CHAPTER 5. PROBLEMS & LIMITATIONS

After working during some months with the application and having tested and analysed its outputs, we could observe some problems and limitations in NEPI. In this chapter, we sum up all the problems that have arisen while developing the project and the way we solved them. Some of these problems could be solved thanks to the help of NEPI creators, who, taking into account it was still in its early stages of development, were very willing to help us and were grateful for our comments.

5.1. Problems with NEPI

The first problems using NEPI appear at the very first installation. When we have to build NS3, we need an appropriate version of g++. NEPI NS-3.9 version compiles well using g++ version 4.4.5, but we get an error when using g++ version 4.6.1 for ubuntu. There was an NS-3.11 version for NEPI almost ready, and this new version should not have these compilation problems.

While they were finishing this new version, the webpage links to download 3.9 version were broken; however version 3.11 works properly, so we succeeded to compile NS-3.11 or 3.12. Nonetheless, the problems now arise when running NEPI, as some libraries are missing.

We had to wait until the NS version in NEPI was updated to 3.11. The installation instructions on the experiment page were also updated and an installation script was available to make things easier.

Moreover, we have to take into account that NEPI is still in an unstable stage, so we might bump into some bugs. NEPI developers were pleased to receive any suggestions we had to improve NEPI from the user/researchers’ perspective.

Further evidence that the project is still in an unstable stage became clear when they told us to register in the NEPI users’ mailing list. We thought it could be a good idea to discuss problems there, so the discussion/solutions could be available for future users. However, when we tried to register in the mailing list, we discovered no one had ever registered as the mail address was not complete and nobody had reported that error.

Once we succeeded in installing NEPI, some problems arose when using NETNS and trying to connect two nodes. The NEPI team explained us how to do it. First, they supposed the problem was related to the switch device and the kernel version. As a first solution they recommended us to use Tap devices and TunChannels. Two days later, they concluded saying the problem had to do with the iptables configuration for bridge devices, as NETNS switch component uses, amongst other things, a bridge device. Therefore, the problem was not related to the Linux version but to the machine configuration.
They told us what commands to use for the switch configuration.

```
cd /proc/sys/net/bridge
# echo 0 > bridge-nf-call-iptables
```

Furthermore, we had some problems using PlanetLab and all the web site examples, and they told us that the examples were outdated. Some days later, the web site was updated and the NEPI team told us the right configuration of PlanetLab.

At this point, the main problem was connecting NETNS and PlanetLab, because we needed to set the tags in our PlanetLab slice, logging into the webpage and adding them. We could not do this because our user did not have enough privileges to complete the operation.

We waited for some days searching for a UPC PlanetLab member who could add these tags in our slice, but we were not successful. We informed NEPI developers about our problem and they gave us a solution. If all slice users agreed, they could add the tags to the "upcple_confine" slice themselves to investigate this problem.

While all the PlanetLab problems were being solved, we were building some experiments using only PlanetLab nodes. Tags are only required in scenarios mixing NETNS and PlanetLab. So, we had more doubts in its use when we tried to see a remote shell that was running in PlanetLab node. But "xterm", does not work for experimentation environments that are not local to our computer. When we run a NETNS instance locally (without indicating a remote host in which to run the controller), the xterm command was executed in our machine, but in PlanetLab the "xterm" command was executed always in a remote node. So, without X forwarding we could see nothing and NEPI doesn't yet offer support to such operations.

Knowing that and once the tag problem was solved, we mixed some NETNS and PlanetLab nodes, but we came across another problem. In our slice we had a list of available nodes, but not all of them were able to work in this type of scenario. We had to try different hostnames, one by one, until we found those that worked.

Moreover, some nodes failed if we did not state the operating system, so we had to indicate the operating system properly. We had to try the different operating systems available in NEPI for each node in order to find the combination that worked. Thus, we can say that not all nodes support such scenarios.

Another type of questions we had to face were related to putting a node temporary off and returning it up later. The NEPI team told us that the way to "put a node off" in PlanetLab was to set down a network interface to disconnect the node from the network. Real network interfaces cannot be put off because they are shared by many slices. Virtual interfaces (TAP/TUN devices) can be put off by executing:
Testing NEPI usability and features

```
        echo "<$iface_name>" > /vsys/vif_down.in
```

When we want to put them on again we have to execute:

```
        echo -e "<iface_name>\n<ip>\n<net prefix (ex: 24)>" > /vsys/vif_up.in
```

But currently NEPI offers no support to do this transparently. So the only way to work around this was to explicitly add PlanetLab Application boxes with these commands.

For instance, we added an Application box with the command "sleep 5; echo '<iface_name>' > /vsys/vif_down.in", which puts off the tap named 'iface_name' 5 seconds after the command is executed.

To reference the iface_name during design time (when you don't know it), you need to use 'wildcards' like {#tap-cli.[if_name]#}. Where 'tap-cli' is the value set in the 'label' attribute of the tap interface box. The command will then look like:

```
        sleep 5; echo '{#tap-cli.[if_name]#}' > /vsys/vif_down.in
```

However, this did not work properly because, although the node put the interface off, it did not come up again after the "sleep time". Therefore, we reported the problem again and the NEPI team told us that the vsys/vif_down.in script that is responsible for switching the link down leaves the interface unusable. Because of PlanetLab deployment constraints it is not possible to do right now a world wide deployment of a fixed script to solve the problem. However, it will be done as soon as possible.

The last problems were related to the final scenario. When we mixed the three platforms, there were a lot of nodes and, as we explained in 3.1.1.3, the only way to connect PlanetLab and NETNS is using tunnels.

Another question arose from the number of interfaces that a node can support. When we tried to connect 5 interfaces to the same node, the latter was not able to add so many routes in its IP route table. So we had to change our scenario.

Apart from all of these problems, we have to take into account NEPI usability. When running a scenario, it did not work properly. Errors often come up, forcing us to close the program and open it again. Nowadays, the experiment runs and works perfectly, but it sometimes has a bug. This error has not been reported because maybe due to having the experiment started and stopped far too many times. Sometimes, we could not stop it in a proper manner, so it did not work again when restarted. This problem sometimes happened the first time we run the experiment, so we had to close the application and start it again. However, from time to time we even had to reboot the machine to make it work again.

Finally, we have received an e-mail informing us that most of the errors reported here have already been solved in a new version of NS-3 and now works perfectly in NEPI. The updated version is 3.13, but we did not have time enough to test or check how these adjustments work in this new version. Our project has been using version 3.11.
By means of conclusion, we can say that, even if this program is still in an unstable state, and we find some failures, developers make it easy to work with because they answer very fast and steadily. They explain everything you need, and apply changes in the website. While we have been doing this project, NEPI has change a lot, which means future users will not have to go through the same doubts.

5.2. **PlanetLab Limitations**

Slices are created on demand and expire after two months. When a slice expires, it is destroyed. When a slice is destroyed, all files in the slice are removed from all nodes assigned to that slice.

The expiration date of a slice may be extended by using the Renew Slice form. There is no limit on the number of times a slice may be renewed, but the expiration date may never be set to more than two months into the future.

Slices also have resource limits. Disk space, memory usage, file descriptors, and bandwidth are controlled on a per-slice, per-node basis.
CHAPTER 6. FUTURE LINES & CONCLUSIONS

In this last chapter, we mention some possible further developments to our project. We indicate how to continue this project in the future. Testing a tool is not a task that finishes the day we perform the test, let alone if the tool is still in an "unstable" state. We believe that in every new version the state will be better and if the number of users increases the problems will drastically decrease. In this chapter we emphasize the possible future projects that could be planned after finishing this one.

6.1. Future work

This project could move forward in many ways. One of the first things to do is to check the changes that have been implemented in NEPI version 3.13 or later.

Despite we have been testing NEPI, there is still a lot of work to do in order to fix it and test it. I do not think many users nowadays have used NEPI because during the project I have not received a single user’s mail from the mailing list. Therefore, starting to use it to test real scenarios and get rid of all the bugs would be a good way to help the project advance, and make it a better program.

Another future project line would be testing NEPI along with other applications. In this project we used a P2P application, but NEPI can be tested using many other applications such as streaming video in order to learn how NEPI behaves in these environments. If the new changes and developments work and the nodes can jump more easily and have you connected again, we could perform many more tests or on-demand video streaming.

NEPI is a tool in constant growth and development. We do not know how it will evolve in the future, but having it re-tested and compared taking into account the conclusions that we draw from this project may be another interesting line of work.

6.2. Conclusions

During the course of this project we became aware of the problems related to the design and implementation of testbeds, as the need to learn different programming languages and different work environments can make it very costly. First of all, we want to express our gratitude to the NEPI team for the great job they are doing in trying to unify different platforms and tools into a single one using a single programming language. Although the web page says that “NEPI is in an ‘unstable’ version”, they keep on developing and improving it.

During this project, we have been able to test NEPI and build up different scenarios, from the simplest ones to far more complex ones that mix different platforms.
In my opinion, NEPI performs rather satisfactorily because, as we managed to confirm, it can be used to create different kinds of scenarios and gather all wanted results thanks to its flexibility.

Apart from that, we want to highlight the important role that NEPI developers played in this project. When any problem arose, they helped us very fast by modifying, searching or reproducing our error until a solution was found. They made their best to try to solve every problem. The average response time has been of 24 hours. They may not have found the right solution in such a short time, but they answered us giving some advices on the problem.

On the other hand, as NEPI users, we have realised that there is still a lot of work to do in the development and improvement of NEPI, but, at the same time, we can confirm that the NEPI team works hard every day to make their application better. For example, they update their web page once a week adding, modifying or clarifying information. Moreover, in my opinion, NEPI needs more test users to advance further and fastest. The work conducted under the present project has allowed us to detect and report some fails, errors and limitations that will surely improve future versions of NEPI. This entire project is based on the version that is currently available on the NEPI web page, but we have been informed that in the very near future they will upload a new and updated version. This new version will include some patches that solve some of our reported errors as well as other modifications.

To sum up, NEPI is an application still in development, but that can already be put to work. We might label it a BETA version as some errors still arise while using it. However, NEPI has exceeded our expectations with respect to flexibility and its results. Furthermore, it includes a very helpful interface that greatly simplifies the tasks of configuring and programming testbed scenarios.
BIBLIOGRAPHY

Web pages

[1] Network Emulation

[2] Network Simulation

[3] Distributed Testbed

[4] NEPI Project Homepage
<http://nepihome.org/> [Retrieved on June, 21st 2012]

[5] NS-3
<http://www.nsnam.org/> [Retrieved on May, 30th 2012]

[6] PlanetLab
<http://www.planet-lab.org/> [Retrieved on May, 30th 2012]

[7] PlanetLab Europe
<http://www.planet-lab.eu/> [Retrieved on May, 30th 2012]

[8] Overlay Weaver
<http://overlayweaver.sourceforge.net/> [Retrieved on May, 30th 2012]

[9] Fedora Project Homepage
<http://fedoraproject.org> [Retrieved on May, 30th 2012]

[10] Java

<http://ant.apache.org/> [Retrieved on May, 30th 2012]

[12] VLC Command-line

[13] Linux Manual Online
<http://linux.die.net/man> [Retrieved on May, 30th 2012]

[14] The Linux Foundation
<http://www.linuxfoundation.org/> [Retrieved on May, 30th 2012]

[15] Python Programming Language
<http://www.python.org/> [Retrieved on May, 30th 2012]

[16] Wireshark
<http://www.wireshark.org/> [Retrieved on May, 30th 2012]

Mailing Lists

[17] NEPI Mailing List: nepi-users@nepihome.org

[18] Overlay Weaver List: overlayweaver-discuss@lists.sourceforge.net

Others: