Internship in the Universidad Politécnica de Valencia : Validation and improvement of a MANET emulation tool.

Wannes VOSSEN (wanvos@posgrado.upv.es)

June 22, 2010

English : The present document is a detailed report of a five months internship in the computer networks research group[1] of the *UPV*[2] in Valencia, Spain. The described work placement is part of the first year of a Master degree in computer networks and telecommunications (Master STRI[3] Toulouse, France). In this report, I will focus on two aspects: (I) relate the experience of such an international internship and (II) present the results and conclusions of the achieved work. Readers might be interested in only one of those two aspects and should consider the table of contents to sort relevant information.

Español : Este documento es la memoria de una practica de cinco meses en el grupo de investigación de redes de la *UPV*, Valencia. Esta practica es parte de un primero ano de Master en redes y telecomunicaciones (Master STRI, Toulouse, Francia). En la memoria, describiré dos aspectos en particular: (I) Intentaré transcribir la experiencia de tal instancia al extranjero y (II) presentaré los resultados y conclusiones de mi trabajo. El lector pueda tener interés en uno solo de estos dos aspectos y puede referirse al index para encontrar su contenido.

Français : Ce document est un rapport détaillant cinq mois de stage dans le groupe de recherche sur les réseaux informatiques à l'*UPV*, Valencia, Espagne. Le dit stage a été effectué dans le cadre d'une première année de master en réseaux et télécommunications (Master STRI à Toulouse). Ce rapport ce concentrera principalement sur deux aspects: (I) il s'attachera à retranscrire l'expérience d'un stage à l'étranger et (II) il présentera le travail accompli et les résultats obtenus durant le stage. Le lecteur pourrait n'être intéressé que par un seul de ces deux aspects et devrait se référer à la table des matières pour trouver l'information souhaitée.

[1]Grupo de Redes de Computadores. See `http://www.grc.upv.es`
[2]Universidad Politécnica de Valencia. see `http://www.upv.es`
[3]Systèmes de Télécommunicatons et Réseaux Informatiques. See `http://www.stri.net`

Contents

List of Figures

Part I.

Personal Experience

1. Background and motivations

The *ERASMUS* program allows two European universities to settle an agreement and to exchange their students for one or two academical semester(s). Achieving such an exchange is probably a unique experience in one's life. It gives the opportunity to discover a new culture. The student is dropped in a totally new environment, with few landmarks, forced to review his bounds and to open himself to new habits and a different educational system. It is an opportunity that, in my point of view, can't be ignored.

For the past 3 years, I have been studying a computer science degree in Toulouse. The two first years of one's academical career are - for obvious reasons - not appropriate for an exchange. During my third year, I reoriented my studies from pure computer science to networks and telecommunications. As I then entered a new school, I probably was in a too tricky situation to consider the overload of an exchange. Never the less, I undertook the preparation of such a project. Achieving exchange is not that common in my home school. Consequently, there was no specific exchange agreement settled with the university of Valencia or even with any other fully Spanish speaking university[1]. Setting up such an agreement from scratch was quite an heavy task and requested several months of administrative procedures.

Finally, the reason why I choose Spain for my *ERASMUS* among other countries is that I wanted to learn a second foreign language, which I believe is interesting for my professional career.

[1]There is a partnership settled with the university of Barcelona but I was told that Catalan was the main language there.

2. The Host University

2.1. Universidad Politécnica de Valencia (UPV)

UNIVERSIDAD POLITECNICA DE VALENCIA

The *UPV* is a Spanish public university oriented to sciences and technology. It is spread over three campuses in the region of Valencia, Spain. It also regroups 37000 students, 2800 teachers and 2200 professionals to offer a large variety of services.

During the first semester of the academical year, I experienced the *UPV* as a student and my global impression was quite positive. In particular, I appreciated how students were considered as the center of any concern: classes did generally regroup a maximum of 20-30 persons and teachers organized tutor sessions in their office twice a week. The whole forward planning of the semester was very often announced from the very beginning, which allowed efficient personal organization. The university also provides numerous services like a very efficient and unified intranet, many sport facilities, conferences, libraries, places to study and so on.

2.2. Grupo de redes de computadores

The network research group *GRC* was founded in 2000 and it regroups researchers from the *Computer Engineering Department*. Currently, the group's efforts are focused on Mobile Ad-Hoc Networking (*MANET*s) and more precisely on the management of such networks. For example, it is concerned about routing, security and quality of service in *MANET*s.

Figure 2.1 shows the organization of the *GRC* research group. The faculty regroups several professors, mainly working at the *UPV* but also coming from other Spanish universities. Professors have several tasks like research, education and administration. For example, professors are in charge of *PhD* students, who are working on their thesis and participating the research work. Professors can also appeal to Master students as a support for their research: It is a possibility, for Spanish Master students, to work in their university's laboratories and receive specific grants in return. Master students can also be foreign students who have to perform an internship during their exchange. This is my situation.

For my every days work in the laboratory, I collaborated with *Jorge Hortelano* and *Alvaro Torres*. *Jorge* is a *PhD* student and he is also the creator[1] of the software I worked on. He introduced me to its functioning and answered many of my questions. *Alvaro* is a Master student, he also worked on the program I had to validate. We thus had to work together as one's changes could interfere with another's one. As he had been working on the software for a longer time, he also answered many of my questions. *Alvaro* and I also collaborated to write the research article described in chapter 6.5 on page 49.

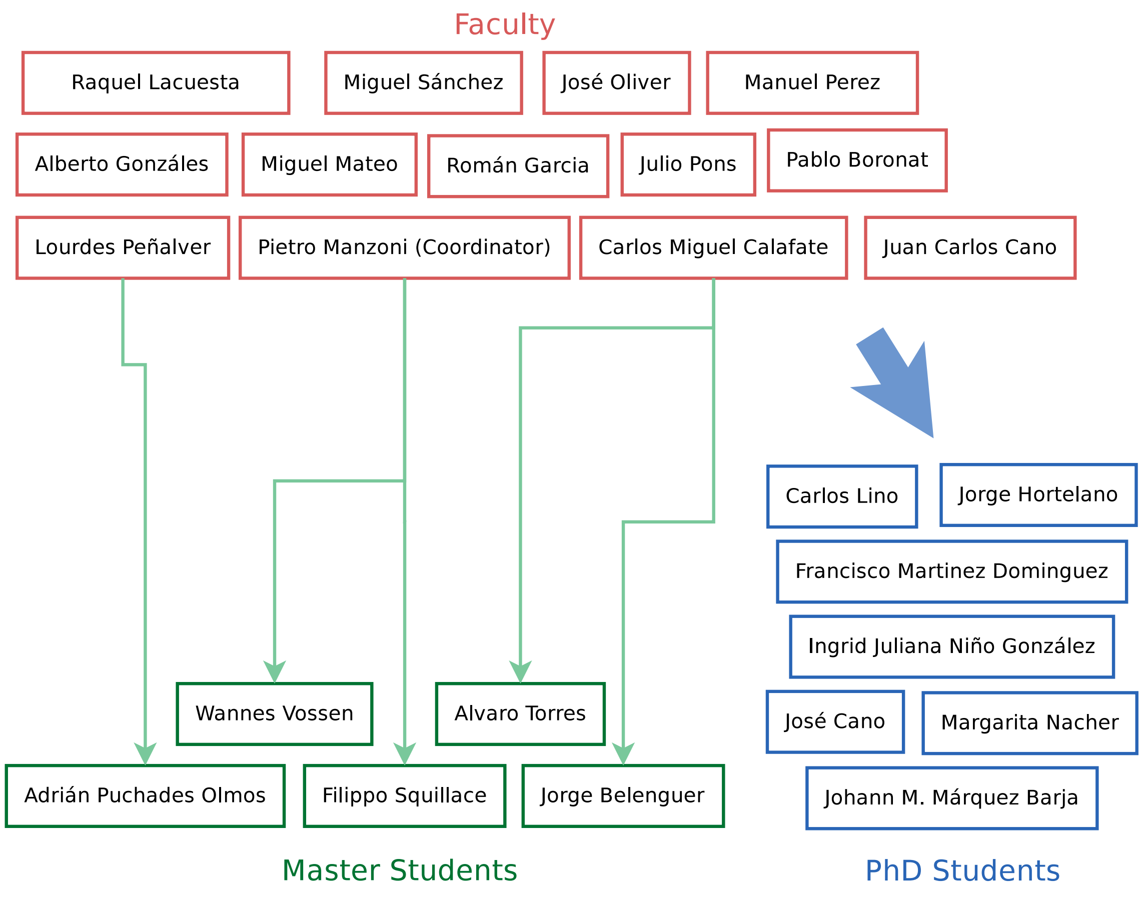


Figure 2.1.: Organization chart of the GRC research group.

2.3. An internship opportunity

On exchange, one should stay as close as possible to the material he would have studied in his home university. My classmates in France had to perform a four month internship in some company. On the occasion of an exchange, it is quite common for a student to work in a university lab. When I created the exchange agreement, I asked my Spanish exchange coordinator (*Marta Caballero*) for such a placement and she put me in touch with *Pietro Manzoni*, coordinator of the *Computer Networks Group* and also dealing with exchange in the computer science department. A few emails later, we had an overview of what I could possibly work on.

2.4. Technical aspects

On my arrival, I was assigned to a desktop in the network research lab. The material and the environment were modern and quite comfortable to work with. We were more or less 6 Master and PhD students working int the laboratory and the atmosphere was pleasing. One could easily ask another for advice and everybody was always pleased to help.

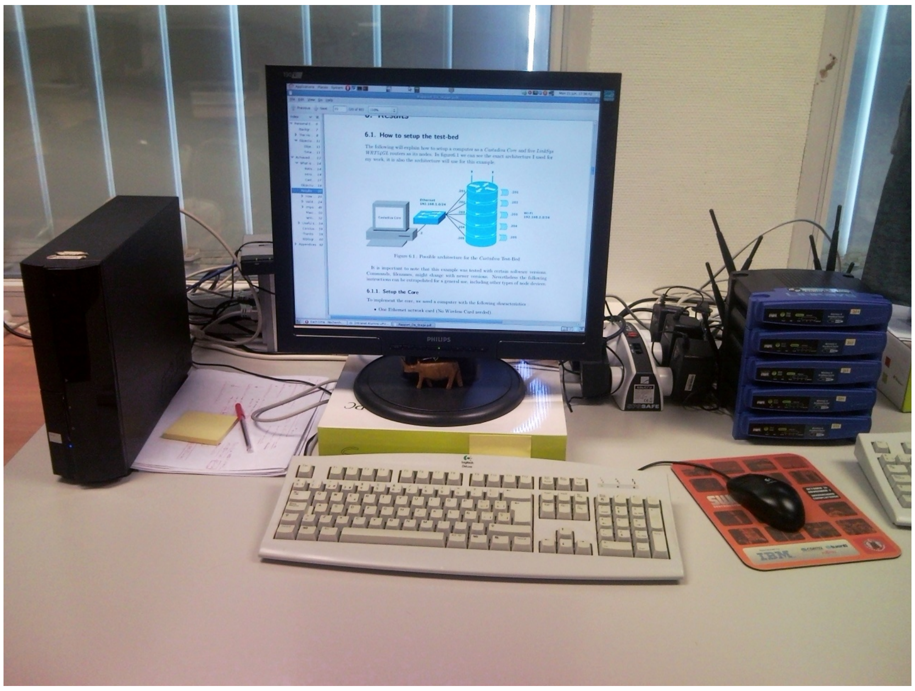


Figure 2.2.: Working material with the stacked routers.

3. Objectives & time management

3.1. Objectives

Figure 3.1 shows how I spread my different tasks over the whole internship. Tasks will be explained with more detail in Part II. For now, I will focus on time management and objectives.

On my first day, I met *Pietro Manzon*i, my supervisor. He introduced me to the team and especially to *Jorge*, who guided my first tasks: I had to setup my work environment and get used to the software that I was suposed to validate. Two weeks later, on February the 10th, I met *Pietro Manzoni* angain and we set my main objectives for the internship. My work was theoretically quite simple: validate a few new functionalities of a wireless simulation test-bed. Nevertheless, we did not know exactly how far their developpement had been taken.

As shown in Figure 3.1, those objectives finaly spread over the whole internship, with a few additionnal works like the participation to a resarch article. This is due to the fact that the functions I had to validate were left in a very early developpement stage, with no documentation and none of those funcionalities acutally functioned out of the box. My work finally consisted in a circle of investigating, fixing errors, trying to validate and investigating again for the new errors.

Each time I focused on a new part of the software, I had to analyse and understand it. During my first days, the program apeared to me as something huge I would never understand. Slightly, my knowledge of the program increased and I started assimilating pieces of the puzzle. After 5 month, I felt a very important progression of knowledge and effectiveness. Through the present report, I would like to transmit that experience.

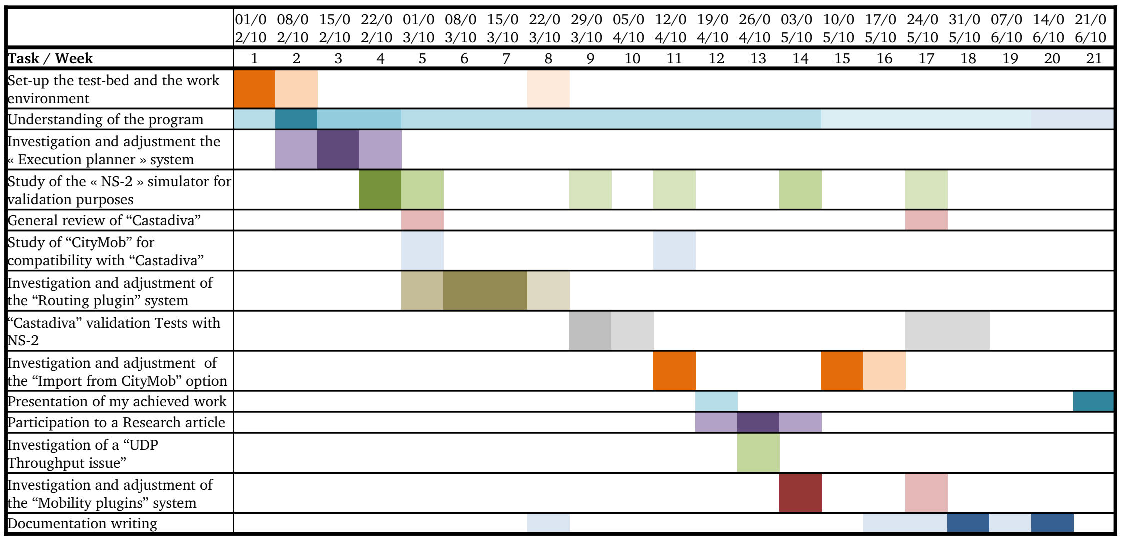
3.2. Time management

Something important regarding time management is that I have attended classes during the entire internship. I attended 9 hours of class every week, homework and projects represented at least another 9 hours of personal work.

This taken in account, the instructions I received from my home university were to balance, as good as possible, among those two activities. *Pietro Manzoni* also warned me that he would not be watching at my timetable.

I thus had to discipline myself, trying to be as efficient as possible in my university work in order to be able to spend as much time as possible in the laboratory. I also spent fewer hours in the laboratory during the exam period, which was the week before the 11th and 15th of June.

Every month, on Wednesday, the whole group meets and discusses the student's works. Unfortunately, I had a class on Wednesday, overlapping with the meeting. I gave up one of those classes to participate one of those meetings, I think it is an important experience to understand how the laboratory works. When I presented my achieved work, I was pleased to hear critics and interesting suggestions to deal with my problems.



	01/02/10	08/02/10	15/02/10	22/02/10	01/03/10	08/03/10	15/03/10	22/03/10	29/03/10	05/04/10	12/04/10	19/04/10	26/04/10	03/05/10	10/05/10	17/05/10	24/05/10	31/05/10	07/06/10	14/06/10	21/06/10
Task / Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Set-up the test-bed and the work environment	🟧	🟧						🟧													
Understanding of the program	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Investigation and adjustment the « Execution planner » system		🟪	🟪	🟪																	
Study of the « NS-2 » simulator for validation purposes				🟩	🟩				🟩		🟩			🟩			🟩				
General review of "Castadiva"					🟥												🟥				
Study of "CityMob" for compatibility with "Castadiva"					🟦						🟦										
Investigation and adjustment of the "Routing plugin" system					🟫	🟫	🟫	🟫													
"Castadiva" validation Tests with NS-2									⬜	⬜							⬜	⬜			
Investigation and adjustment of the "Import from CityMob" option											🟧				🟧	🟧					
Presentation of my achieved work												🟦									🟦
Participation to a Research article												🟪	🟪	🟪							
Investigation of a "UDP Throughput issue"													🟩								
Investigation and adjustment of the "Mobility plugins" system														🟥			🟥				
Documentation writing								🟦								🟦	🟦	🟦	🟦	🟦	

Figure 3.1.: Task distribution over time

Part II.

Achieved work

4. What is Castadiva ?

It is mandatory, for further reading, to understand what *Castadiva* is. In Section 4.2 , I will make a short description[1] of it. Please note that a more extended description of *Castadiva* is also available, it can be read in the *Castadiva Journal*[1].

4.1. MANETs

Mobile ad-hoc networks (*MANETs*) are packet radio networks composed by independent and heterogeneous stations that cooperate in routing and packet forwarding tasks. All the nodes which are part of a *MANET* can act as routers, allowing communication among several out-of-range end-points. As a *MANET*'s nodes can also be moving around, the the network topology can suffer constant changes. In such situation, dynamic routing protocols are very interesting to automatically reconfigure routes.

Testing and evaluating *MANET* management protocols is important to guarantee their effectiveness in real situations. Researchers have three options to do so: to use simulation tools, to use emulators, or to use test-beds.

4.2. Introduction to Castadiva

Castadiva is a *MANET* emulator which provides a cost-effective alternative to simulation tools. This can be achieved by allowing certain critical components of a simulation to be real. For example, *Castadiva* relies on real wireless communications, using *IEEE 802.11* interfaces.

[1]The description is based on my work in a previous article[2]. See Appendix B

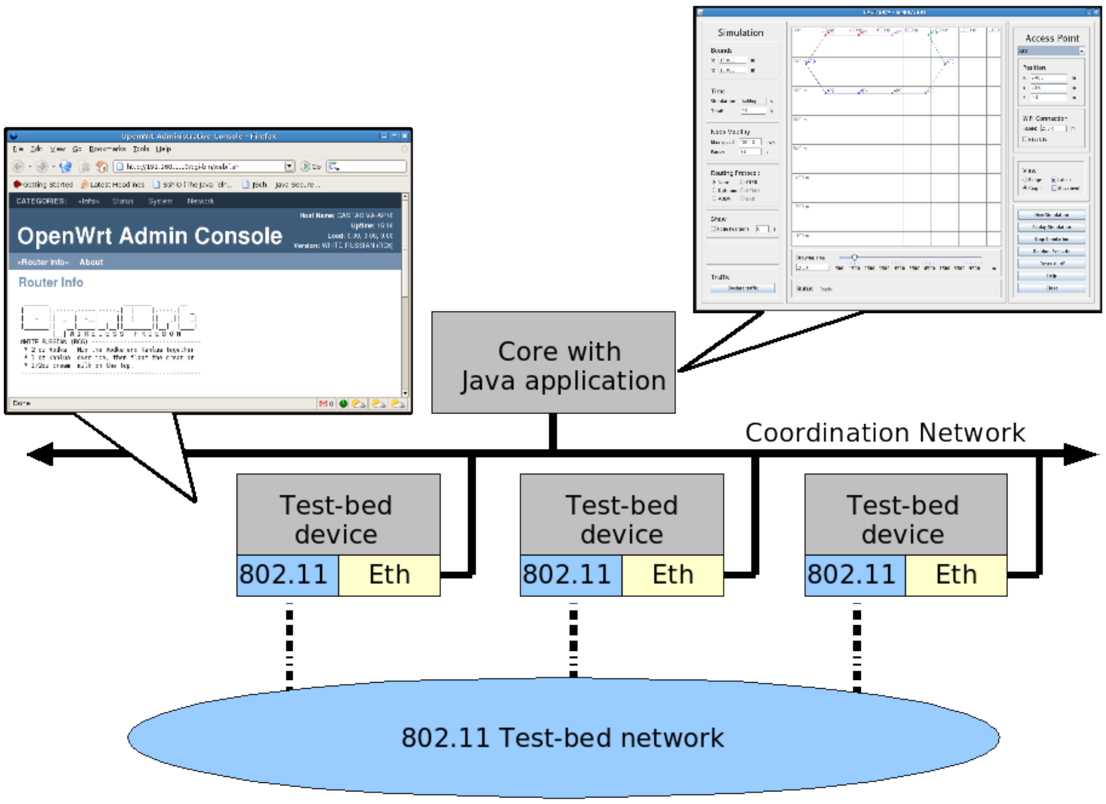


Figure 4.1.: *Castadiva*'s architecture

Castadiva's architecture is shown in Figure 4.1. It is based on two major elements: (a) The *Core*, which orchestrates the simulation and coordinates the *Nodes*; (b) The *Nodes* which are able to communicate among themselves, for simulation purposes, using their *IEEE 802.11* wireless interface. Communication among the *Core* and the *Nodes* is performed over a typical *Ethernet* network. *Ethernet* allows reduced delays and guarantees no interferences with wireless signals.

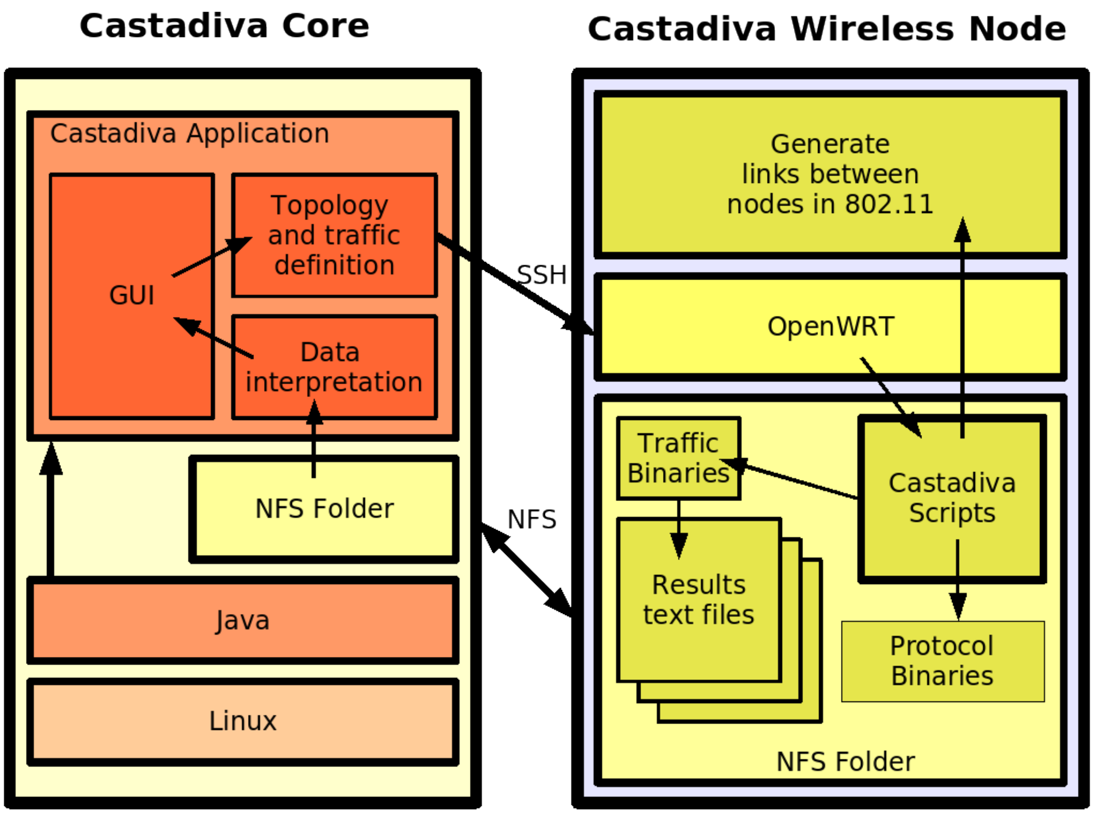


Figure 4.2.: Software components for *Castadiva*

As shown in figure 4.2, the *Castadiva* core is written with *JAVA* and can be run on a simple computer. The *Nodes*, on their part, must rely on a *Linux* operating system. For example, *Castadiva*'s development team used *Wi-Fi* routers[2] or even *Net-books* [3]. *Net-books* offer better hardware performances than routers and can be easily moved around.

Figure 4.2 offers a detailed overview over the protocols used in *Castadiva*. It represents the situation where the *Core* would be a *Linux PC* and where the nodes would be wireless routers using a *Linux* based operating system (*OpenWRT*). Since the *Node* devices may be very specialized (routers, *PDAs*, ...) and have limited storage resources, we use *NFS* to store and access any simulation content on the nodes. The *NFS* server must be located on the *Core*, allowing easy file sharing among the *Castadiva* software and every *Node*. Finally, *Secured SHell* (*SSH*) is a good option to control the simultaneous start of a simulation on every *Nodes*.

[2]LinkSys WRT54GL with the OpenWRT Linux based Operating system.
[3]Asus eee PC 901

4.3. Castadiva's features

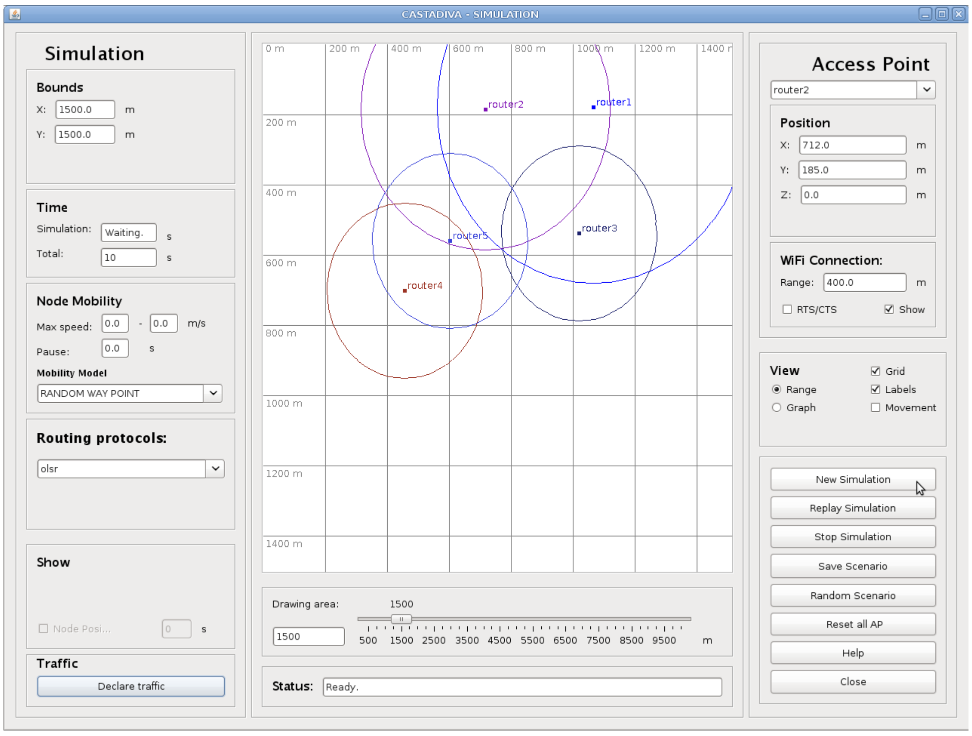
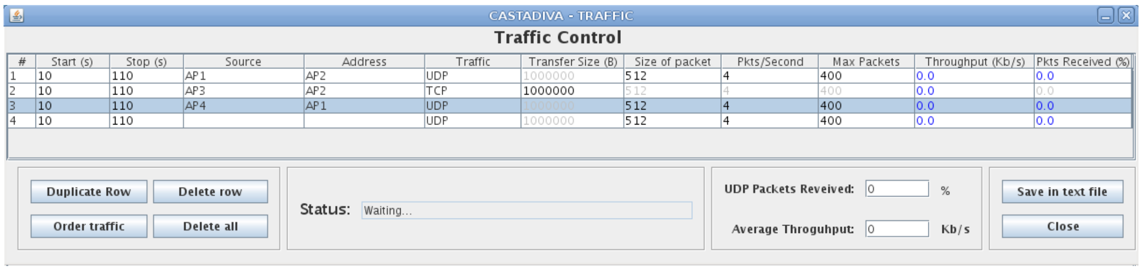


Figure 4.3.: *Castadiva's s*imulation window

Figure 4.3 shows the graphical interface that was designed to easily control the previously described system. As we can see, it is possible to place each *Node* on a canvas and to define its range. According to those ranges, *Castadiva* makes sure that two out-of-range *Nodes* do not communicate, even if their real interfaces are able to do so. Among the options available in figure 4.3, we can see that some are related to *Mobility* and *Routing Protocols*. Those features are part of my work on *Castadiva* and will be described in section 6.2 on page 25.


CASTADIVA - TRAFFIC

Traffic Control

#	Start (s)	Stop (s)	Source	Address	Traffic	Transfer Size (B)	Size of packet	Pkts/Second	Max Packets	Throughput (Kb/s)	Pkts Received (%)
1	10	110	AP1	AP2	UDP	1000000	512	4	400	0.0	0.0
2	10	110	AP3	AP2	TCP	1000000	512	4	400	0.0	0.0
3	10	110	AP4	AP1	UDP	1000000	512	4	400	0.0	0.0
4	10	110			UDP	1000000	512	4	400	0.0	0.0

Status: Waiting...

UDP Packets Reveived: 0 %

Average Throguhput: 0 Kb/s

Figure 4.4.: *Castadiva*'s traffic window

When a simulation ends, the traffic window is automatically displayed (figure 4.4). That window has two purposes: it can be used to configure the communications among nodes and, later on, to display simulation results.

For a more detailed overview of the *Castadiva User Interface*, which reflects its features, refer to Appendix A on page 59. Section 6.1 on page 21 can also help for comprehension, it explains *Castadiva* through a practical example. The *Castadiva Journal*[1] is also a good source for further information.

5. Objectives

My objectives for the internship were based on a previous work realized by *Ignacio Muñoz Vera* in 2009. Basically, I had to validate his work, making sure that the results generated with his new functionalities were correct and consistent. But as we did not know how far his project had been carried out, it was also possible that I would have to fix eventual errors. The main novelties I had to validate were the following:

- The *Execution Planner* allows to setup a list for automatic simulations.
- The *Routing Plugin System* should enable the user to automatically start and stop any routing plugin on his *Access-Points*.
- The *Mobility Plugin System* should allow to generate mobility scenarios.
- The *CityMob Importation* should allow to import a *CityMob* random mobility scenario.

Apart from those systems I had to validate, I also had to perform editorial work:

- Participate the writing of a research article dealing with *Castadiva*.
- Write a report and documentation dealing with the knowledge I gathered along the internship.

Of course, these are the main tasks and they oriented my work. They led me to achieve other subsequent tasks like:

- Setup of the test-bed.
- Understanding of the existing program.
- Learning how to use simulation tools in ordre to validate *Castadiva*'s results.

In the next chapter, each of those tasks is described with more detail.

6. Results

6.1. How to setup the test-bed

The following will explain how to setup a computer as a *Castadiva Core* and five *LinkSys WRT54GL* routers as its nodes. In figure 6.1 we can see the exact architecture I used for my work, it is also the architecture that I will use for this example. In figure 2.2 on page 11 we can also see a picture of the real test-bed.

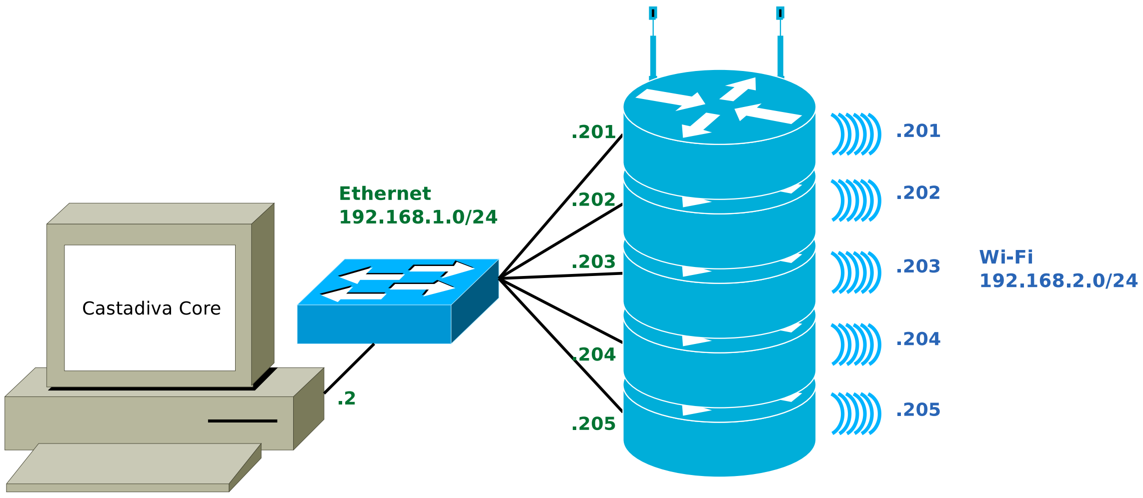


Figure 6.1.: Possible architecture for the *Castadiva* Test-Bed

It is important to note that this example was tested with certain software versions. Commands, filenames might change with newer versions. Nevertheless the following instructions can be extrapolated for a general use, including other types of node devices.

6.1.1. Setup the Core

To implement the core, we need a computer with the following characteristics :

- One Ethernet network card (No Wireless Card needed).
- Support of *Network File System* (*NFS*) as a Server.
- Support of *JAVA* to run the Core program.
- Support of the *javac* (*JAVA* Compiler) and *jar* (Creation of a *.jar* archive) commands to use *Castadiva*'s plug-in system.

In our example, we use a *PC* with a *Debian* distribution. Here is how to install the *NFS* Server and how to export a directory.

1. Install the *NFS* Server.

```
apt-get install portmap nfs-common nfs-kernel-server
dpkg -l portmap nfs-common nfs-kernel-server
dpkg-reconfigure portmap
```

2. Create and export the */CASTADIVA* directory.

```
mkdir /CASTADIVA
echo "/CASTADIVA␣192.168.1.0/255.255.255.0(rw,sync,no_subtree_check,
    no_root_squash)" >> /etc/exports
exportfs -ra
```

3. Configure the *hosts.deny* and *hosts.allow* files.

```
cat /etc/hosts.deny
lockd:ALL
mountd:ALL
rquotad:ALL
statd:ALL
```

```
cat /etc/hosts.allow
lockd: 192.168.1.1, 192.168.1.201, ...
rquotad: 192.168.1.1, 192.168.1.201 , ...
mountd: 192.168.1.1, 192.168.1.201 , ...
statd: 192.168.1.1, 192.168.1.201 , ...
```

6.1.2. Setup the Nodes

It is mandatory that the nodes run a *Linux* based operating system, moreover they should be able to :

- Support C compilation. A cross-compiler might be a solution.
- Support *Network File System* (*NFS*) as a client.
- Have two network interfaces: One supporting Ethernet and the being wireless.
- Support routing (*route* command) for static routing.
- Support *iptables*. They are used to drop packets when nodes are logically out of range but fiscally in range. Which typically occurs with stacked nodes.

In our example, we use *LinkSys WRT54GL*[1] wireless routers with an *OpenWRT*[2] Linux-Based operating system.

OpenWRT is a *Linux* distribution for embedded devices. It is a full operating system, based on a *Linux* kernel, that supports several packages and offers an important customization level. The operating system is supplied as a pre-compiled firmware that can be sent to the device using the original manufacturer's firmware update system. A detailed and up-to-date tutorial of how to install *OpenWRT* is available at `http://wiki.openwrt.org/oldwiki/openwrtdocs/hardware/linksys/wrt54gl`. The following is a commented summary of the *tftp* method.

1. Warning: You MUST install the 2.4 kernel before you install any more recent kernel. Otherwise, you wont be able to flash using *tftp* anymore.

[1] *A Linux-Based router that costs approximately 50€.//See : `http://www.linksysbycisco.com/EU/en/products/WRT54GL`.*

[2] See `http://openwrt.org/`.

2. Download the desired version of the firmware on the *OpenWRT* website. It should come as a *.bin* file. For example: *openwrt-wrt54g-squashfs.bin*.

3. Install a *Trivial File Transfer Protocol* (*TFTP)* client on the computer. For example, *atftp* is a good option for *Debian*.

4. Configure your computer's network card to *192.168.1.x* . Where $x \neq 1$. This is important, the router will be automatically configured to *192.168.1.1* after a successful firmware update and you would have an *IP* conflict.

5. Unplug the power-cord of the router and make sure the router is connected to the computer's network. Using a switch, for example.

6. Run your *tftp* program and run the following command

   ```
   tftp > connect 192.168.1.1
   tftp > mode octet
   tftp > trace
   tftp > timeout 1
   tftp > put openwrt-wrt54g-squashfs.bin
   ```

7. Immediately plug the router's power-cord: The transfer should start soon.

You might have to repeat those steps a few times before you succeed. You may want to vary the time between the last *tftp* command and the router's start-up. If you do not succeed with the *TFTP* method, It might be because of a previous wrong installation as stated in step 1. If this happens, I recommend to use the *mtd command line method* described in the official *OpenWRT* wiki.

We now need to establish a first connection, using *telnet*, in order to set the *root* password on the router. Then, we will need to configure the routers for *Castadiva*. As I had 5 routers to manage and as I have had to reinstall routers several times to obtain a clean configuration, I designed a script shell that automates the installation of the router. That script shell is a good example to understand how to configure a router.

```
#!/bin/sh

CASTADIVADIR="/castadiva"
NFSDIR="/castadiva/nfs"
GW="192.168.1.2"
ROUTER="201"

# Creation of the castadiva directories
mkdir $CASTADIVADIR
chmod 0777 $CASTADIVADIR

mkdir $NFSDIR
chmod 0777 $NFSDIR


# Configuration of temporary internet access for package install
route del default
route add default gw $GW

echo "nameserver 158.42.249.8" > /etc/resolv.conf

# NFS install
opkg update
opkg install kmod-fs-nfs
```

```

# Set automatic NFS mount at boot
echo "mount␣-t␣nfs␣$GW:/CASTADIVA␣$NFSDIR␣-o␣nolock" >> /etc/rc.d/S90custom
    -user-startup.sh
chmod 0777 /etc/rc.d/S90custom-user-startup.sh

# SSH Public-Key configuration
cat id_*.pub >> /etc/dropbear/authorized_keys
chmod 0600 /etc/dropbear/authorized_keys

# Network configuration
cp network /etc/config/network
cp system /etc/config/system
cp wireless /etc/config/wireless
cp firewall /etc/config/firewall

echo "Install␣successfull,␣please␣reset␣router"
```

The script and its attached files can be copied on the router using the *scp -r*[3] command and it should be run on the router using *ssh*. Let us now review the different files and commands that are used in this script.

- As we configured a default gateway, *NAT* and routing should be enabled on the gateway so that an Internet connection can be established. The *DNS* value mus point to a valid *DNS* server.

  ```
  echo "1" > /proc/sys/net/ipv4/ip_forward
  iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
  ```

- *opkg* is the packet manager in the *OpenWRT* systems.

- */etc/rc.d/S90custom-user-startup.sh* is *OpenWRT*'s specific file for user commands at startup.

- The *.pub* file is a public key for *SSH*. It is recommended to configure *Open SSH public key authentication* among your *Core* and your routers. That will prevent you entering a password each time you connect to a router. To generate a public key for *root* on the core computer, do the following as *root*.

  ```
  ssh-keygen -t rsa
  ```

  Set no paraphrase.

- The files in */etc/config* are used to properly configure the router. Those files are read at router's start-up and overwrite any common Linux configuration. I would recommend to copy and configure those files from the first updated router, as they can change from one to another version.

6.1.3. Compile for the routers

Now that the routers are correctly configured, there is one last step we have to fulfill to obtain a *Castadiva*-ready system. In order to simulate traffic among nodes, *Castadiva* uses binary files and runs them directly on the nodes. Those files are simple *TCP* and *UDP* clients[4] and servers, written in *C*. We thus need to compile those files for our

[3] *scp* allows to transfer files over ssh. *-r* stands for *recursive*, it allows to copy entire folders.
[4] The source files should be provided with the program, they should be located in a "*bin*" directory.

OpenWRT operating system. To do so, we will use a cross-compiler that can be found on the *OpenWRT* site[5]. Make sure that the compiler corresponds to your version. In the previously downloaded archive, the cross-compiler is located at *staging_dir/toolchain-mipsel_gcc3.4.6/bin/mipsel-linux-gcc.*

```
./mipsel-linux-gcc -DDEBUG /CASTADIVA/bin/UdpFlowClient.c /CASTADIVA/bin/
    dacme.c -o /CASTADIVA/bin/UdpFlowClientMIPS
./mipsel-linux-gcc /CASTADIVA/bin/UdpFlowServer.c -o /CASTADIVA/bin/
    UdpFlowServerMIPS
```

Finally, the compiled files must be placed in the *NFS_SERVER_PATH/bin/* directory. Note that there are *MIPS* and *X86* files, this stands for the node's processor's architecture.

6.2. Validation of new functionalities

6.2.1. The execution planner

6.2.1.1. Introduction

An overview of the current execution planer's *Graphical Users Interface* (*GUI*) can be found in appendix A on page 59. It was designed to allow automatic execution of various and possibly different simulation scenarios. The user can generate a list of previously saved scenarios, indicate how many times each simulation should be run and let the system process his list. Results for each simulation are written in a text file.

The version I was asked to validate offered almost the same interface than the one shown in appendix A. Unfortunately, I did not manage to get it working, it is to say to load a scenario nor to run the simulations. All my intents were solved with some *JAVA* errors.

I thus started looking for what could be wrong and finally reviewed most of the source code. *Alvaro* also participated that task as he needed the execution planner functionality for his own work. Particularly, he added the deferred simulation start system and the *Save/Load list* function.

6.2.1.2. Design

I tried to understand and conserve as much as possible the existing code and design of the execution planner.

Figure 6.2 shows a very general overview of the current execution planer's functioning.

[5]See `http://downloads.openwrt.org/kamikaze/8.09.2/brcm-2.4/OpenWrt-SDK-brcm-2.4-for-Linux-i686.tar.bz2`.

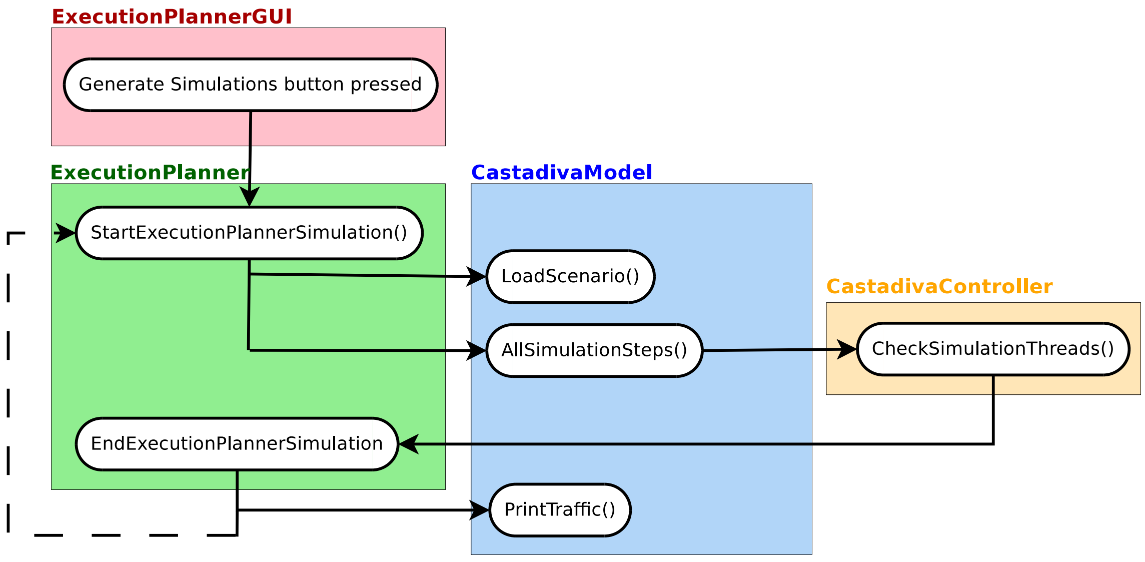


Figure 6.2.: Global design of the Execution Planner

LoadScenario() and *AllSimulationSteps()* are two main functions of *Castadiva*. What the *Execution Planner* does, is to take advantage of *Castadiva*'s *Save* and *Load* functionalities in order to load and process many simulations in a loop. Once the scenario is loaded and when its simulation ends, we need to intercept its ending instructions (*CheckSimulationThreads*) and write the results to a file (*PrintTraffic*). Eventually we do also need to start the next run or simulation.

Further information for more detailed comprehension can be found directly into the *ExecutionPlanner.java* file as I fully commented any piece of code I worked on.

6.2.1.3. Usage

There are two options to add a simulation to the execution planer's simulation list : You can create a new scenario (1) or load a previously saved one (2).

1. If you decide to create a new scenario, the simulation window will be displayed and you will be able to setup the scenario to your needs. You will then need to save it: what the execution planner does, basically, is to use *Castadiva*'s load function for each simulation of it's list. If you do not save the scenario, *Castadiva* wont be able to load it.

2. To load a previously saved scenario, you need to select either the folder containing your scenario or the *scenario* folder itself in the file selector. When a scenario was successfully loaded, it is displayed in the simulation window. Be careful that if you make any changes on the scenario, they wont be kept if you simply press the Accept button. You need to use the save option and erase the previously loaded file or create a new one in order to keep your changes.

Important note: In points 1 and 2 I introduced how the execution planner saves and loads scenarios. Consequently, you should also notice that, if you use the execution planner to load any scenario, your previous settings and scenario will be erased.

Once a scenario is loaded, a new line is added to the execution planner. To edit the line and change the *source folder*, the *results folder* or the amount of *runs*, use the *Edit Simulation* button (See figure 6.3).

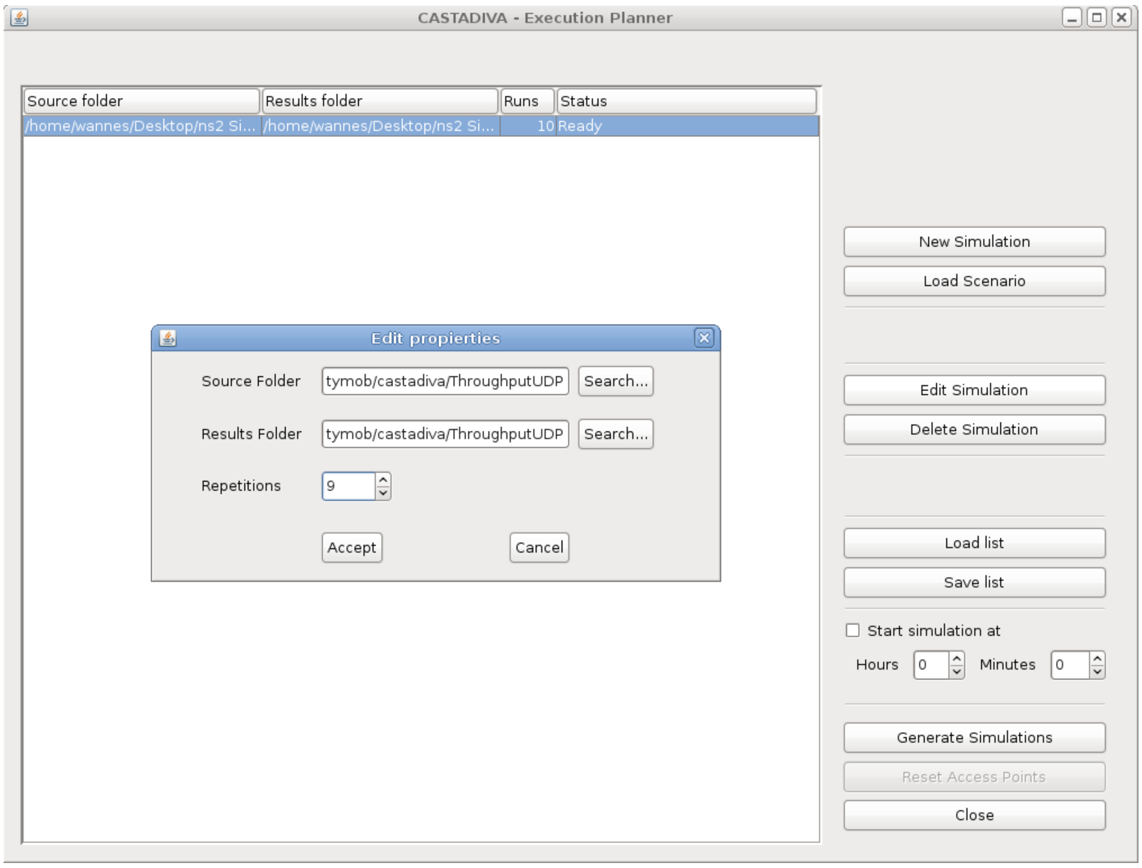


Figure 6.3.: Simulation edition in the execution planner

The *Source folder* is the path to your scenario, and the *Result folder* will contain the simulation results. The results are written in a text file named *X_definedTraffic.txt* where *X* corresponds to the run. For example, if you have 10 runs in your simulation, the first run will be written in the *[Result Folder]/Iterations/10_definedTraffic.txt* and the last run to *[Result Folder]/Iterations/1_definedTraffic.txt*.

The *Load list* and *Save list* buttons allow to save and load the execution planer's simulation list to a file. It might be interesting to save a large list in order to save time in the case something goes wrong during simulation.

The *Start simulation at* checkbox offers to defer the execution according to the system time : the execution planner will figure out how long it must wait before the configured time of the day (*Hours* and *Minutes*). The simulation can thus be planned up to 24 hours after it's configuration. Note that the real start time can be delayed of up to one minute and that the timer can be canceled as long as the simulations are not started.

Finally, the *Generate Simulations* button starts the execution planer's process. During simulation, the *runs* will be decremented and the *Status* information for each row will be updated. The status can take one of the following values :

- *Ready*: The scenario is ready to be simulated.
- *Done* when the simulations where successfully played for this scenario.
- *Waiting for the AP...* when *Castadiva* is collecting simulation information. It means that the simulation is ending. If the *Execution Planner* gets blocked on this message,

it might mean that something went wrong during simulation and that it is not able to collect the simulation information.

- *Simulation in process...* : the scenario is being simulated.
- *Canceled* : the scenario's simulation was interrupted.

When simulating the list, the *Reset Access Points* is enabled. Clicking this button simply sends a *reboot* instruction to every access point. This might be a solution to unlock a problematic simulation or cancel the execution planer's process.

6.2.1.4. Validation

As commented in the introduction, I could not validate the original version of the *Execution Planner* as I did not succeed to run it. Taken in account the absence of any comment in the code, I would say the *Execution Planner* was in a beta stage. It might have functioned but probably in a very precise situation with very precise arguments.

The execution planner has suffered many changes since the initial version and we discovered new problems later on. The difficulty with the execution planner is that it needs to take in account any possibility of *Castadiva* : Plugins, routing protocols, types of traffic... All those functionalities are spread all over the original program. I mean that it is not sufficient to call a *loadCastadiva()* function and later on *startSimulation()*. There are numerous settings that have to be separately taken in account.

As I also commented in the introduction, *Alvaro* made a regular use of the execution planner for his own work. Most of the time it worked just fine and the benefits of the execution planner are really significant when the amount of simulations increases. Nevertheless, with over 200 runs spread over various simulations, a problem occurred and I did not manage to solve it until now. It might be something related with memory overflow.

6.2.2. Routing Plugins

6.2.2.1. Introduction

With *MANETs*, mobility can make links among nodes unstable. It is important to have strong routing protocols to reconfigure routes in a short time and allow efficient communication. In *Castadiva*, there are two options for routing:

1. Static routing, it is to say that routes are calculated with *Castadiva* for the whole simulation and automatically set on the nodes. For now, *Castadiva* only supports the *OPTIMUM* static routing which is based on the *Dijkstra algorithm.*

2. Dynamic routing. In such case, nodes are responsible for the routing process. In other words, each access point must run its own and appropriate routing software. Routing is thus depending on the routing software available for the *Access-Point*'s platform.

Point 1 is implemented since the very beginning of *Castadiva.* It is based on some *route* instructions that are sent to the *Access-Points.* Point 2 was also available in legacy versions of *Castadiva* : the user could simply install and run routing protocols on all of his *Access-Points.*

Routing Plugins allow something simple but also very useful. When a routing plugin is configured, it receives the instructions of how to start and stop a specific routing protocol on the *Access-Points.* It can also replace one configuration file on the *Access-Points.*

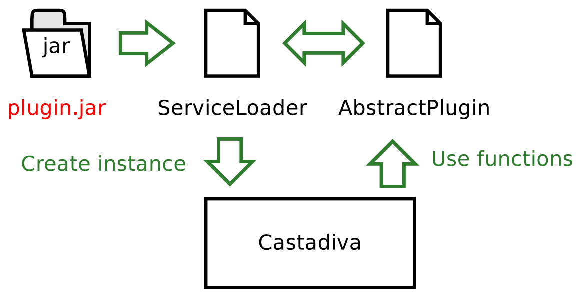


Figure 6.4.: ServiceLoader class usage in *Castadiva*

My work here was to validate a previously implemented *Routing Plugin System*. Unfortunately, it was absolutely not functional for some reasons I will describe later in this section. I thus reviewed the whole system in order to obtain what we just described.

Important note: *Routing Plugins* do not install any routing software on the *Access-Points*. It is mandatory to previously install the adequate routing software on every *Access-Point*. The only thing routing Plugins can do is to send the start and stop instructions for the routing protocol and replace one configuration file on the *Access-Points*.

6.2.2.2. Routing on OpenWRT

With *OpenWRT* based *Access-Points*, a few routing protocols are originally supplied among the original packages. For example, we have: *OLSR*, *Babel*, *B.A.T.M.A.N*, *BGPv4*, *BGPv4+*, *BGPv4*, *IS-IS*, *OSPFv2*, *OSPFv3*, *RIP*, *RIPNG*.[6]

It is also possible to compile some more protocols for *OpenWRT Kamikaze* but this is a quite difficult and random process.

6.2.2.3. OLSR

OLSR[7] (Optimized Link State Routing) is one of the routing protocols available for *OpenWRT* and designed for *MANETs*. It is also the protocol that was used for the routing plugin system's development.

6.2.2.4. Design

The following describes the technical implementation of *Routing Protocol Plugins* in *Castadiva*. First of all, you should know that the design of the *Routing Plugin System* is quite complicated compared to what it actually does. This is probably because of the *Mobility Plugin System* (See section 6.2.3). The *Mobility Plugin System* has higher requirements than the *Routing Protocol Plugin System* but they were designed together and do therefore share a similar design. In the validation section I will make suggestions for an alternative design.

The *Java* programming language offers a mechanism to allow a user to insert custom code directly into a compiled program. More generally, the term of *Plugin* can be used.

[6]This is a non exhaustive list for the *Kamikaze* version of *OpenWRT*. It is the result of the *opkg list | grep routing* command.

[7]See `http://www.olsr.org` for further information

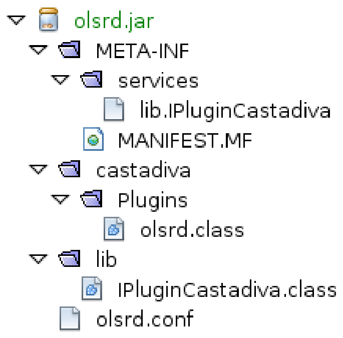


Figure 6.5.: Structure of a routing plugin's *.jar* file

As shown in figure 6.4, the *ServiceLoader*[8] class can load a *.jar* archive directly into the a *Castadiva* running program. The *.jar* archive contains a special set of files as shown in figure 6.5.

The structure of the *.jar* file is depending on the structure of the *Castadiva* source folder and it contains several files:

- *lib.IPluginCastadiva* is related to the *ServiceLoader* class and must point to the class (regarding packages) where the plugin should be loaded. In our example, it would contain the following line: *castadiva.Plugins.olsrd*
- *MANIFEST.MF* is an automatically generate file.
- *olsrd.class* contains the plugin implementation. It will be described further on.
- *IPluginCastadiva.class* is the abstract class that *olsrd.class* implements.
- *olsrd.conf* has no direct relation with the plugin system. *olsrd.conf* is the configuration file for the routing protocol and I put it into the archive to have a unique plugin file.

The *.class* file naturally comes from a *.java* source file which must match a previously established pattern. In *Castadiva*, the pattern is an abstract class named *IPluginCastadiva*. It can be found in the *lib* package and suggests the following methods :

- *getBin()* : Where is the binary file located?
- *getFlags()* : What arguments should be used to run the binary file?
- *getPathConf()* : Where is the configuration file located?
- *getConf()* : How is the configuration file named ?
- *getKillInstruction()* : How to stop the protocol from running?

The user must implement those methods for the routing protocol he wants to use. To do so, the *Custom Routing Protocol* user interface allows the user to set *String* values for each of those function, except for the *kill* instructions : to stop a running protocol, *Castadiva* uses the following command : *killall protocol* where protocol is the binary file passed as *getBin()*. See figure 6.6.

[8]See *Javadoc* for further information and tutorials.

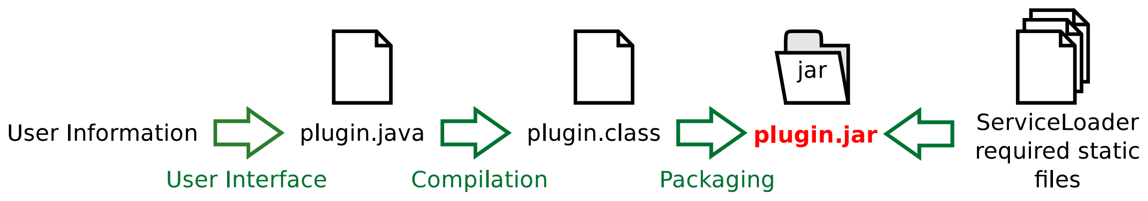


Figure 6.6.: ServiceLoader plugin generation in *Castadiva*

Important note: The user does not have to compile or package anything. This is done automatically by *Castadiva*, using the *javac* and *jar* system commands. This supposes that the *javac* and *jar* are available on *Castadiva*'s host system.

6.2.2.5. Usage

How to setup a routing protocol on an *Access-Point* depends on its operating system. On an *OpenWRT* operating system, *OLSR* can be installed using the following commands.

```
opkg update
opkg install olsrd
```

Nevertheless, there are a few points that should be considered on every system to guarantee compatibility with *Castadiva*.

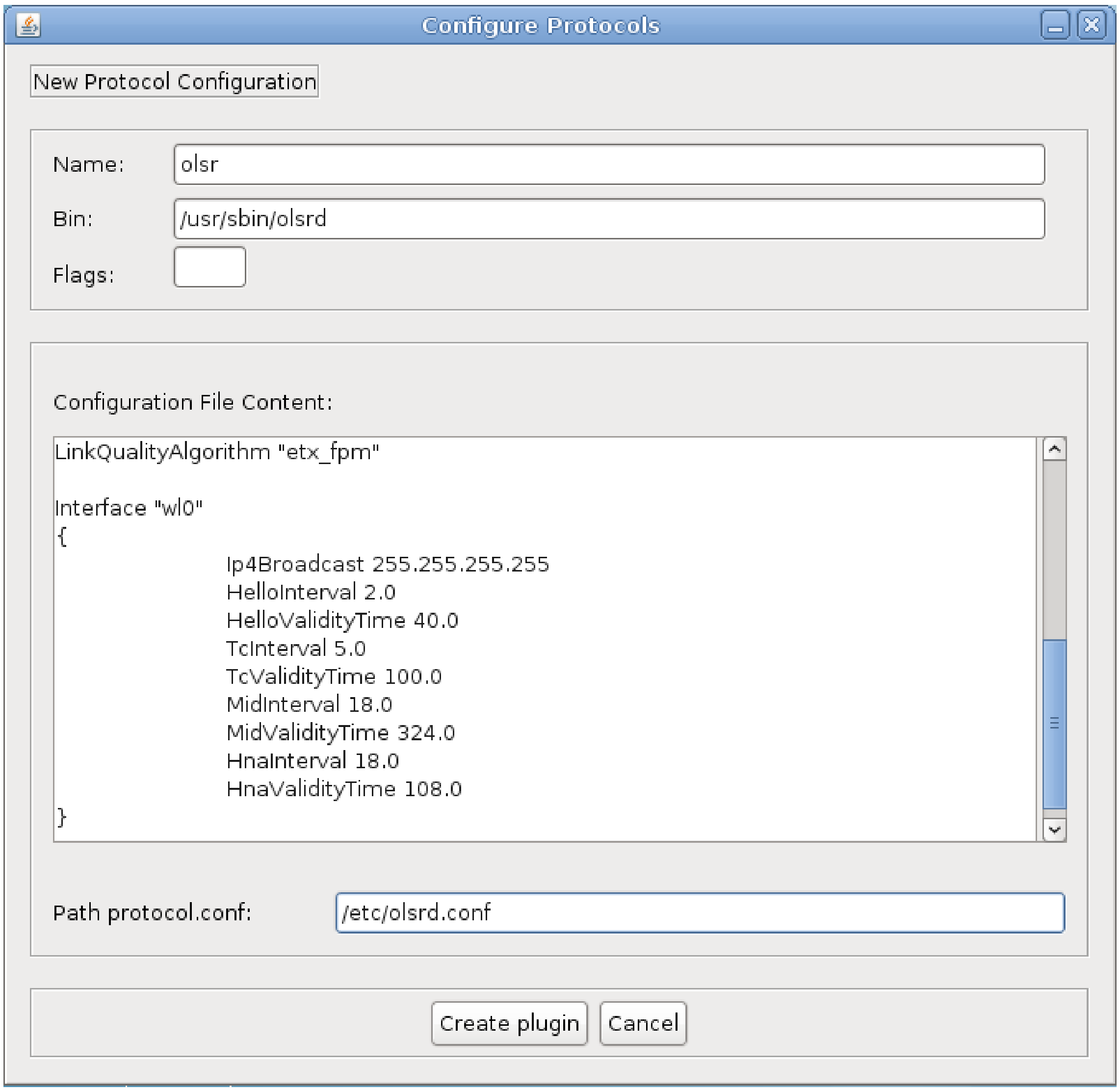
- The routing protocol should not start automatically with the *Access-Point*. In our example, the following command allows to enable or disable the protocol on router's startup */etc/init.d/olsrd [ENABLE|DISABLE]*. Note that it is also possible to configure startup settings in /etc/rc.d
- The routing protocol must be configured for the correct interface. With *OLSR* on an *OpenWRT* operating system, the interface can be selected in */etc/config/olsrd*.
- It might be interesting to backup the original protocol configuration file. While using a plugin for the first time, *Castadiva* may erase that file. It is also interesting to use the original configuration file as a basis for custom configuration. With OLSR on an OpenWRT operating system, the *olsrd.conf* configuration file is located in : */var/etc/olsrd.conf*

To ease compilation of new *Routing Protocol Plugins*, *Castadiva* offers a *Graphical User Interface* (*GUI*). Let us review each option offered by figure 6.7.

The *Name* field allows us to identify the protocol and it's configuration. It must be unique and if an existing name is re-used, it will be overwritten. *Bin* must point to the location of the binary file, on the *Access-Point*, which allows to start the routing protocol. Note that the whole path must be given and that the last part of that path, which should be the filename, is also used to kill the protocol when the simulation ends. This is done trough the *killall binName* command. Therefore, flags will be inserted just after *Bin* and for protocol startup only. *Configuration file content* is the content of the configuration file that will be written in the *Path protocol.conf*, on each *Access-Point*.

Important note: The automated compilation uses certain files in the source (*/src*) folder of the project. Make sure that the following constants are correct in *Castadiva-Model.java*:

- *PLUGIN_WORKFOLDER = "pluginTemporaryFiles";* is the name of the temporary folder that is created for compilation purpose.



Name: olsr

Bin: /usr/sbin/olsrd

Flags:

Configuration File Content:

```
LinkQualityAlgorithm "etx_fpm"

Interface "wl0"
{
			Ip4Broadcast 255.255.255.255
			HelloInterval 2.0
			HelloValidityTime 40.0
			TcInterval 5.0
			TcValidityTime 100.0
			MidInterval 18.0
			MidValidityTime 324.0
			HnaInterval 18.0
			HnaValidityTime 108.0
}
```

Path protocol.conf: /etc/olsrd.conf

Figure 6.7.: Routing plugin configuration window

- *PLUGIN_JAR_FOLDER = "src/castadiva/Plugins";* is the path to the Plugin folder, where *.jar* files are copied after compilation and where Plugins are loaded on startup.

- *PLUGIN_INCLUDE_FOLDER = "src/lib";* must point to the folder containing the *IPluginCastadiva.java* abstract class.

6.2.2.6. Troubleshooting

The following is presented as a Question - Answer list.

1. **I just installed or used a custom routing protocol on my Access-Points. Now, I can no longer execute a simulation with the *NONE* routing settings.**
   Two things should be checked. First, the routing protocol should not start automatically when your Access-Point starts. See section 6.2.2.5 to disable the routing protocol on startup. Secondly, try to run the *route* command on your access points. A previous simulation may have stopped unexpectedly and did not stop the routing protocol. Use the *killall protocolBinFileName* command to make sure that the protocol is no longer running.

2. **After I tried to configure a plugin, *Castadiva* wont launch anymore.**
   Try removing the *.jar* file from the plugin directory

3. **A folder is created in *Castadiva*'s execution directory. What about it?**
   A folder usually named *pluginTemporaryFiles* is created to compile and then package the plugin. This folder should be deleted automatically and it can be harmlessly deleted at any time.

4. **I just created a new routing protocol plugin but it does not work properly.**
   There can be many reasons for a plugin not to work. Nevertheless, there are a few recurrent things you can try. First, consider the plugin as the execution of the following command. Your specific commands can be found in */CASTADIVA/instructionsForNodeX*.

```
cp /castadiva/nfs/olsrd.conf /etc/olsrd.conf
#Start routing protocol.
/usr/sbin/olsrd -d 0
#Wait for the end of simulation.
sleep 70
#Kill the protocol
killall olsrd 2>/dev/null
#Clean protocol configuration file
rm /etc/olsrd.conf
```

   Try to run each command separately, with *ssh*, to find out where the problem is.

5. **The routing protocol is correctly activated but simulation results are not consistent.**
   Consider that most routing protocols require time to exchange routing information and establish routes. Try to configure traffic a reasonable time after the simulation starts.

6.2.2.7. Validation

As previously commented, the original program could not be validated as it was unable to generate any plugin. A deep analysis of the code revealed several errors :

1. Paths were *hard-coded* into the program. Therefore, I introduced global variables to define the execution environment and give more flexibility to the program.
2. Some folders where required for the plugin creation and were not automatically generated. The whole structure for plugin creation was finally reviewed. Temporary folders are now deleted after each plugin creation.
3. The initial plugin system did not remove any temporary file. As a result, any plugin contained all the files generated for all the previously generated Plugins. Currently, a single folder (*PLUGIN_WORKFOLDER*) is created and removed for each plugin creation.
4. The initial plugin system was unable to handle any configuration file of more than one line. It was also unable to handle odd " symbols. As each plugin contained all of his ancestors (see 3), I could find out that the typical "asdfasdfasdfasdf" chain was recurrently used to test the *Configuration File Content* text area. Any carriage return crashed *Castadiva*. This was solved using a separate *protocol.conf* file to store the protocol configuration data. I also included the *protocol.conf* file into the jar archive in order to have a single file for a plugin. Moreover, it allows to easily modify the configuration file.
5. The code was partly Spanish written and there was practically no comment. We can suppose that it was still under development. I translated the code and wrote comments for the whole routing plugin functionality.
6. The *Flags* text filed was not used in the program. It is now usable.
7. The configuration field was removed by the program after the simulation. If the simulation failed, the configuration file was not reset. As for any other simulation element, I planned the configuration file removal using a *sleep* command.
8. The routing instructions were sent too early. The routing protocol was started seven seconds before the client and visibility settings. This was no problem with static routing as routes did not change along the simulation. Moreover, separating those instructions was interesting to reduce the Access-Point's load. But with dynamic routing Plugins, the routing protocol had 7 seconds to initialize itself with no visibility restrictions. As the plugin system should be generic for any routing protocol, it is not possible to evaluate the initialization time. The initialization time should thus be part of the simulation.

I made a few simple tests in order to check the creation of a plugin and the loading/unloading of a *Routing Plugin* during a simulation. Therefore, I aligned 5 nodes so that each node only saw his direct neighbor. I then connected to every *Access-Point* using *SSH* and checked that the routes were set correctly using the *route* command. Finally, i connected to the lefter *Access-Point* and ran the *traceroute* command to the righter one. The results are correct :

```
root@router1:~ traceroute 192.168.2.205
        traceroute to 192.168.2.205 (192.168.2.205), 30 hops max, 38 byte
            packets
```

```
1  192.168.2.202 (192.168.2.202)  2.370 ms  2.953 ms  8.983 ms
2  192.168.2.203 (192.168.2.203)  5.417 ms  4.306 ms  5.376 ms
3  192.168.2.204 (192.168.2.204)  6.888 ms  6.809 ms  6.434 ms
4  192.168.2.205 (192.168.2.205)  8.055 ms  9.685 ms  7.111 ms
```

Even though the previous test succeeded, the validation of the *Routing Protocol Plugin* is not achieved, it would probably be interesting to try to implement another protocol. I was suggested to implement *AODV* but I did not succeed as there is no official implementation for *AODV* in *OpenWRT Kamikaze*.

Let us finally comment the design of the *Routing Plugin System*. Using the *ServiceLoader* class to simply store a few text lines is probably a very complicated way to enable *Routing Plugins* in *Castadiva*. A simple text file would have been an easier solution. Moreover, it would be safer as it would not require to compile and package the plugin files. Calling system commands might lead to cross-platform compatibility issues.

Also to make it more flexible, the routing plugin system could be replaced by a larger system that would simply allow the execution of custom user code before and after a simulation.

6.2.3. Mobility Plugins

6.2.3.1. Introduction

I previously commented the *plugin* concept in section 6.2.2 on page 28. A *Mobility Plugin* is supposed to enable custom mobility pattern definition in *Castadiva*. In other words, *Mobility Plugins* can be configured to add controlled and dynamic mobility to *Castadiva*'s scenarios. In previous versions ,*Random Mobility* already was an option, as well as manual mobility which was not handled by *Castadiva*. Controlled mobility allows the usage of an algorithm to define the positions of the nodes during the simulation. The *Mobility Plugin* system's dynamism comes from the the possibility to use variable parameters like the amount of nodes, the maximum speed in the algorithm and so on.

The following section will describe the design, usage and validation for the *Mobility Plugin System*.

6.2.3.2. Design

As for *Routing Plugins*, the *Mobility Plugin System* is based on the *ServiceLoader* class which allows to insert code into a running program. Figure 6.8 on the next page shows how to create a *.jar* plugin file that can be loaded using the *ServiceLoader* class. The process is quite similar to the one described for routing Plugins (6.2.2.4 on page 29) but it is slightly more complicated regarding two points.

1. The *MobilityPluginWrapper* class has some dependencies with *Castadiva*'s files. Therefore, we need to copy those files, compile them and insert them in the *.jar* package. To avoid any confusion for *Integrated Development Environment* like *NetBeans* which scan folders for *.java* files, those files are stored as *.txt* files.

2. The Mobility plugin system is based on a wrapper. A wrapper that is nothing more that the final *.java* file with a few */*code*/* and **/name*/* tags that should be replaced with the user code.

As for *routing Plugins*, *mobility Plugins* are automatically generated and packaged. They rely on the *jar* and *javac* system commands.

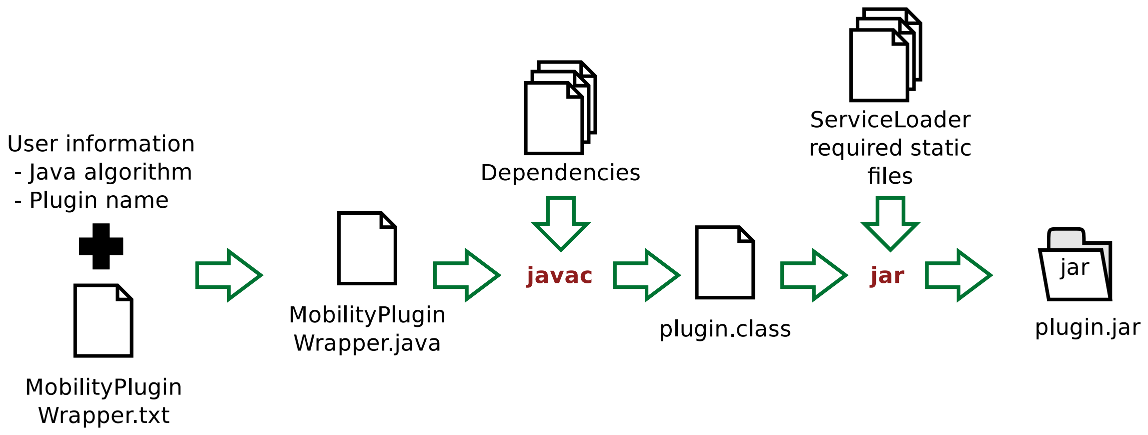


Figure 6.8.: Creation of a *.jar ServiceLoader Mobility Plugin*

Important note: The automated compilation uses certain files in the source (*/src*) folder of the project. Make sure that the following constants are correct in *Castadiva-Model.java*:

- *PLUGIN_WORKFOLDER = "pluginTemporaryFiles";* is the name of the temporary folder that is created for compilation purpose.
- MOBILITY_PLUGIN_JAR_FOLDER= *"src/castadiva/MobilityPlugins";* is the path to the Plugin folder, where *.jar* files are copied after compilation and where Plugins are loaded on startup.
- *PLUGIN_INCLUDE_FOLDER = "src/lib";* must point to the folder containing the *IPluginCastadiva.java* abstract class.

6.2.3.3. Usage

The *Mobility Plugin System* is quite simple to use when it is correctly understood. Figure 6.9 shows the unique user interface that can be used to generate a custom mobility plugin.

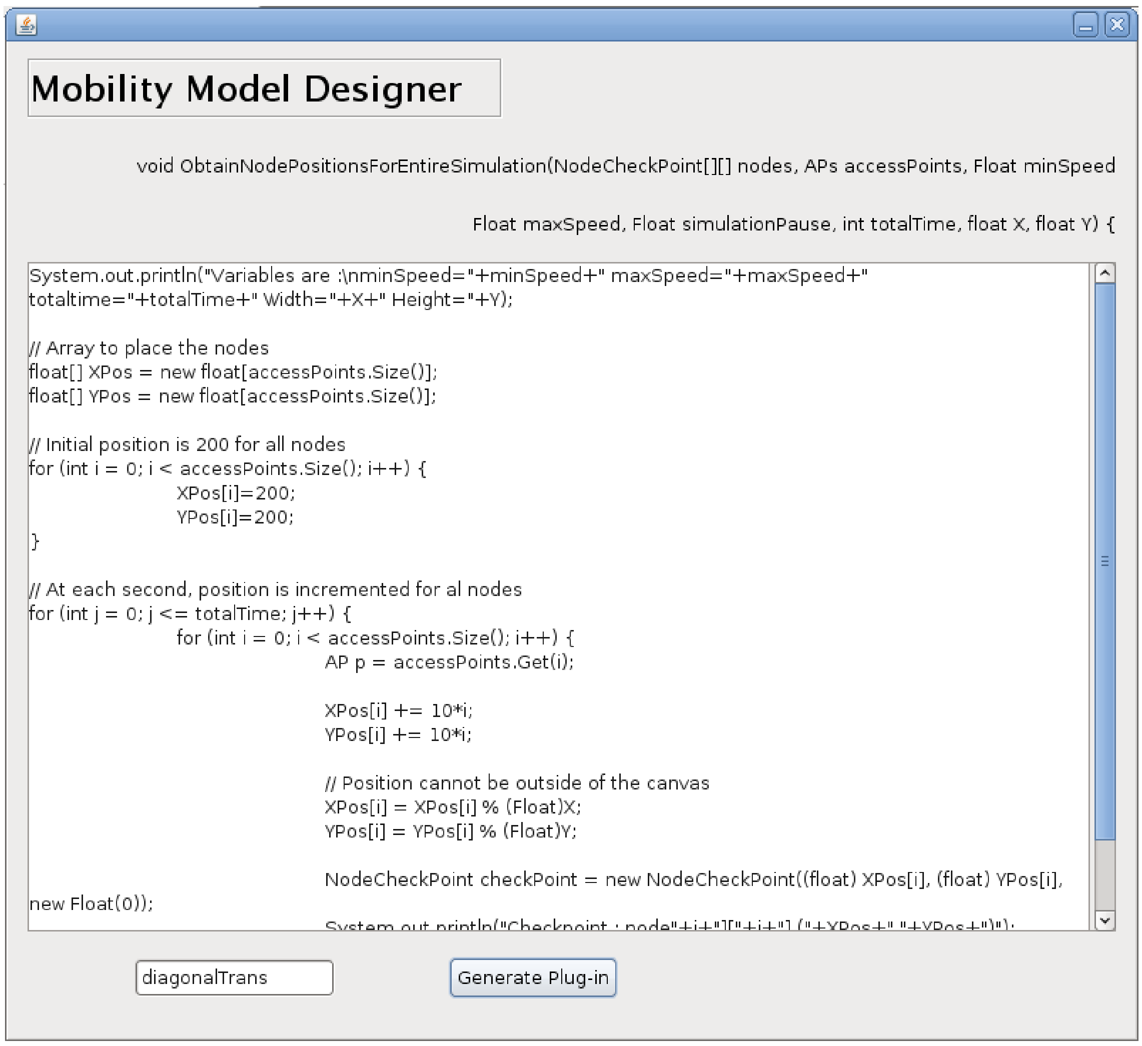
In the upper part of the figure, we can see the header of a *JAVA* function. The variables shown in that header are the one that the user can use in order to design his algorithm. The only requirement for the final algorithm is that it entirely fills the *NodeCheckPoint[][] nodes* matrix. In order to create a *NodeCheckPoint* instance, the *NodeCheckPoint* class can be used as follows:

```
NodeCheckPoint checkPoint = new NodeCheckPoint(float XPosition,float
    YPosition,float ZPosition);
```

After the *Generate Plug-in* button was pressed, the program will try to compile the code. If there is any compilation error, it is displayed in a special window.

The second part of the *Mobility Plugin System* takes place in the *Simulation window*[9]. First of all, the *Mobility Plugin* should be available in the *Mobility Model* drop-list. Then, it is important to configure the different parameters that are used in the *Mobility Plugin*: *Min Speed*, *Max Speed*, *Pause* and *Simulation time*. Finally, pressing the *New Simulation* button will show the node's movement on the canvas. Be aware that this is not a live replay

[9]See appendix A for more information about Castadiva's *GUI*.



void ObtainNodePositionsForEntireSimulation(NodeCheckPoint[][] nodes, APs accessPoints, Float minSpeed

Float maxSpeed, Float simulationPause, int totalTime, float X, float Y) {

```
System.out.println("Variables are :\nminSpeed="+minSpeed+" maxSpeed="+maxSpeed+"
totaltime="+totalTime+" Width="+X+" Height="+Y);

// Array to place the nodes
float[] XPos = new float[accessPoints.Size()];
float[] YPos = new float[accessPoints.Size()];

// Initial position is 200 for all nodes
for (int i = 0; i < accessPoints.Size(); i++) {
			XPos[i]=200;
			YPos[i]=200;
}

// At each second, position is incremented for al nodes
for (int j = 0; j <= totalTime; j++) {
			for (int i = 0; i < accessPoints.Size(); i++) {
						AP p = accessPoints.Get(i);

						XPos[i] += 10*i;
						YPos[i] += 10*i;

						// Position cannot be outside of the canvas
						XPos[i] = XPos[i] % (Float)X;
						YPos[i] = YPos[i] % (Float)Y;

						NodeCheckPoint checkPoint = new NodeCheckPoint((float) XPos[i], (float) YPos[i],
new Float(0));

						System.out.println("Checkpoint : node"+i+"]["+j+"] ("+XPos+" "+YPos+")");
```

diagonalTrans

Figure 6.9.: *Mobility Plugin* configuration window in *Castadiva*

and that it is not synchronized with the real simulation. After a simulation was successfully processed, you can review each second of the simulation using the *Show* feature.

Important note: Even if the *maxSpeed* parameter is not set in the algorithm, it is mandatory to set a positive *Max Speed* in the Simulation interface. Otherwise, mobility is not used.

6.2.3.4. Validation

As previously commented, the original program could not be validated as it was unable to generate any plugin. A deep analysis of the code revealed several errors :

1. Some variable names were displayed with different case and sometimes different names in the graphical interface. The variable pause was also proposed but not present in the code.

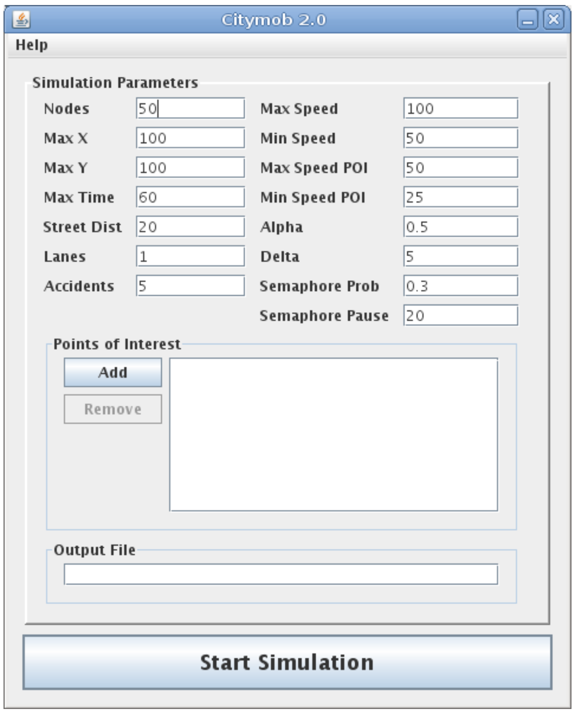
2. When several plugins were available in the plugin list, whatever plugin was selected, the first plugin (alphabetical order) was always loaded and used. This came from a bad usage of the ServiceLoader class. I corrected this issue by cleaning up and recoding the Plugin structure.

3. Even when two nodes remained out of range during the whole simulation, they had some data transferred during the simulation. This problem came from an error with the synchronization between client startup and visibility settings. With static simulations, it was not a problem to set visibility instructions 10 seconds before the simulation started: those instructions did not change during the whole simulation. On the contrary, it allowed to spread the *Access-Points* system load on simulation startup. With mobility, visibility instructions are dynamic and change over time. Those 10 seconds of delay introduced incoherent simulation results: visibility settings were always 10 seconds earlier than traffic instructions.

4. *totalTime = 111;* was hard-coded somewhere in the code. This caused an out of array exception if the simulation lasted less than 111 seconds.

In order to validate the *Mobility Plugin System*, I generated a mobility scenario that makes nodes move in diagonal with different speeds for each node. The *JAVA* code for that algorithm can be found in Appendix H on page 85. I then manually calculated the theoretical position of a few nodes at a determined time and checked it was well imported in *Castadiva*. This is probably not a full validation of the *Mobility Plugin System*, and it would probably be interesting to compare results. But I did not succeed in obtaining any valid results with mobility in *Castadiva*.

Important note: The *Mobility Plugin system* is functional but the results of any subsequent simulation can not be considered as valid. This is because of a problem that was discovered in the *Access-Point*'s client and server programs, in particular with TCP. This problem is affecting all mobility in *Castadiva*, even random mobility. The issue was not fixed but it is detailed in section 6.3.1 on page 44.



Nodes	50	Max Speed	100
Max X	100	Min Speed	50
Max Y	100	Max Speed POI	50
Max Time	60	Min Speed POI	25
Street Dist	20	Alpha	0.5
Lanes	1	Delta	5
Accidents	5	Semaphore Prob	0.3
		Semaphore Pause	20

Figure 6.10.: *CityMob 2.0* User Interface

6.2.4. CityMob Import

6.2.4.1. Introduction

CityMob[10] (See figure 6.10) is a mobility pattern generator for *VANET*s (*Vehicular Ad-hoc NETworks*).It allows to generate random mobility scenarios in *ns-2* format. An example of *CityMob* output can be found in appendix D on page 74.

Next sections will test, fix, explain and validate *CityMob* importation in *Castadiva*.

6.2.4.2. Design

This section shows the current design of the *CityMob* importation system. The global behavior and design respects *Ignacio Muñoz Vera*'s initial work but the implementation of each function suffered many changes, as it corrects a very important semantic errors, as well as a syntactic errors[11].

Figure Figure 6.11 on page 40 shows how *Castadiva* deals with *CityMob* files (*cityMob.out*) and converts them into a set of *Access-Points* and a *mobility matrix*, which is a matrix that contains the position of each node at every second of the simulation.

When a *CityMob* importation is processed, *Access-Points* parameters like its name, its *IP address*, ..., can come from two possible sources:

1. A first and priority option is the *aps.txt* file which can be configured in the *configuration* folder of *Castadiva*. That file contain settings for a list of node. The first line will be used for the first node and so on. Note that *aps.txt* is similarly used with the node creation utility.

[10]See: `http://www.grc.upv.es/Software/citymob.html`
[11]See 6.2.4.4 for more details about the errors

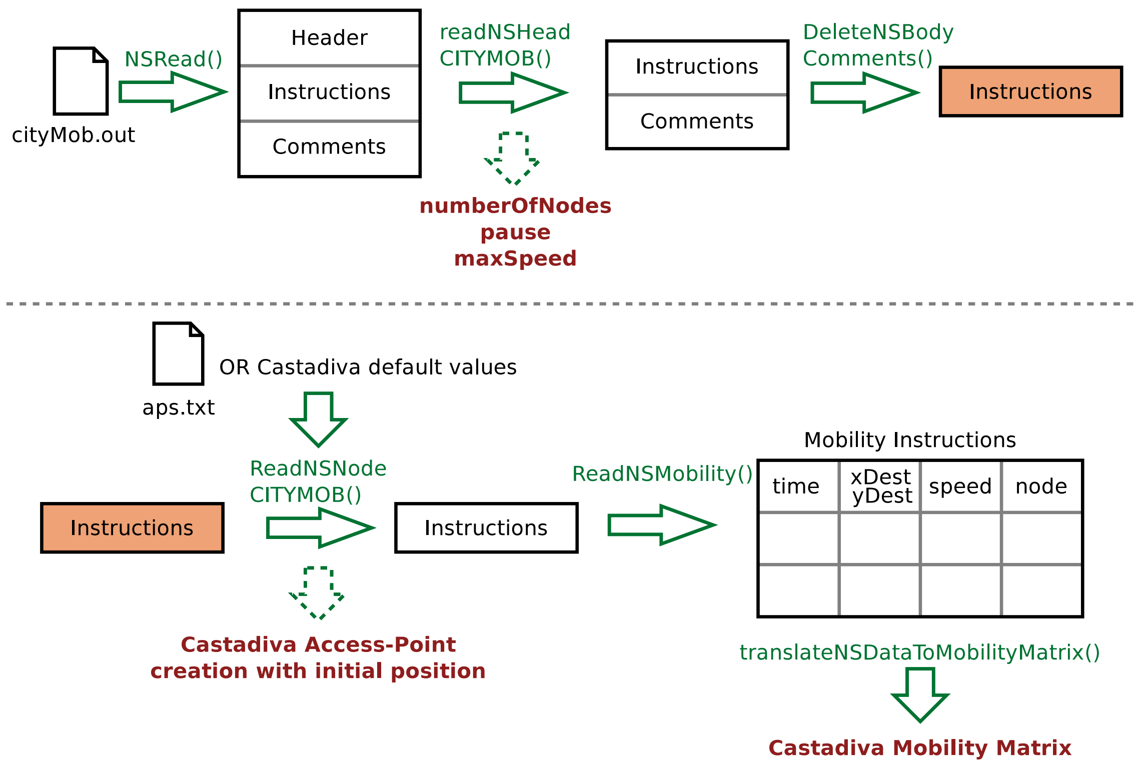


Figure 6.11.: *CityMob* importation process

2. A second option is to use *Castadiva* built-in default values, they are defined in *CastadivaModel.java*'s header.

Let us now comment the most important function in figure Figure 6.11 on page 40 : *translateNSDataToMobilityMatrix()*. That function takes a list of movement instructions, for example :

```
#Node 1: TARGET REACHED (540.0, 1200.0) (75.06782928526441 Km/h)
$ns_ at 52.18759071966314 "$node_(1)␣setdest␣540.0␣1200.0␣75.06782928526441
   "
```

and converts it to a *Mobility Matrix*[12] which knows for every node its position at any second of the simulation. It is important to notice the granularity of the *Mobility Matrix*, one second, which is different from the granularity of the *CityMob* output file. This will lead to approximations.

In order to generate the *Mobility Matrix*, we will need to take in account that a movement can be interrupted by another movement instruction. Therefore, while reading each instruction, we will write it entirely to the mobility matrix. If no later instruction interferes with the current instruction, the matrix is consistent. If a later instruction interferes, it will will consider the position of the node at the instructions time rounded to one second. Here, we have an approximation that is transmitted along the whole simulation. The importance of such an approximation can be seen in figures 6.12 and 6.13 on the facing page.

It is probably possible to obtain strictly correct results but it would require important changes on the original program. In order to reduce the error, I simply added the following

[12]It would be a matrix with *numberOfNodes* rows and *totalSimulationTime* columns. The same matrix is used for *Random Mobility* and *Mobility Plugins*.

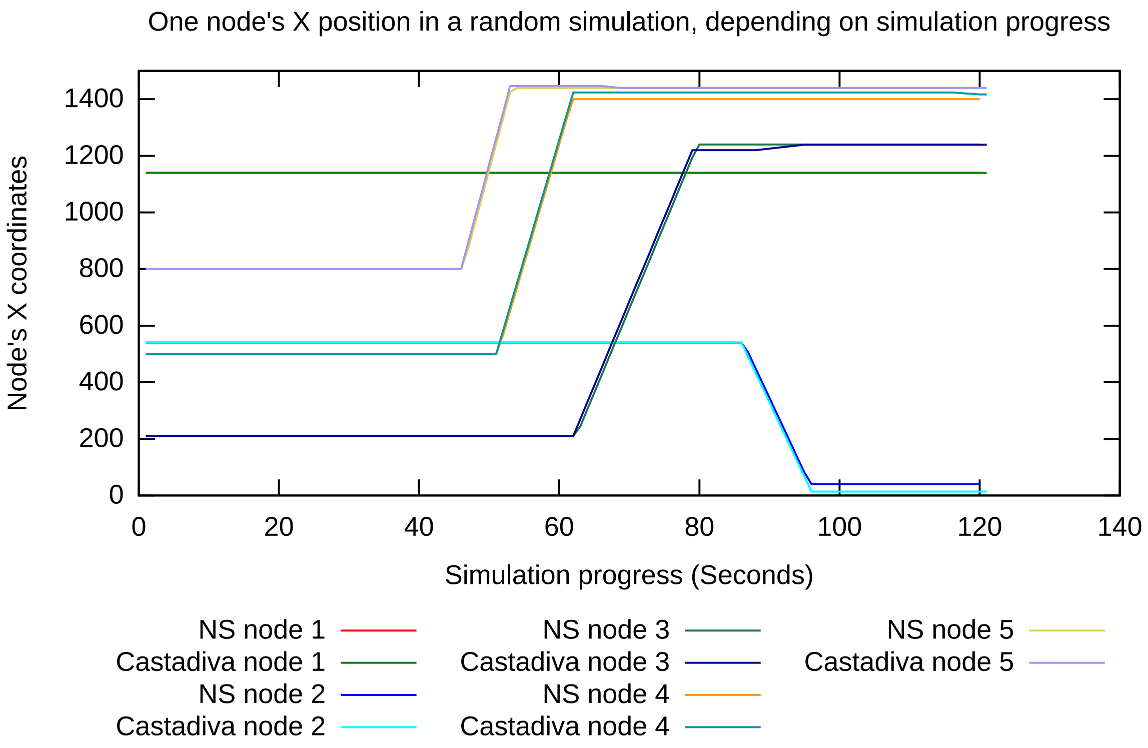


Figure 6.12.: One node's X position in a random simulation without error correction

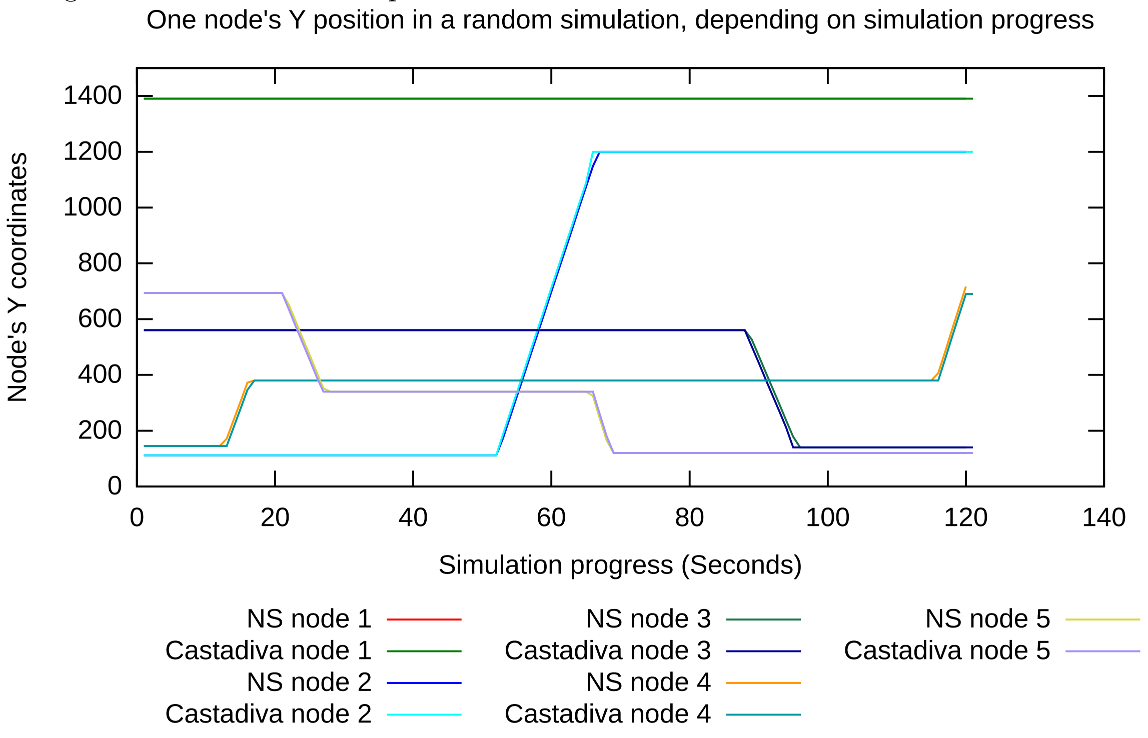


Figure 6.13.: One node's Y position in a random simulation without error correction

condition : if we are processing the last move of the node (which would be the last second of his movement), then we set the node at his theoretical destination (which is given by the movement instruction). Results for the same simulation can be found in figures 6.14 and 6.15 on the next page

Important note: The *CityMob* user interface show speeds as *Km/h*. In the generated *ns-2* files, comments for each movement instruction also announce *Km/h*. The same values are inserted in the *ns-2*'s code beneath the comment (see code in section 6.2.4.2). As *ns-2* handles *meters/seconds*[13], there probably is something wrong. For now, we have configured *Castadiva* so that it behaves similarly to *ns-2*.

6.2.4.3. Usage

There is no particular requirement to import a *CityMob* scenario to *Castadiva*. As it is the current version, we used *CityMob 2* for the development of this functionality. Importation might no longer work if the *CityMob* output file structure undergoes any change.

Once you generated the scenario, use the *import > CityMob* menu and select *CityMob*'s output file. If you want to set custom parameters for the imported *Access-Points*, configure the *configuration/aps.txt* file in *Castadiva*'s folder to do so. Each line of that file represents an *Access-Point*, the CityMob importation system will consider line one for it's first *Access-Point* and so on. Finally, note that a *CityMob* mobility model is not saved when the save function is used in *Castadiva*. You will thus need to load the *CityMob* file before you load a scenario.

6.2.4.4. Validation

The original *CityMob* import did not work out of the box. It assumed (*hard-coded*) that the nodes where named with a string followed by a number from *0* to *N-1*. For example, my nodes where named from *1* to *N* as recommended in the *Castadiva Journal*[1]. This was corrected simply removing this requirement and calling nodes with their number instead of their name.

Castadiva's results were then compared to *ns-2* for a random *UDP* scenario: results were very different. Indeed, the original importation misunderstood *CityMob*, it assumed that when a movement instruction was given at *time T* to move from *position A* to *position B* in *C* seconds, the node would necessarily be in *position B* at *time T+C*. In fact, a node can receive new instruction during its travel. In such case, it interrupts its current movement and starts moving according to the new instructions.

Correcting this made me review a very large portion of the code. Investigating the existing code was a hard task as there were no comments. I thus fully commented the *CityMob* import system.

In order to validate my results, I finally compared *Castadiva*'s new node positions to *ns-2*'s ones. As shown in figures Figure 6.14 on page 43 and Figure 6.15 on page 43, results are now quite conclusive. Note that the slight differences come from *Castadiva*'s sampling which only occurs every second.

Important note: The *CityMob* importation is functional and correct but the results of any subsequent simulation can not be considered as valid. This is because of a problem that was discovered in the *Access-Point*'s client and server programs, in particular with *TCP*. This problem is affecting mobility in *Castadiva* since it's very beginning and is detailed in section 6.2.3.4 on page 38.

[13] *See* `http://www.isi.edu/nsnam/ns/tutorial/` *section IX.2.*

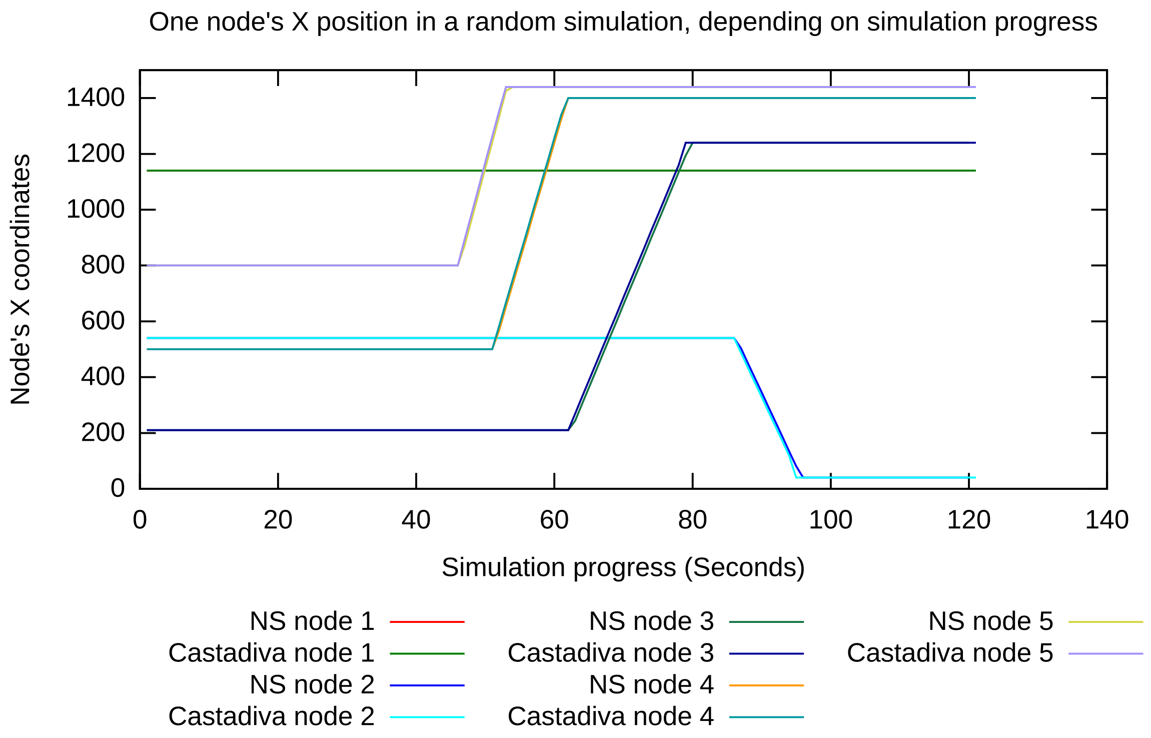


Figure 6.14.: One node's X position in a random simulation

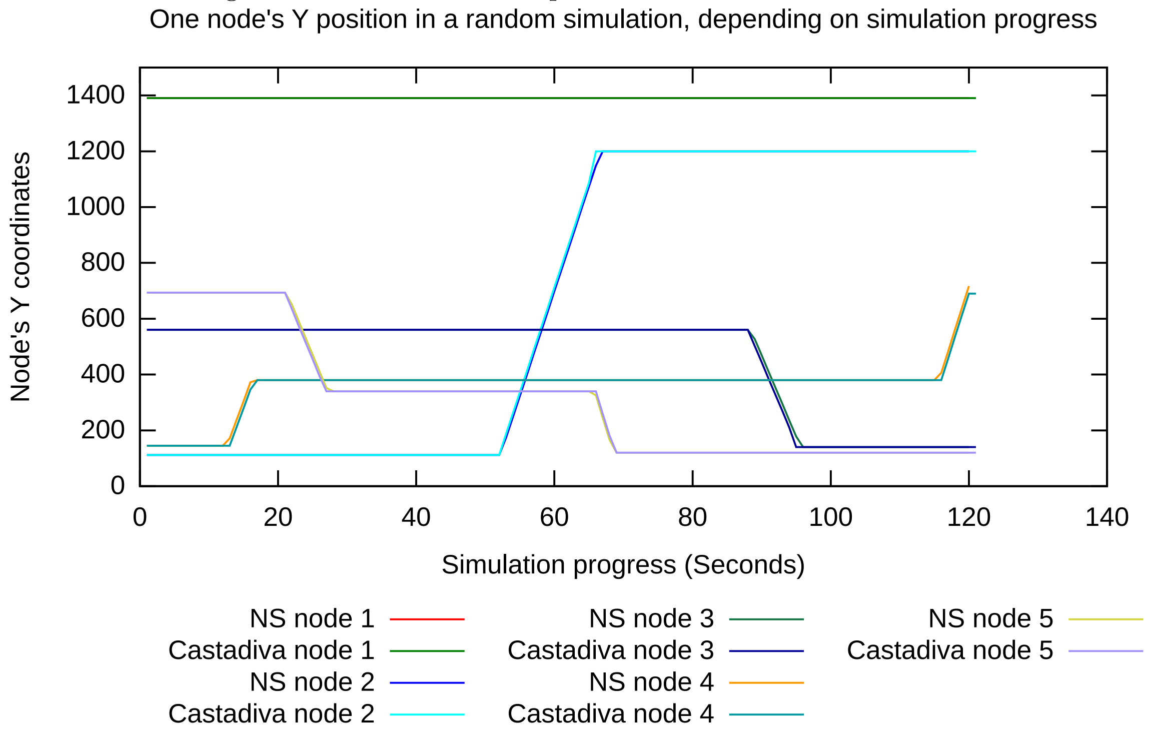


Figure 6.15.: One node's Y position in a random simulation

6.2.5. Minor issues in Castadiva

This section contains a list of issues I encountered in *Castadiva*. Those issues are minor issues that are not directly related with the parts of the program I worked on. I fixed some of those issues (6.2.5.1) but had no time to fix all of them (6.2.5.2).

6.2.5.1. Fixed issues

1. When the *Execution Planner*, the *Routing protocol designer*, the *Simulation* or the *Mobility plugin designer* windows where closed using the upper right cross, the whole program was exited with no warning.
2. The path to the *NFS* folder was not loaded when a scenario was loaded. As a result, if a non default path was used, it had to be reconfigured manually after each load.
3. The *Net device* value in the *Computer configuration* window was not updated after a new scenario was loaded.
4. The *Stop simulation* button was no longer compatible with mobility. I thus replaced it with a *Reset Access-Points* which should send a reboot command to all the *Access-Points*.
5. The buttons *Import Citymob* and *Import NS-2* in the execution planner had no associated code. They were removed.

6.2.5.2. Unfixed issues

1. The *Net device* value in the *Computer configuration* window not seem to be used in the program.
2. The export NS-2 function exports TCP traffic as UDP. TCP is not implemented in the code.

6.3. Important mobility issues in Castadiva

While trying to compare *Castadiva*'s results with *ns-2*'s results for data transfers with mobility, none of my comparisons were conclusive. During my last days investigating the possible origin of the problem, I found an error with mobility in *Castadiva* (6.3.1). I also found an issue with *ARP* that can be problematic in certain situations (6.3.2). Unfortunately, I had no sufficient time to fix those problems and this should probably be done as soon as possible as the problem also probably affects the current release of *Castadiva*.

6.3.1. Bad throughput calculation

Originally, *Castadiva* was designed for static scenarios. And so were *Castadiva*'s client and server programs that are used on the *Access-Points* to generate *TCP* or *UDP* traffic flows. See *Protocol Binaries* in figure 4.2 and section 6.1.3 for more information about those programs.

We are now going to detail how those programs calculate throughput, especially how they calculate the time for the throughput calculation. Please note that the following explanations might be difficult to understand. They are written for someone who already knows about *Castadiva*'s implementation. The two points at the end of this section shows the consequences of such an implementation with mobility.

With TCP, the server program is started from the very beginning of the simulation. The parameters for the server programs are

```
use /castadiva/nfs/bin/TcpFlowServerMIPS <Port> <secs> <times>
```

secs represents the time that the server should wait for connections before it exits. For example, for a simulation of 120 seconds, if we have a transfer from second 10 to 60 then *secs == 120*.
The client program is started exactly when the data transfer should start. It will try to connect to the server and then transmit for the determined time. In our example it is started at second 10 and will transfers for *60-10=50* seconds. When the server receives the first connection from the client, it will record the transfer start time and continue receiving as long as he has not received for *secs* seconds. **To calculate throughput, the server will use the time between the first and the last received packets**.

With UDP, the server program is also started from the very beginning of the simulation. The parameters for the server program are

```
use /castadiva/nfs/bin/UdpFlowServerMIPS <Port> <waiting secs> <
    receiving seconds> <times>
```

Where *waiting secs* indicates how long the server should wait for connections and *receiving seconds* shows how long the reception should last. The client programs starts simultaneously with the transfer.
The client program transfers for the desired time and then exits. As it is UDP, it wont try to get any confirmation from the server. When the server receives its first packet, it starts counting for *receiving seconds* before it shuts down.

Now, there is a problem when both those server programs are used with mobility. As it is possible that two nodes get in range and out of range at any time, two problems can arise.

1. If nodes are out of range when the transfer starts. In a static simulation, they remain out of range for the whole simulation. With a mobile simulation, they might get in range later in the simulation. If they do so, Servers do only start counting transfer time from the moment the first packet is received. For example, if a simulation lasts 120 seconds and if there is a transfer among two nodes from second 10 to second 110. If the nodes are out of range until second 109, they will start transferring for 100 seconds at second 109 and exceed the simulation time of (109+100)-110 = 99 seconds.
   With UDP, as the client do not know about the reception of its packets, it wont transfer more than 100 seconds. During the 99 seconds out of the simulation time, nothing will be transmitted. Results should thus be correct, but this generates a useless waiting time.
   With TCP, the client will need to connect to the server before it can transfer its data, and it will start counting down its transfer time when it gets its first connection with the server. As a result, during the 99 seconds out of the simulation, TCP will continue transferring data. It is not to mention that visibility is reset after the simulation time. This causes wrong results.

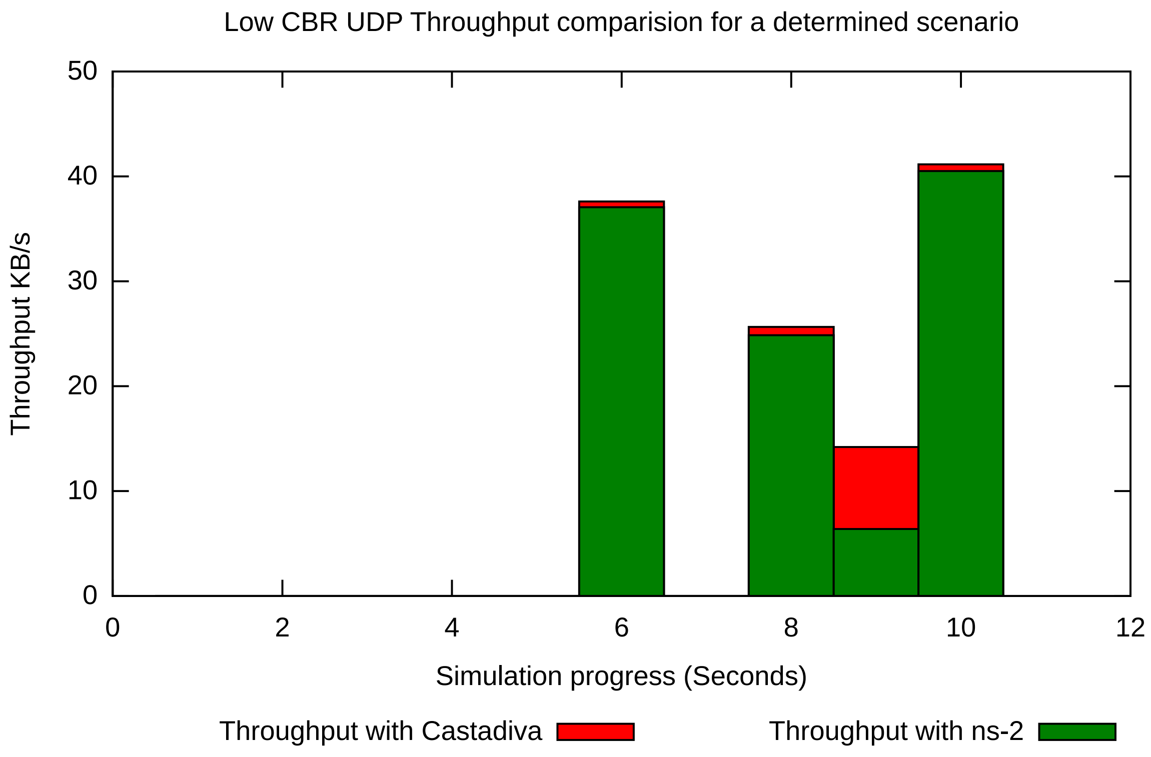


Figure 6.16.: Low *CBR UDP* throughput for a mobile scenario in *Castadiva* ans *ns-2*

2. With *TCP*, If a transfer is planned for 100 seconds and if the nodes get in range for only 2 seconds at second 98, the throughput will be calculated over 2 seconds (This is not taking in account problem 1). If another transfer is planned for the same 100 second and if the nodes get in range for the two first seconds, the throughput will be calculated over 100 second. As a result, two nodes that transfer the same amount of data over the same time get very different results. The nodes that first enter in contact are disadvantaged for TCP throughput calculation.

I will now present my last validation results that lead me to the present conclusions. In figure6.16 we have a comparison of a low *CBR* (*Constant Bit Rate*) scenario in *Castadiva* and *ns-2*. It is the same scenario imported from *CityMob* and commented in figures 6.14 and 6.15 on page 43 in the *CityMob Import* validation (see section6.2.4.4). In that scenario, the following transfers were configured: 1-2, 1-3, 1-4, 1-5, 2-3, 2-4, 2-5, 3-4, 3-5, 4-5 from second 10 to 110. In figure 6.16, they are numbered from 1 to 10 in the same order.

The first observation that we can make is that visibility is correct. When *ns-2* can transfer, *Castadiva* can also transfer. Figure 6.17 shows the distance between the different nodes, still for the same simulation. For readability, only nodes that get in range a least once in the simulation are shown. This confirms correct mobility management in *Castadiva* retarding visibility as the higher throughput are associated with the nodes that spend more time in range.

6.3.2. Bad ARP emulation with TCP

Castadiva's visibility rules do not block *ARP_REQUEST*s. Therefore, two theoretically out-of-range nodes do constantly know their respective *MAC* addresses even-tough they can not communicate at *layer 3*. In *ns-2* if there is no special routing protocol enabled

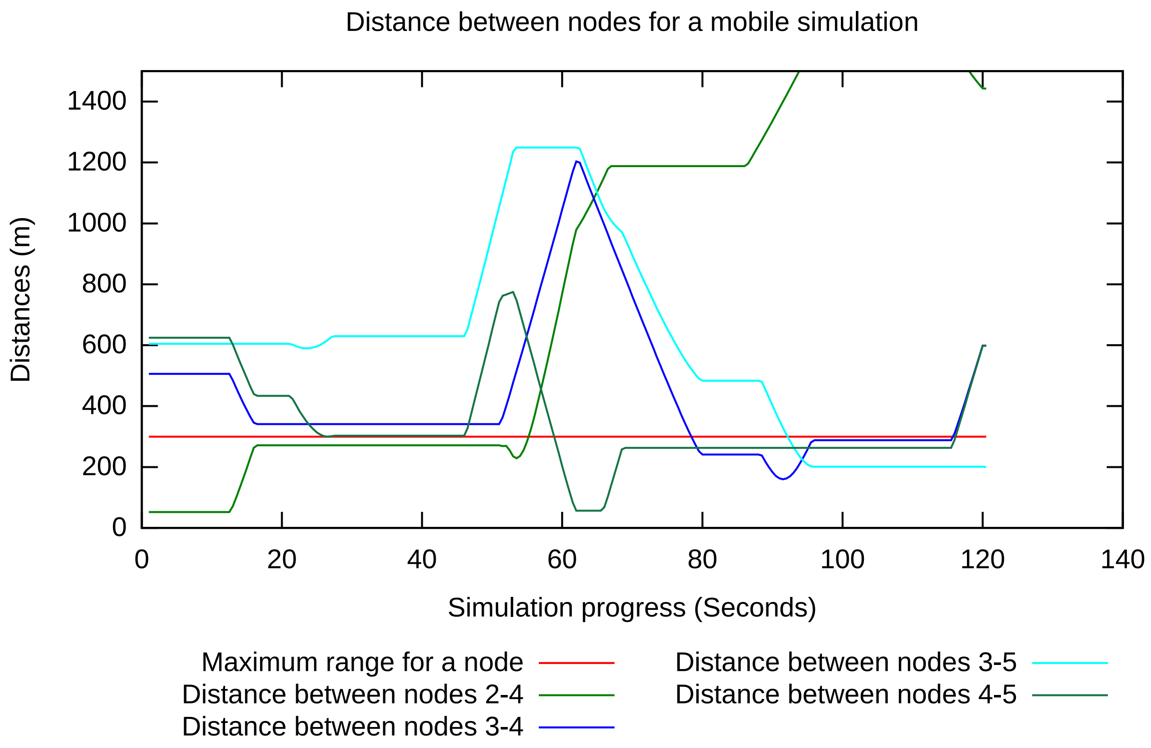


Figure 6.17.: Distance between nodes for a mobile scenario.

and if a node receives the command to transfer some data to another out-of-range node, it will make an *ARP_REQUEST* and wont get any response. After a short time, it will make another *ARP_REQUEST*. As long as the *ARP_REQUEST* gets no response, the waiting time increases. If two nodes get in range later in the simulation and if the client is still waiting for a reply to his *ARP_REQUEST*, transfer wont start before the next *ARP_REQUEST*. On the contrary, in *Castadiva*, it starts immediately. The difference of time can lead to a difference of up to 128 seconds[14].

Important note: With *UDP*, *ns-2* keeps sending *ARP* requests as long as it gets no answer.

6.4. Maximal UDP bit rate issue

As explained in section 7.2.3 on page 51, one of my first concerns to compare *Castadiva* with *ns-2* was to setup *ns-2* so that it behaves similarly to my *802.11g* network. Therefore, I made a speed comparison between *Castadiva* ans *ns-2* with *UDP Constant Bit Rate* traffic. Results can be found in figure 6.18 on the next page.

When I presented my first results to the group, some commented the very low transfer rate of *Castadiva*. Later on, *Alvaro* and I investigated the possible reasons of such a speed difference. We finally found out that there probably was an error with the *OpenWRT* router's drivers and *UDP*. Let us now describe the different tests that led us to such a conclusion.

1. Our first reaction to that low throughput was interference, probably due to the

[14]The delay is depending on the *ARP_PERSISTENCE* parameter. The default delay is 31 seconds. See `http://www.rabbit.com/documentation/docs/manuals/tcpip/usersmanualv1/arp_dns.html`.

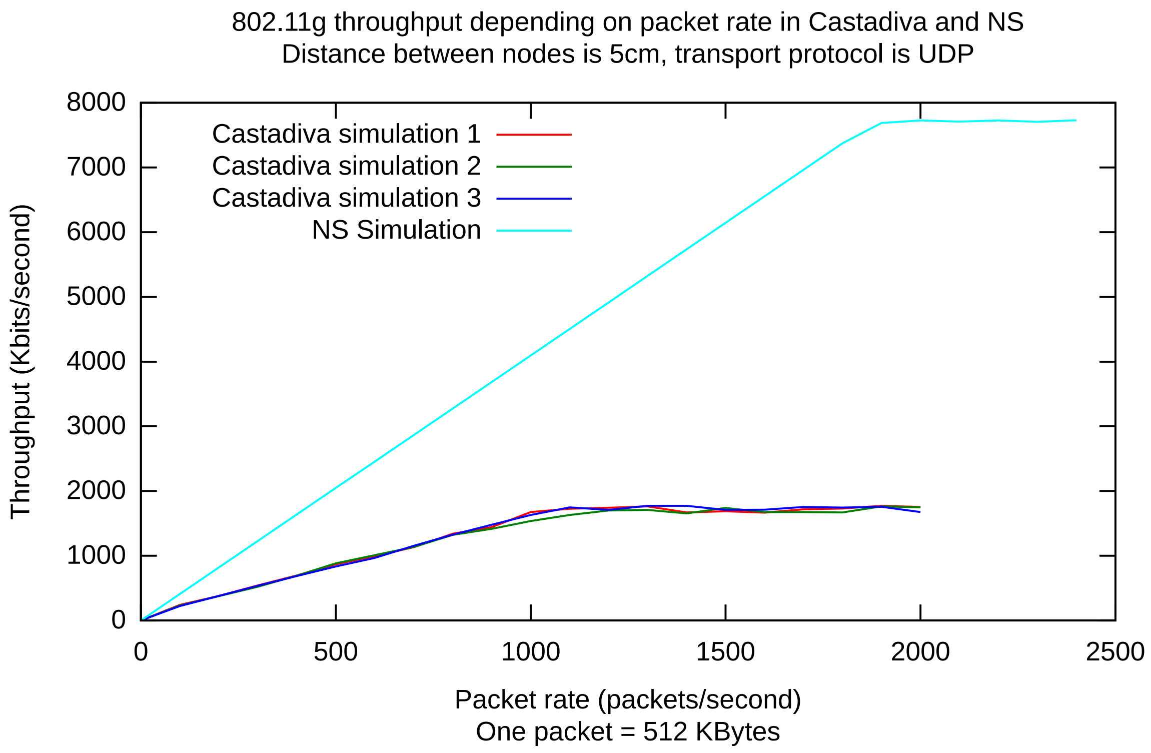


Figure 6.18.: UDP CBR Throughput comparison between *ns-2* and *Castadiva*

election of a bad channel. But *TCP* transfers do reach much higher speeds, we reached up to 8*Mbps* with *TCP* instead-of *UDP*.

2. In Alvaro's configuration, with *EEEPCs* instead of routers, everything is fine.

3. We then tried to connect the routers with *Ethernet* and ran the same *UDP* simulation. We reached around 25Mbps, which is acceptable.

4. In order to know if the problem came from the *UDP* client software, we used an *EEEPC* as a client to generate the packets. This was suggested by *Carlos*. Those packets where then re-routed over a wireless link. We got only 2Mbps throughput. We did the same with an EEEPC as a server connected to the second router. Again, we got only 2*Mbps* with *UDP*.

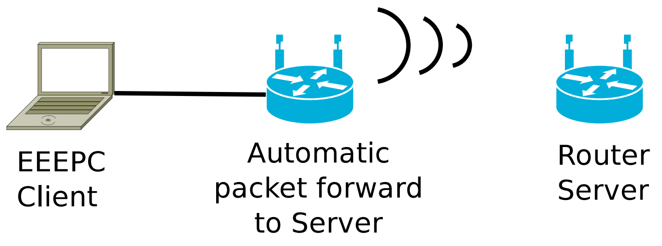


Figure 6.19.: Test architecture for *UDP* routing on wireless routers

5. We found out that the processor of client routers do rapidly saturate with *UDP* transfers. Optimizing the client source code and optimizing the compilation, we got

a slightly better throughput with *UDP*. But still much lower than *TCP*'s throughput and processor saturation.

As a conclusion, it seems that *LinkSys WRT54GL* routers have problems handling wireless *UDP* packets. At least with *OpenWRT*.

6.5. Writing a research Article

It was an unsuspected task. On April the 20th, I was asked to participate in a research article, in collaboration with *Alvaro Torres*. The article comments the new functionalities in *Castadiva* and it is supposed to be presented during the *CEDI 2010* congress in September[15]. A copy of the final article can be found in appendix B.

As it was a relatively urgent task, I partly suspended my previous work to dedicate about one week to the writing of the article[16]. In fact, writing was probably just a minor task, even-though we had to write in English. What really gathered my efforts was to get a clear idea of *Castadiva*, my work and what I was going to write about. I think it was an interesting approach that forced me to reenforce the basements of my knowledge about *Castadiva*.

I finally wrote the following sections of the article:

3 Castadiva which is a presentation of the test-bet.

4.3 Plugin system comments the routing and mobility plugin systems.

When we got a first version of what the article could be, we submitted our draft to *Carlos T. Calafate*, *Juan-Carlos Cano* and *Pietro Manzoni*. Their revision was very helpful, they corrected many grammatical errors but they also pointed out some semantic errors. For example, we should not name "*Castadiva*" before we describe it. This might of course seem obvious but it is not that simple to correctly explain something you have been working on for months. It thus was an interesting exercise and I tried to remember of it for this report. I also remembered *LyX*[17], a LaTeXedition tool that I discovered for the occasion and that I appreciated a lot.

[15]Our article is part of the section *XXI Jornadas de Paralelismo*. See the following URL for further details: `http://cedi2005.ugr.es/2010/contenido.php?apartado=actividades&sub=simposios`.

[16]See figure 3.1 on page 13 for time management details

[17]See `www.lyx.org`

7. Useful software

7.1. Netbeans

Netbeans is an *IDE* (*Integrated Development Environment*) which supports *JAVA*, the language *Castadiva*'s *Core*[1] is written with. It is a very efficient tool which really empowers the programming process. It integrates advanced functionalities, the ones I used most are

- Code completion based on the imported classes.
- Search for all the usages of a function or variable in the project.
- Search for the declaration of a function or variable in the project.
- Support of *Subversion*, a version management system to enable effective collaborative work.
- Live error correction.
- *WYSIWYG*[2] graphical user interface editor.
- Automated variable and function renaming.

For example, when a I had to investigate a new functionality, I systematically searched the corresponding code using the *Find usages* or *Go to source* options. When the code was written in Spanish, I could simply rename variables and functions in the whole project. Having a list of available functions for a variable is very handful to handle very large classes.

The subversion system that was set up for *Castadiva* was also very well integrated in *Netbeans*. It was easy to compare versions of the programs, keep track of all the changes and release new versions of the software.

7.2. Network Simulator 2 (NS-2)

7.2.1. Introduction

The Network Simulator - ns-2 is developed by various researchers and institutions. It is frequently used in network research and was appropriate, in our situation, to compare *Castadiva*'s results to other results. In this section, we wont describe all of *ns-2*'s possibilities but review use-full tips for ns-2 with *Castadiva*, MANET's and routing in general.

[1]See section 4.2 to learn more about the *Core* in *Castadiva*.
[2]*What You See Is What You Get*

7.2.2. Learning about ns-2

There are many tutorials dealing with any aspect of *ns-2* spread all over the Internet. For example, I recommend *Marc Greis's tutorial*[3] which offers an introduction to ns-2's basics and also teaches the basics of wireless simulation with *ns-2*. In general, I would recommend to focus on this[4] page to encounter information about *ns-2*. As there are several versions of *ns-2*, I rarely succeeded in finding appropriate information using any search engine. I thus recommend to use as much as possible the *NS Manual*[5] in order to find precise and probably version-related information.

7.2.3. Simulating 802.11g with ns-2

In order to validate results that were generated using *802.11g* routers, a first step is to configure ns-2 with the correct parameters to simulate *802.11g*. Surprisingly, I did not find any set of parameters to achieve this on the Internet. I finally investigated two different options to solve this problem.

1. *Johann M. Márquez Barja*[6], also working in the network laboratory, implemented *802.11g* for his own work. See appendix C for more detail. I used his implementation as a basis of my own configuration file that does not support distance losses, and thus closer to *Castadiva*. See appendix E on page 77 for the complete configuration file.

2. Later on a new version of *ns-2* (*ns-2.34*) was released. One of the important improvements in that version is the support of packet loss due to interferences on wireless links. Another improvement is the native support of *802.11g*. Fortunately, one of the example scripts (*adhoc_tcp.tcl*) that can be found in the *ns-allinone-2.34/dei80211mr-1.1.4/samples* folder precisely simulates an *Ad-Hoc 802.11g* network. That script inspired my final ns-2 settings. Those settings can be found in appendix F on page 80.

7.2.4. Setting ranges with ns-2

In *Castadiva*, ranges can be configured through the user interface (See appendix A on page 59). If the distance among two nodes is lower or equal to their range, they can communicate without any distance-related loss. If the distance is greater, then *Access-Points* can not communicate at all.

In order to set the range of a signal in *ns-2*, the *threshold.cc*[7] file, located at *~ns/indep-utils/propagation/threshold.cc* can be compiled and used as follows:

```
./threshold -m TwoRayGround -fr 2400000000 -ht 1 -hr 1 300
distance = 300 propagation model: TwoRayGround
Selected parameters:
 transmit power: 1.58489
 frequency: 2.4e+09
 transmit antenna gain: 1
 receive antenna gain: 1
 system loss: 1
 transmit antenna height: 1
```

[3]See: http://www.isi.edu/nsnam/ns/tutorial/index.html
[4]See: http://www.isi.edu/nsnam/ns/
[5]See: http://www.isi.edu/nsnam/ns/ns-documentation.html
[6]See: http://www.marquez-barja.com/
[7]See: http://www.isi.edu/nsnam/ns/doc/node223.html

```
 receive antenna height: 1

Receiving threshold RXThresh_ is: 1.95665e-10
```

I made a few experiments with nodes separated with 300, 299.9 or 301 to confirm that result.

7.2.5. Analyzing ns-2 results

I first used a method from *Evan Jones's blog*[8] to collect *UDP* simulation results. It consists in recursively running a function with a fixed interval of time and to write temporary results to a text file. This was quite efficient for *UDP* and other data like distances between nodes or node's position but I could not translate the technique to *TCP*. Carlos Miguel Calafate is the one who taught me an interesting method based on *ns-2*'s default output files and simple *Script Shell* scripts. For example, the following command:

```
cat simple.tr | grep ^r | grep "Md␣4" | grep "Ms␣2" | grep cbr | wc -l
```

Will return the number of *UDP CBR* packets that were transmitted from node 2 to node 4. There are, of course, many other possibilities[9] and this method allows a deeper analysis and better understanding.

7.2.6. Castadiva and ns-2

Castadiva offers two *ns-2* related functions*:*

- *import NS-2* was probably meant to offer an *Export to Castadiva* functionality in other network related software. As an input, it takes a very precisely formatted file. Any space should be respected and a special header is also requested.

  ```
  #
  # nodes: 5, pause: 0.0, max speed: 0.0, max x = 1500.0, max y: 1500.0
  #
  ```

- *export NS-2* function is supposed to allow traffic ans scenario export for *ns-2*. For now, *TCP* traffic is exported as *UDP*. See Unfixed issue 2 on page 44.

Nevertheless, the export function is very interesting when the simulation has many nodes with complicated movements.

7.3. Gnuplot

Gnuplot[10] is a powerful tool that allows to generate any kind of graphical charts. For my validation work, I often had to compare the results of a *Castadiva* simulation with the results of a *ns-2* simulation. Some of those charts can be viewed in this report, for example Figures 6.12 and 6.14. Those figures were exported directly in vector graphical format (*SVG*), which offers many advantages as an excellent image quality, a light weight and infinite scalability.

As for *ns-2*, *Gnuplot* changes a lot along versions and it is not always easy to find the adequate information. But the Gnuplot user manual[11] has an index which makes it quite

[8] *See:* *http://evanjones.ca/basic-80211-stats.html*
[9] The following legend of ns-2 output format might be helpful: http://nsnam.isi.edu/nsnam/index.php/NS-2_Trace_Formats
[10] *See* *www.gnuplot.info*
[11] See http://www.gnuplot.info/documentation.html

easy to use. As an example of how to generate a chart with Gnuplot, you can refer to appendix G on page 83 which is the source file of figure 6.14.

7.4. LaTeX

I discovered LaTeX during this internship. I first used it to write the *Routing Plugin System*'s user manual. LaTeX has the advantage of producing high quality documents as it respects many typographical rules which are also effective for professional edition.

Later on, I discovered *LyX*, a Graphical User Interface for LaTeX which makes LaTeX writing much more productive, even-tough it probably also introduces some limits.

I used LaTeX with *LyX* for this report and got fully satisfied. LyX handles many graphical file formats which allows to insert vectorized drawings directly into a *pdf* document. It offers to generate fully operational *pdf* documents with *hyperlink* and index management. Finally, LaTeX also handles appendices with great effectiveness.

8. Conclusion

I did not achieve all of the tasks I was given for this internship. At least, it did not succeed in positively validating all of *Castadiva*'s new options. When we defined my work, we did not know exactly how hard it would be. It finally came up to be much more difficult than expected: the functionalities that I had to validate were in a beta stage. On one hand, the code was chaotic, sometimes Spanish written, not commented and contained errors, omissions. On the other hand, the previous designs were incomplete as they did not take in account all parts of the program. For sure, It is not an easy task to enable mobility in *Castadiva*, which was initially designed without mobility. The complexity makes it quite difficult and long to assimilate a new functionality. It is also hard to find the origin of an error which can be as vague as wrong results. Nevertheless, regarding the task I did not entirely validate, I believe I made an important progress and I probably identified the source of most of the remaining errors. I hope someone will continue my work and therefore, I wrote this report with as much detail I could.

My experience as an trainee in the *GRC* group and in the *UPV* in general is definitively positive. Even-though I had some hard times desperately looking for an error, I enjoyed most of the hours I spent in the laboratory. The people I met there are nice, friendly and ready to help. This is probably also an important aspect when it comes to one's professional life. Regarding knowledge and competences, I probably augmented my analysis abilities as well as the ability to progressively handle a large problem. Of course, I dealt a lot with *JAVA*, *Linux* and *ns-2* but I also increased my general computer network knowledge as well as... my command of the Spanish language.

9. Thanks

There are a few persons I would like to thank for their help with this internship project.

Pietro Manzoni (Coordinator of the *GRC*) For being my internship director and for offering me a job in the *Computer Network Lab* at the *UPV*.

André Aoun (Director of *STRI*) For his voluntary support all along my project, from it's very beginning to it's last days.

Marta Caballero (Exchange Coordinator at the *UPV*) For her patience an her support with administrative tasks.

Carlos Miguel Calafate (Professor at the *UPV*) for his pertinent technical advice.

Alvaro Torres & Jorge Hortelano (Master and *PhD* students in the *GRC*)for their patience an their answers to many of my questions about *Castadiva*.

Index

Bibliography

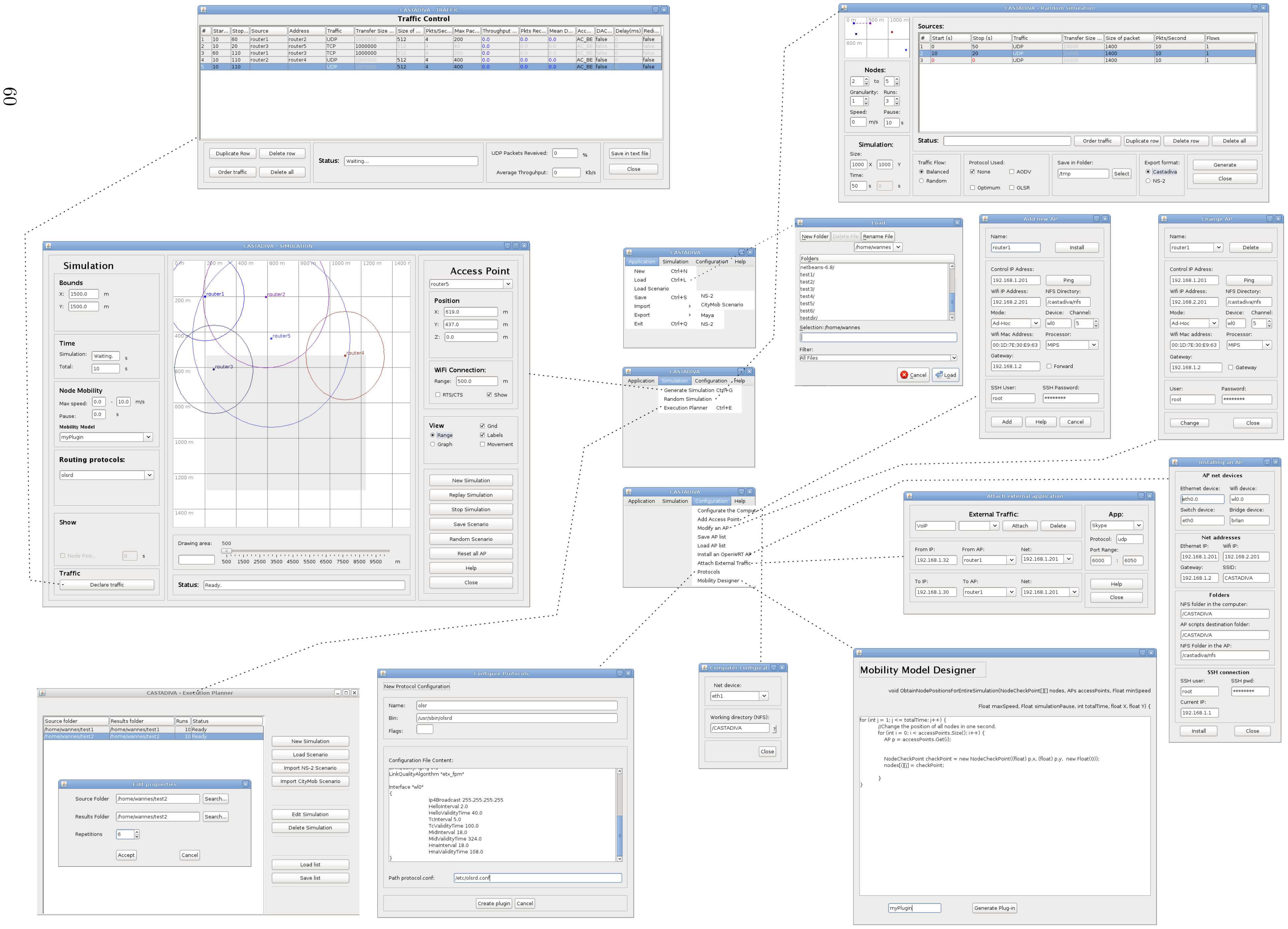
[1] Jorge Hortelano, Juan-Carlos Cano, Carlos T. Calafate, and Pietro Manzoni. Testing applications in manet environments through emulation. *EURASIP Journal on Wireless Communications and Networking, vol. 2009, Article ID 406979, 20 pages, 2009. doi:10.1155/2009/406979.*

[2] Wannes Vossen, Alvaro Torres, Jorge Hortelano, Carlos T. Calafate, Juan-Carlos Cano, and Pietro Manzoni. Extending an emulation platform for automatized and distributed evaluation of qos in manets. April 2010.

Part III.

Appendices

A. Castadiva's GUI splash view



CASTADIVA - TRAFFIC

Traffic Control

#	Star...	Stop...	Source	Address	Traffic	Transfer Size ...	Size of ...	Pkts/Sec...	Max Pac...	Throughput ...	Pkts Rec...	Mean D...	Acc...	DAC...	Delay(ms)	Redi...
1	10	60	router1	router2	UDP	1000000	512	4	200	0.0	0.0	0.0	AC_BE	false	0	false
2	10	20	router3	router5	TCP	1000000	512	4	40	0.0	0.0	0.0	AC_BE	false	0	false
3	60	110	router1	router3	TCP	1000000	512	4	200	0.0	0.0	0.0	AC_BE	false	0	false
4	10	110	router2	router4	UDP	1000000	512	4	400	0.0	0.0	0.0	AC_BE	false	0	false
5	10	110			UDP	1000000	512	4	400	0.0	0.0	0.0	AC_BE	false	0	false

Status: Waiting...
UDP Packets Reveived: 0 %
Average Throguhput: 0 Kb/s

CASTADIVA - Random simulation

Sources:

#	Start (s)	Stop (s)	Traffic	Transfer Size ...	Size of packet	Pkts/Second	Flows
1	0	50	UDP	10000	1400	10	1
2	10	20	UDP	10000	1400	10	1
3	0	0	UDP	10000	1400	10	1

Nodes: 2 to 5
Granularity: 1 Runs: 3
Speed: 0 m/s Pause: 10 s
Simulation: Size: 1000 X 1000 Y
Time: 50 s 0 s
Traffic Flow: [✓ Balanced] Random
Protocol Used: [✓ None] AODV Optimum OLSR
Save in Folder: /tmp
Export format: [✓ Castadiva] NS-2

CASTADIVA - SIMULATION

Simulation
Bounds X: 1500.0 m Y: 1500.0 m
Time Simulation: Waiting... s Total: 10 s
Node Mobility Max speed: 0.0 - 10.0 m/s Pause: 0.0 s
Mobility Model: myPlugin
Routing protocols: olsrd
Access Point: router5
Position X: 619.0 m Y: 437.0 m Z: 0.0 m
WiFi Connection: Range: 500.0 m RTS/CTS [✓ Show]
View [✓ Range] Graph [✓ Grid] [✓ Labels] Movement
Drawing area: 500
Status: Ready.

Load
Selection: /home/wannes
Folders: netbeans-6.8/ test1/ test2/ test3/ test4/ test5/ test6/ testdir/
Filter: All Files

Add new AP
Name: router1
Control IP Adress: 192.168.1.201
Wifi IP Address: 192.168.2.201 NFS Directory: /castadiva/nfs
Mode: Ad-Hoc Device: wl0 Channel: 5
Wifi Mac Address: 00:1D:7E:30:E9:63 Processor: MIPS
Gateway: 192.168.1.2 Forward
SSH User: root SSH Password: ********

Change AP
Name: router1
Control IP Adress: 192.168.1.201
Wifi IP Address: 192.168.2.201 NFS Directory: /castadiva/nfs
Mode: Ad-Hoc Device: wl0 Channel: 5
Wifi Mac address: 00:1D:7E:30:E9:63 Processor: MIPS
Gateway: 192.168.1.2 Gateway
User: root Password: ********

Installing an AP
AP net devices: Ethernet device: eth0.0 Wifi device: wl0.0
Switch device: eth0 Bridge device: brlan
Net addresses: Ethernet IP: 192.168.1.201 Wifi IP: 192.168.2.201
Gateway: 192.168.1.2 SSID: CASTADIVA
Folders: NFS folder in the computer: /CASTADIVA
AP scripts destination folder: /CASTADIVA
NFS Folder in the AP: /castadiva/nfs
SSH connection: SSH user: root SSH pwd: ********
Current IP: 192.168.1.1

Attach external application
External Traffic: VoIP
App: Skype Protocol: udp Port Range: 6000 : 6050
From IP: 192.168.1.32 From AP: router1 Net: 192.168.1.201
To IP: 192.168.1.30 To AP: router1 Net: 192.168.1.201

CASTADIVA - Execution Planner

Source folder	Results folder	Runs	Status
/home/wannes/test1	/home/wannes/test1	10	Ready
/home/wannes/test2	/home/wannes/test2	10	Ready

Edit properties
Source Folder: /home/wannes/test2
Results Folder: /home/wannes/test2
Repetitions: 6

Configure Protocols
Name: olsr
Bin: /usr/sbin/olsrd
Flags:
Configuration File Content:

```
LinkQualityAlgorithm "etx_fpm"

Interface "wl0"
{
            Ip4Broadcast 255.255.255.255
            HelloInterval 2.0
            HelloValidityTime 40.0
            TcInterval 5.0
            TcValidityTime 100.0
            MidInterval 18.0
            MidValidityTime 324.0
            HnaInterval 18.0
            HnaValidityTime 108.0
}
```

Path protocol.conf: /etc/olsrd.conf

Computer configuration
Net device: eth1
Working directory (NFS): /CASTADIVA

Mobility Model Designer

void ObtainNodePositionsForEntireSimulation(NodeCheckPoint[][] nodes, APs accessPoints, Float minSpeed
Float maxSpeed, Float simulationPause, int totalTime, float X, float Y) {

```
for (int j = 1; j <= totalTime; j++) {
      //Change the position of all nodes in one second.
      for (int i = 0; i < accessPoints.Size(); i++) {
        AP p = accessPoints.Get(i);


        NodeCheckPoint checkPoint = new NodeCheckPoint((float) p.x, (float) p.y,  new Float(0));
        nodes[i][j] = checkPoint;

      }
}
```

myPlugin

B. Extending an emulation platform for automatized and distributed evaluation of QoS in MANETs [2]

Extending an emulation platform for automatized and distributed evaluation of QoS in MANETs

Wannes Vossen, Alvaro Torres, Jorge Hortelano,
Carlos T. Calafate, Juan-Carlos Cano and Pietro Manzoni

Department of Computer Engineering

Universidad Politécnica de Valencia

Camino de Vera, S/N, 46022 Valencia, Spain

wanvos@posgrado.upv.es, atcortes@batousay.com, jorgehortelano@gmail.com

{calafate,jucano,pmanzoni}@disca.upv.es

Abstract

Nowadays Mobile Ad Hoc Network (MANET) testing is being done using simulation models, like ns-2, but such simulations tend to be too optimistic. In this work we extend Castadiva, a test-bed architecture that allows validating software solutions (both applications and network protocols) for real ad-hoc environments using low-cost, off-the-shelf devices and open source software. Our extensions are twofold: in the one hand we improve the platform to enhance support for real-time traffic. On the other hand we make distributed tests possible and fully automatic.

1 Introduction

Mobile ad-hoc networks (MANETs) are packet radio networks composed by independent and heterogeneous stations that cooperate in routing and packet forwarding tasks, conforming a dynamic multi-hop network.

All the nodes which are part of a MANET can act as end-points for data interchange or as routers when end-points are not in direct contact. The topology of the network changes dynamically as mobile nodes join or depart the network, or when radio links between nodes become unusable, and so networks can be deployed easily and cheaply by using efficient routing protocols to operate correctly. The importance of MANETs becomes more evident by noticing the wide application area that MANET scenarios embrace. Special situations require communication networks to be available without any previous infrastructure, like emergency missions, military operations or ad-hoc meetings.

Testing and evaluating protocols for MANETs is a mandatory request to guarantee its success in a real world application. Researchers in this field have three options for testing their MANET protocols: using simulation tools, using emulators, or using test-beds.

Emulators provide an attractive middle ground between pure simulators and wireless test-beds, allowing scalable and repeatable experimentation using real devices. In section 2 we review the different emulators available.

Castadiva [8] is one of those emulators, it is also the platform that we are extending in this paper. Further details can be found in section 3.

Also, the proliferation of devices with multimedia and wireless networking capabilities increase the demand of audiovisual communications among MANET nodes, requiring certain levels of QoS. To meet this need, the IEEE 802.11e [7] working group has enhanced the IEEE 802.11 [6] MAC to provide QoS at the MAC layer. So, in order to meet these new requirements we have updated Castadiva to be

QoS capable.

The rest of this work is organized as follows: a few related works are presented in section 2. Section 3 briefly describes the original Castadiva architecture and features. In section 4 we present the different extensions made to Castadiva to offer QoS support. Finally, in section 5 we present some concluding remarks.

2 Related work

The idea of creating an emulator for MANET environments is not new and, in fact, several platforms for generating real ad hoc network experiments have been proposed and can be found in the literature.

We will only focus on those platforms that, similarly to Castadiva operate with real code, since our main purpose is to test and validate real code.

ORBIT is an indoor radio grid emulator for controlled experimentation and an outdoor field trial network for end-user evaluations in real-world settings [4]. This emulator needs an expensive noise generator since it emulates high node distances by reducing the signal-to-noise ratio. It also requires investing a high budget to create the grid of nodes (each computer is a possible position of the node in a simulation), as well as extra support servers for data storage. Thus, deploying the entire infrastructure requires a lot of room.

Carnegie Mellon U. Wireless Emulator supports real devices, applications, MAC and PHY layers on a network-wide scale, while maintaining experimental control and repeatability [5]. The disadvantages of this emulator are that it does not use commercial off-the-shelf devices, using an FPGA for digital emulation instead.

MobiEmu an emulator to test an ad hoc network of any scale and with any mobility scenario without actually moving the adhoc nodes physically [10]. We discarded this emulator for our test since it relies on expensive clusters to emulate the scenario.

Taking the limitations of these platforms into consideration, Castadiva relies instead into a cheap architecture based on low-cost devices to generate a test-bed.

Castadiva outperforms the other emulators in terms of: (a) the variety of devices that can be used as nodes, (b) the initial budget needed to deploy the emulator and (c) the ease of use provided by it's GUI; this means that it makes a clear contribution to the research community.

To our knowledge none of the emulators mentioned above support QoS traffic.

3 Castadiva

Castadiva is a MANET test-bed platform wich provides a cost-effective alternative to simulation tools. This can be achieved by allowing certain critical components of a simulation to be real. For example, Castadiva relies on actual wireless communications, using IEEE 802.11 interfaces.

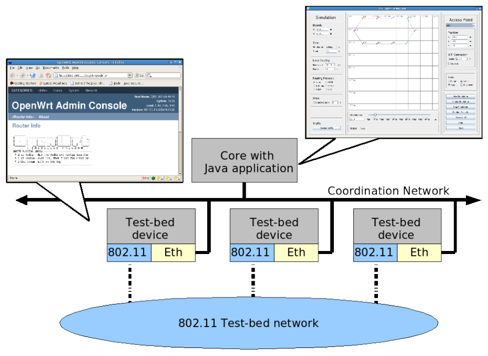


Figure 1: Castadiva architecture

Castadiva's architecture is shown in Figure 1. It is based on two major elements: (a) The core, which orchestrates the simulation and coordinates the Nodes; (b) The nodes which are able to communicate among themselves, for simulation purposes, using their IEEE 802.11 wireless interface. Communication among the core and the Nodes is performed over a typical Ethernet network. Eth-

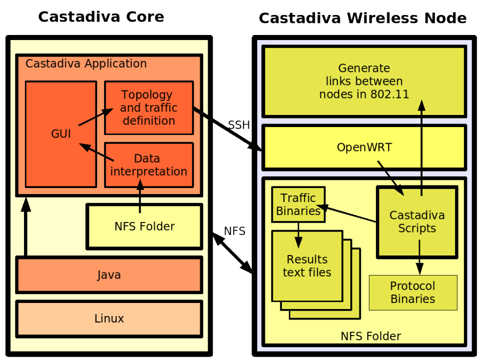


Figure 2: Software components for Castadiva

ernet allows reduced delays and guarantees no interferences with wireless signals.

As shown in figure 2, the Castadiva core is written with JAVA and can be run on a simple computer. The nodes, on their part, must rely on a Linux operating system. For example, Castadiva's development team used Wi-Fi routers[1] or even Netbooks [2]. Netbooks offer better hardware performances than routers and can be easily moved around.

Figure 2 offers a detailed overview over the protocols used in Castadiva. It represents the situation where the Core would be a Linux PC and the nodes are wireless routers using a Linux based operating system (OpenWRT). Since the Node devices may be very specialized (routers, PDAs, ...) and have limited storage resources, we use NFS to store and access any simulation content on the nodes. The NFS server must be located on the Core, allowing easy file sharing among the Castadiva software and every node. Finally, Secured Shell (SSH) is a good option to control the simultaneous beginning of a simulation on every nodes.

Figure 3 shows the graphical interface that was designed to easily control the previously described system. As we can see, it is possible to place each node on the canvas and to define its range. According to those ranges, Castadiva makes sure that two out-of-range nodes do not communicate, even if their real interfaces are able to do so. The graphical user interface also allows picking a routing protocol and mobility instructions. Those possibilities will be further described in this paper.

Finally, figure 4 is a capture of the traffic window. Traffic among nodes can be configured with a few options related to UDP or TCP, such as the UDP rate or the TCP file size to transfer.

4 Proposed improvements

In this section we explain all the new features added to Castadiva. These features include both QoS integration and automatization of testing parameters.

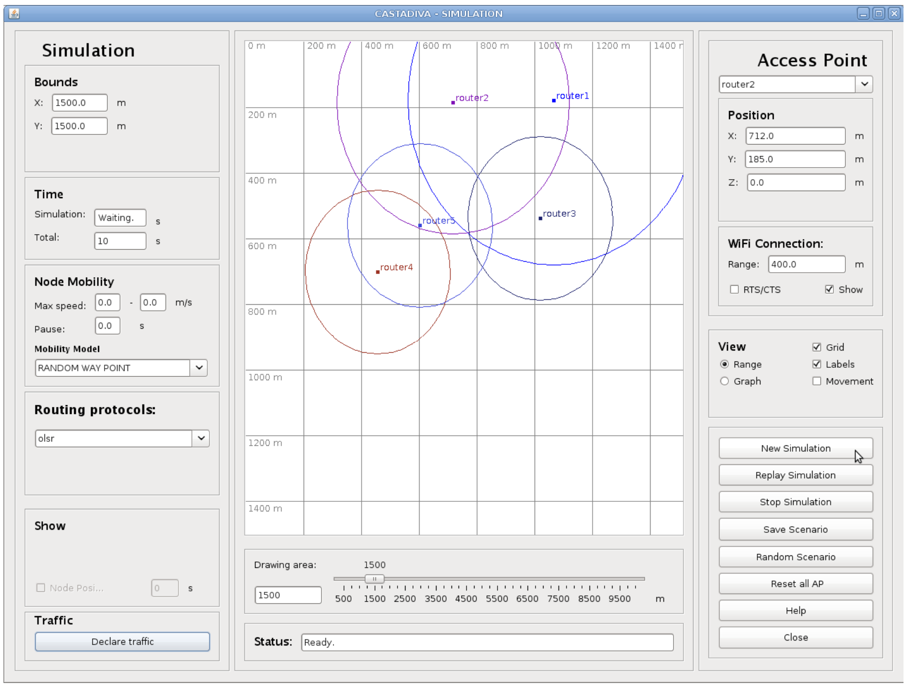
This section is divided in four parts, the first two parts describe how we added QoS support to Castadiva, and the other two describe the automatization and scheduling of a batch of tests. In particular, the first part explains how we included support for IEEE 802.11e; the second part explains how we managed to measure delay in UDP packets. Part three describes the new plugins system that allows switching between routing protocols and mobility models. Finally, in the fourth part, we describe the new execution planner and it's statistics collection system.

4.1 IEEE 802.11e support

IEEE 802.11e is an extension to the IEEE 802.11 standard which was released in 2005 to add QoS support. QoS is achieved by replacing the DCF function by EDCA (enhanced distributed channel access). With this new function (fully compatible with DCF) QoS is achieved through the introduction of different access categories (ACs), and their associated backoff entities.

Contrarily to the legacy IEEE 802.11 stations, where all packets have the same priority and are assigned to a single backoff entity, IEEE 802.11e stations have four backoff entities (one per AC) so that packets are sorted according to their priority. The different access categories available in IEEE 802.11e stations

[1]Linksys WRT54GL with the OpenWRT Linux based Operating system.
[2]Asus eee PC 901



Bounds
X: 1500.0 m
Y: 1500.0 m

Time
Simulation: Waiting. s
Total: 10 s

Node Mobility
Max speed: 0.0 - 0.0 m/s
Pause: 0.0 s
Mobility Model: RANDOM WAY POINT

Routing protocols: olsr

Access Point: router2
Position
X: 712.0 m
Y: 185.0 m
Z: 0.0 m

WiFi Connection:
Range: 400.0 m
☐ RTS/CTS ☑ Show

View
● Range ☑ Grid
○ Graph ☑ Labels
☐ Movement

Drawing area: 1500

Status: Ready.

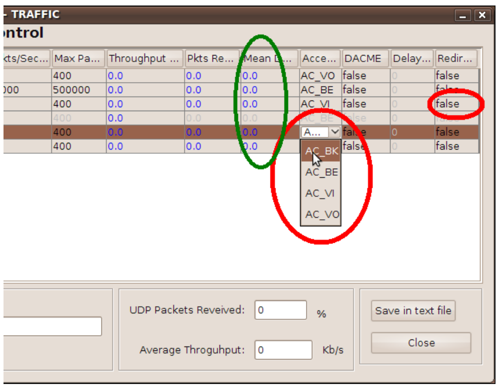
Figure 3: Simulation window

are: Voice (AC_VO), Video (AC_VI), Best-effort (AC_BE) and Background (AC_BK).

For applications to take advantage of the IEEE 802.11e technology, datagrams should have their IP Type of Service (TOS) header field set according to the desired user priority. When delivered to an IEEE 802.11e enabled wireless card driver, those datagrams will be handled according to the priority defined.

In Castadiva we have implemented the packet priority support by adding TOS information to UDP packets in our generator. Figure 5 shows how the user can select one of the four categories for each UDP flow.

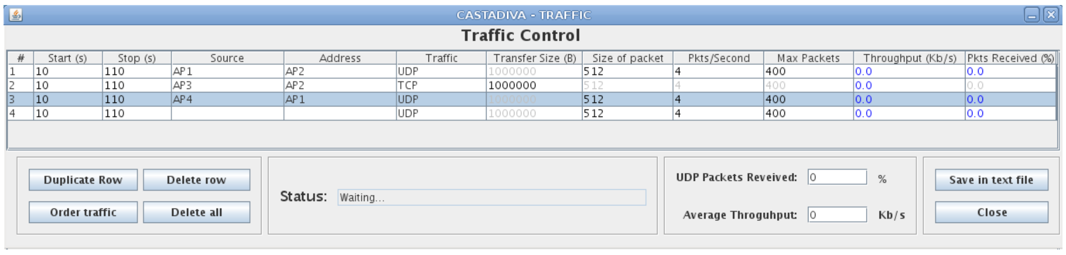
If the subjacent network is not IEEE 802.11e enabled, setting the TOS won't make any difference; however if the network is IEEE 802.11e enabled, traffic differentiation should be effective.



kts/Sec...	Max Pa...	Throughput ...	Pkts Re...	Mean D...	Acce...	DACME	Delay...	Redir...
	400	0.0	0.0	0.0	AC_VO	false	0	false
000	500000	0.0	0.0	0.0	AC_BE	false	0	false
	400	0.0	0.0	0.0	AC_VI	false	0	false
	400	0.0	0.0	0.0	AC_BE	false	0	false
	400	0.0	0.0	0.0	A...	false	0	false
	400	0.0	0.0	0.0		false	0	false

UDP Packets Reveived: 0 %
Average Throguhput: 0 Kb/s

Figure 5: IEEE 802.11e category selection and Delay measurement.



#	Start (s)	Stop (s)	Source	Address	Traffic	Transfer Size (B)	Size of packet	Pkts/Second	Max Packets	Throughput (Kb/s)	Pkts Received (%)
1	10	110	AP1	AP2	UDP	1000000	512	4	400	0.0	0.0
2	10	110	AP3	AP2	TCP	1000000	512	4	400	0.0	0.0
3	10	110	AP4	AP1	UDP	1000000	512	4	400	0.0	0.0
4	10	110			UDP	1000000	512	4	400	0.0	0.0

Status: Waiting...

UDP Packets Reveived: 0 %

Average Throguhput: 0 Kb/s

Figure 4: Traffic window

4.2 Delay measurement for UDP traffic

Measuring the delay of UDP packets is a problem which is not easy to solve in a real environment. Notice that in a simulation environment this is very easy to do, as we can determine when an UDP packet leaves the source and when it arrives to the destination; However, in a real environment, we have to deal with clock desynchronization, which significantly complicates the process.

To cope with this problem we implemented two different methods, one more precise and another just orientative. The basis of both methods is our UDP flow generator. At the source node we read the local time, and stamp it on each sent packet. On the sink application, again, we obtain the time, and then compare it with the packet's time-stamp.

The first method counters clock desynchronization by relying on ntp [1] at the beginning of each emulation. This method is just orientative since it is not able to solve the clock drift problem during tests. Additionally, the low margin of error required can not be achieved with ntp.

The second method we adopted was the solution proposed in [2]. It employs the control network (Fast-Ethernet based) used by Castadiva to redirect UDP packets back to the source; when using this method, both source and sink programs are on the same machine. This way we avoid clock desynchronization, but we introduce a new delay. This delay should be measured and taken into account when results are processed.

In terms of implementation and integration with Castadiva we can select the second method indepently for each flow by setting the "Redirect" flag to true, as we can see in figure 5.

4.3 Plugins system

The first version of Castadiva was fully functional, but only allowed using a single mobility model and three routing protocols. With the dynamic creation and loading of plugins we have managed to support almost any routing protocol and mobility model.

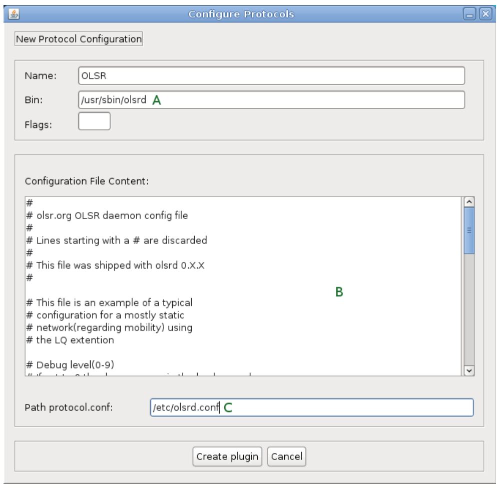
Routing protocols

Previous versions of Castadiva already supported a few common routing protocols. For example OLSR [9] and AODV [3].

For those protocols to be functional, they had to be previously installed and configured on each node. What Castadiva did, when a simulation started, was to activate the selected protocol, sending a simple command line instruction via Secure Shell (SSH). When the simulation ended, it was deactivated the same way.

However, a few constraints come up when that command line is hard-coded: what if it changes with different nodes or different operating system versions?

Therefore, a plugin system was developed and implemented in Castadiva. The idea is to allow the user to define the needed information to manage routing on the different nodes. Moreover, it offers the advantage of possibly designing a routing protocol plugin for any routing protocol supported by the nodes.



Name: OLSR

Bin: /usr/sbin/olsrd

Flags:

Configuration File Content:

```
#
# olsr.org OLSR daemon config file
#
# Lines starting with a # are discarded
#
# This file was shipped with olsrd 0.X.X
#

# This file is an example of a typical
# configuration for a mostly static
# network(regarding mobility) using
# the LQ extention

# Debug level(0-9)
```

Path protocol.conf: /etc/olsrd.conf

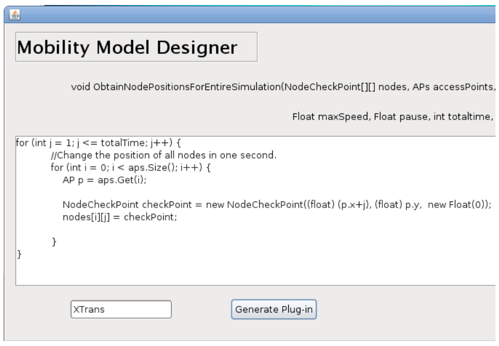
Figure 6: Routing plugin configuration window in Castadiva

Figure 6 shows how routing plugins can be configured in the current version of Castadiva. The command in text-field A is sent to each access point when the simulation starts, followed by eventual flags. All text entered in text-area B is saved as a file that is copied on the nodes at the path location stated in text-field C. All existing files are overwritten and the configuration file is removed after the simulation.

Finally, in order to find out how to stop the routing protocol, Castadiva will automatically find out the name of the binary file stated in A and use a *"killall binaryFileName"* instruction on each node.

Mobility

We also extended Castadiva's to define custom mobility models during a simulation. Through the graphical interface shown in figure 7, a user can take advantage of the JAVA programming language to calculate the position of each node at each second of the simulation. Given simulation information (Access Points list, minimum speed, ...), the user can write any desired code to meet the only requirement: fill in the NodeCheckPoint[i][j] array where i represents



```
void ObtainNodePositionsForEntireSimulation(NodeCheckPoint[][] nodes, APs accessPoints,
Float maxSpeed, Float pause, int totaltime,
```

```
for (int j = 1; j <= totalTime; j++) {
        //Change the position of all nodes in one second.
        for (int i = 0; i < aps.Size(); i++) {
           AP p = aps.Get(i);

           NodeCheckPoint checkPoint = new NodeCheckPoint((float) (p.x+j), (float) p.y,  new Float(0));
           nodes[i][j] = checkPoint;

        }
}
```

Figure 7: Mobility plugin configuration window

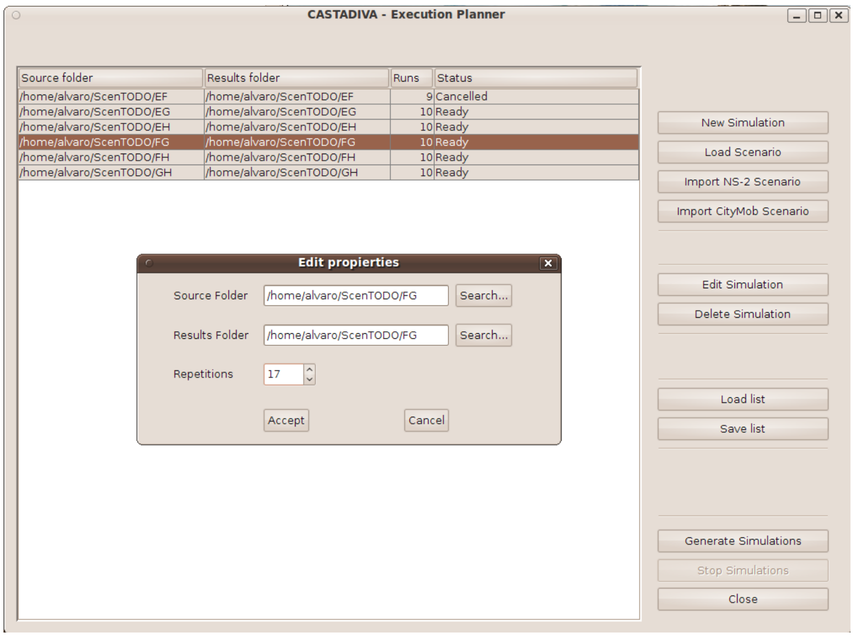
an access point and j a second in the simulation. Note that every access-point must have a defined position at any time of the simulation. If the code is correct, the plugin will be compiled and integrated directly into Castadiva, thus becoming available in the Simulation window's mobility plugin drop list (see Figure 3).

4.4 Execution planner

The first version of Castadiva had had the following drawbakcs: tests should be performed one-by-one, causing the creation of a huge set of experiments to be very inefficient. With this new feature we have managed to run up to 200 tests in a row, allowing us to collect and process lots of statistics.

The execution planner alone is very powerful, but if we combine it with the new plugin system (Section 4.3) we can automatize the experiment launching process, varying not only the traffic sources/destinations, but also the routing protocol or the mobility model. Thus, we consider that it is a good platform for evaluating ns-2 results and performing comparisons.

As we can see in figure 8, the GUI of the execution planner provides an easy way to create and automatize the MANET evaluation. It includes features like "Load list", which imports a list of scenarios previously saved with the "Save list" option. The main component of the window is a table that reflects the dif-


CASTADIVA - Execution Planner

Source folder	Results folder	Runs	Status
/home/alvaro/ScenTODO/EF	/home/alvaro/ScenTODO/EF	9	Cancelled
/home/alvaro/ScenTODO/EG	/home/alvaro/ScenTODO/EG	10	Ready
/home/alvaro/ScenTODO/EH	/home/alvaro/ScenTODO/EH	10	Ready
/home/alvaro/ScenTODO/FG	/home/alvaro/ScenTODO/FG	10	Ready
/home/alvaro/ScenTODO/FH	/home/alvaro/ScenTODO/FH	10	Ready
/home/alvaro/ScenTODO/GH	/home/alvaro/ScenTODO/GH	10	Ready

Edit propierties

Source Folder: /home/alvaro/ScenTODO/FG

Results Folder: /home/alvaro/ScenTODO/FG

Repetitions: 17

Figure 8: Execution planner GUI

ferent simulations that Castadiva will process (column "Source folder"), and it also includes the path where results will be saved ("Results folder"), the numbers of runs per simulation ("Runs" column) and, at the end, it includes a message that informs the user about the status of the simulation (Ready, Simulation in progress, Canceled, Retrieving results, or Done).

The results are stored in a directory named "Iterations" inside the folder we mentioned in the column "Results Folder". Inside that directory every simulation creates a file named "X_DefinedTraffic.txt" where $X \epsilon \{1, Runs\}$, where each file contains the data of the table that we can see in figures 5 and 4. An example of this file can be seen in table 1.

5 Conclusions and future work

In this work we present some extensions to the Castadiva platform to improve research in the MANETs field by allowing to make real test-bed experiments in a simple and straightforward manner.

With the proposed improvements we have created a simple and powerful tool to automatize the evaluation of MANETs in general and QoS enabled MANETs in particular. Now, the VoIP and Videoconferencing experiments can be done taking into account QoS and delay parameters, which are usually available for simulation studies but only in a very few real experiments.

With this new features, we plan to create a real test-bed to analyze the behavior of various routing protocols when their packets are prioritized via IEEE 802.11e.

Castadiva is free software developed under the GNU GPL license, and can be downloaded at **http://castadiva.sourceforge.net**

Acknowledgments

This work was partially supported by the *Ministerio de Educación y Ciencia*, Spain, under Grant TIN2008-06441-C02-01, by the Generalitat Valenciana under Grant GV/2009/010.

Strt	Stop	Src	Addr	Traff	Transf	P/sec	Thrgpt	Received	Delay	AC	Redir
10	130	6	3	TCP	10737		5823.931				
10	130	6	3	UDP	512	125	505.941	99.64	121.64	VI	true
10	130	6	3	UDP	256	10	20.277	100.0	87.209	VO	true
10	130	3	6	UDP	512	125	504.497	99.52	84.506	VI	true
10	130	3	6	UDP	256	10	20.156	99.4	53.654	VO	true
Total UDP packets received: 99.64											
Average throughput: 1374.9603											

Table 1: Sample results obtained by the execution planner

References

[1] Network Time Protocol. http://datatracker.ietf.org/wg/ntp/charter/. Acessed: May 3, 2010.

[2] J.C Cano, J.M. Cano, C. Calafate, E. Gonzalez, and P. Manzoni. Evaluation of the trade-off between power consumption and performance in bluetooth based systems. *Sensor Technologies and Applications, International Conference on*, 0:313–318, 2007.

[3] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir R. Das. Ad hoc on-demand distance vector (AODV) routing. Request for Comments 3561, MANET Working Group, http://www.ietf.org/rfc/rfc3561.txt, July 2003. Work in progress.

[4] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, and K. Ramachandran. Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols. *Wireless Communications and Networking Conference, 2005 IEEE*, 3:1664–1669, 2005.

[5] Glenn Judd and Peter Steenkiste. Design and implementation of an rf front end for physical layer wireless network emulation. In *IEEE 2007 IEEE 65th Vehicular Technology Conference (VTC2007)*, April 2007.

[6] IEEE 802.11 WG. International Standard for Information Technology - Telecom. and Information exchange between systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications, ISO/IEC 8802-11:1999(E) IEEE Std. 802.11, 1999.

[7] IEEE 802.11 WG. 802.11e IEEE Standard for Information technology-Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements, 2005.

[8] Jorge Hortelano, Juan-Carlos Cano, Carlos T. Calafate, and Pietro Manzoni. Testing applications in manet environments through emulation. *EURASIP Journal on Wireless Communications and Networking, vol. 2009, Article ID 406979, 20 pages, 2009. doi:10.1155/2009/406979.*

[9] T. Clausen and P. Jacquet. Optimized link state routing protocol (OLSR). Request for Comments 3626, MANET Working Group, http://www.ietf.org/rfc/rfc3626.txt, October 2003. Work in progress.

[10] Y. Zhang and W. Li. An integrated environment for testing mobile ad hoc networks. Available at: http://www.wins.hrl.com/projects/adhoc.

C. Johann M. Márquez Barja's implementation for 802.11g in ns-2

```
# HTTP en VANETs

#  Simulation script for NS2
#  Johann Marquez (johann@marquez-barja.com)


puts "--> Top-Down Scripting!! \n"


# ======================================================================
# Define options
# ======================================================================

set val(chan)       Channel/WirelessChannel         ;# channel type
set val(prop)       Propagation/TwoRayGround        ;# radio-propagation model
set val(netif)      Phy/WirelessPhy                ;# network interface type
set val(mac)        Mac/802_11                    ;# MAC type
set val(ifq)        Queue/DropTail/PriQueue          ;# interface queue type;  ((Queue/DropTail/PriQueue ;#(for AODV)))    ((CMUPriQueue ;#(for DSR)))
#set val(ifq)        CMUPriQueue ;#(for DSR)
set val(ll)         LL                             ;# link layer type
set val(ant)        Antenna/OmniAntenna             ;# antenna model
set val(x)              2000                           ;# X dimension of the topography
set val(y)              2000                           ;# Y dimension of the topography
set val(ifqlen)         50                                ;# max packet in ifq
set val(seed)           0
set val(rp)             AODV                ;#one of OLSR, AODV, AODVUU, DSR, TORA, OLSR(v7)
set val(nn)             200                             ;# how many nodes are simulated
set val(nn_traffic_sources) 50                 ;#nodes que navegaran por web
set val(nn_servers) 1                 ;#nodes servidores
#set val(cp)             "traffic-trace-n50-c5... file"     ;#patron de conexiones, en web no utilizar
set val(sc)             "./scenarios/m2-n200-t6000-s17-w2000-h2000-d80-a0-v1.scn"; #escenario de trafico CAMBIAR
set val(stopns)           20                           ;# simulation time  set val(trfile) out.tr
set val(namfile)        out.nam
#set val(httpfile)       out.http
set val(random_mobility)   FALSE
set val(random_number_generator)   TRUE
set val(bw)             54e6
set val(completion)     0

set val(nn_pages)     300
set val(nn_session)   100
set val(interSession) 1
set val(interPageOption) 0
set val(pagesize) 10
set val(objsize)  10
# ============================================================================
# check for boundary parameters and random seed
if { $val(x) == 0 || $val(y) == 0 } {
   puts "-->  No X-Y boundary values given for wireless topology\n"
}
if {$val(random_number_generator) == TRUE} {
   puts "-->  Seeding Random number generator with $val(seed)\n"
   ns-random $val(seed)
}

proc usage { argv0 }  {
       puts "Usage: $argv0"
       puts "\tmandatory arguments:"
       puts "\t\t\[-x MAXX\] \[-y MAXY\]"
       puts "\toptional arguments:"
       puts "\t\t\[-cp conn pattern\] \[-sc scenario\] \[-nn nodes\]"
       puts "\t\t\[-seed seed\] \[-stop sec\] \[-tr tracefile\] \[-rp protocol\]\n"
}


proc getopt {argc argv} {
   global val
   lappend optlist nn nn_traffic_sources sc rp trfile httpfile stopns pagesize objsize ;#lista de parametros de ejecucion de sim ns -parametro valor -otroparametro valor

```

```
        for {set i 0} {$i < $argc} {incr i} {
                set arg [lindex $argv $i]
                if {[string range $arg 0 0] != "-"} continue

                set name [string range $arg 1 end]
                set val($name) [lindex $argv [expr $i+1]]
        }
}



#
# If routing protocol is DSR, adjust queue type
#
if {$val(rp) == "DSR" &&  $val(ifq) == "Queue/DropTail/PriQueue"} {
   set val(ifq)        CMUPriQueue
}


# to set one session per traffic source
set val(nn_session) $val(nn_traffic_sources)


puts "\n\n\n"
puts "-->  Configuration:"
puts "-->  set val(chan)       Channel/WirelessChannel         ;# channel type"
puts "-->  set val(prop)       Propagation/TwoRayGround        ;# radio-propagation model"
puts "-->  set val(netif)      Phy/WirelessPhy                ;# network interface type"
puts "-->  set val(mac)        Mac/802_11                    ;# MAC type"
puts "-->  set val(ifq)        Queue/DropTail/PriQueue          ;# interface queue type;
((Queue/DropTail/PriQueue ;#(for AODV))    ((CMUPriQueue ;#(for DSR)))"
puts "-->  #set val(ifq)        CMUPriQueue ;#(for DSR)"
puts "-->  set val(ll)         LL                             ;# link layer type"
puts "-->  set val(ant)        Antenna/OmniAntenna             ;# antenna model"
puts "-->  set val(x)              $val(x)                          ;# X dimension of the
topography"
puts "-->  set val(y)              $val(y)                          ;# Y dimension of the
topography"
puts "-->  set val(ifqlen)         50                               ;# max packet in ifq"
puts "-->  set val(seed)           0 "
puts "-->  set val(rp)             $val(rp)                     ;#one of AODV, AODVUU, DSR,
TORA, OLSR(v7)"
puts "-->  set val(nn)             $val(nn)                           ;# how many nodes are
simulated "
puts "-->  set val(nn_traffic_sources) $val(nn_traffic_sources)                ;#nodes que
navegaran por web "
#puts "-->  #set val(cp)             $val(cp)     ;#patron de conexiones, en web no utilizar"
puts "-->  set val(sc)             $val(sc);            #escenario de trafico CAMBIAR "
puts "-->  set val(stopns)           $val(stopns)                        ;# simulation time  "
puts "-->  set val(trfile)         $val(trfile) "
puts "-->  set val(namfile)        $val(namfile)"
#puts "-->  set val(httpfile)       $val(httpfile)"
puts "-->  set val(random_mobility)   $val(random_mobility) "
puts "-->  set val(random_number_generator)   $val(random_number_generator)"
puts "-->  set val(bw)             $val(bw)"
puts "-->  set val(completion)  $val(completion)"
puts "-->  set val(nn_pages)    $val(nn_pages)"
puts "-->  set val(nn_session)  $val(nn_session)"
puts "-->  set val(interSession) $val(interSession)"
puts "-->  set val(interPageOption) $val(interPageOption)"
puts "-->  set val(pagesize) $val(pagesize)"
puts "-->  set val(objsize) $val(objsize)"
puts "\n\n\n"


# ======================================================================
# Main Program
# =======================================================================



#
# Initialize Global Variables
#

```

```
LL set mindelay_                50us
LL set delay_                   25us
LL set bandwidth_               0       ;# not used
LL set off_prune_               0       ;# not used
LL set off_CtrMcast_            0       ;# not used

Agent/Null set sport_           0
Agent/Null set dport_           0

Agent/CBR set sport_            0
Agent/CBR set dport_            0

Agent/TCPSink set sport_        0
Agent/TCPSink set dport_        0

Agent/TCP set sport_            0
Agent/TCP set dport_            0
Agent/TCP set packetSize_       1460

#give preference to routing protocols
Queue/DropTail/PriQueue set Prefer_Routing_Protocols    1

# unity gain, omni-directional antennas
# set up the antennas to be centered in the node and 1.5 meters above it
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 1.0
Antenna/OmniAntenna set Gr_ 1.0

# Initialize the SharedMedia interface with parameters to make
# it work like the 914MHz Lucent WaveLAN DSSS radio interface
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.559e-11
Phy/WirelessPhy set RXThresh_ 3.652e-10
Phy/WirelessPhy set Rb_ 11*1e6
Phy/WirelessPhy set Pt_ 0.2818
Phy/WirelessPhy set freq_ 914e+6
Phy/WirelessPhy set L_ 1.0
# Phy/WirelessPhy set bandwidth_ 11e6
Phy/WirelessPhy set bandwidth_ 54e6        ;# for 802.11a and 802.11g


# Initialize the 802.11 MAC
#Mac set bandwidth_ $val(bw)
#puts "Bandwidth set to $val(bw)"
#Mac/802_11 set dataRate_ $val(bw)


#Configuration for 802.11a and 802.11g

Mac/802_11 set CWMin_         15
Mac/802_11 set CWMax_         1023
Mac/802_11 set SlotTime_      0.000009            ;# 20us
Mac/802_11 set SIFS_          0.000016            ;# 10us
Mac/802_11 set PreambleLength_        96          ;# 144 bit
Mac/802_11 set PLCPHeaderLength_      40     ;# 48 bits
Mac/802_11 set PLCPDataRate_  6.0e6            ;# 1Mbps
Mac/802_11 set RTSThreshold_  3000              ;# bytes
Mac/802_11 set ShortRetryLimit_       7             ;# retransmittions
Mac/802_11 set LongRetryLimit_        4             ;# retransmissions
Mac/802_11 set basicRate_ 6e6
Mac/802_11 set dataRate_ 54e6
```

D. CityMob2 output file example

```
#
# MODEL 4: Enhanced Downtown Traffic Simulation Model
#
#
#        SEED = 1273842116345
#
#
# nodes number= 5, max time= 120.0, max_speed= 100.0, max X= 1500.0, max Y= 1500.0
#
# d= 20.0, damaged= 1, alpha= 0.5, delta= 5.0
#
# min_speed_d= 25.0, max_speed_d= 50.0
#
#Node 0 (1140.0, 1390.6241869639089) --> (1140.0, 280.0) 58.88932070560723 Km/h
$node_(0) set X_ 1140.0
$node_(0) set Y_ 1390.6241869639089
$node_(0) set Z_ 0.0
#Node 1 (540.0, 111.77523584641058) --> (540.0, 1200.0) 75.06782928526441 Km/h
$node_(1) set X_ 540.0
$node_(1) set Y_ 111.77523584641058
$node_(1) set Z_ 0.0
#Node 2 (210.11858578806059, 560.0) --> (1240.0, 560.0) 59.38569098715311 Km/h
$node_(2) set X_ 210.11858578806059
$node_(2) set Y_ 560.0
$node_(2) set Z_ 0.0
#Node 3 (500.0, 145.20410617171447) --> (500.0, 380.0) 67.02568486533434 Km/h
$node_(3) set X_ 500.0
$node_(3) set Y_ 145.20410617171447
$node_(3) set Z_ 0.0
#Node 4 (800.0, 693.2904817726454) --> (800.0, 340.0) 59.76214259985875 Km/h
$node_(4) set X_ 800.0
$node_(4) set Y_ 693.2904817726454
$node_(4) set Z_ 0.0
#
#-- END OF INITIAL POSITION CONFIGURATION --
#
# Movements:
#
##### NODE 0 MOVEMENTS #####
#Node 0 had an ACCIDENT:
$ns_ at 0.0 "$node_(0) setdest 1140.0 1390.6241869639089 0.0"
#END ACCIDENT
##### NODE 1 MOVEMENTS #####
#Node 1: TARGET REACHED (540.0, 1200.0) (75.06782928526441 Km/h)
$ns_ at 52.18759071966314 "$node_(1) setdest 540.0 1200.0 75.06782928526441"
#Node 1: NEW TARGET (40.0, 1200.0)
#Node 1: TARGET REACHED (40.0, 1200.0) (52.77465990538741 Km/h)
$ns_ at 86.30727806161097 "$node_(1) setdest 40.0 1200.0 52.77465990538741"
#Node 1: RED SEMAPHORE UNTIL 104.4088489660594
#Node 1: GREEN SEMAPHORE
#Node 1: NEW TARGET (40.0, 520.0)
$ns_ at 104.45 "$node_(1) setdest 40.0 1200.0 0.0"
#Node 1: SPEED CHANGE (0.0 Km/h --> 63.40445188023556 Km/h)
$ns_ at 120.0 "$node_(1) setdest 40.0 926.1279925728561 63.40445188023556"
##### NODE 2 MOVEMENTS #####
#Node 2: TARGET REACHED (1240.0, 560.0) (59.38569098715311 Km/h)
$ns_ at 62.432094828450175 "$node_(2) setdest 1240.0 560.0 59.38569098715311"
#Node 2: NEW TARGET (1240.0, 140.0)
#Node 2: TARGET REACHED (1240.0, 140.0) (58.1799813877065 Km/h)
$ns_ at 88.43832044864641 "$node_(2) setdest 1240.0 140.0 58.1799813877065"
#Node 2: NEW TARGET (60.0, 140.0)
$ns_ at 120.0 "$node_(2) setdest 718.4845036632628 140.0 59.5073149544258"
##### NODE 3 MOVEMENTS #####
#Node 3: TARGET REACHED (500.0, 380.0) (67.02568486533434 Km/h)
$ns_ at 12.611064243209228 "$node_(3) setdest 500.0 380.0 67.02568486533434"
#Node 3: NEW TARGET (1400.0, 380.0)
#Node 3: TARGET REACHED (1400.0, 380.0) (84.05963188698215 Km/h)
$ns_ at 51.19406600728597 "$node_(3) setdest 1400.0 380.0 84.05963188698215"
#Node 3: RED SEMAPHORE UNTIL 66.41438630266265
#Node 3: GREEN SEMAPHORE
#Node 3: NEW TARGET (1400.0, 1440.0)
$ns_ at 66.45 "$node_(3) setdest 1400.0 380.0 0.0"
#Node 3: SPEED CHANGE (0.0 Km/h --> 77.55136935175538 Km/h)
#Node 3: TARGET REACHED (1400.0, 1440.0) (77.55136935175538 Km/h)
```

```
$ns_ at 115.65609438489076 "$node_(3) setdest 1400.0 1440.0 77.55136935175538"
#Node 3: RED SEMAPHORE UNTIL 129.19274902868204
$ns_ at 120.0 "$node_(3) setdest 1400.0 1440.0 0.0"
##### NODE 4 MOVEMENTS #####
#Node 4: TARGET REACHED (800.0, 340.0) (59.76214259985875 Km/h)
$ns_ at 21.281796117941216 "$node_(4) setdest 800.0 340.0 59.76214259985875"
#Node 4: NEW TARGET (1440.0, 340.0)
#Node 4: TARGET REACHED (1440.0, 340.0) (92.42208279251665 Km/h)
$ns_ at 46.22910709632431 "$node_(4) setdest 1440.0 340.0 92.42208279251665"
#Node 4: RED SEMAPHORE UNTIL 55.864743625517406
#Node 4: GREEN SEMAPHORE
#Node 4: NEW TARGET (1440.0, 120.0)
$ns_ at 55.9 "$node_(4) setdest 1440.0 340.0 0.0"
#Node 4: SPEED CHANGE (0.0 Km/h --> 79.91145948724842 Km/h)
#Node 4: TARGET REACHED (1440.0, 120.0) (79.91145948724842 Km/h)
$ns_ at 65.8109690284956 "$node_(4) setdest 1440.0 120.0 79.91145948724842"
#Node 4: NEW TARGET (20.0, 120.0)
$ns_ at 120.0 "$node_(4) setdest 44.37136896264302 120.0 92.78417491661162"
#
#-- EXITING PROGRAM --#
```

E. Configuration for 802.11g in ns-2 without distance losses

```
# ======================================================================
# Define options
# ======================================================================
set val(chan)         Channel/WirelessChannel  ;# channel type
set val(prop)         Propagation/TwoRayGround ;# radio-propagation model
set val(ant)          Antenna/OmniAntenna      ;# Antenna type
set val(ll)           LL                       ;# Link layer type
set val(ifq)          Queue/DropTail/PriQueue  ;# Interface queue type
set val(ifqlen)       50                       ;# max packet in ifq
set val(netif)        Phy/WirelessPhy          ;# network interface type
set val(mac)          Mac/802_11              ;# MAC type
set val(rp)           DumbAgent                ;# ad-hoc routing protocol
set val(nn)           5                        ;# number of mobilenodes

# ======================================================================
# Simulation
# ======================================================================


# Create simulator
set ns_    [new Simulator]

# Set up trace file
$ns_ use-newtrace
set tracefd [open simple.tr w]
$ns_ trace-all $tracefd

# Setup Xgraph files
set f0 [open out0.tr w]
set f1 [open out1.tr w]
set f2 [open out2.tr w]
set f3 [open out3.tr w]
set f4 [open out4.tr w]
set f5 [open out5.tr w]
set f6 [open out6.tr w]
set f7 [open out7.tr w]
set f8 [open out8.tr w]
set f9 [open out9.tr w]

# Create the "general operations director"
# Used internally by MAC layer: must create!
create-god $val(nn)




LL set mindelay_                50us
LL set delay_                   25us
LL set bandwidth_               0       ;# not used
LL set off_prune_               0       ;# not used
LL set off_CtrMcast_            0       ;# not used

Agent/Null set sport_           0
Agent/Null set dport_           0

Agent/CBR set sport_            0
Agent/CBR set dport_            0

Agent/TCPSink set sport_        0
Agent/TCPSink set dport_        0

Agent/TCP set sport_            0
Agent/TCP set dport_            0
Agent/TCP set packetSize_       1460

# unity gain, omni-directional antennas
# set up the antennas to be centered in the node and 1.5 meters above it
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 1.0
Antenna/OmniAntenna set Gr_ 1.0

# Initialize the SharedMedia interface with parameters to make
# it work like the 914MHz Lucent WaveLAN DSSS radio interface
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.559e-11
Phy/WirelessPhy set Rb_ 11*1e6
Phy/WirelessPhy set freq_ 914e+6
Phy/WirelessPhy set L_ 1.0
# Phy/WirelessPhy set bandwidth_ 11e6
Phy/WirelessPhy set bandwidth_ 54e6        ;# for 802.11a and 802.11g


# Initialize the 802.11 MAC

#Configuration for 802.11a and 802.11g

Mac/802_11 set CWMin_         15
Mac/802_11 set CWMax_         1023
Mac/802_11 set SlotTime_      0.000009            ;# 20us
Mac/802_11 set SIFS_          0.000016            ;# 10us
Mac/802_11 set PreambleLength_        96          ;# 144 bit
Mac/802_11 set PLCPHeaderLength_      40     ;# 48 bits
Mac/802_11 set PLCPDataRate_  6.0e6            ;# 1Mbps
Mac/802_11 set RTSThreshold_  3000              ;# bytes
Mac/802_11 set ShortRetryLimit_       7             ;# retransmittions
Mac/802_11 set LongRetryLimit_        4             ;# retransmissions
Mac/802_11 set basicRate_ 6e6
Mac/802_11 set dataRate_ 54e6


# The RX treshold determines the range
# Processed with threshold.cc
Phy/WirelessPhy set RXThresh_ 9.90556e-10
# 32dBm
Phy/WirelessPhy set Pt_ 1.58489


# Create and configure topography (used for mobile scenarios)
set topo [new Topography]
# 1000x1000m terrain
$topo load_flatgrid 1500 1500

# Configure the future nodes
$ns_ node-config -adhocRouting $val(rp) \
        -llType $val(ll) \
        -macType $val(mac) \
        -ifqType $val(ifq) \
        -ifqLen $val(ifqlen) \
        -antType $val(ant) \
        -propType $val(prop) \
        -phyType $val(netif) \
        -channel [new $val(chan)] \
        -topoInstance $topo \
        -agentTrace OFF \
        -routerTrace OFF \
        -macTrace ON \
        -ifqTrace OFF \
```

F. Configuration for 802.11g in ns-2 with distance losses

```
# ======================================================================
# Default NS2 header for Wannes's work on Castadiva's validation       =
# Simulates a 802.11g wireless network (Parameters from Yojan)         =
# ======================================================================
# = Version 1.1 =========================================== 2010.04.16 =
# ======================================================================

 # ***********************************************************
 # Define general simulation options
 # ***********************************************************
        # sensing threshold in dB above noise power
        set sensingTreshdB 5

        set system_type [exec uname -s]
        # Libraries have different names in some operating systems
        if {[string match "CYGWIN*" "$system_type"] == 1} {
            load /usr/bin/ns-allinone-2.34/dei80211mr-1.1.4/src/.libs/cygdei80211mr-0.dll
        } else {
            load /usr/bin/ns-allinone-2.34/dei80211mr-1.1.4/src/.libs/libdei80211mr.so
        }

        set val(chan)         Channel/WirelessChannel/PowerAware  ;# channel type
        set val(prop)         Propagation/FreeSpace/PowerAware    ;# radio-propagation model
        set val(ant)          Antenna/OmniAntenna                 ;# Antenna type
        set val(ll)           LL                                  ;# Link layer type
        set val(ifq)          Queue/DropTail/PriQueue             ;# Interface queue type
        set val(ifqlen)       10                                  ;# max packet in ifq
        set val(netif)        Phy/WirelessPhy/PowerAware          ;# network interface type
        set val(mac)          Mac/802_11/Multirate                ;# MAC type
        set val(rp)           DumbAgent                           ;# #one of OLSR, AODV, AODVUU, DSR, TORA, OLSR(v7)
        set val(nn)           5                                   ;# number of mobilenodes
        set PHYDataRate       Mode54Mb
 # ***********************************************************
 # Define advanced options
 # ***********************************************************
        LL set mindelay_                1us
        LL set delay_                   1us
        LL set bandwidth_               0       ;# not used

        Node/MobileNode instproc getIfq { param0} {
            $self instvar ifq_
            return $ifq_($param0)
        }

        Node/MobileNode instproc getPhy { param0} {
            $self instvar netif_
            return $netif_($param0)
        }

        set noisePower 7e-11
        set per [new PER]
        $per loadPERTable80211gTrivellato
        $per set noise_ $noisePower
        set val(CSThresh) [expr $noisePower *  pow ( 10 , $sensingTreshdB / 10.0 ) ]
        set val(AffectThresh) [expr $noisePower ]
        # Transmission power is 32dBm
        Phy/WirelessPhy set Pt_ 1.58489
        Phy/WirelessPhy set freq_ 2437e6
        Phy/WirelessPhy set L_ 1.0

        # The RX treshold determines the range
        # If the received signal strength is
        # greater than this threshold, the packet can be successfully received.
        # Processed with threshold.cc
        # This threshold is for 300 m with 32dbm transmission power
        Phy/WirelessPhy set RXThresh_ 9.90556e-10

        Mac/802_11 set bSyncInterval_ 20e-6
        Mac/802_11 set gSyncInterval_ 10e-6

        Mac/802_11 set ShortRetryLimit_ 3
        Mac/802_11 set LongRetryLimit_ 5

        Mac/802_11/Multirate set RTSThreshold_ 100000

        Mac/802_11/Multirate set dump_interf_ 0

        $per set debug_ 0
        PowerProfile set debug_ 0
        Phy/WirelessPhy set debug_ 0
        Mac/802_11/Multirate set debug_ 0
        Phy/WirelessPhy set CSThresh_ $val(CSThresh)

# ======================================================================
# End of the default header                                            =
# ======================================================================
```

G. Simple Gnuplot example file

```
set terminal svg size 600,400 enhanced fname 'arial'  fsize 11 butt solid
set output 'CityMob - PositionX.svg'
set key horizontal
set title "One node's X position in a random simulation, depending on simulation progress"
set xlabel "Simulation progress (Seconds)"
set ylabel "Node's X coordinates"
set yrange [0:1500]

plot "positionNode0.tr" using 1:2 title 'NS node 1' with lines, \
"positionNode0.txt" using 1:2 title 'Castadiva node 1' with lines, \
"positionNode1.tr" using 1:2 title 'NS node 2' with lines, \
"positionNode1.txt" using 1:2 title 'Castadiva node 2' with lines, \
"positionNode2.tr" using 1:2 title 'NS node 3' with lines, \
"positionNode2.txt" using 1:2 title 'Castadiva node 3' with lines, \
"positionNode3.tr" using 1:2 title 'NS node 4' with lines, \
"positionNode3.txt" using 1:2 title 'Castadiva node 4' with lines, \
"positionNode4.tr" using 1:2 title 'NS node 5' with lines, \
"positionNode4.txt" using 1:2 title 'Castadiva node 5' with lines
unset output
```

H. Mobility Plugin algorithm for the Mobility Plugin System

```java
System.out.println("Variables are :\nminSpeed="+minSpeed+" maxSpeed="+maxSpeed+"
totaltime="+totalTime+" Width="+X+" Height="+Y);

// Array to place the nodes
float[] XPos = new float[accessPoints.Size()];
float[] YPos = new float[accessPoints.Size()];

// Initial position is 200 for all nodes
for (int i = 0; i < accessPoints.Size(); i++) {
        XPos[i]=200;
        YPos[i]=200;
}

// At each second, position is incremented for al nodes
for (int j = 0; j <= totalTime; j++) {
        for (int i = 0; i < accessPoints.Size(); i++) {
                AP p = accessPoints.Get(i);

                XPos[i] += 10*i;
                YPos[i] += 10*i;

                // Position cannot be outside of the canvas
                XPos[i] = XPos[i] % (Float)X;
                YPos[i] = YPos[i] % (Float)Y;

                NodeCheckPoint checkPoint = new NodeCheckPoint((float) XPos[i], (float) YPos[i],
new Float(0));
                System.out.println("Checkpoint : node"+i+"]["+j+"] ("+XPos+","+YPos+")");
                nodes[i][j] = checkPoint;
        }
}
```