Preliminary Notes

The instructions in this document are applicable to personal computers running the following Operating Systems:

- Microsoft Windows from version 7 up to 10
- Apple Mac OS X from versions 10.8 'Mountain Lion' up to 10.11 'El Capitan'
- Debian-based Linux, such as Ubuntu (recommended) from versions 12.04 'Precise' to 14.04 'Trusty Tahr'.

It is not recommended to apply the setup to a virtual machine, although possible, as the configuration requires an hypervisor environment (Virtualbox) in the host system.

For the aforementioned Operating Systems, the setup will use 'package managers' in order to turn the process well controlled (versions, updates), simple to 'cleanup' and not 'affecting' the host environment.

1 Setting up the Environment

This section describes the minimum setup of the experimental lab environment for systems running Windows, Mac OS X and Linux.

1.1 Microsoft Windows Setup

The recommended Package Manager for Microsoft Windows is Chocolatey, like the Linux apt-get or yum [https://chocolatey.org/]. It was designed to be a decentralized framework for quickly installing applications and tools, and is built on the NuGet infrastructure, currently using Windows PowerShell.

First, create a directory for your projects, for example:

```
C:\> mkdir C:\Users\userid\Desktop\MyProjects
```

To install Chocolatey, open an an elevated (i.e. Admin) Windows Command Prompt and paste the text string (following the command prompt) from the box below and press Enter.

```
C:\> @powershell -NoProfile -ExecutionPolicy Bypass -Command "iex ((new-object net.webclient).DownloadString('https://chocolatey.org/install.ps1'))" & & SET PATH=%PATH%;\%ALLUSERSPROFILE%\chocolatey\bin
```
For a better command prompt alternative for Windows, install ConsoleZ, a program that provides many useful features like tabs, layout changes, support for Windows clear type, keyboard shortcuts and many other:

```
C:\> choco install consoleZ
```

Once installed, launch ConsoleZ by calling ‘console’ from the command prompt. Optionally, ConsoleZ can be ‘pinned’ as icon to the taskbar. In the menu of Console select Edit – Settings, and under ‘Console’, set your default Startup Directory, for example, C:\Users\Admin\Desktop\MyProjects. Under ’Appearance’, set the font to Consolas 11, Black background. Under Behavior set ‘Copy on Select’. Under Hotkeys, change the ‘New Tab 1’ hotkey to Ctrl-T, ‘Copy Selection’ to Ctrl-C and ‘Paste’ to Ctrl-V. Under ‘Tabs’, verify that the ‘Main’ is set to PowerShell, and if run as ‘administrator’.

It is now time to install the other packages for the Lab experiments environment. Start a Console and install the following programs using Chocolatey:

```
C:\> choco install wget
C:\> choco install git
C:\> choco install virtualbox
C:\> choco install vagrant
```

After this point, and in order to settle everything (Windows is quite tricky in updating environment variables), close the Console and relaunch it again. Additionally, as an SSH (secure shell) client is not distributed with Windows, it is necessary to tell the system where to find the SSH client installed with GIT:

1. Open the Control Panel
2. Go to System and Security
3. Click on System, then on the Change Settings button
4. Display the Advanced tab and click on Environment Variables...
5. Look for the Path variable in the System variables list, select it then Edit...

At the end of the string, add the following path to Git’s bin:

```
C:\Program Files\Git\bin\;C:\Program Files\Git\usr\bin
```

Note: You can uninstall Chocolatey by simply removing the (invisible) folder C:\ProgramData\chocolatey (environment variable %ProgramData%), since it is not actually ‘installed’ on the system.
1.2 Apple Mac OS X

The recommended Package Manager for Mac OS X is Homebrew, like the Linux `apt-get` or `yum` [http://brew.sh]. Homebrew installs packages to their own directory and then symlinks their files into `/usr/local`.

As a first step you may need to install **XCode** from Apple (free from the App Store), in order for Homebrew to be able to compile applications from their source code. Macs do not have any of the developer's 'Command Line tools' installed by default, so we need to install them before we can get anywhere. After having XCode run the following command in terminal:

```
$ xcode-select --install
```

To install Homebrew, open Terminal and at the shell prompt paste the text (starting at the `ruby` invocation) from the box below and press enter.

```
$ ruby -e "$(curl -fsS https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

The Terminal will tell what it is about to do, and ask you if you want to proceed: press **Enter** to do so. The Terminal may then ask for a password: this is the password to the Admin account on your computer. Type your password, and press **Enter**. When it's done, the Terminal will say that the installation was successful, and ask you to run `brew doctor`. Do as it suggests:

```
> $ brew doctor
```

This will make Homebrew inspect your system and make sure that everything is set up correctly. If the Terminal informs of any issues, you will need to fix them, and then run `brew doctor` again to verify that all was correctly fixed. When everything is set up correctly, you will see the message 'Your system is ready to brew', and you can move on.

Homebrew installs packages to their own directory under `/usr/local/Cellar/` and then symlinks their files into `/usr/local`. Homebrew Cask extends Homebrew and brings its elegance, simplicity, and speed to GUI Mac OS X applications, installing the OS X Apps under `/opt/homebrew-cask/Caskroom/` and then symlinking their files the user ~/Applications/ folder.

It is now time to install the other packages for the Lab experiments environment. Open Terminal and install the following programs using Homebrew:

```
$ brew install wget
$ brew install git
$ brew install caskroom/cask/brew-cask
$ brew cask install virtualbox
$ brew cask install vagrant
```
For each command the Terminal will start displaying lots of information, as Homebrew is keeping you updated on what it is doing. This flow of information will be a guide to let you know whether or not the computer is still working, and so, do not interrupt it.

Now it is time to create a directory for your Lab projects, for example:

```bash
$ mkdir ~/Desktop/MyProjects
```

Note: You can uninstall Homebrew and all packages with the following command:

```bash
$ rm -rf /usr/local/Cellar /usr/local/.git && brew cleanup
```

### 1.3 Linux

For a Linux system, namely Ubuntu, the standard package manager ‘`apt-get`’ is already present, as well as some packages, such as ‘wget’.

To install the other packages open a Terminal and issue the following commands (answer Y to the prompts, and several dependencies will also be installed).

```bash
$ sudo apt-get install git
```

Please note that for latest distributions of Ubuntu, such as 14.04, 14.10 or 15.04, a careful install of virtualbox is required as the package architecture has to match the Linux kernel architecture. Additionally, the VirtualBox package in Ubuntu (as of release 14.04) has some issues when running in RAM-constrained environments.

For that purpose you need to ensure that you have the adequate sources referenced for your kernel architecture (14.04 'Trusty'/ 14.10 'Utopic' / 15.04 'Vivid').

Start by editing the sources list issuing the following command:

```bash
$ sudo nano /etc/apt/sources.list
```

Add the following line to the list (exemplified for 14.04 'Trusty' kernel):

```bash
deb http://download.virtualbox.org/virtualbox/debian trusty contrib
```

You also need to add and register the Oracle public key (combined command):

```bash
$ wget -q https://www.virtualbox.org/download/oracle_vbox.asc -O - | sudo apt-key add -
```

Now you can add Virtualbox:

```bash
$ sudo apt-get install virtualbox-4.3
```
For Vagrant there is still an issue and we need to get the latest version (not in Ubuntu repository) by downloading the package from the Vagrant website [http://downloads.vagrantup.com/](http://downloads.vagrantup.com/) and then issuing the following command (from the downloads folder, adapt to your version):

:::$\text{dpkg\ -i\ vagrant\_1.7.4\_x86\_64\_deb}$

Reboot the system and create a directory for your Lab projects, for example:

:::$\text{mkdir\ ~/Desktop/MyProjects}$

## 2 First Experiment

The first experiment, valid for all the systems (Windows, Mac and Linux) will consist on the launch of an Emulated network using Mininet [http://mininet.org](http://mininet.org). Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine (in this case, a Virtual Machine).

For that purpose it is necessary to provision through Vagrant a Mininet system, using the following commands, issued on the system Console/Terminal. The first command will position the creation of the system in the adequate working folder under MyProjects, for example in Windows `C:\Users\Admin\Desktop\MyProjects\mininet`, in Mac or Linux `~/Desktop/MyProjects/mininet`:

:::$\text{cd\ mininet}$

:::$\text{vagrant\ init\ ktr/mininet}$

The above command will fetch an already prepared Mininet system in a Ubuntu 14.04.2. When successfully created the system needs to be started up with the command:

:::$\text{vagrant\ up\ --provider\ virtualbox}$

This phase may take up a few minutes, depending on the speed of the host system, with an output similar to the following:

```
Bringing machine 'default' up with 'virtualbox' provider...
  ==> default: Checking if box 'ktr/mininet' is up to date...
  ==> default: Clearing any previously set forwarded ports...
  ==> default: Clearing any previously set network interfaces...
  ==> default: Preparing network interfaces based on configuration
      ...  
  ==> default: Adapter 1: nat
  ==> default: Forwarding ports...
  default: 22 => 2222 (adapter 1)
  ==> default: Booting VM...
```
== > default: Waiting for machine to boot. This may take a few minutes...
  default: SSH address: 127.0.0.1:2222
  default: SSH username: vagrant
  default: SSH auth method: private key
  default: Warning: Connection timeout. Retrying...
== > default: Machine booted and ready!
== > default: Checking for guest additions in VM...
  default: Guest Additions Version: 4.3.10
  default: VirtualBox Version: 5.0
== > default: Mounting shared folders...

When ready we will then establish a session with the system using the following command:

```bash
vagrant@mininet :~$ vagrant ssh
```

The session is established and we will get the machine prompt:

```
Welcome to Ubuntu 14.04.2 LTS (GNU/Linux 3.13.0-54-generic x86_64)
Last login: Sun Oct 18 22:50:28 2015 from 10.0.2.2

vagrant@mininet :~$
```

A very simple test can then be performed to verify that all is working correctly by issuing the command ‘sudo mn’ to create a simple network topology of two hosts (h1, h2) and a switch (s1):

```
vagrant@mininet :~$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
  (h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```
From this point we can issue the command 'pingall' to verify if we have connectivity between the hosts:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

With the command 'net' we can verify how the nodes are connected, i.e., host1 is connected by its eth0 port to the switch1 on eth1 port, etc.:

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
```

Issuing the command 'dump' we can see the IP addresses of the hosts (10.0.0.1 and 10.0.0.2):

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=1708>
<Host h2: h2-eth0:10.0.0.2 pid=1712>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=1717>
<Controller c0: 127.0.0.1:6633 pid=1701>
```

To end the Mininet emulation we use the command 'exit' and it is also advisable to 'clean' any phantom Mininet processes in the system with the command 'sudo mn':

```
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 2 links..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 5030.655 seconds
vagrant@mininet:~$ sudo mn -c
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noses
*** Cleanup complete.
vagrant@mininet:~$
```
We can now exit the session:

```
vagrant@mininet:~$ exit
logout
Connection to 127.0.0.1 closed.
```

In order to stop the Virtual Machine and to verify the global state of all active Vagrant environments on the system, we can issue the following commands:

```
:~$ vagrant halt
==> default: Attempting graceful shutdown of VM...
:~$ vagrant global-status
```

Confirm that the statuses of the VMs is 'powered off'.