Evaluating the goodness of MANETs performance results obtained with the ns-2 simulator

Jorge Hortelano, Marga Nácher, Juan-Carlos Cano, Carlos Calafate, Pietro Manzoni
Polytechnic University of Valencia
Dept. of Computer Engineering
Camino de Vera, s/n, 46071 Valencia, SPAIN
pmanzoni@disca.upv.es

ABSTRACT

The development of new protocols and applications for Mobile Ad Hoc Networks (MANET) requires a thorough testing before their deployment in a real environment. Testing is typically based on simulation, even though this might "oversimplify" the problem thus leading to incorrect results. In this work we describe *Castadiva*, a test bed based on low-cost off-the-shelf devices, which is used to test protocols developed for MANETs. *Castadiva* is completely compatible with the file format used by the ns-2 simulator, so simplifying and allowing a fair comparison between the two systems. We compare the results obtained with *Castadiva* and the ns-2 network simulator, and demonstrate that the ns2 simulator is a reliable tool for the evaluation of MANETs related proposal.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol verification*; D.2 [**Software Engineering**]: Software/Program Verification

Keywords

Simulation, ad hoc networks, validation.

1. INTRODUCTION.

Mobile ad-hoc networks (MANET) [1] are wireless networks with no fixed infrastructure. Nodes belonging to a MANET can either be end-points of a data interchange or can act as routers when the two end-points are not directly within their radio range. The growing research efforts focusing on this new technology requires the availability of tools that allow researchers to evaluate their proposals. Testing and evaluating any of the proposed protocols for MANETs is a mandatory request to guarantee its success in any real word application.

Researchers have two options for testing their proposals, either through simulation or using real test-beds platform. Among the available networks simulators for MANETs, ns-2 [2] is the most popular in the research community. It is a discrete event network simulator, which can be easily extended and which is well documented. Simulation tools are used because they are flexible and inexpensive. A real test-bed platform on the contrary permits to obtain more rigorous and realistic results.

In this work we use *Castadiva*, a test-bed architecture that simplify carrying out realistic experiments; it relies on low-cost, off-the-shelf devices combined with a Linux platform. *Castadiva* allows generating network topologies, exporting them to real devices and obtaining the test results. It can also generate different types of traffic between nodes, and offers support for some well-known ad-hoc routing protocols. It relies on a cheap architecture that includes two different networks: a wired network, called connection network, that connects the core with a group of wireless nodes, and a wireless network where the actual test-bed experiments are made. We developed a group of tools for administration purposes, with a friendly user interface design to help the user to define the scenario of the network and the desired traffic connections between MANET nodes. All of these tools were developed with open source software.

Since *Castadiva* is completely compatible with the ns-2 file format we used it to perform the comparison in a simple and straightforward manner. We selected a range of both static and dynamic MANETs scenarios and compared the obtained results in *Castadiva* with those obtained using ns-2. We confirmed that the ns-2 simulator is a reliable tool that provides results which are comparable, although not identical, to those obtained with a realistic MANET.

The rest of this paper is organized as follows: Section 2 describes *Castadiva*'s architecture. Section 3 presents the implementation details and, in Section 4, we compare this tool with the ns-2 simulator. Finally, in Section 5 we present our concluding remarks.

2. CASTADIVA OVERVIEW

Castadiva is a test-bed developed to evaluate and analyze protocols and applications for MANETs. The test-bed relies on an actual wireless network between nodes for testing purposes.

The main application, developed in Java, controls all devices and manages the links among them according to a

NSTools 2007 - First International Workshop on Network Simulation Tools 2007. October 22, 2007, Nantes France

pre-defined network topology. It also manages traffic generation between node pairs. *Castadiva* combines two different networks: the coordination network (wired), that connects the *Castadiva* core with the wireless nodes, and the wireless network, where actual tests are run. Figure 1 shows a schema of *Castadiva*'s components.

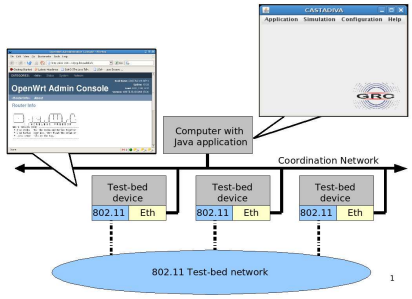

Figure 1: Schema of *Castadiva*'s components.

The coordination network is a wired network that connects *Castadiva*'s core server with the wireless nodes. This network allows the main application to send configuration messages to all the nodes without creating any interference within the wireless network itself. It is based on Fast-Ethernet technology, avoiding large latency. Basically, this network requires a switch connected to the main server and to all nodes. Through this network the main application sends instructions to nodes, allowing them to re-configure so as to create the desired network topology. It also allows running traffic-generating applications available on each wireless node. The SSH protocol is used to send instructions through this network. Using a fast network means that all nodes will start an experiment at about the same time, avoiding significant latency and maximizing result accuracy. The wireless network is composed of wireless nodes, and the actual topology of this network is defined using the GUI of *Castadiva*.

Castadiva core has two main functions: (a) to allow a user to interact with the system so as to define all the test parameters required and (b) to coordinate the wireless nodes during an experiment. By using *Castadiva*'s GUI a user can control all of *Castadiva*'s functionality, defining the network topology and the traffic flow among nodes. *Castadiva* allows setting the scenario area where nodes will be deployed. When selecting a node, its location is highlighted and it can be changed according to the desired network topology. When all nodes are deployed the user can press the button *Simulate,* and each physical node will be re-programmed so as to enforce the chosen network topology. Figure 2 shows how *Castadiva* allows a user to interact with the network. We describe the whole functionality offered by *Castadiva*'s GUI in Section 3.2.1.

Castadiva's server executes the application and configures the network devices. It consists of a Pentium IV with 1 GBs of RAM memory, and has a Linux Debian Etch distribution installed. Concerning the wireless nodes used, they can be any sort of computing device, like a laptop, a PDA or a wireless router. The main requirement for a node is that it must have a Linux/Unix operative system installed, and two network cards: an Ethernet card and an IEEE 802.11 card. If the node is a wireless router, the OpenWRT [3] kernel is a good solution. OpenWRT is an open source operating system available for a wide range of router manufacturers. This embedded Linux system natively offers SSH connections, along with the possibility of running shell scripts. Moreover, a programmer can develop its own application in a standard Linux distribution and export it to this operative system. In our case, we developed some applications in C for traffic generation/control purposes.

Figure 3: Castadiva's physical network.

Figure 3 shows our test-bed. One switch connects CASTADIVA's server with all the wireless nodes for coordination purposes. On the right hand of the picture the group of wireless nodes being used are shown. It consists of ten Linksys routers (models WRT56G and WRT56GL) and a Buffalo router (model WZR-RS-G54). The wireless ad hoc network conformed by these nodes is the one used in *Castadiva*'s testbed experiments.

3. *CASTADIVA*'S IMPLEMENTATION DETAILS

In this section we detail the requirements of *Castadiva* on the server and on the wireless nodes. We describe the software tools we have developed to connect all the wireless nodes with the server, and how *Castadiva* allows making connections among them. We also explain the process of designing network topologies by using the Scenario Generation tool, an interactive and user-friendly interface that allows defining the network's scenario and the desired traffic connections among nodes.

Castadiva requires some libraries and services to operate. The requirements of *Castadiva* are different for the server and the wireless nodes. The server must be a standard Linux-based system and must have a Java Virtual Machine, an SSH client and an NFS server installed. Each node must be a Linux based system with an SSH server and an NFS client; besides, it must include the *libgcc* library and have the *Iptables* toolset installed (see figure 4).

The connection between *Castadiva*'s core element (server) and each node is made using both SSH and NFS connections. On *Castadiva*'s server, the user interacts with the application by defining the network topology, the traffic and selecting the desired routing protocol. Then, through SSH, the application sends a starting instruction to each node

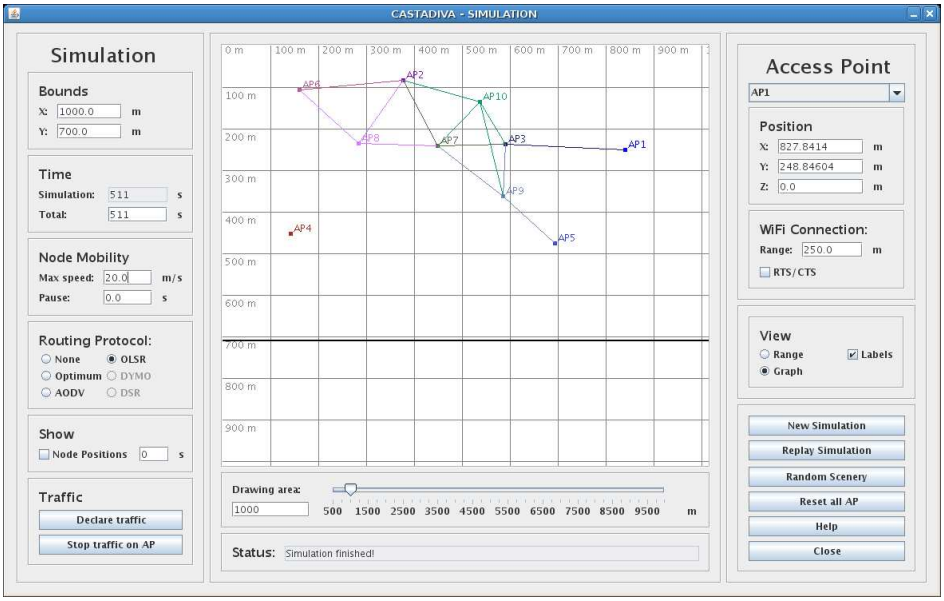

Figure 2: A view of *Castadiva*'s GUI.

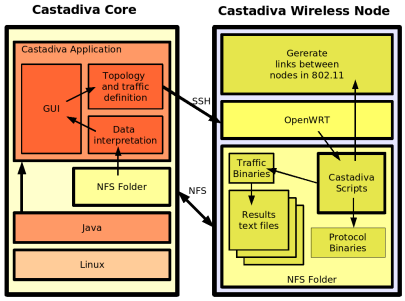

Figure 4: Software components for *Castadiva*.

through the coordination network (wired). Wireless nodes achieve coordination among themselves by executing the required binaries, which are stored into a server folder shared through NFS. This is a easy way to spread instructions to all nodes, and it solves storage limitation problems on nodes. When tests start, a group of files with the results are created and stored into *Castadiva*'s server by again relying on the NFS filesystem. We find that Ethernet connections are fast enough to export these files to the routers without significant delays.

The main application parses the results, obtaining the different testbed statistics. Finally, the application displays results to the user.

The application was developed using the Java programming language and the BASH scripting language. To make SSH connections through Java we use JSch [4], by JCraft. It is required to coordinate all the nodes during experiments. *Castadiva*'s implementation can be divided into two parts: the main application and the light-weight applications running on wireless nodes. The latter are the focus of the next section.

3.1 Wireless nodes' software

Each node has a set of requirements that must be met for successful operation: a Linux-based operative system, a set of special-purpose scripts, some specific applications and connectivity to *Castadiva*'s server.

The operating system installed on each router is OpenWRT. OpenWRT allows executing BASH scripts natively; besides, it includes Dropbear, a simple SSH server used to receive instructions from *Castadiva*'s server. Concerning the set of *Castadiva*'s scripts, they are generated automatically by *Castadiva*'s main application. Their purpose is to configure the wireless network topology.

Each node makes use of three applications: Iptables, TcpFLow, and UdpFlow. The first one is open source and exist in most Linux distributions, while the other two were developed by us.

Network topology configuration is made through the Iptables [5] tool. According to the selected topology, Iptables allows us to dynamically break the network links between pairs of nodes. This tool exists for all Linux distributions, including the OpenWRT embedded system.

To generate traffic we create the UdpFlow and TcpFlow tools. Both tools are designed to create a traffic flow between two nodes, each tool create one class of traffic. To create a flow of data we must specify a source/destination pair, the starting and ending times for this flow, and the maximum amount of bytes to be sent.

Castadiva also includes routing agents for well-known routing protocols, such as the AODV [6] provided by the Uppsala University and the OLSR [7] which is included with the

OpenWRT distribution.

3.2 Main application

Castadiva's core element, a Java application running at the server, includes all the control functions required for test-bed experimentation. It is responsible for network topology maintenance, traffic control, as well as reporting the obtained simulation results and graphical representation. When the user define the network scenario and the selected traffic, *Castadiva* configures the wireless nodes according to that topology. The application communicates with each node through SSH connections to send the required instructions. The traffic flow between nodes and the routing protocol used are also set through this technique. When all experiments are finished, *Castadiva*'s core calculates the performance results, and finally it shows these results to the user.

Castadiva's main application was created using Java's Swing library. We consider that it is a good solution for visual design since most basic components are already created, and can be easily modified by the programmer. In its development we have used the Model - View - Controller [8] design as reference.

3.2.1 Scenario Generation Tool

Castadiva is designed to be a test-bed where network scenarios and traffic between nodes are generated so as to resemble a real MANET. Therefore, it is expected to be an easy and useful tool for the study of MANETs.

To start a new experiment we only need to define the network topology in the corresponding window and then define the traffic flow and the routing protocol used. By pressing the start button tests begin, and *Castadiva* returns the test results. We now offer more details about *Castadiva*'s GUI.

Main menu

A standard menu allows accessing the different options of *Castadiva*. Basic options were added, allowing a user to save and load a project, or export it to other test environments such as ns-2. It actually generates all the files required as input to this particular simulator, allowing to compare *Castadiva*'s test results with those obtained through simulation. By selecting the *Application* menu you may start a new testbed experiment, load a previous one, or save the last one defined. It also allows to exchange files between ns-2 and *Castadiva*, allowing to use the same scenario and traffic in both applications.

Adding Nodes To The Testbed

Before starting an experiment the user needs to define the number of participating nodes, along with their configuration. Such information allows *Castadiva* to access nodes and manipulate them to generate a scenario. All the information is defined automatically when the user wishes to add a new one, though it can be changed by the user or can be read from a file. An internal identifier is required to distinguish a node from others in *Castadiva*'s framework. Such identifier is then referenced when defining the network topology and data connections. The remaining parameters will be used by *Castadiva*'s main application to connect nodes among themselves and with the main server. The MAC address is required for *Castadiva* to enforce topology changes.

All the executable files and the scripts are stored in an NFS directory that is accessible by all nodes. This way *Castadiva* makes storage capacity independent of wireless nodes' memory.

Castadiva relies on its own tools to generate traffic between nodes. Such tools are run on each node, and must be compiled for all types of CPU used. Currently, tools are compiled for MIPS and Intel processors, though the list can be easily augmented.

The SSH user and password fields are used by the main application to connect to each individual router and submit commands. Also, a Ping button was included to allow testing the connectivity between the server and routers.

Ad Hoc Network Scenario Generation

Once all the nodes are defined, they can be distributed to conform a scenario. *Castadiva* supports both manual and random topology generation, and the scenario is set through *Castadiva*'s blackboard. The blackboard is a representation of a virtual environment where nodes are located. Nodes are differentiated through different colors and labels. If the appropriate option is selected, the radio communication range is also shown through a circle of the same colour.

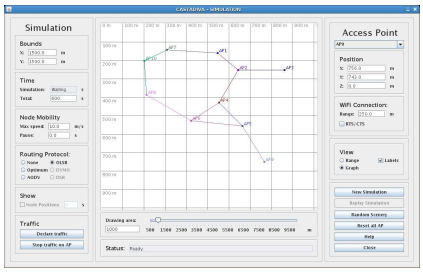

Figure 5: Scenario definition with CASTADIVA.

Figure 5 shows the topology generation environment. We can see ten nodes located in a scenario of 1000 x 1000 meters (scenarios bounds are marked with a darker line).

At the right hand we may edit node properties, such as position and signal range. *Castadiva* also can activate or deactivate the RTS/CTS 802.11 option for each node. The group of buttons appearing below allow starting a new test, stopping it and rebooting nodes to reset all values.

At the left hand, *Castadiva* offers scenario option editing. We can define the scenario bounds, the test time, node mobility and the routing protocol used. The *Declare traffic* button allows setting traffic, as shown below, and the stop button halts it.

A status bar provides general information to inform the user about what is being done, and the horizontal scrolls allows zooming in and out. Finally, the user may alternate between two different views: radio ranges or a graph. Every edge of the graph represents an IEEE 802.11 link connection, which is a more intuitive view.

Mobility in Castadiva.. It is important to point out the speed and pause option of the Scenario Window. If a user write a value greater than zero in the speed option, each node acquire a random movement with a speed between zero and the inserted value. When a node arrives to a destination point, it wait for a selected pause time and then select a new random destination point to move on. This behavior is also completely compatible with the one selected in the ns-2 simulator.

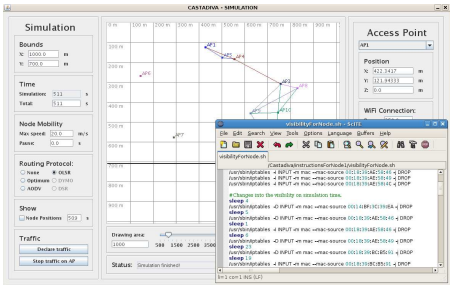

Figure 6: Mobility implementation.

Castadiva generate all node movements needed for the simulation before it starts. For each simulated second, it calculates the visibility range for each node. The obtained visibility is translated into Iptables rules and sleep commands. Algorithm 1 shows the behaviour of *Castadiva* when a node (with MAC 00:14:BF:3C:39:EA) go out range at second 10 and return in range at second 25.

Algorithm 1: Iptables rules for a simulation between two nodes.

```
sleep 10
iptables -I INPUT -m mac –mac-source 00:14:BF:3C:39:EA
-j DROP
sleep 15
iptables -D INPUT -m mac –mac-source
00:14:BF:3C:39:EA -j DROP
```

Castadiva also allows a user to capture the network topology at any simulated second, which could be useful to do a post-processing of the changes occurred in a network topology when the mobility has been activated.

3.2.2 *Network Traffic Declaration*

Castadiva's traffic generation tool allows defining different types of traffic flows between pairs of nodes. With that purpose *Castadiva* provides a table where each row defines a connection. Traffic parameters for each connection can be set depending on the type of protocol selected, and invalid values are marked with red. Examples of parameters are: *packet size*, *packets per second*, *start time*, *end time* and *maximum number of packets sent*. When an experiment finishes, *Castadiva* fills in this table with results, including throughput and, if traffic is UDP based, the percentage of packets received. *Castadiva* also shows the percentage of UDP packets correctly received and the aggregated throughput.

3.2.3 *Random test generator*

Sometimes it is useful to automate the test-bed evaluation process varying different parameters. With that purpose *Castadiva* includes functionality to generate random tests, where a user can define traffic and automatically test with different number of nodes and randomly-generated network topologies.

The user must specify the bounds of the scenario and the routing protocol used. The minimum and maximum number of nodes for testing must also be defined, along with the increase granularity. (e. g., with a node interval between 4 and 10 nodes and a granularity of 2, *Castadiva* executes four tests with 4, 6, 8, and 10 nodes). *Castadiva* allows also to specify how many times each test will be repeated.

At the top left the current scenario generated is displayed, though it can not be modified. Again, all the tests can be stored in either *Castadiva*'s or ns-2's format.

4. PERFORMANCE RESULTS COMPARISON

In this Section we describe the comparison of the performance results obtained using *Castadiva* and the ns-2 simulator. We use a representative scenario where nodes are located so that the maximum number of hops between them is four (see Figure 7).

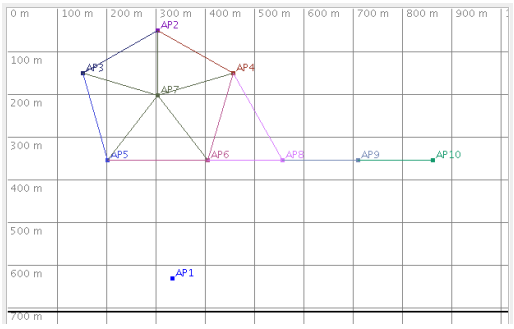

Figure 7: The scenario used for the test.

The scenario spans a 1000m x 700m area, and the test time is of 510 seconds. We set the wireless nodes' range to 250 meters. In terms of traffic, we define both UDP and TCP connections between each participating node and node AP7. For TCP connections, the maximum transfer size is of 1000 MB. UDP flows generate 4 packets per seconds, and packet size is fixed at 512 bytes. The traffic start at the simulation time of 10 seconds, allowing some previous node movement.

Figure 8 shows a by-node comparison between *Castadiva* and ns-2 node. In this test, each node has a maximum speed of 5 m/s. and no routing protocol is selected, therefore only traffic between directly connected nodes is allowed. The selected scenario was generated by ns-2 and imported to *Castadiva*.

The results show that the obtained results are quite similar. Since we have not selected any routing protocol, only those connections which go through directly connected nodes can successfully delivered. We also observe that *Castadiva* has a lower throughput than ns-2. When *Castadiva* is selected, the

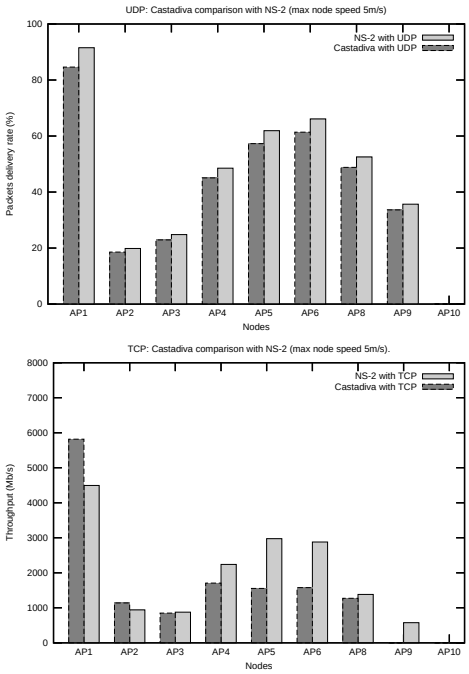

Figure 8: Performance comparison between *Castadiva* and ns-2. Node speed is 5m/s, and no routing is selected.

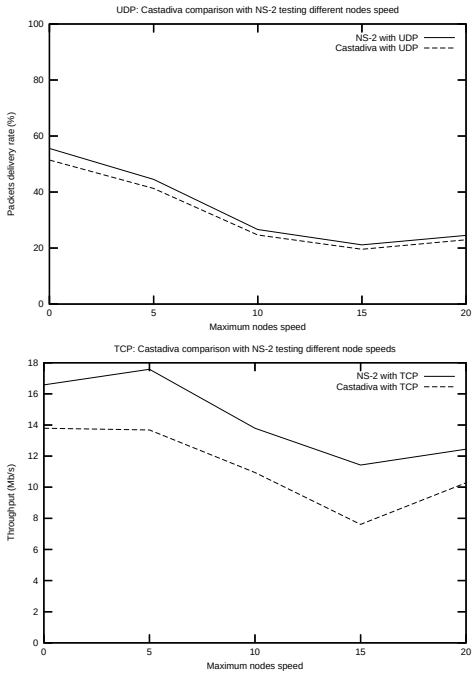

Figure 9: Impact that node speed has over the aggregated traffic in *Castadiva* and ns-2.

shared wireless media is prone to both transmission errors and contentions among stations. In the case of ns-2, only contention effects are simulated, which explains the observed discrepancy.

We evaluate now the impact of node speed over the aggregated UDP and TCP traffic. We vary node speed among 0, 5, 10, 15 and 20 m/s and repeate all the tests both in *Castadiva* and using ns-2. Figure 9 shows the obtained results. We observe that when we selected TCP traffic the differences among *Castadiva* and ns-2 increase. In TCP, when a packet is lost or it arrives out of order, it is transmitted again. We observe that the initial retransmission timeout differs between Castasdiva and ns-2 and so the obtained differences. In ns-2 the timeout is around 5 second while *Castadiva* uses a timeout of 8 simulated seconds.

We now repeat all the previous experiments but enabling the OLSR routing protocol, so that nodes behaves also as routers. We configure the OLSR parameters according to those proposed in the RFC standard. We set the various OLSR parameters as indicated in Table 1.

Parameter	Value used
HELLO_INTERVAL	2
REFRESH_INTERVAL	2
TC_INTERVAL	5
MID_INTERVAL	TC_INTERVAL
HNA_INTERVAL	TC_INTERVAL

Table 1: OLSR Parameters.

Figure 10 shows the obtained results node by node where each node has a maximum speed of 5 m/s. The average percentage of UDP packets correctly received increases since now all the selected connections can continue using a multihop communication. If we select TCP traffic, we observe that the average throughput does not increase, because the network was already saturated. However, since we are using the OLSR protocol, now the throughput is shared among all the nodes.

Finally, Figure 11 shows the obtained results when node speed varies from 0, 5, 10, 15 and 20 m/s using OLSR protocol. We observe that when UDP traffic is selected, as node speed increases, the packet delivery ratio decreases both in ns-2 and in *Castadiva*. On the other hand when the traffic is based on TCP flows, the obtained results are mainly affected by high bandwidth requirements of TCP.

5. CONCLUSIONS

In this paper we compared the results obtained with *Castadiva* and the ns-2 network simulator. *Castadiva* is a testbed architecture based on low-cost off-the-shelf devices, which is used to test protocols developed for MANETs. *Castadiva* is completely compatible with the file format used by the ns-2 simulator, so simplifying and allowing a fair comparison between the two systems.

Using both TCP and UDP data traffic and under a variety of static and dynamic MANETs scenarios we show that *Castadiva* is able to obtain confident results while using real wireless off-the-shelf devices and we also demonstrate that the ns2 simulator is a reliable tool for the evaluation of MANETs related proposal.

6. ACKNOWLEDGMENTS

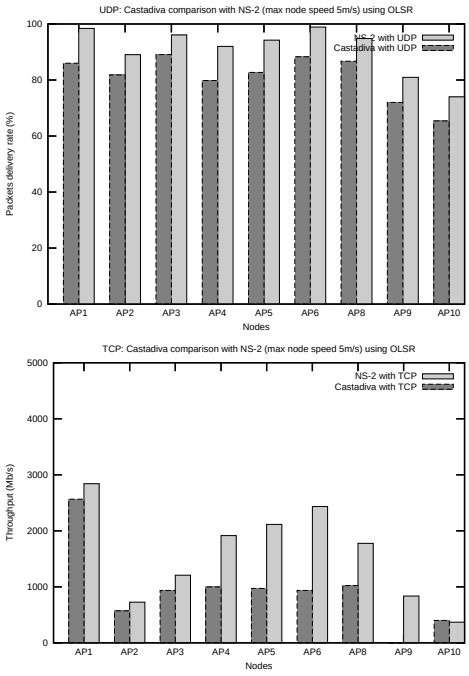

Figure 10: Performance comparison between *Castadiva* and ns-2. Node speed is 5m/s, and OLSR is selected.

This work was partially supported by the *Ministerio de Educación y Ciencia*, Spain, under Grant TIN2005-07705-C02-01 and by the *Generalitat Valenciana, Ayudas complementarias para proyectos de I+D+i*, Spain, under Grant ACOMP07/237.

7. REFERENCES

[1] S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic. *Mobile Ad Hoc Networking*. IEEE Press, 2004.

[2] USC/ISI UC Berkeley, LBL and Xerox PARC researchers. Network simulator - ns (version 2). Available at: http://www.isi.edu/nsnam/ns/, 1998.

[3] Openwrt, wireless freedom. Available at: http://openwrt.org.

[4] A. Yamanaka and JCraft Inc. Jsch, the java secure channel. Available at: http://www.jcraft.com/jsch/.

[5] B. Hubert. *Linux Advanced Routing & Traffic Control HOWTO*. Available at: http://lartc.org/, 1.43 edition, October 2003.

[6] Uppsala Universitet. Aodv-uu. Available at: http://core.it.uu.se/core.

[7] P. Jacquet T. Clausen. Optimized link state routing protocol (olsr). *RFC: 3626*, October 2003.

[8] T. M. H. Reenskaug. The model-view-controller (mvc) its past and present. In *Java Zone*, Oslo, September 2003.

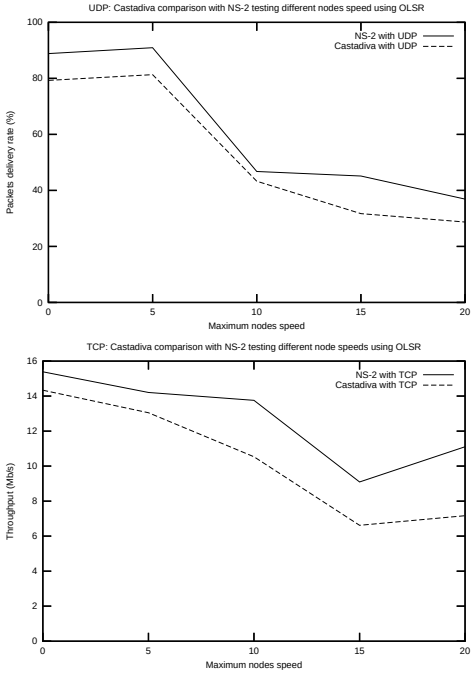

Figure 11: Impact that node speed has over the aggregated traffic in *Castadiva* and ns-2 when using the OLSR protocol.