imalse Documentation

Release 0.0 alpha

Jing Conan Wang

August 19, 2012

CONTENTS

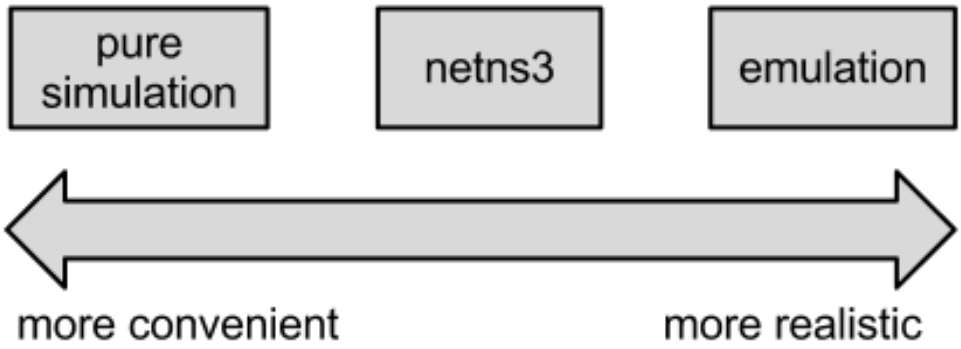
CHAPTER

ONE

INTRODUCTION

IMALSE (**I**ntegrated **MAL**ware **S**imulator and **E**mulator) is a framework to help researchers to implement prototype of botnet based network malware. Researchers just need to implement the malware behaviour once and then it can run the following three modes:

- **emulation mode**: In this mode, each copy of imalse will behave exactly like a real malware. You can install it in a real machine, or in a virtual machine and set up a testbed to test the characteristic of the malware.(Don't use it to attack other people's machines;)) [**Note**: you can potentially work with Common Open Research Emulator to emulate a lot of nodes in one machine]
- **netns3 simulation mode**: You can specifiy the topology of the network and the ip addresses of each node in this mode. IMALSE will launch virtual machines (linux namespace) for each node in the network and construct the network automatically. All virtualized nodes will connect to **NS3** through tapbridge and all traffic will consume there. The simulation will be in real time. It is based on netns3 project.
- **pure ns3 simulation mode**: No virtual machince will be launched for the pure ns3 simulation mode, the whole simulation will be done in ns3. ns3 default scheduler will be used instead of the real time scheduler in netns3 case, which saves much time. One simulation day may only consume several real seconds.



1.1 Typical Use Case

Suppose Conan is a Ph.D student who has proposed a novel anomaly detection technique for Internet traffic. He wants to demostrate the usefulness of this approach. To do this, he designs a scenario that 100 client computers accessing a server through the internet, 10 of which had already been compromised and controlled by botmaster through botnet. At some point, the botmaster will initiate a ddos attack by asking all compromised computers to send ping requests to the servers. The anomaly detection technique requires all the incoming and outcoming traffic of the server for at least two days.

How can he collect the data he want? imalse provides different solutions at different abstract level. He decides to use **TopoSimExperiment** in which he can load some topology file generated by Inet topology generator and select **ddos_ping_attack** attacking scenario from the imalse software which provide exactly what he wants.

The first question is since the method is not mature, Conan wants to test it under different parameter combinations. It will be forever if each simulation takes more than two days. Fortunately, by running the simulation under **pure ns3 simulation mode** Conan can finish one simulation with less 100 real seconds, though the time has past for more than two days in the simulator.

After extensive testing, Conan has been quite confident about the performance of the anomaly detection techinique now. But he is still a little bit worried about whether the result of ns3 is convincing enough. As a result, he run a complete simulation under **netns3 simulation model** and collect data. Of course, this time it runs more than two days, but he doesn't care that much because he only need to run it for very few times. Conan generates some plots and writes a paper with data of **netns3 simulation model** and satisfied with this.

A rich company named NetSecurity reads this paper and think it is a good method. They want to deploy it but need more realistic test before deployment, so they decide to test it under their intranet. They ask Conan for a copy of the code and select several computer in the intranet to join the botnet, each computer run an independent copy of imalse under **emulation client mode**, there is a computer serving as botmster and running a imalse under **emulation server model**(the server refers to the C&C server in the botnet). The data of attacked server is recorded and analyzed with Conan's tools. It turns out to be good, and the Company decide to use this method.

As a lazy Ph.D student, Conan just need to write one copy of code to describe the secnario during the whole process. With the help of imalse, he can have more time to sleep and enjoy the classical music. :)

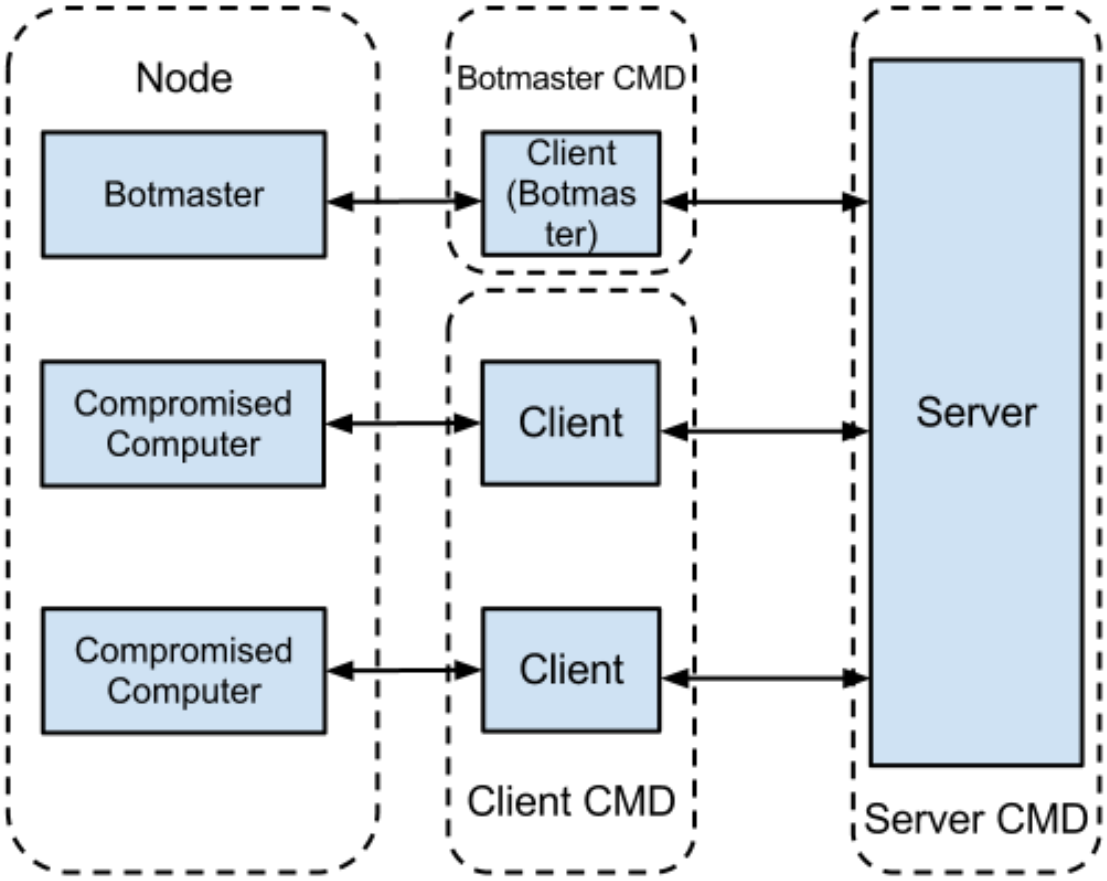
CHAPTER

TWO

DESCRIPTION:

To support the variety of modes noted above, Imalse design in a way that to seperate the botnet mechanism and the network.

The general structure of Imalse is shown in the following figure:



Node and **C**ommand **M**eta **D**escription are the two key concepts in the design. **Node** is the abstraction of a real computer. A node should support:

- **Basic Utility Functions**: serveral basic utility calls, including getting the system time, make node sleep, so on so forth

- **Socket API** this is require to implement basic
- **File System**
- **Higher Level Application**

There is a abstract base class in core module named **BaseNode** that define the all APIs a node need to overload. If you want to extend the framework to support other type of simulator, please subclass **BaseNode** and implement all the virtual functions.

a command is a basic event. **C**ommand **M**eta **D**escription defines a set of commands for a node, namely it defines what event a node can generate. There are three types of **CMD**s:

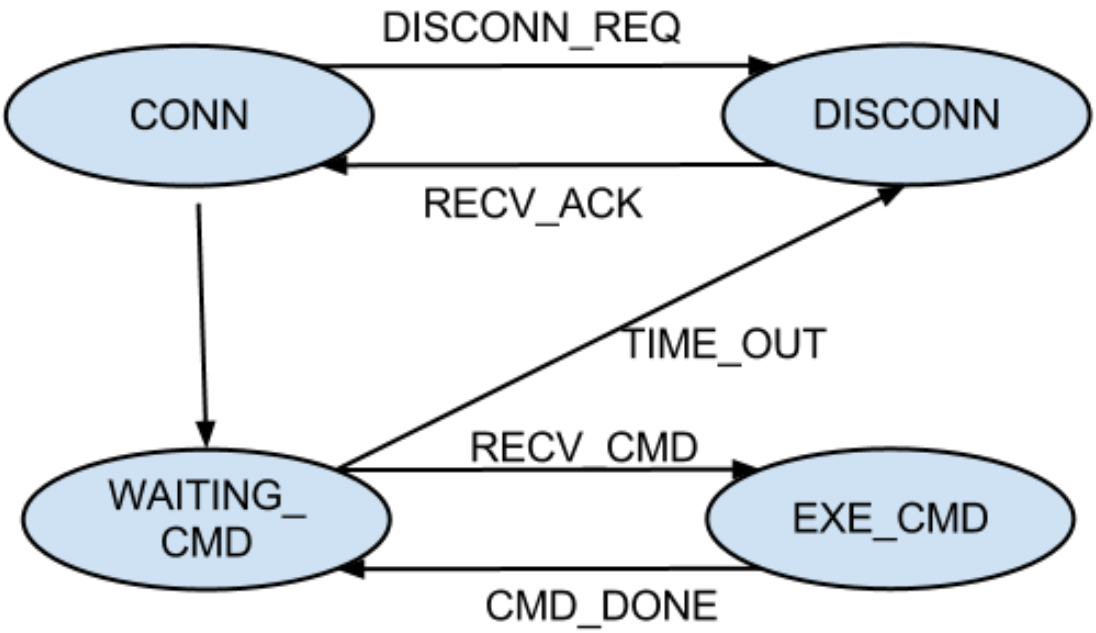
1. Server **CMD**: Command meta description for the server. Server usually will waiting attack command from botmaster and send corresponding commands to clients.
2. Client **CMD**: Client will wait command from server and do correspondingactions in user's comoputer. What a Client can do depends on the Node **API**
3. Botmaster **CMD**: Botmaster is actually a special client with higher privilege. Botmaster should verify itself by sending verify command and send attack commands to server.

The basic procedure is that the botmaster send commands to the server, the server translates the commands and send commands to Clients.

2.1 Basic Botnet Mechanism:

In **core** module, imalse provides a basic framework for the botnet and the support to real network, netns3 and NS3. User can create their own flavor of botnet by subclassing **ServerCMD**, **ClientCMD** and **BotmasterCMD**, each flavor is called a new **scenario**.

The basic botnet mechanism can be described as **F**inite **S**tate **M**achine. The **FSM** for the **ClientCMD** is as follows:



2.2 Scenario:

As a noted above, user can create their own flavor of **Botnet**, which is so called **scenario**. Currently, Imalse provides two sample **scenarios**:

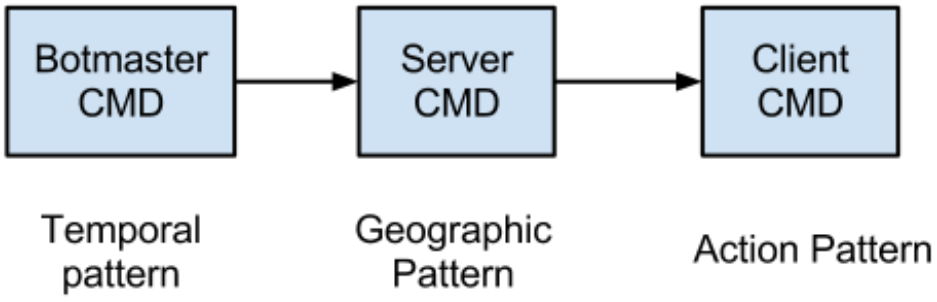
1. **ddos_ping_flooding: in this scenario, botmaster can issue send_pings** command to initiate a ddos ping flooding attack to a specific server.
2. **file_exfiltration: in this scenario, botmaster can request bots to search** in the file system with any file that contains a certain pattern, like **password**. Whenver an interesting is found, the bot will upload the file to a ftp server.

To implement a new scenario, you need to create a new folder in scenario folder. This folder actually is a new module in python. This module should provide the following classes: **BotMaster**, **ClientCMD**, **ServerCMD**.

You can subclass the BotMaster ClientCMD and ServerCMD in core, which provide a botnet communication scheme and make the difference of real node, netns3 node and ns3 sim node to be transparent.

An recommendation of implementation

1. put temporal pattern of attack into Botmaster **CMD**. In other words, in botmaster **CMD**, you can specify which attack should happen at what time. Botmaster can issue a **file_scan** attack command every 24 hours, or according to a possion distribution, etc.
2. put geographical pattern of attack into Server **CMD**. In other words, which clients should join this attack.
3. put action pattern of attack into Client **CMD**. A Client command will define a sequence of node actions, which is unique to this attack.



2.3 Experiment

Experiment is only used in **simulation mode**. Experiments need to do Topology and network configuration, user behaviour specification. This is usually the only part user need to code if he is using an existing scenario.

In ROOT/experiments folder, we provide three samples of experiment.

- **StaticRouteExperiment** is a experiment in which a small diamond shape topology in demoed and the routing tables for all nodes are manually specificed.
- **TopoExperiment** is a experiment in which you can load topology generated by some topology generators, like Inet , the ip address of each node will be automatically assigned. The parameters you can customize are: 1. *NETWORK_BASE*, 2. *SERVER_ADDR*, 3. *IP_MASK*, 4. *server_id_set*, 5. *botmaster_id_set*, 6. *client_id_set*, 7. *Delay*, 8. *DataRate*. The delay and data rate for all links are the same. All parameters are the static member

of **TopoExperiment** class and you just need to change them in the code accordingly. You will need this if you only care about topology and too lazy to configure the network.

- **ManualTopoExperiment** is simular to **TopoExperiment**, however, in **ManualTopoExperiment**, you can specify the delay and data rate of each link and also the ip address of every interface. Instead of hard code all parameters in the code, it will load a configuration script with python syntax. We provide a GUI editor to create such configuration script.

When you want to implement you own version of experiment, you can put it in ROOT/experiments folder, there is class factory function in the ROOT/experiments/API.py, when you input the different **–mode** parameter in the command line, the class factory function will generate a new experiment class with all the methods and members you definde with corresponding base class.

Some things you need to pay attention are:

1. **There is only one experiment for each file. The name of experiment file should** have the same name with the experiment class.
2. The experiment class should declare **BaseClass** as the base class

For example, we define the **TopoExperiment** in *ROOT/experiments/TopoExperiment.py*, the definition of topology experiment is as follows:

```
class TopoExperiment(BaseClass):
```

It declares BaseClass as the base class, though **BaseClass** is not defined or imported in this file. It probably won't pass the syntax checker like *pyflake* and *pylint*. Actually **BaseClass** is a placeholder that will later be replaced by real implementation. To clarify this 'strange' code , I will introduce the implementation details a little bit. If you don't want to know the reason, just ignore this part.

```
def experiment_factory(experiment, mode):
    BaseClass = mode_map[mode]
    exper_fname = settings.ROOT + '/experiments/' + experiment + '.py'
    execfile(exper_fname, locals())
    return locals()[experiment]
```

The code above is the code of experiment_factory. It will dynamically select the base class according to following map:

```
mode_map = {
        'netns3':ImalseNetnsExperiment,
        'sim':ImalsePureSimExperiment,
        }
```

and then execuate the experiment file according to the name of the experiment it will load, which is the reason why the name of the file and the experiment class must be consistant.

CHAPTER

THREE

GUI SUPPORT

One important principle of Imalse is "not reinventing the wheels". So instead of implementing a GUI by myself, I take advantage of existing GUI tools, particually NS3 PyViz and CORE network topology Editor (which is again based on IMUNES network topology editor)

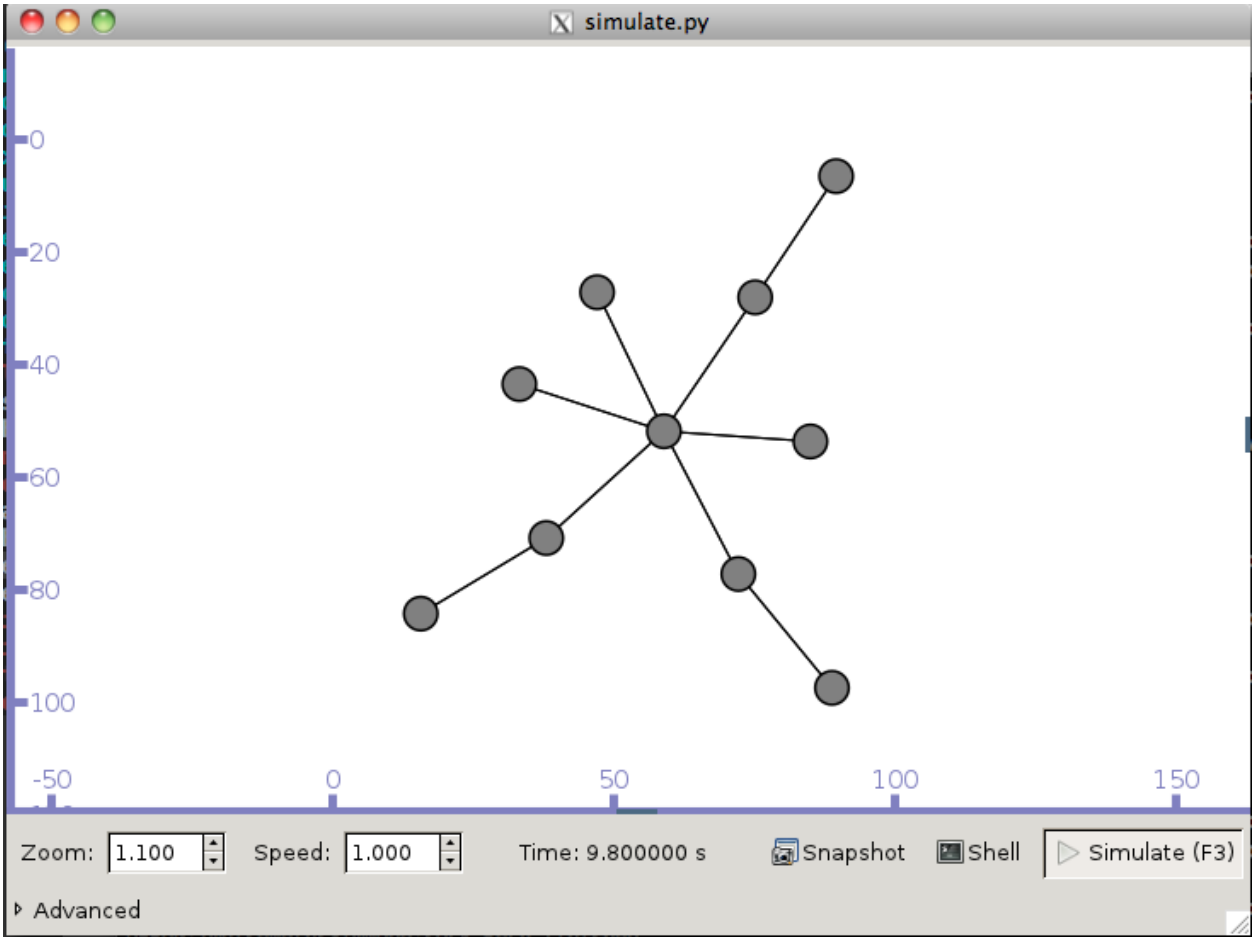
The GUI support of Imalse are two folds:

1. It provides a simple visualizer that can visualize the topology and traffic dynamically. This is based on PyViz.
2. It provides a complete Network Topology Editor in which you can "draw" the network topology, config the network and export the configuration scripts for command line to use. This part is based on CORE Network Topology Editor

3.1 Simple Visualizer

The first function is suitable for those who are accustomed to work under command lines but just need a very simple visualization to check everythings goes right. You have total control using this, you can implement new command line settings, and editing network topology and ip address.

Currenly only NS3 pure simulation can be visualized using PyViz. To visualize the simulation, you just need run the simulation in *sim* mode and add the *–SimulatorImpl=Visual* in the command. The screenshot of simple visualizer is as follows:

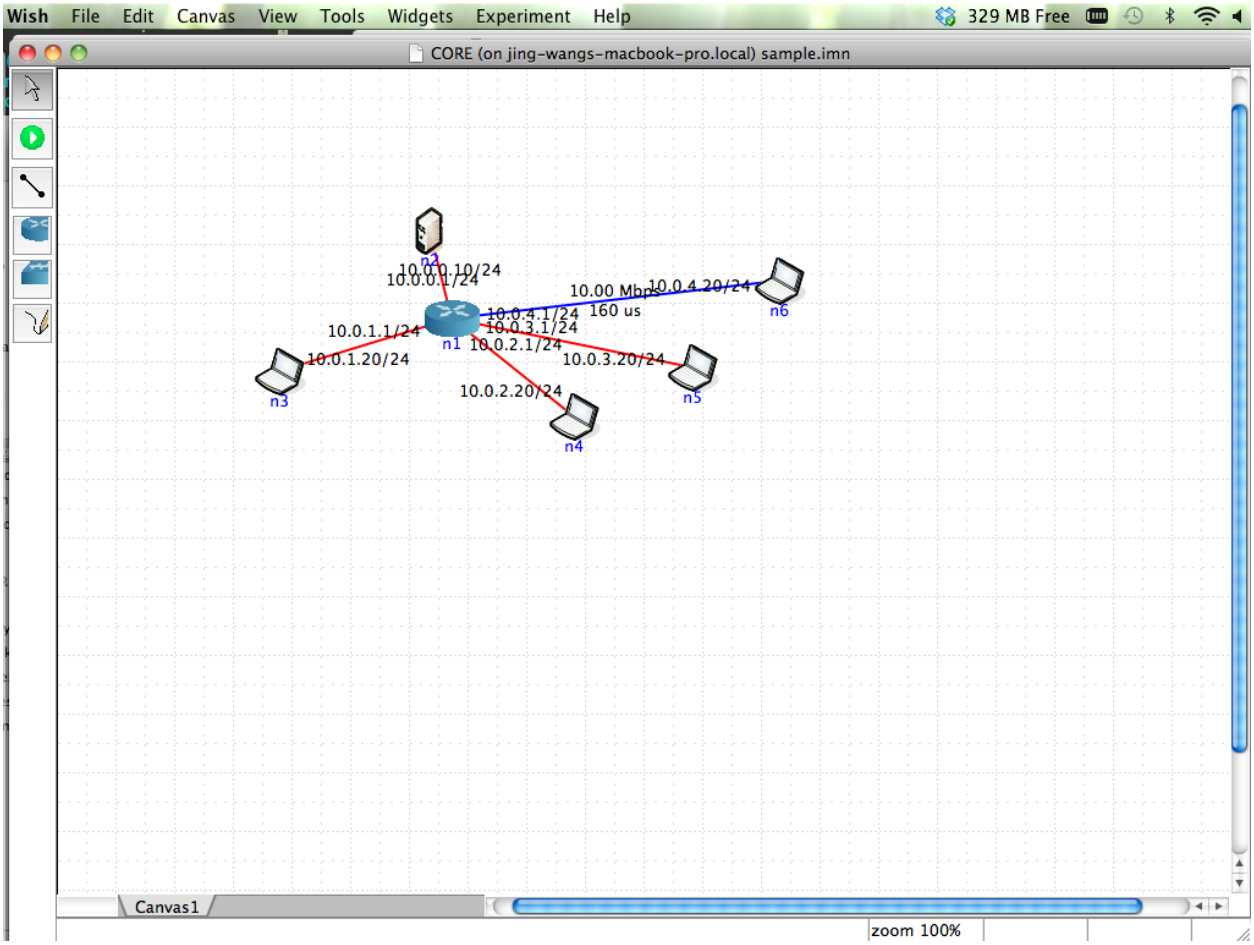


3.2 Network Topology Editor

One of the most cubersome part of configuration is to 1. create a suitable network topology. 2. Configure the ip address. 3 and specify the role of each node. You can definitely do this using commandl line and actually I have tried my best to make the process easier. For example, in TopologyExperiment, the IP address of each subnetwork can be automatically assigned. The inet topology generator tool has been integrated to generate reasonable topology.

However, in many cases, you would prefer to configure by youself instead of using automated tools. In this case, you will have a lot of tedious typing work if you use command line tools.

To make the proces easier, the Imalse provide a GUI of topology editor. This GUI editor is revised from GUI editor of CORE . A screenshot is as follows:



please refer to the CORE GUI Manual Page and IMUNES Manual for basic usuage of the editor. You can look at the GUI Demo for the difference.

CHAPTER

FOUR

DEMO:

Here are some demo videos

- Command Line Usage
- GUI usage

CHAPTER

FIVE

SOURCE CODE:

the project is currently hosted on www.bitbucket.org. To get a copy of the code, please install mercurial first and type

```
hg clone https://hbhzwj@bitbucket.org/hbhzwj/imalse
```

in the command line.

CHAPTER

SIX

INSTALLATION:

6.1 Installation of NS3

We recommend you to install a revised version NS3 based on NS3.14.1. We add the imalse module to this to deal with the packet manipulation. Run the following command in the bash.

```
wget https://bitbucket.org/hbhzwj/imalse/downloads/ns-allinone-3.14.1-with-imalse.tar.gz
tar -xzvf ns-allinone-3.14.1-with-imalse.tar.gz
cd ns-allinone-3.14.1-with-imalse
./build.py
```

It will check the dependencies first. be careful about the message of and install the corresponding dependencies. Under Ubuntu 12.04, you can install the dependencies by typing

```
sudo apt-get install g++ python-dev gccxml python-pygccxml python-pygraphviz python-pygoocanvas
```

After building the ns-allinone successfully. There is one more thing you need to do. The ns3.14.1 has a bug in python binding of dsr, the most recently added module. You need disable the import of dsr binding in ns3.py.

```
cd ns-allinone-3.14.1-with-imalse/ns-3.14.1/build/bindings/python/
vi ns3.py
```

then comment the

```
from ns.dsr import *
```

line.

6.2 Installation of Common Open Research Emulator

We use netns3 to vituralize the node in which requires common open research emulator. Since netns3 has been integrated into imalse, you just need install CORE

Refer to the following http://pf.itd.nrl.navy.mil/core/core-html/Installing-from-Packages-on-Ubuntu.html for installation of common open research emulator.

6.3 Download imalse

Then download the tarbar for the imalse

```
wget -O imalse.tar.bz2 https://bitbucket.org/hbhzwj/imalse/get/94d1ff15736f.tar.bz2
tar -xvf imalse.tar.bz2
```

or you can use hg clone command in the previous section to get the lastest version. The last thing you need to to is to change the ROOT and NS3_PATH in settings.py. ROOT should be the directory of the imalse source code and NS#_PATH should be the directory for the NS3.

CHAPTER

SEVEN

ACKNOWLEDGEMENT

First, I need to thank Hugo González, my advisor in the *Google Summer of Code 2012* for his guidance and support in this memorable summer. He made the general direction and provided tons of usefule suggestions during the developing process.

Second, I need to thank my advisor Yannis Paschalidis for his continuous support since I joined the lab. As a experienced researcher and renowned scholar, he has taught me so much in the past two years.

Third, I need to thank Google for sponsoring this program. Google is a really awesome company. I wish I can have more opportunites to work with them in the future.

Last, I need to thank my girlfriend for her understanding and support.

CHAPTER

EIGHT

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*