Hindawi Publishing Corporation
EURASIP Journal on Wireless Communications and Networking
Volume 2009, Article ID 406979, 20 pages
doi:10.1155/2009/406979

Research Article

Testing Applications in MANET Environments through Emulation

Jorge Hortelano, Juan-Carlos Cano, Carlos T. Calafate, and Pietro Manzoni

Department of Computing Engineering, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain

Correspondence should be addressed to Juan-Carlos Cano, jucano@disca.upv.es

Received 22 July 2009; Revised 18 November 2009; Accepted 15 December 2009

Recommended by Amiya Nayak

Mobile Ad Hoc Network solution testing is typically done using simulation. The simulated code is generally a simplified version of the real code, and thus code porting to actual operating systems lacks a strong validation. In this work we present Castadiva, a test-bed architecture that allows validating software solutions for real ad hoc network environments using low-cost, off-the-shelf devices and open source software. Castadiva is also compatible with the ns-2 simulator, allowing a combined and more complete evaluation. In this work we present our tool and the results obtained when evaluating videocalls in different scenarios, both static and dynamic.

1. Introduction

Mobile ad hoc wireless networks (MANETs) [1] consist of mobile nodes interconnected by multihop communication paths. Unlike conventional wireless networks, ad hoc networks have no fixed network infrastructure or administrative support. Nodes belonging to a MANET can either be end-points of a data interchange or act as routers when the two end-points are not directly within their radio range. The topology of the network changes dynamically as mobile nodes join or depart the network, or when radio links between nodes become unusable. This means that users have the freedom to create their own networks, which can be deployed easily and cheaply. The importance of MANETs becomes evident by noticing the wide application area that these embrace. Special situations require communication networks to be available without any previous infrastructure, like emergency missions, military operations [2], or ad hoc meetings. However, a price for all those features is paid in terms of complex technological solutions.

MANETs require efficient routing protocols to operate correctly. A routing protocol specifies how nodes communicate with each other to disseminate routing information, allowing them to create routes between any two nodes on a network. Many of the academic papers dealing with ad hoc and mesh networks [3–6] evaluate protocols assuming varying degrees of mobility within a bounded area, usually with all nodes within a few hops of each other, and usually with nodes sending data at a constant rate. Different protocols are then evaluated based on the packet drop rate and the overhead introduced by the routing protocol. The growing research efforts focusing on this new technology require 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 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. A simulation tool is an application which behaves or operates like a given real system when provided with a set of controlled inputs. Currently several network simulators exist, including ns-2 [7], OPNET [8], Seawind [9], and REAL [10]. As shown in [11], ns-2 is the most extended in the research community. ns-2 is a discrete event network simulator that is popular in academia for its extensibility (due to its open source model). It is typically used for the simulation of MAC, routing and transport protocols, and is heavily used in ad hoc networks research. ns-2 supports an array of popular network protocols, offering simulation results for wired and wireless networks alike. Simulation offers four important advantages. First, it enables experimentation with large networks. Second, it enables experimentation with configurations that may not be possible with existing technology. Third, it allows rapid prototyping. Finally, it makes reproducible experiments in a controlled environment possible. Simulations also have some disadvantages. First, researchers can not test their own real world implementation of a protocol in a realistic scenario. Second, simulators typically do not incorporate realistic models of node mobility and radio signal propagation [11].

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

A test-bed is a platform for experimentation which allows for rigorous, transparent, and replicable testing of scientific theories, computational tools, and other new technologies [12].

In this work we present *Castadiva*, a test-bed architecture that simplifies carrying out realistic experiments; it relies on low-cost, off-the-shelf wireless routers 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 server 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 define the scenario of the network and the desired traffic connections between MANET nodes in a simple and straightforward manner. All of these tools were developed with open source software and they are freely available for the research community at http://castadiva.sourceforge.net/.

Castadiva was designed to be completely compatible with the ns-2 file format, which allows us to compare both in a simple and straightforward manner. Hence, we relied on ns-2 to validate our test-bed. We selected a range of both static and dynamic MANET scenarios and compared the results obtained using *Castadiva* with those obtained using ns-2. We confirmed that *Castadiva* is a reliable tool that provides results which are comparable, although not strictly identical, to those obtained in a realistic MANET.

After validating *Castadiva* we proceeded to assess the performance of a real videoconferencing tool in MANET environments. The application being tested is Ekiga [13], a platform offering real-time audio/video communication between users supporting both SIP [14] and H.323 [15] standards. The metrics analyzed were the packet loss rate and the end-to-end delay. We also characterized the source traffic under different conditions since the data rate generated is dynamically adjusted by Ekiga.

The rest of this work is organized as follows. Related works are presented in Section 2. Section 3 describes *Castadiva*'s architecture. Section 4 presents the implementation details. Section 5 presents the performance evaluation we have done to validate our tool, comparing this tool with the ns-2 when using TCP and UDP traffic. In Section 6 we show the results obtained with real audio/video traffic in different scenarios. Finally, in Section 7 we present our concluding remarks.

2. Related Work

The idea of automatic computer-controlled routing in a mesh network is not new; in fact the main idea dates back to Baran in the early 1964 [16]. Since then, several prototypes for generating real ad hoc network experiments were proposed and can be found in the literature.

ORBIT. An indoor radio grid emulator for controlled experimentation and an outdoor field trial network for end-user evaluations in real-world settings [17]. This emulator needs an expensive noise generator since it emulates higher 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.

mLab. An emulator that strikes a balance between desktop simulations and outdoor field tests by allowing users to develop and test ad hoc protocols and applications in a laboratory environment [18]. This emulator can only generate network topologies and capture packets.

Carnegie Mellon University Wireless Emulator. Supports real devices, applications, MAC and PHY layers on a network-wide scale while maintaining experimental control and repeatability [19]. The disadvantages of this emulator are that it does not use commercial off-the-shelf devices, using a 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 ad hoc nodes physically [20]. We discarded this emulator for our test since it relies on expensive clusters to emulate the scenario.

Seawind. Another emulator designed for performance studies of real protocols and applications on wireless networks [9]. It uses an emulated link and router to generate the network topology and noncommercial off-the-shelf devices, making it difficult to scale up the network. Also, it can only emulate one traffic flow.

WHYNET. Is a hybrid wireless testbed environment targeting realistic, scalable, and flexible evaluation of wireless technologies and applications [21]. These hybrid experimentation modes use physical and simulated elements (e.g., protocol layers, subnets) in different combinations.

Table 1: Comparative of existing emulators.

Name	GUI	Real devices	Real code	Flexibility	Size of emulator	Initial budget	Scalable	Free
Castadiva	Yes	Routers, PDAs and computers	Yes	Yes	fits in a desktop	<1000€	Yes	Yes
MobiEmu	No	Computers	Yes	Yes	fits in a room	>3000€	Yes	Yes
mLab	No	No (is almost a simulator)	No	Yes	fits in a desktop	<1000€	Yes	Yes
Carnegie Mellon University Wireless Emulator	Yes	computers, but needs FPGAs	Yes	Yes	fits in a desktop	>3000€	Yes	No
ORBIT	No	Computers	Yes	Only mobility in grid.	Huge, needs a warehouse	>10000€	Yes but very expensive	No
Seawind	Yes	Computers	No	No, only emulate two nodes	fits in a desktop	—	No	No
WHYNET	No	Hybrid simulator	Yes	Yes	fits in a desktop	<1000€	Yes (by simulation)	No

GUI: a graphical interface for easy use of the emulator. Real Devices: Which devices can be used on the emulation tests? Real Code: Can users introduce real implementations of application and protocols on its emulations without changing anything, or must they rewrite the code to fit the emulator? Flexibility: Can any topology be implemented? Size of the Emulator: Can the emulator be deployed on a simple desktop of a lab, or does it need an entire room for its deployment? Initial Budget: The initial (approx.) inversion for generating a test-bed of 10 nodes. Scalable: Is it easy to increase the number of nodes without increasing the cost significantly? Free: Can it be used freely by anyone without purchasing?

WHYNET uses a geographically distributed set of physical wireless testbeds, making results difficult to be replicated by other researchers.

We developed *Castadiva* to deploy a cheap architecture that relies on low-cost devices to generate a test-bed. The test bed obtained effectively uses low-cost hardware to achieve a complex, yet reliable wireless network simulation. Table 1 compares the different emulators available against *Castadiva*. As can be observed, only three emulators are free: *Castadiva*, MobiEmu, and mLab. mLab is almost a simulator that does not use real devices. *Castadiva* can use any devices with a wireless card, which offers flexibility when purchasing devices, thus reducing the budget to deploy the emulator. MobiEmu is more expensive than *Castadiva* and does not have a GUI. Castadiva implements a GUI to guide the users when creating the scenario and the traffic pattern.

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

3. Castadiva's Architecture Overview

Castadiva is a test-bed designed to deal with the development and the performance evaluation of protocols and applications for MANETs. The test-bed relies on actual IEEE 802.11 wireless interfaces for communication among nodes. *Castadiva* is composed by a server that runs the main application, several wireless nodes, two different networks (wired and wireless), and an application that coordinates all devices.

Castadiva's server executes the application and configures the network devices. Concerning the wireless nodes used, they can be any sort of computing device, like a laptop, a PDA, or a wireless router. In our prototype, each node is a Linksys router to minimize costs. The main requirement for a node is that it must have a Linux/Unix operating system installed and two network cards: an Ethernet card and an IEEE 802.11 card. If the node is a wireless router, the OpenWRT [22] kernel is an excellent 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 operating system. In our case, we developed some applications in C for traffic generation and control purposes.

Figure 1 shows a schema of *Castadiva*'s components. The main application, developed in Java, controls all devices and dynamically manages the links among them according to the desired network scenario. It also manages traffic generation between pairs of nodes. Since the controlling application requires communicating with nodes to send control packets, *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 run.

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, to avoid large latencies. Basically,

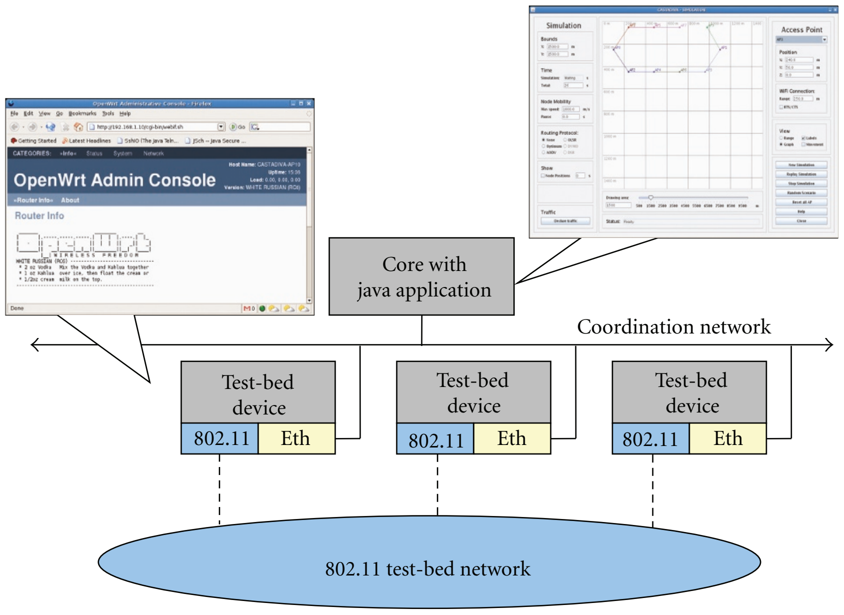


Figure 1: Schema of *Castadiva*'s components.

the coordination 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 reconfigure periodically according to the desired network topology and also to run lightweight traffic-generating applications available on each wireless node. For communication purposes, we rely on the SSH protocol [23] to send instructions through this network. Using a fast network means that all nodes participating in a test will start at about the same time, avoiding significant latency effects and maximizing result accuracy.

The wireless network is composed by *Castadiva*'s wireless nodes, and the topology of this network is defined by the GUI of *Castadiva*, so that it can change at runtime. Nodes communicate in ad hoc mode using IEEE 802.11g technology.

Castadiva's 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 fixing 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 *Simulate* button, and each physical node will be reprogrammed so as to enforce the chosen network topology. Figure 2 shows the GUI application of *Castadiva*. We describe the whole functionality offered by *Castadiva*'s GUI in Section 4.2.

Figure 3 shows our test-bed. One switch connects *Castadiva*'s server with all the wireless nodes for coordination purposes. On the center of the picture the group of wireless nodes being used is shown. It consists of eleven Linksys routers (models WRT54G and WRT54GL). The wireless ad hoc network conformed by these nodes is the one used in *Castadiva*'s testbed experiments. The experiments presented in this work required extending *Castadiva* to support traffic injected from outside applications. In particular, we connected laptops running videoconference software to our MANET, allowing us to assess the performance of video conference sessions as experienced by users.

4. 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 emulating 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. Concerning nodes, each one must be a Linux-based system with an SSH server and an NFS client; besides, they must have 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 GUI application by defining the network topology and the traffic and selecting the desired routing protocol. Then, through SSH, the application sends a *start* instruction to each node through the coordination network (wired).

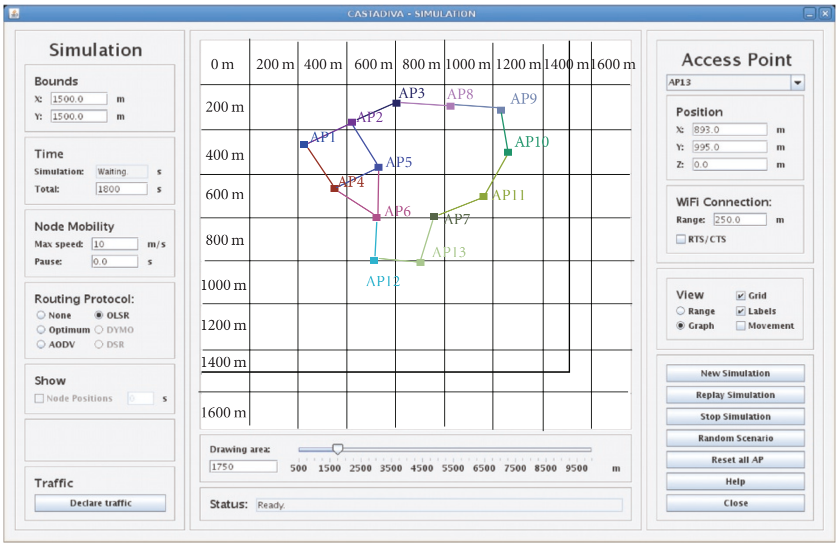

```
Simulation
Bounds
X: 1500.0 m
Y: 1500.0 m
Time
Simulation: Waiting. s
Total: 1800 s
Node Mobility
Max speed: 10 m/s
Pause: 0.0 s
Routing Protocol: ○ None  ● OLSR  ○ Optimum  ○ DYMO  ○ AODV  ○ DSR
Show
☐ Node Positions 0 s
Drawing area: 1750
Status: Ready.
Access Point
AP13
Position
X: 893.0 m
Y: 995.0 m
Z: 0.0 m
WiFi Connection:
Range: 250.0 m
☐ RTS/CTS
View  ○ Range  ● Graph  ☑ Grid  ☑ Labels  ☐ Movement
```

Figure 2: Scenario definition with *Castadiva*.

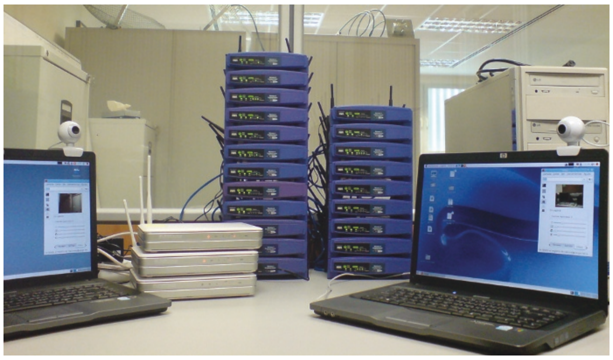


Figure 3: *Castadiva*'s physical network.

Wireless nodes achieve coordination among themselves by executing the required binaries, which are stored in a server folder shared through NFS. This is an easy way to spread instructions to all nodes, and it also solves storage limitation problems on nodes. When tests start, a group of files with the results is created and stored into *Castadiva*'s server. Each individual router accesses its own configuration file by 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, thus obtaining the different testbed statistics. Finally, the application displays results to the user.

Next we present the *Castadiva*'s implementation, which can be divided into two parts: the light-weight applications running on wireless nodes, and the main application.

4.1. Wireless Nodes' Software. Each node has a set of requirements that must be met for successful operation: a Linux-based operating system, a set of special-purpose scripts, and 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 exists in most Linux distributions, while the other two were developed by us.

Network topology configuration relies on the Iptables [24] 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 created the UdpFlow and TcpFlow tools. Both tools are designed to generate a traffic flow between two nodes, being that each tool creates a single traffic flow. 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 AODV and OLSR, that are included with the OpenWRT and initiated according to user settings.

4.2. The Main Application. *Castadiva*'s core element, a Java application running at the server, includes all the control functions required for testbed experimentation. It is responsible for network topology maintenance, traffic control, as well as result calculation and presentation. A user can define the characteristics of wireless nodes. Each node is deployed at a specific position in a simulated area as chosen by the user.

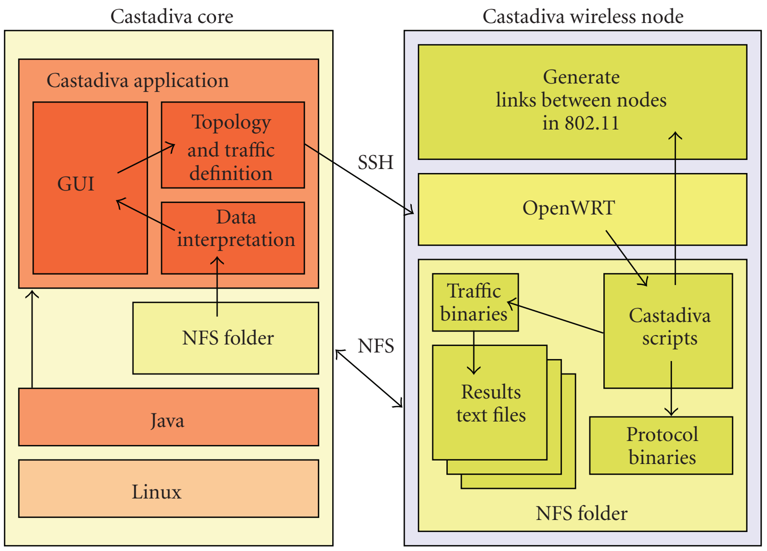


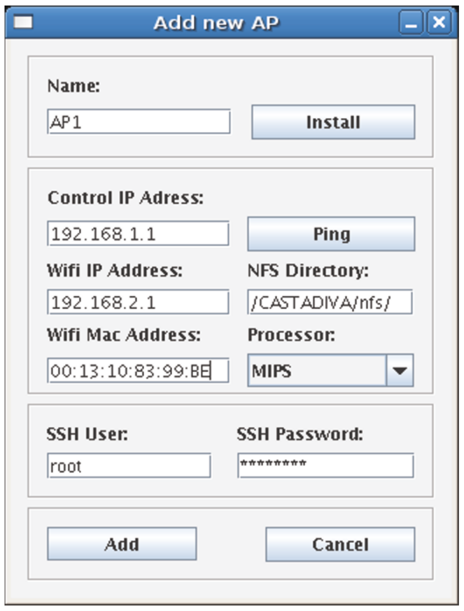
FIGURE 4: Software components for *Castadiva*.

Once the topology is defined, *Castadiva* must configure 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 the experiments are finished, *Castadiva*'s core must calculate the result statistics for the experiment by gathering all the data obtained and finally show 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.

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 automatically at the end of the simulation. We now offer more details about the services offered by *Castadiva*.



Name: AP1

Control IP Adress: 192.168.1.1

Wifi IP Address: 192.168.2.1

NFS Directory: /CASTADIVA/nfs/

Wifi Mac Address: 00:13:10:83:99:BE

Processor: MIPS

SSH User: root

SSH Password: *********

FIGURE 5: Node configuration interface.

4.2.1. 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. Figure 5 shows an example of the definition of a node in the system.

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 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 run at each node and must be compiled for all types of CPUs 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 the routers.

4.2.2. Ad Hoc Network Scenario Generation. Once all the nodes are defined, they can be configured 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 adequate option is selected, the radio communication range is also shown through a circle of the same colour.

Figure 6 shows the topology generation environment. We can see ten nodes located in a scenario of 850×1000 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 of each node. The group of buttons appearing below allows 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, later on, 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.

4.2.3. Mobility in Castadiva. Castadiva offers the user two different approaches for emulating mobility: (a) any scenario from the ns-2 simulator could be directly used within *Castadiva*, and (b) the user can take advantage of the GUI to select their own mobility model. We already included the random way point mobility model [25]. This behavior is similar to the one provided by the "setdest" tool embedded in the ns-2 simulator: if a user picks a value greater than zero in the speed option, each node acquires a random movement with a speed between zero and the inserted value. When a node arrives to a destination point, it waits for a selected pause time and then select a new random destination point to move to. *Castadiva* allows users to easily add new mobility models implementations.

For the emulation of mobility, *Castadiva* generates all node movements required for the test before it starts. In particular it generates, for each node, a mobility vector according to the selected user option in the GUI or imports it from an ns-2 scenario. Also, *Castadiva* calculates the variation of the topology continuously in time. Therefore, our emulator obtains the new topology of the network, updating the wireless links on each real device whenever necessary. The granularity used to upgrade the wireless links is of one second, but it can be changed. Since the real devices do no experience real mobility, we emulate the wireless links connectivity between nodes using the Iptables tool. This tool allows each node of *Castadiva* to only receive packets from nodes that are within range according to the emulated topology, and thus it blocks the reception of packets when nodes are out of range. All Iptables rules are stored into script files. These scripts will be loaded on each node through NFS when the simulation starts. Figure 7 shows the file loaded by AP1. We can see in this figure different Iptables rules interleaved with *sleep* rules. The sleep time allows enforcing Iptables' rules at suitable times. Hence, each rule is loaded only when the emulation requires a node to change its connectivity state towards other node. For example, Algorithm 1 shows what is the behaviour of *Castadiva* when a node (with MAC 00:14:BF:3C:39:EC) goes out of range at second 15 and comes back into range at second 35.

Castadiva also allows a user to see all node positions at a certain instant of time. When a simulation finishes, the user can activate the *Show* option and pick an instant of time. Immediately *Castadiva* shows the network topology at that time. This option is useful to do a later evaluation of the changes occurred in the network topology when mobility was activated.

4.2.4. 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 in red. Examples of parameters are *packet size*, *packets per second*, *start time*, *end time,* and *maximum number of packets sent*. Figure 8 shows a usage example of this tool, where rows define seven traffic connections. It contains some helpful buttons that allow making row operations (delete, order by starting time, or copy). Traffic settings are exportable to ns-2 format also.

When an experiment finishes, *Castadiva* fills this table with results, including the average throughput, in the case of TCP traffic, or the percentage of packets received in the case of UDP traffic.

4.2.5. Random Test Generator. Sometimes it is useful to automate the testbed 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 numbers of nodes and randomly generated network topologies. This is achieved through the *Random test window* shown in Figure 9.

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 cannot be modified. Again, all the tests can be stored in either *Castadiva*'s orns-2's format.

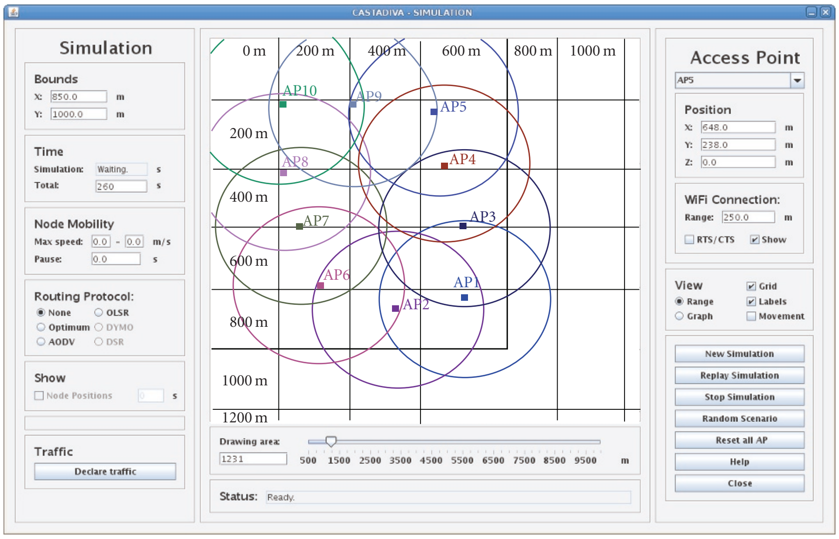
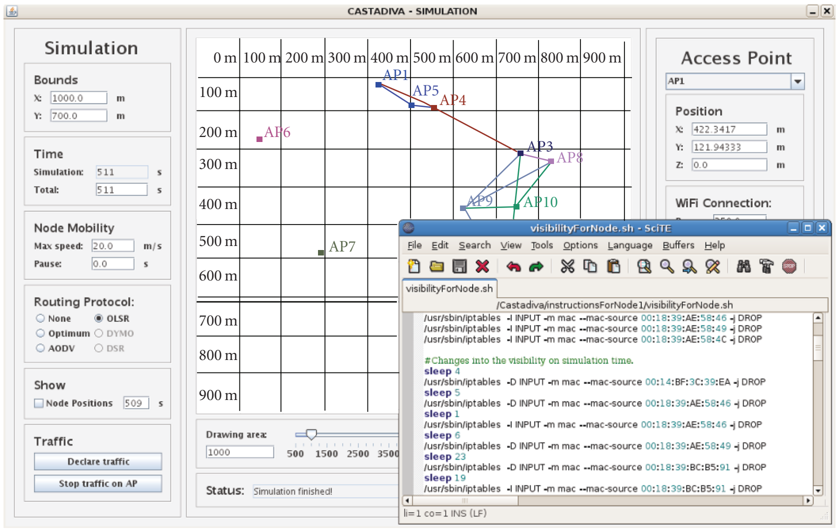


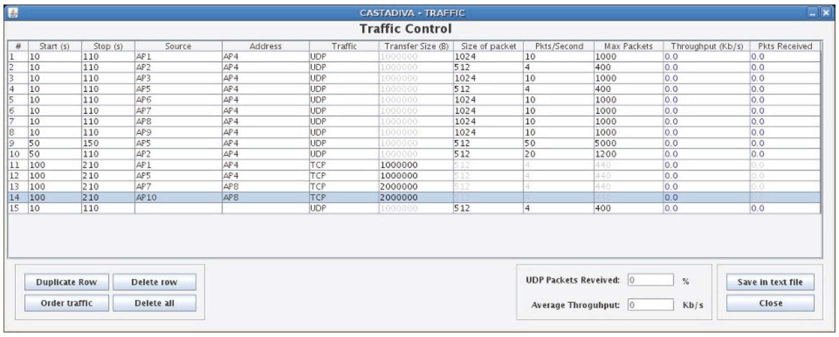
Figure 6: Scenario definition with *Castadiva*.



```
/usr/sbin/iptables  -I INPUT -m mac --mac-source 00:18:39:AE:58:46 -j DROP
/usr/sbin/iptables  -I INPUT -m mac --mac-source 00:18:39:AE:58:49 -j DROP
/usr/sbin/iptables  -I INPUT -m mac --mac-source 00:18:39:AE:58:4C -j DROP

#Changes into the visibility on simulation time.
sleep 4
/usr/sbin/iptables  -D INPUT -m mac --mac-source 00:14:BF:3C:39:EA -j DROP
sleep 5
/usr/sbin/iptables  -D INPUT -m mac --mac-source 00:18:39:AE:58:46 -j DROP
sleep 1
/usr/sbin/iptables  -I INPUT -m mac --mac-source 00:18:39:AE:58:46 -j DROP
sleep 6
/usr/sbin/iptables  -D INPUT -m mac --mac-source 00:18:39:AE:58:49 -j DROP
sleep 23
/usr/sbin/iptables  -D INPUT -m mac --mac-source 00:18:39:BC:B5:91 -j DROP
sleep 19
/usr/sbin/iptables  -I INPUT -m mac --mac-source 00:18:39:BC:B5:91 -j DROP
```

Figure 7: Mobility implementation.



Traffic Control

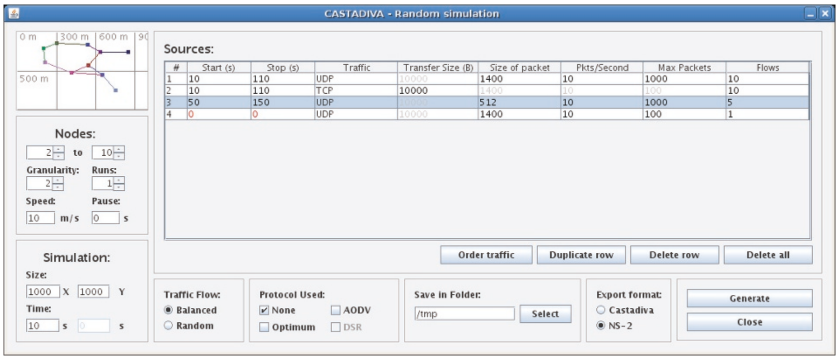
#	Start (s)	Stop (s)	Source	Address	Traffic	Transfer Size (B)	Size of packet	Pkts/Second	Max Packets	Throughput (Kb/s)	Pkts Received
1	10	110	AP1	AP4	UDP	1000000	1024	10	1000	0.0	0.0
2	10	110	AP2	AP4	UDP	1000000	512	4	400	0.0	0.0
3	10	110	AP3	AP4	UDP	1000000	1024	10	1000	0.0	0.0
4	10	110	AP5	AP4	UDP	1000000	512	4	400	0.0	0.0
5	10	110	AP6	AP4	UDP	1000000	1024	10	1000	0.0	0.0
6	10	110	AP7	AP4	UDP	1000000	1024	10	1000	0.0	0.0
7	10	110	AP8	AP4	UDP	1000000	1024	10	1000	0.0	0.0
8	10	110	AP9	AP4	UDP	1000000	1024	10	1000	0.0	0.0
9	50	150	AP5	AP4	UDP	1000000	512	50	5000	0.0	0.0
10	50	110	AP2	AP4	UDP	1000000	512	20	1200	0.0	0.0
11	100	210	AP1	AP4	TCP	1000000	512	4	440	0.0	0.0
12	100	210	AP5	AP4	TCP	1000000	512	4	440	0.0	0.0
13	100	210	AP7	AP8	TCP	2000000	512	4	440	0.0	0.0
14	100	210	AP10	AP8	TCP	2000000	512	4	440	0.0	0.0
15	10	110			UDP	1000000	512	4	400	0.0	0.0

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

Figure 8: Traffic declaration tool.

```
sleep 15
iptables -I INPUT -m mac –mac-source 00:14:BF:3C:39:EC -j DROP
sleep 20
iptables -D INPUT -m mac –mac-source 00:14:BF:3C:39:EC -j DROP
```

Algorithm 1: Iptables rules to emulate when a node goes out of range between seconds 15 and 35.



Sources:

#	Start (s)	Stop (s)	Traffic	Transfer Size (B)	Size of packet	Pkts/Second	Max Packets	Flows
1	10	110	UDP	10000	1400	10	1000	10
2	10	110	TCP	10000	1400	10	100	10
3	50	150	UDP	10000	512	10	1000	5
4	0	0	UDP	10000	1400	10	100	1

Nodes: 2 to 10
Granularity: 2 Runs: 1
Speed: 10 m/s Pause: 0 s

Simulation:
Size: 1000 X 1000 Y
Time: 10 s 0 s

Traffic Flow: [✓ Balanced] Random
Protocol Used: [✓ None] AODV Optimum DSR
Save in Folder: /tmp
Export format: Castadiva [✓ NS-2]

Figure 9: Random test window.

4.2.6. Compatibility with the ns-2 Simulator. *Castadiva* can also import/export scenarios to/from ns-2, making it compatible with the most widely used MANET simulator. This characteristic offers the possibility of comparing results and reaching more meaningful conclusions.

4.2.7. Castadiva Extensions for External Traffic Injection. Such functionality allows external nodes to generate real traffic of any kind and redirect it to specific nodes of *Castadiva*. For this reason *Castadiva* also incorporates an extra component that allows attaching a laptop or a computer to a node. Figure 10 shows an example of this functionality.

This way we can use laptops to generate any flow of traffic and redirect it to specific nodes of *Castadiva*. For example, you can use real applications like Ekiga to launch a videoconference and study the behaviour of H.323, SIP, and video streaming protocols in MANETs.

Figure 11 shows an example of two laptops connected to *Castadiva*. Both laptops have a webcam and run the Ekiga application to generate a videocall. *Castadiva* redirects all traffic related to this video-conference through the emulated MANET, from the entry to the output point.

5. Validation of Castadiva

In this section we validate the functionality and accuracy of *Castadiva* in different MANET scenarios. We divide our tests in two groups: first, we compare *Castadiva* with the ns-2 simulator using both TCP and UDP traffic in a static scenario; second, we then proceed with a similar analysis in a mobile scenario.

5.1. Evaluation of Castadiva with a Static Scenario. To validate the functionality of the proposed tool, and to test its effectiveness, we have chosen a representative scenario where nodes are located in a way such that the maximum number of hops between nodes is of four. The topology used in our evaluation is shown in Figure 12.

The scenario is defined in a 1000 m × 700 m area, and the test time is of 100 seconds. The selected scenario was generated by ns-2 and imported to *Castadiva*.

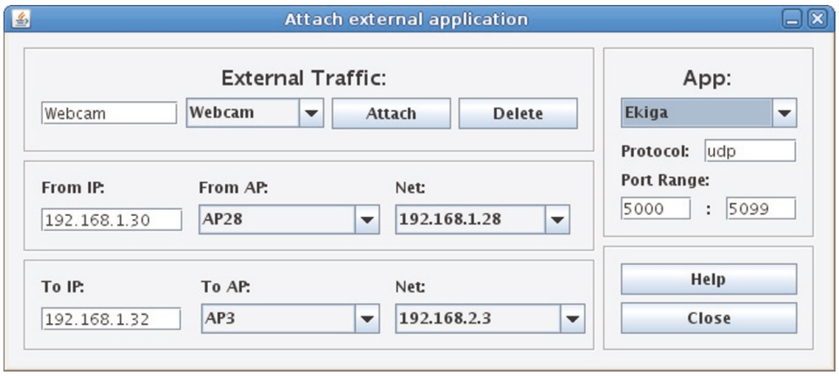
Since *Castadiva* is completely compatible with the ns-2 file format for scenarios, we were able to compare both in a simple and straightforward manner.

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 6. For TCP connections, the maximum transfer size is of 100 MB. UDP flows generate 55 packets per second, and packet size is fixed at 512 bytes.

In our first evaluation no routing protocol is being used. Figure 13 shows the results obtained in these tests. We can see that, as expected, wireless nodes that are out-of-range from node 6 are not able to communicate with it. This shows that both network topology and traffic definitions of *Castadiva* are being enforced correctly.

On both tests we observe that *Castadiva* has a lower throughput than ns-2. We must take into account that the shared wireless media is prone to both transmission errors and contention among stations, which is due to the wireless devices being placed physically close to each other. In the case of ns-2, only contention effects are simulated, which explains the observed discrepancy.

We now repeat the previous experiment with routing and forwarding enabled. We pick the Optimum routing option, so that *Castadiva* is responsible for calculating the best route to reach a destination node and modifying the routing tables of nodes to enforce the chosen topology. Figure 14 shows the results for this test.



Protocol: udp
Port Range: 5000 : 5099

From IP: 192.168.1.30
From AP: AP28
Net: 192.168.1.28

To IP: 192.168.1.32
To AP: AP3
Net: 192.168.2.3

Figure 10: External traffic declaration.

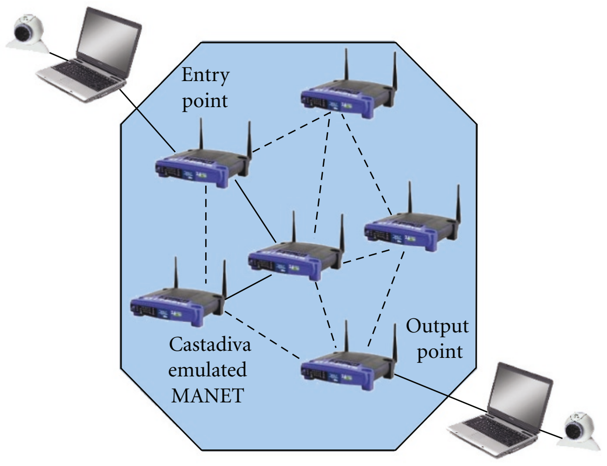


Figure 11: Example of how to add external traffic injection.

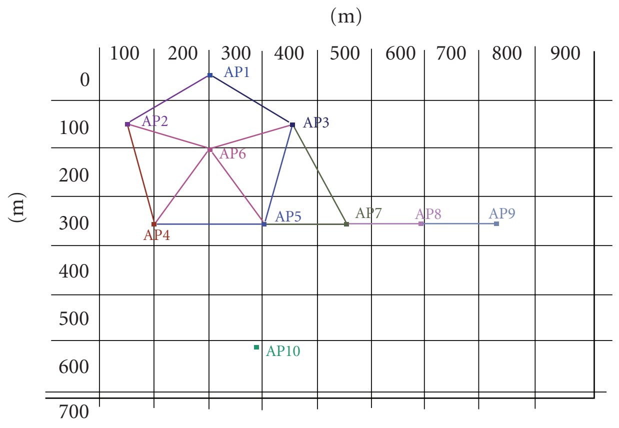


Figure 12: Scenario used for evaluation purposes.

UDP tests show that traffic from nodes AP7, AP8, and AP9 is now able to reach the destination, while node AP10 remains isolated as intended. Notice that, for the former nodes, the packet loss ratio increases slightly with the number of hops to destination. With ns-2 we do not observe this behaviour for the reasons referred above.

Results with TCP traffic show that the throughput for nodes 1 to 5 is reduced compared to the previous experiment (Figure 13), which is due to competing traffic from AP7, AP8, and AP9. For these nodes throughput decreases with increasing number of hops, as expected. With ns-2 we observe that both AP8 and AP9 suffer from starvation; one of the reasons for this behaviour is that, with *Castadiva*, all nodes share a same medium and so packet collisions between out-of-range nodes do not occur.

Figure 15(a) shows how the effect of collisions as the number of hops increases in a 802.11 g wireless network. We make different tests, varying the load from 256 kb/s to 3 Mb/s. As can be seen, the number of hops makes the packet loss increase. This behaviour is normal in a wireless network due to interferences among devices but in *Castadiva* it can be aggravate by the fact that all devices are close to one another. To better study how the proximity of the devices of *Castadiva* affects performance, Figure 15(b) compares the capacity offered by *Castadiva* with respect to other capacity models proposed in the literature for different scenarios. We select the following scenarios: (a) a grid topology [26], (b) a random topology [26], and (c) a chain topology [27]. We observe that, as expected, the capacity of the network obtained in *Castadiva* is lower than the other

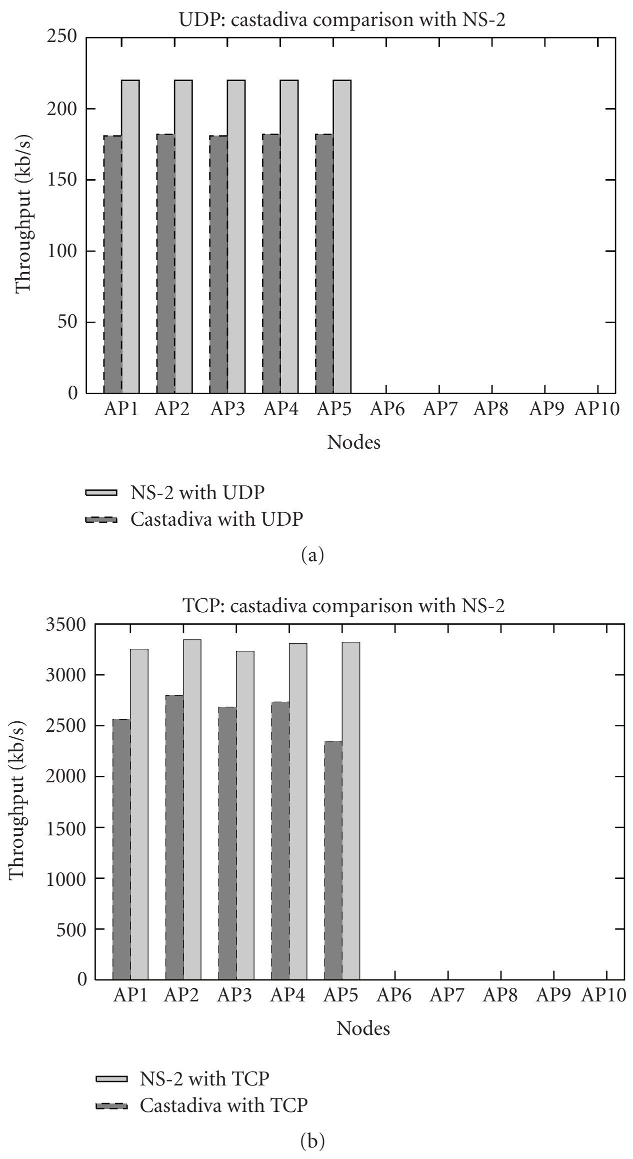


Figure 13: Performance comparison between *Castadiva* and ns-2 in a static scenario using CBR/UDP traffic (a) and FTP/TCP traffic (b). (routing disabled).

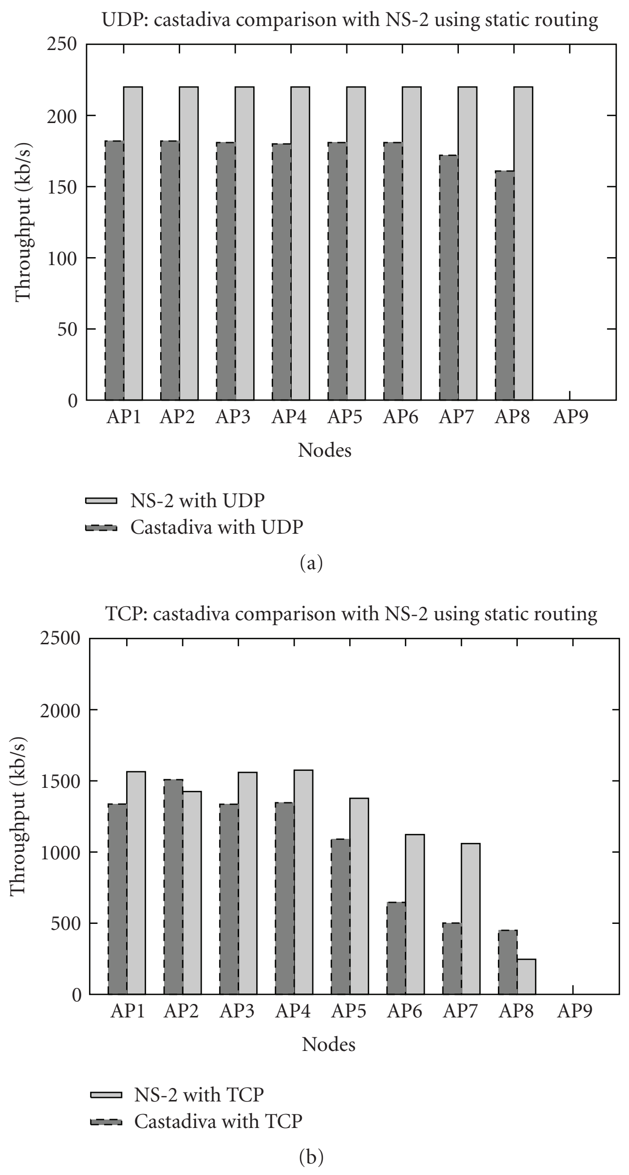


Figure 14: Performance comparison between *Castadiva* with ns-2 in a static scenario. Using CBR/UDP traffic (a) and FTP/TCP traffic (b). Routing enabled.

ones since all devices of *Castadiva* interfere with each other. Solutions such as ORBIT try to cope with this problem by mitigating the effect of the noise among nodes; however, it introduces new problems into the emulation as limiting the network topology and limiting the transmission range of all nodes, which is also far from the real behaviour of a MANET. *Castadiva* makes a trade-off between accuracy and price, offering a cheap and portable platform that is good enough for almost all tests.

In terms of the control network (wired), we observe that the bandwidth consumption of the SSH protocol over Ethernet is far from approaching saturation, and that latency is low enough to allow adequate coordination of all nodes.

5.2. Evaluation of Castadiva with a Mobile Scenario. We now define a mobile scenario with an area of 1000 m $\times$ 700 m and simulation time of 500 seconds and 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 second, and packet size is fixed at 512 bytes. In the first test, each node has a maximum speed of 5 m/s and no routing protocol is used.

Figure 16 shows a node-by-node comparison between *Castadiva* and ns-2 for both UDP and TCP traffic. The selected scenario was generated by ns-2 and imported to *Castadiva* to have the same mobility pattern.

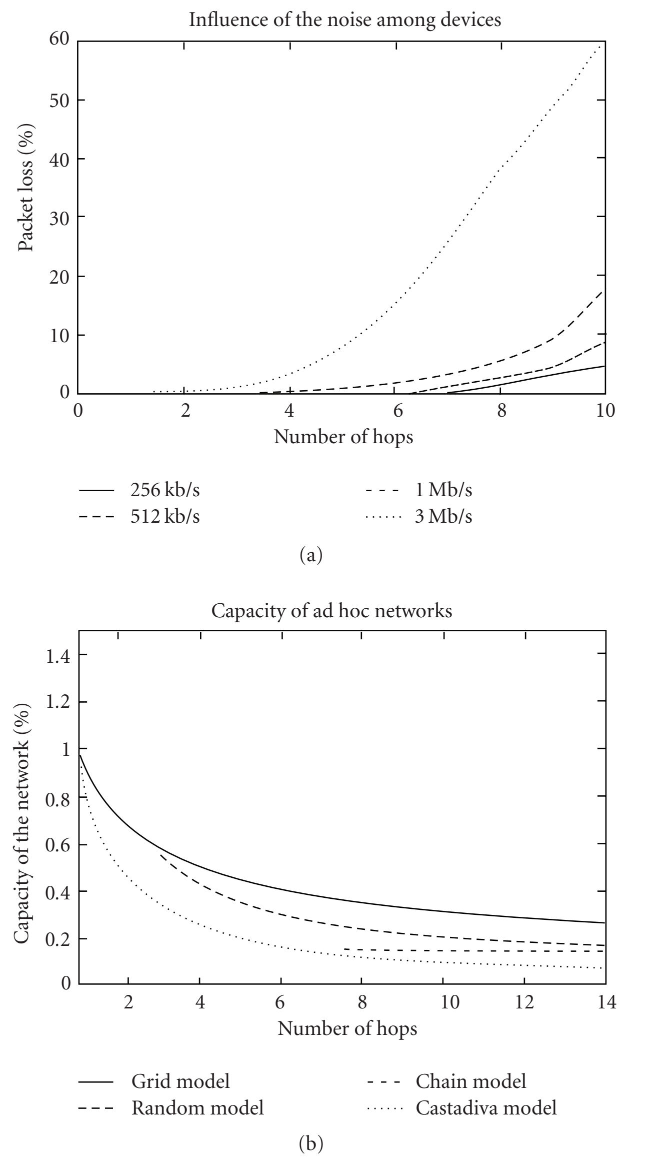


Figure 15: Packet loss due to the proximity of the devices in an emulation (a) and capacity of an ad hoc network compared with Castadiva (b).

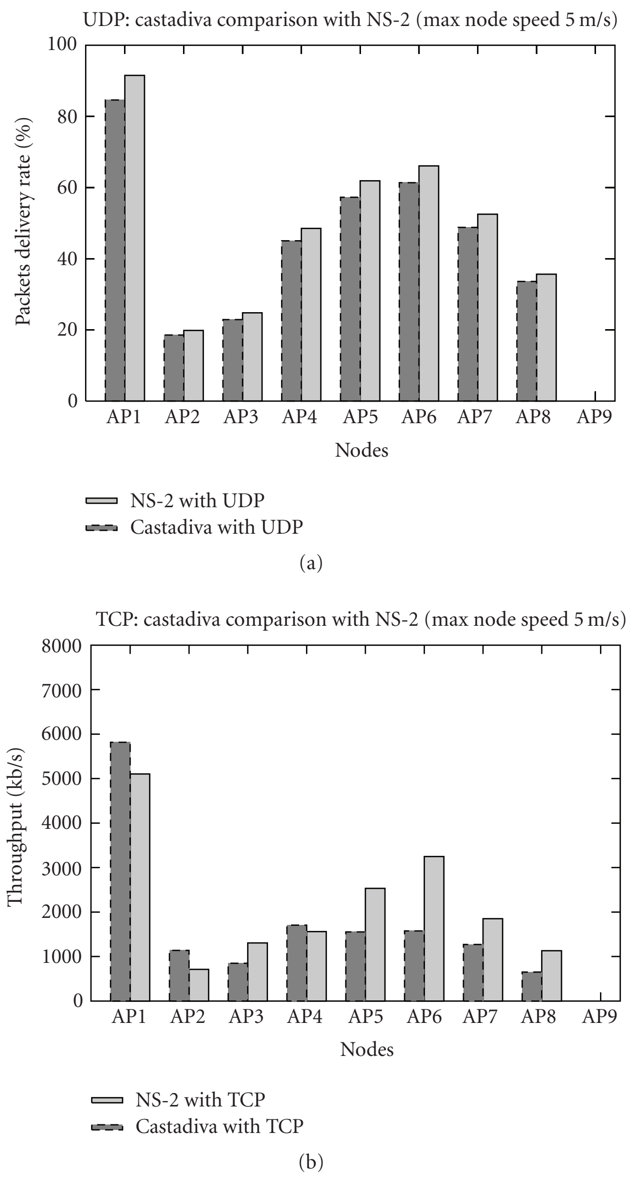


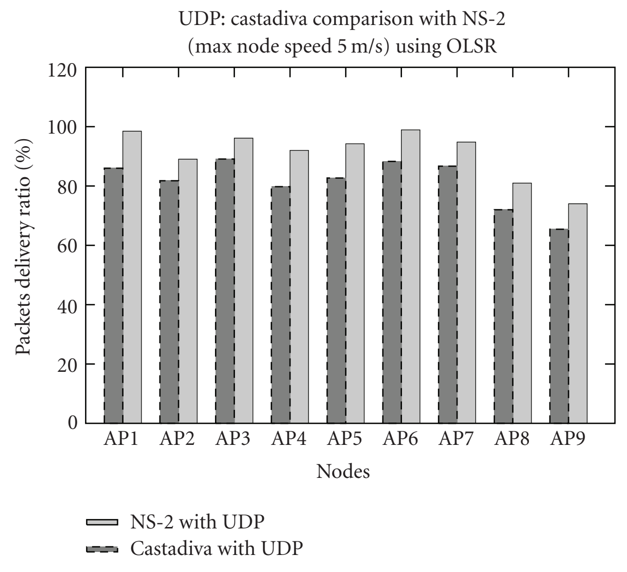
Figure 16: Result comparison of *Castadiva* with ns-2 without routing.

The figure shows that the obtained results are quite similar, which validates *Castadiva*'s implementation. Since we have not selected any routing protocol, transmissions are successful only when the destination is a 1-hop neighbour. We also observe that *Castadiva* has, in general, a lower throughput/delivery rate than ns-2. When *Castadiva* is used, the 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 for UDP traffic. In the case of TCP traffic, results are more heterogeneous since it is a stateful, bandwidth-greedy protocol prone to present nonlinearities, specially in mobile ad hoc network environments. In terms of routing, we picked the OLSR routing protocol since its implementation is available for both testing environments (simulated and real), being quite similar.

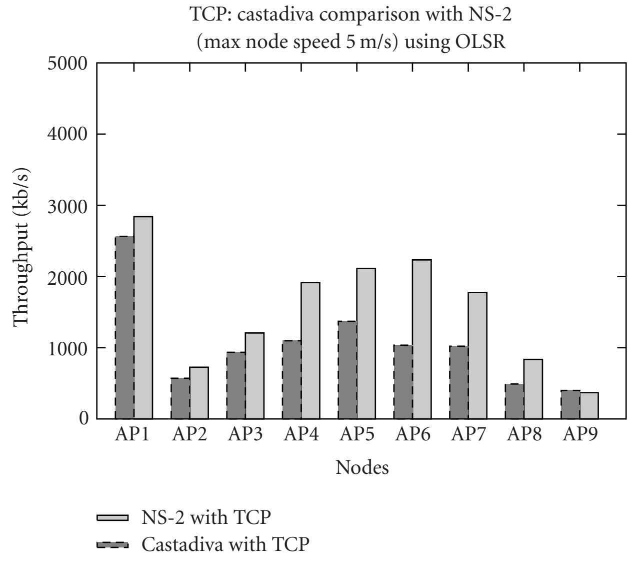
Concerning OLSR-related parameters choices, we use the values proposed in the RFC, shown in Table 2 for the reader's convenience.

Table 2: Default OLSR parameter values.

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

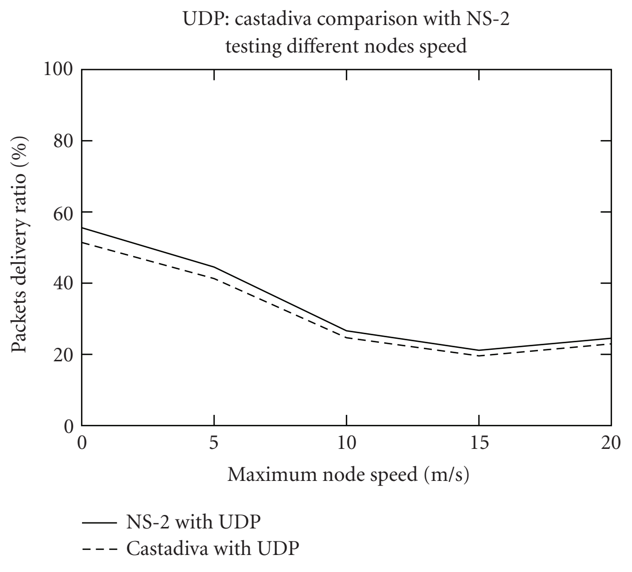


(a)

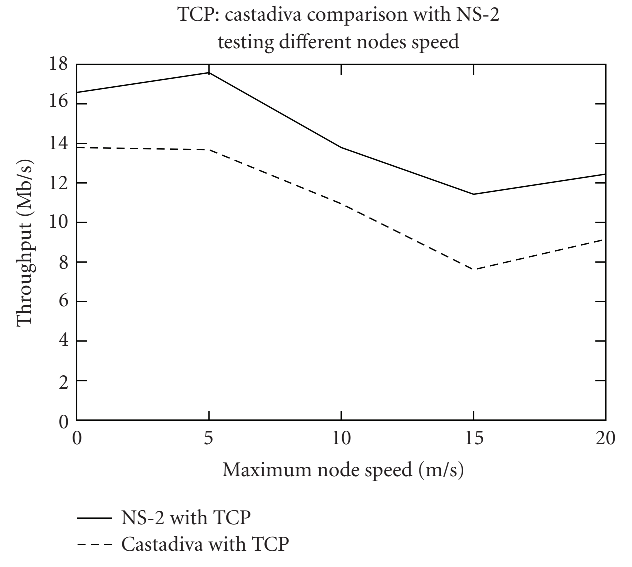


(b)

FIGURE 17: Result comparison of *Castadiva* with ns-2 for UDP (a) and TCP (b) traffic with routing.



(a)



(b)

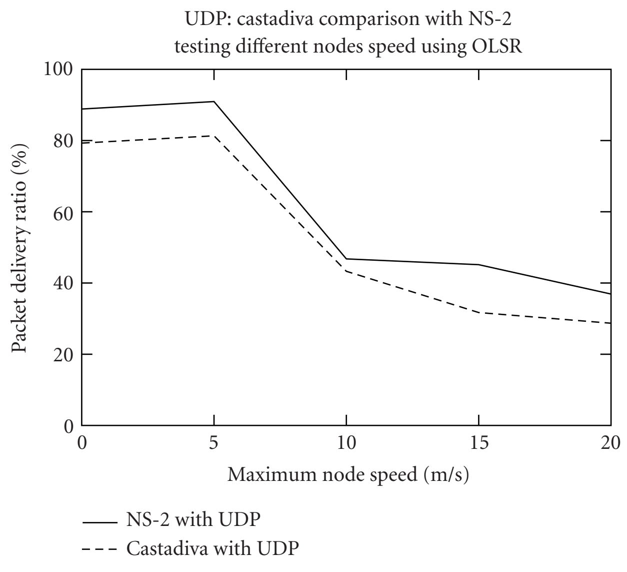
FIGURE 18: Comparison of *Castadiva* and ns-2 at different node speeds with both UDP (a) and TCP (b) traffic. (routing disabled).

Figure 17 shows the similitude between *Castadiva* and ns-2 for both UDP and TCP traffic. In these tests, nodes virtually situated more than 1-hop away are also able to send traffic to the destination node thanks to the routing protocol. Similarly to the test without a routing protocol, *Castadiva* has a lower throughput/delivering rate than ns-2, for the same reasons explained before. In the case of TCP traffic, results are more heterogeneous as in the test without routing.

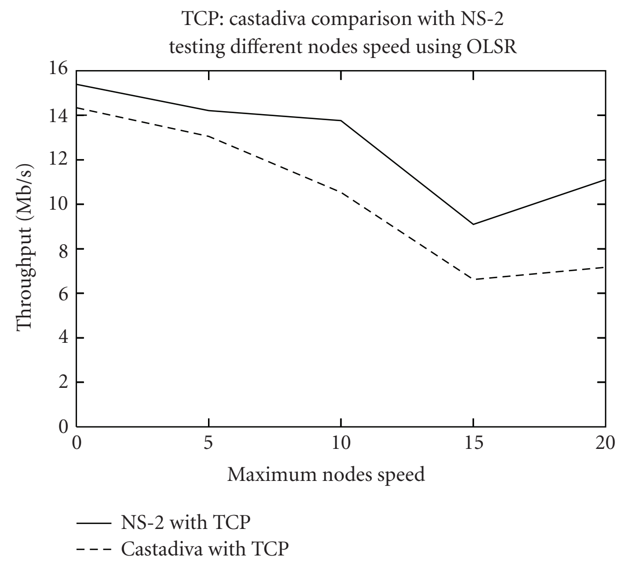
We now evaluate the impact of node speed. To do that we vary the degree of mobility in different scenarios, testing with maximum node speeds of 0, 5, 10, 15, and 20 m/s. As for the previous test, each scenario was generated by ns-2 and imported to *Castadiva* to have exactly the same mobility patterns. Figure 18 shows the results obtained in this test.

On both tests we observe that *Castadiva* has a lower throughput than ns-2. For TCP we observe that the difference between *Castadiva* and ns-2 is more significant than for UDP. To discover the reason of this, we study a controlled scenario with only two nodes and measure the arrival time of each packet. With this experiment we obtain the mean delay to do a rerouting when the OLSR performs a topology update. In ns-2 it is of five seconds. However, our nodes have a mean delay of almost eight seconds. This three-second difference causes TCP agents in ns-2 to achieve a higher throughput.

We now repeat the previous experiment with routing and forwarding enabled. Figure 19 shows the average percentage of packets received for maximum node speeds of 0, 5, 10, 15, and 20 m/s using the OLSR protocol.



(a)



(b)

Figure 19: Comparison of *Castadiva* and ns-2 using OLSR at different node speeds with both UDP (a) and TCP (b) traffic.

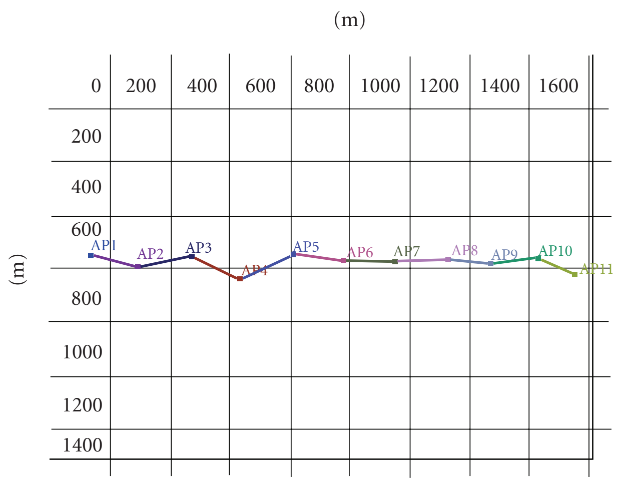


Figure 20: Topology for evaluating video traffic delivery.

Again we observe important similarities between *Castadiva* and ns-2's results. For both UDP and TCP traffic the behaviour of these platforms is quite similar. With *Castadiva* we achieve slightly less throughput than with ns-2 for the reasons explained in Section 5.1. Tests show that the average percentage of UDP packets received is increased when a routing protocol is used, since it allows nodes AP9 and AP10 to reach their destinations. When studying the behaviour of the network using TCP traffic instead we observe that the average throughput is not increased because, in both simulations, the network is saturated. When looking at the overall trend, though, we find that there is a high degree of resemblance between both.

6. Assessing the Performance of Videoconferencing in MANETS with Castadiva

We now describe the performance results obtained when transmitting real-time video traffic through *Castadiva*. We test with several different scenarios where the purpose is to stress the application and evaluate the impact of the number of hops in the quality of the received video stream. We are going to employ two different kinds of video streams: a standard videocall, where the video sequence is almost static, and a more dynamic videocall, where we will point the webcams towards a screen displaying a movie.

Similarly to the previous section, we will perform tests in both static and mobile scenarios.

6.1. Static Scenario Evaluation. Initially we evaluate the performance of a videocall in a static MANET for reference.

The scenario is defined in a 1500 m × 1800 m area, and the test time is of 5000 seconds. We set the wireless nodes' range to 250 meters. To monitor the video traffic we use the tcpdump application [28].

To generate the video traffic we have two laptops with a webcam attached and running Ekiga, an open source VoIP, and videocall application for Linux.

We generate an emulated MANET topology where all nodes are aligned according to a chain topology. Figure 20 shows the topology used when eleven nodes are deployed. In each laptop we redirect all traffic sent to the other laptop through the wireless routers located at both edges of the simulation (AP1 and AP11 in this scenario). We also configure *Castadiva* to route all video traffic through our testbed.

Each node runs the OLSR daemon to obtain consistent routing tables. We initially used the values proposed in its RFC, but the results obtained in static environments were not good since the OLSR was unable to find the routes or the routes were lost too quickly due to the network congestion.

Table 3: OpenWRT parameters values for the OLSR protocol.

Parameter	Used values
HELLO_INTERVAL	5 s
HELLO_VALIDITY	15 s
TC_INTERVAL	2 s
TC_VALIDITY	15 s
MID_INTERVAL	15 s
MID_VALIDITY	300 s
HNA_INTERVAL	15 s
HNA_VALIDITY	300 s

Therefore, instead of using the default RFC parameters, we used the default configuration of OLSR implemented in the OpenWRT system, that are considered optimum for a standard mesh network. Table 3 shows these values.

The effects we want to study are the variations in terms of throughput, delay and jitter when varying the numbers of hops in the network.

6.1.1. Evaluating the Performance with a Standard Videocall. We now present the results obtained when using *Castadiva* to evaluate a standard videocall between two users. Based on traffic traces at source and destination, we measured the throughput and the interpacket delay (jitter).

Figure 21(a) shows the mean throughput obtained for the generated data rate at different number of hops. As we increase the number of hops, we appreciate that the average throughput is decreased to less than a twentieth compared to the one hop scenario (from around of 700 Kb/s to close to 30 Kb/s). Figure 21(b) shows how the packet loss increases when we vary the number of intermediate hops between sender and receiver. In a scenario with more than 9 hops the packet loss rate is significant for the videocall. In particular, for a scenario of 10 hops, the videocall experiences a 10% of packet loss.

Figure 22 shows the cumulative distribution function for the interpacket generation interval and interpacket arrival interval in two scenarios where source and destination are one and twelve hops away, respectively. In the scenario with one hop, the smallest network possible, we can appreciate the differences between the minimum interpacket generation time at the source (around 0.01 millisecond) and the minimum interpacket arrival time at the receiver (around 0.1 millisecond). Also, 50% of the packets have an interpacket generation time of less than 0.03 millisecond, but the interpacket arrival time is typically of more than 1 millisecond.

In the scenario with ten hops, if we compare the interpacket arrival time with the one obtained in the one hop scenario, we can appreciate an increase of more than 2 milliseconds. This is caused by the forwarding time of the additional nodes on the path. This is expected since these packets, usually generated back-to-back by the videoconferencing application, experience significant jitter when traversing an ad hoc network with a high number of hops.

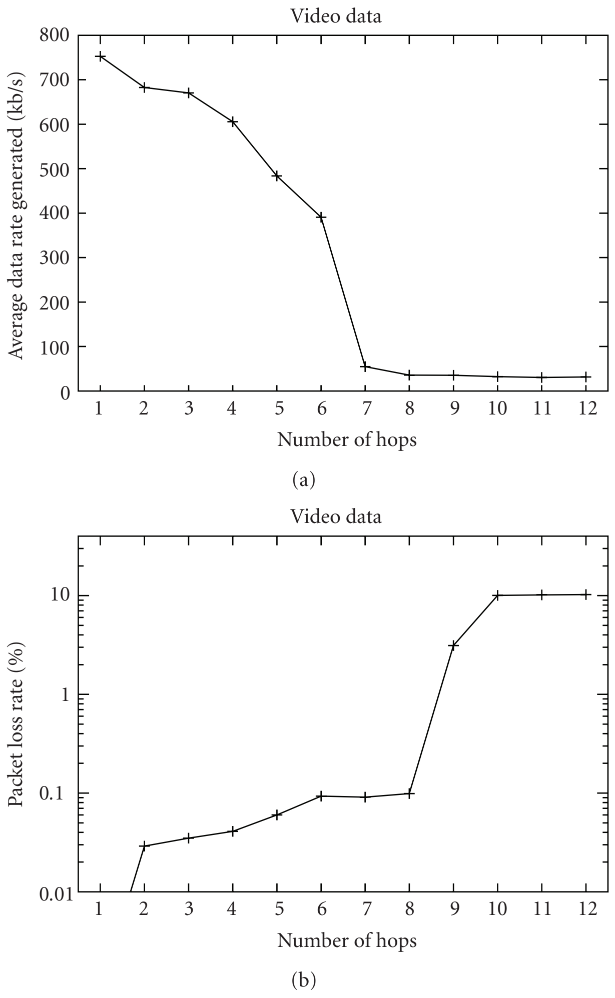
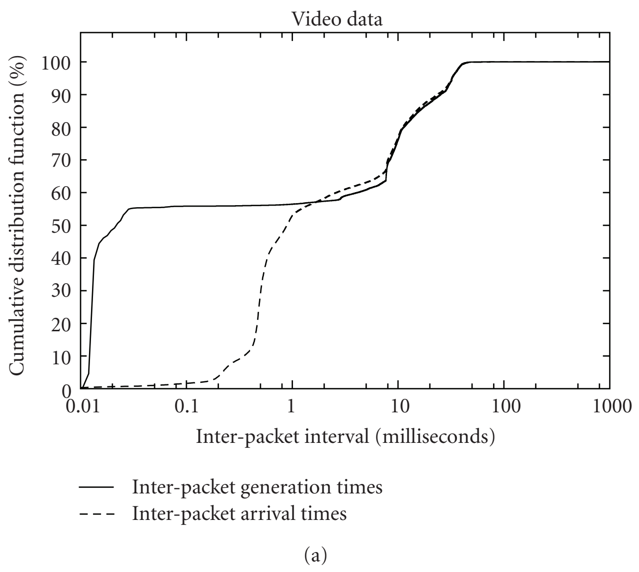


Figure 21: Average data rate generated (a) and packet loss ratio (b) for different numbers of hops.

6.1.2. Evaluating the Performance of a Movie Transmission. In the previous section our experiments relied on a standard videocall. The characteristics of such videoconference—low degrees of video motion—do not impose significant demands in terms of network bandwidth. Therefore, we repeat our experiments with a higher motion video. With that purpose we pointed both webcams (each vinculated to a communication endpoint) to a screen showing a movie. Such strategy also allows running long experiments without the intervention of users. By repeating our experiments and taking long sampling periods we were also able to obtain meaningful values for the end-to-end delay, and so we include them in this section, as well as the measured throughput and jitter for comparison against the previous test. Figure 23 shows the working area used in these tests where we can observe both webcams pointing to a screen showing a film.

Figure 24 shows two frames obtained in the test for different hops, where we can see the quality of the video received in both scenarios. In the scenario with one hop, there is no significant delay between the real video sequence



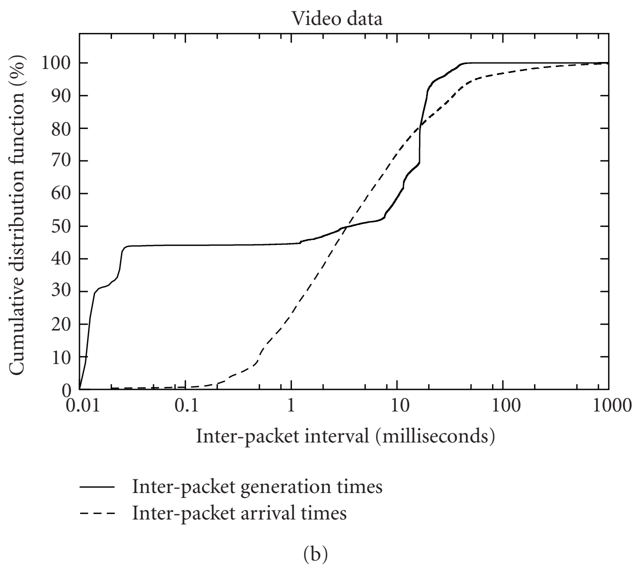


Figure 22: Cumulative distribution function for the interpacket generation interval and interpacket arrival interval in a scenario with one hop (b) and ten hops (a).

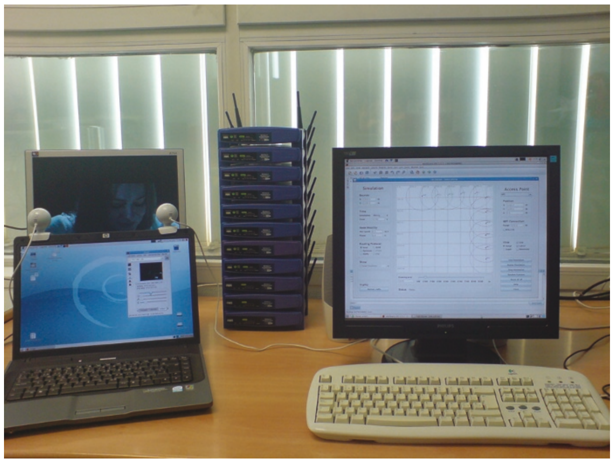


Figure 23: Testing a videocall when both webcams point to screen with a movie.

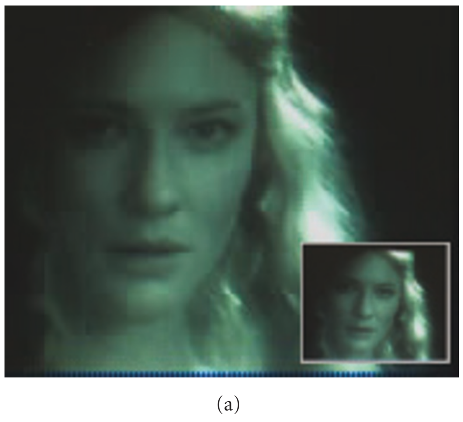
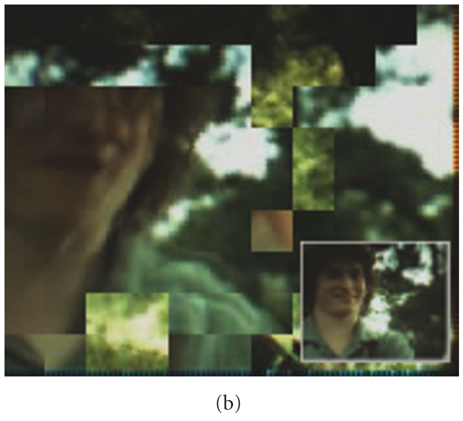
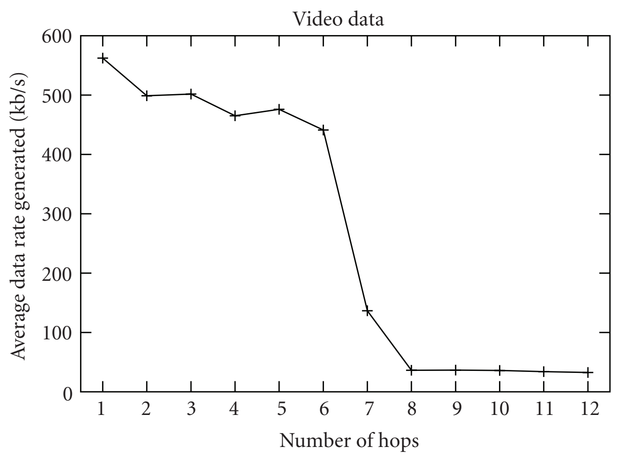



Figure 24: Screenshot of the videocall with a scenario of one hop (b) and ten hops (a).

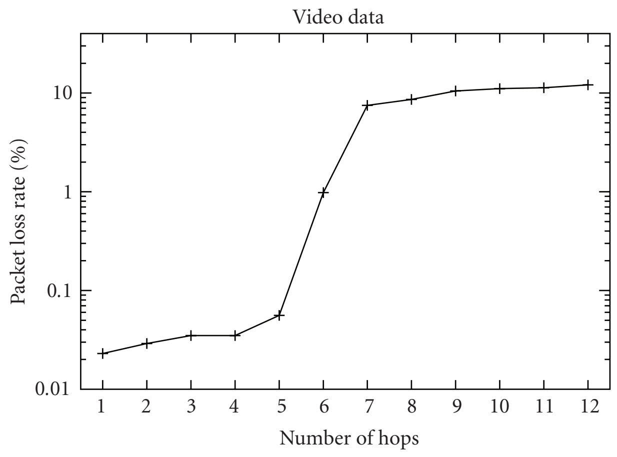
and the decoded one because the path is too short. In a scenario with ten hops, though we can verify the poor performance obtained by comparing the received and transmitted sequences, we also appreciate a delay of the received image compared to the real image (notice that both webcams aim at a same target for comparison purposes).

Figure 25(b) shows the mean throughput obtained for the generated data rate obtained for a different number of hops. In a scenario with one hop, we have a mean data rate above 500 Kb/s. In a scenario with 13 hops, the mean value of the generated data rate decreases to less than 100 Kb/s due to Ekiga's bandwidth throttling mechanism. We also evaluate the packet loss in each scenario. Figure 25(b) shows how the packet loss increases when we vary the intermediate hops between sender and receiver. If we compare this graphic with the one obtained in the evaluation of a real videoconference, we can observe that now the packet loss rate increases more significantly. In fact, for more than 6 hops, the packet loss rate surpasses 10%.

Figure 26 shows the cumulative distribution function for the interpacket generation and arrival times in a scenario with one hop and in a scenario with ten hops. Notice that

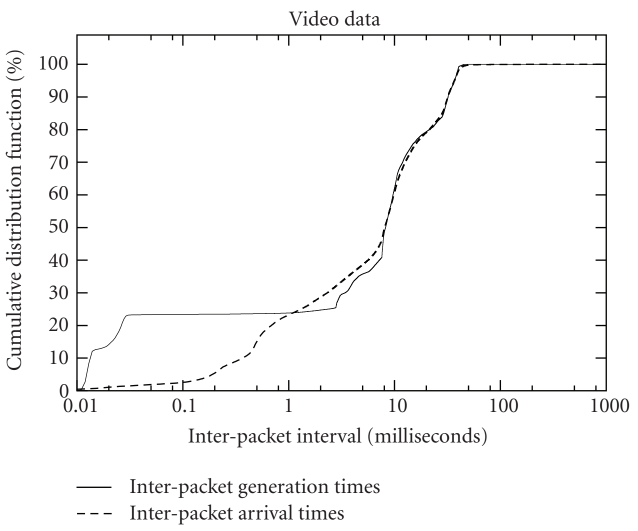


(a)

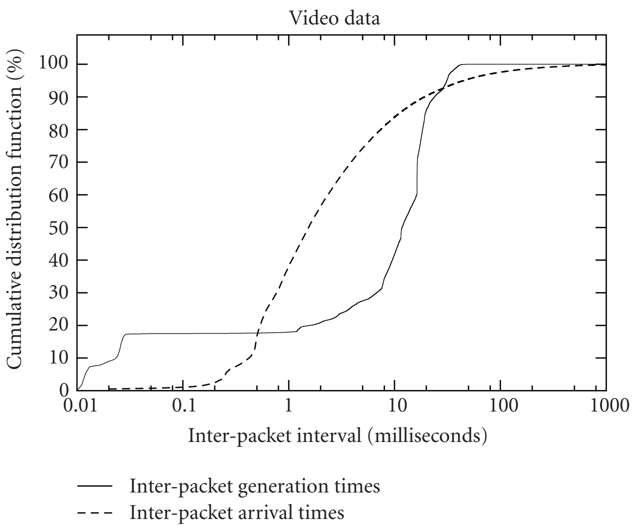


(b)

Figure 25: Cumulative distribution function for the throughput in a scenario with different hops (b) and packet loss rate in different scenarios (a).



(a)



(b)

Figure 26: Cumulative distribution function for the interpacket generation interval and interpacket arrival interval in a scenario with one hop (a) and ten hops (b).

high delay values are quite prone to occur in this case, often introducing significant jitter (above 100 milliseconds).

6.1.3. Evaluating the Round-Trip Time of the Different Scenarios. We also evaluated the impact of increasing the number of hops on delay by using ping sessions and measuring the impact on round-trip time while maintaining videoconference sessions active. We generate a ping session between the two laptops in the chain scenario described earlier, varying the number of hops from one to ten. We obtain the average time of the ping session used to send a packet and receive the answer. To obtain the one-way delay, we divided the results obtained by two.

Figure 27 shows the average delay and the standard deviation for each scenario. As expected, delay increases almost linearly with the number of hops between sender and receiver. Of special interest is the increase in terms of standard deviation, which can be quite problematic for real-time video transmission.

6.2. Dynamic Scenario. In this section we evaluate different scenarios with a dynamic topology. We change the nodes' mobility by testing with the following speed values: 3, 6, 9, 12, 15 meters per second. The scenario is defined in a 1500 m × 900 m area, and the test time is of 5000 seconds. We set the wireless nodes' range to 250 meters. To generate the video traffic we have two laptops with a webcam attached running Ekiga. In this test, as for the static scenario, we also differentiate between two types of scenarios: a standard videocall and a videocall where the webcams point to a movie being displayed.

Regarding to OLSR, and according to [29, 30], we need to tune up the protocol for use in mobile scenarios. The new configuration of the protocol is shown in Table 4.

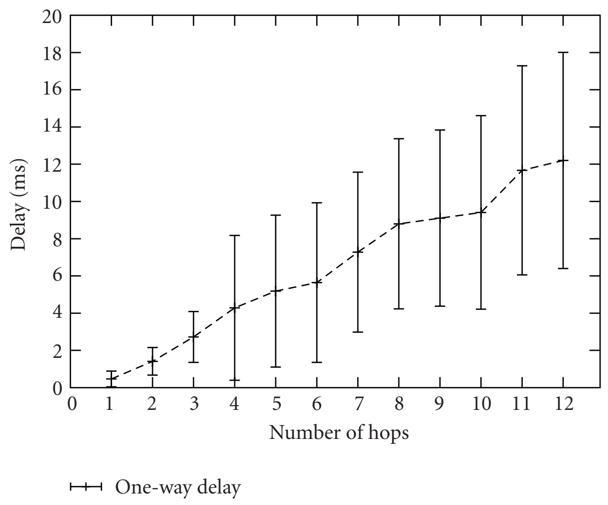


Figure 27: Evaluation of the ping sessions in different scenarios.

Table 4: OLSR values used for the mobility scenarios.

Parameter	Used Values
ELLO_INTERVAL	2 s
HELLO_VALIDITY	6 s
TC_INTERVAL	5 s
TC_VALIDITY	15 s
MID_INTERVAL	5 s
MID_VALIDITY	15 s
HNA_INTERVAL	5 s
HNA_VALIDITY	30 s

Table 5: Percentage of the simulation time when exists a route between both laptops.

Mobility	% time with route
3 m/s	64.72
6 m/s	65.32
9 m/s	85.70
12 m/s	91.77
15 m/s	93.74

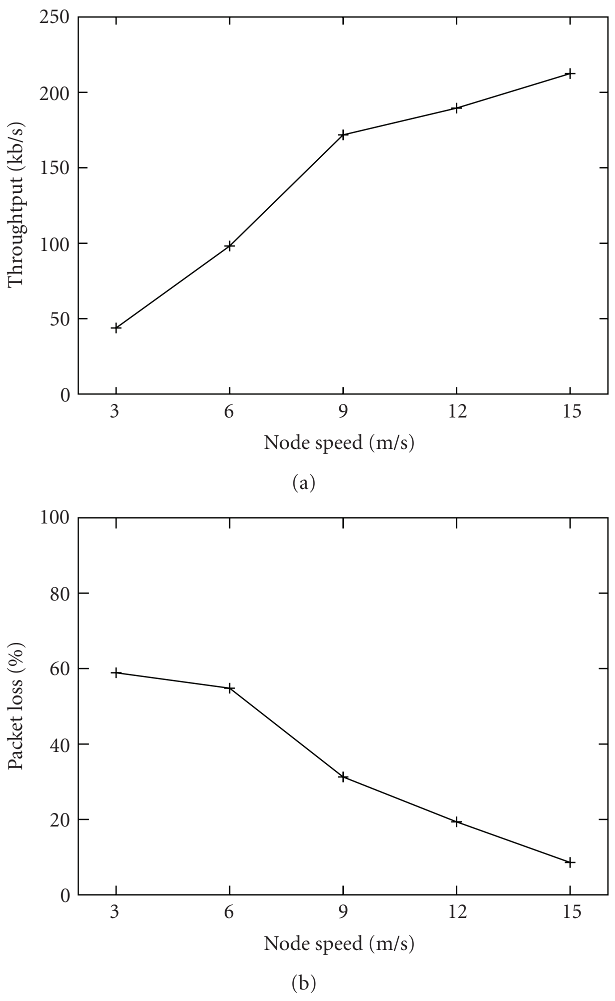


Figure 28: Throughput and packet losses with a standard videocall in a scenario with mobility.

Since, for these tests, we picked a particular OLSR configuration, the first step is to measure the total amount of time that the route between both laptops is established. For this test we emulate 20 random scenarios for each speed, and we measure with a ping if the route has established or not.

Table 5 shows total percentage of time the route is established.

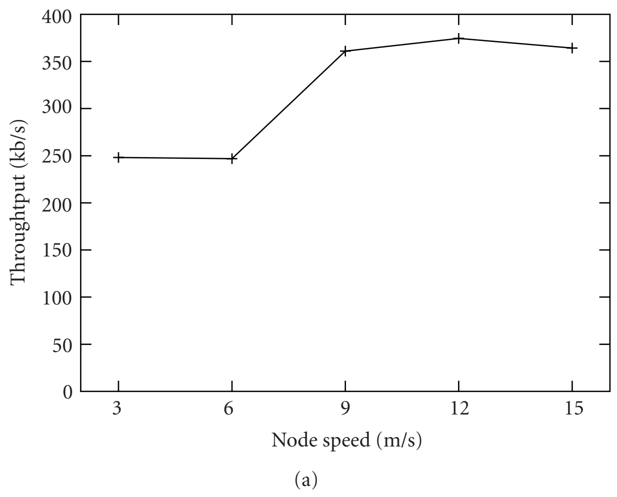
The percentage of time without route is directly related to the amount of time that OLSR needs to obtain a route when the topology is changing plus the time when the network is split, and there is no possible route between the two laptops.

6.2.1. Evaluating the Performance with a Standard Videocall. As in Section 6.1, we first evaluate the behaviour of a standard videocall in a MANET. Since the number of hops is variable in each simulation due to mobility, in this case we study the variation of the throughput with different node speeds as well as the packet losses since the protocol is based on UDP.

Figure 28 shows the results obtained in this set of tests. As we can observe, higher mobility translates into better performance of the network and less percentage of packets lost. This is due to a the higher route availability, as shown in Table 5. Higher speeds avoid lengthy network partitioning effects, which are translated as a better service with a higher average throughput.

6.2.2. Evaluating the Performance of a Movie Transmission. In this section we also measure the throughput obtained by the videoconferencing participants when streaming a movie being displayed.

Figure 29 shows the throughput obtained and the packet losses in a scenario where the webcams point to a screen displaying a movie. As we can see, the throughput is higher than that in the previous test: if the webcams capture



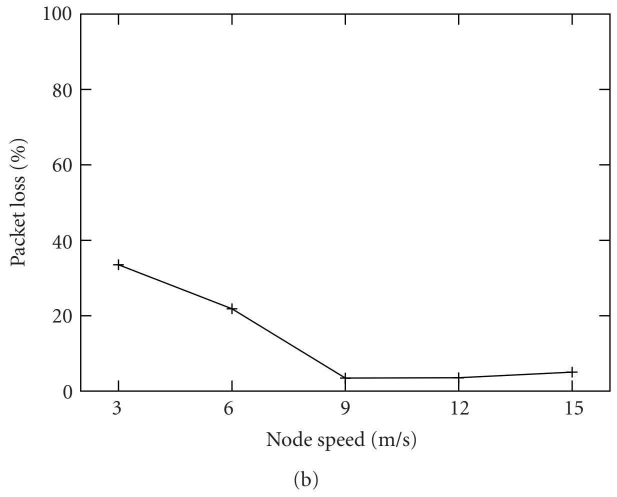


Figure 29: Throughput and packet losses with a movie in a scenario with mobility.

a movie, the video is more complex than the nearly static image of a standard videocall.

If we compare this figure with the one shown when evaluating a standard videocall, we can see a similar trend. The last test shows a higher throughput, as the transmitted video generates a higher data rate. Packet losses are decreased as in the standard videocall because (1) we have a higher throughput when a route is established due the type of video captured, (2) Ekiga's bandwidth throttling mechanism decreases the packet injected when a route is lost, and (3) on average we have the same time periods without routes between the two laptops as in the previous tests. Jointly, these factors explain why the total percentage of packets lost is reduced at higher node speeds (within the speed range tested).

7. Conclusions

In this work we present *Castadiva*, a novel architecture to improve research in the MANETs field by allowing to make real testbed experiments in a simple and straightforward manner.

Castadiva combines the convenience and productivity of Java with the power of the Linux kernel and accompanying tools to emulate an ad hoc network environment. The system was designed to simplify the tasks of scenario generation and starting traffic flows among independent, IEEE 802.11-based, wireless nodes. One of its key advantages is that it is fully compatible with the ns-2 simulator.

The architectural design of *Castadiva* differentiates wireless nodes, used for the actual experiments, from the core application, which has management and control purposes. This core application provides an easy interface to define network topologies and traffic flows between nodes. Those definitions are then translated into run-time instructions sent to testbed nodes when experiments are on-going. We observed that the use of SSH protocols and Fast Ethernet connectivity allows nodes to synchronize the start of an experiment with high accuracy, being all instructions read at once; afterwards, the testbed relies on individual clocks to synchronize instructions throughout the remaining time of an experiment.

An important issue when designing *Castadiva* was that of ns-2 compatibility. *Castadiva* is compatible with the file format used by the ns-2 simulator. In the validation section, we compared the results obtained with *Castadiva* and the ns-2 network simulator. By using both TCP and UDP data traffic, and under a variety of static and dynamic MANETs scenarios, we show that *Castadiva* is able to offer confident results while using cheap wireless off-the-shelf devices.

Castadiva was also extended to allow injecting traffic from external applications. Such functionality enabled us to assess the performance of videoconferencing in an emulated ad hoc network using a real application such as Ekiga. We studied the behavior of a videocall in both static and dynamic scenarios. For the static scenario we change the number of hops between caller and receiver from one to ten. We also vary the characteristics of the video being transmitted. The results obtained in the video evaluation show that, for a large number of hops, delay times are quite high, typically provoking annoyance to the videocall users. Packet losses also become problematic when source and destination are more than six hops away. Additionally, our experiments showed that the OLSR routing protocol is prone to suffer from instability if the hop count is high, despite altering the default parameter values to more efficient ones.

Concerning the mobile scenario, the tests performed show that, on average, a MANET can be used to support a videocall with mobility, being the quality of the video decreased and having a percentage of time without service between 7% and 36% of the total time, depending on the nodes' speed. In general, we can assert that higher degrees of mobility are better to avoid nodes becoming isolated for long periods of time.

To summarize, this work shows that the advantages of using *Castadiva* with respect to other MANET test-beds are that (1) it is a very low-cost test-bed since each node costs about 50$, (2) it is fully compatible with the ns-2 simulator, allowing to compare results between both in a straightforward manner, and (3) it does not occupy a lot of physical space.

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

Acknowledgment

This work was partially supported by the *Ministerio de Ciencia e Innovación*, Spain, under Grant TIN2008-06441-C02-01.

References

[1] M. Conti and S. Giordano, "Multihop ad hoc networking: the theory," *IEEE Communications Magazine*, vol. 45, no. 4, pp. 78–86, 2007.

[2] J. L. Burbank, P. F. Chimento, B. K. Haberman, and W. T. Kasch, "Key challenges of military tactical networking and the elusive promise of manet technology," *IEEE Communications Magazine*, vol. 44, no. 11, pp. 39–45, 2006.

[3] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (AODV) routing," Request for Comments 3561, Network Working Group, http://www.ietf.org/rfc/rfc3561.txt, July 2003.

[4] 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.

[5] D. Johnson, P. Y. Hu, and D. Maltz, "The dynamic source routing protocol (dsr) for mobile ad hoc networks for ipv4," Request for Comments 4728, MANET Working Group, http://www.ietf.org/rfc/rfc4728.txt, February 2007.

[6] I. D. Chakeres and C. E. Perkins, "Dynamic MANET on-demand (DYMO) routing protocol," IETF Internet Draft, November 2007.

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

[8] OPNET Technologies Inc, "OPNET making networks and applications performs," http://www.opnet.com/.

[9] M. Kojo, A. Gurtov, J. Manner, P. Sarolahti, T. Alanko, and K. Raatikainen, "Seawind: a wireless network emulator," in *Proceedings 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB '01)*, Aachen, Germany, September 2001.

[10] S. Keshav, *REAL: a network simulator*, University of California, Berkeley, Calif, USA, 1988.

[11] T. R. Andel and A. Yasinsac, "On the credibility of manet simulations," *IEEE Computer Society*, vol. 39, no. 7, pp. 48–54, 2006.

[12] D. A. Maltz, J. Broch, and D. B. Johnson, "Experiences designing and building a multi-hop wireless ad hoc network testbed," *IEEE Journal on Selected Areas of Communications*, vol. 17, no. 8, pp. 1439–1453, 1999.

[13] Ekiga, free your speech, http://ekiga.org/.

[14] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, and J. Peterson, "Sip: session initiation protocol," Request for Comments 3261, MANET Working Group, http://www.ietf.org/rfc/rfc3261.txt, June 2002.

[15] ITU-T, "Packet-based multimedia communications systems," http://www.itu.int/rec/T-REC-H.323/en, June 2006.

[16] P. Baran, "On distributed communications," The RAND Corporation, 1964.

[17] 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," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC '05)*, vol. 3, pp. 1664–1669, New Orleans, La, USA, March 2005.

[18] A. Karygiannis and E. Antonakakis, "MLab: an ad hoc network test bed," in *Proceedings of the 3rd IEEE Consumer Communications and Networking Conference (CCNC '06)*, vol. 2, pp. 1312–1313, National Institute of Standards and Technology, Las Vegas, Nev, USA, January 2006.

[19] G. Judd and P. Steenkiste, "Design and implementation of an RF front end for physical layer wireless network emulation," in *Proceedings of the 65th IEEE Vehicular Technology Conference (VTC '07)*, pp. 974–979, Dublin, Ireland, April 2007.

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

[21] J. Zhou, Z. Ji, M. Varshney, Z. Su, and Y. Yang, "Whynet: a hybrid testbed for large-scale, heterogeneous and adaptive wireless networks," in *Proceedings of the 1st ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH '06)*, vol. 3, pp. 111–112, Los Angeles, Calif, USA, 2006.

[22] "OpenWRT, wireless freedom," http://openwrt.org.

[23] T. Ylonen and C. Lonvick, "The secure shell (ssh) authentication protocol," Request for Comments 4252, MANET Working Group, http://www.ietf.org/rfc/rfc4252.txt, January 2006.

[24] B. Hubert, "Linux advanced routing & traffic control HOWTO," 1.43 edition, http://lartc.org/, 10 2003.

[25] C. Bettstetter, G. Resta, and P. Santi, "The node distribution of the random waypoint mobility model for wireless ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 2, pp. 257–269, 2003.

[26] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388–404, 2000.

[27] J. Li, C. Blake, D. S. J. De Couto, H. I. Lee, and R. Morris, "Capacity of ad hoc wireless networks," in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom '01)*, pp. 61–69, ACM, New York, NY, USA, 2001.

[28] S. McCanne, V. Leres, and V. Jacobson, "The tcpdump manual page," Lawrence Berkeley Laboratory, 6 89.

[29] C. Gomez, D. Garcia, and J. Paradells, "Improving performance of a real ad-hoc network by tuning olsr parameters," in *Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC '05)*, Murcia, Cartagena, Spain, June 2005.

[30] Y. C. Huang, S. N. Bhatti, and D. Parker, "Tuning olsr," in *Proceedings of the 17th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '06)*, Helsinki, Finland, September 2006.