MOBILEMAN

IST-2001-38113

Mobile Metropolitan Ad hoc Networks

MOBILEMAN

MobileMAN Technical Evaluation

Deliverable D16

Contractual Report Preparation Date: 31 October, 2005
Actual Date of Delivery: 18 November, 2005
Estimated Person Months: 55
Number of pages: 222

Contributing Partners: Consiglio Nazionale delle Ricerche (Italy), University of Cambridge (UK), Institut Eurecom (France), Helsinki University (Finland), NETikos (Italy), Scuola Universitaria Professionale della Svizzera Italiana (Switzerland)

Authors: Giuseppe Anastasi, Emilio Ancillotti, Eleonora Borgia, Raffaele Bruno, Marco Conti, Franca Delmastro, Enrico Gregori, Giovanni Mainetto, Gaia Maselli, Antonio Pinizzotto, Giovanni Turi (CNR); Jon Crowcroft, Andrea Passarella (Cambridge); Claudio Lavecchia, Pietro Michiardi, Refik Molva (Eurecom); Jose Costa Requena (HUT); Piergiorgio Cremonese, Veronica Vanni (Netikos); Ralph Bernasconi, Ivan Defilippis, Silvia Giordano, Alessandro Puiatti, (SUPSI)

Abstract: The aim of this deliverable is to present the validation results of the architecture, protocols and services designed for the MobileMAN project. This deliverable therefore represents the complement of Deliverable D13. Specifically, D13 presents the MobileMAN architecture and protocols, while in this deliverable the solutions we devised are validated. Whenever possible, our evaluations are based on measurements from real small- medium-size testbeds. In addition, simulation results are used to study large scale networks and/or complex mobility scenarios. Results are presented by following a bottom up approach from wireless technologies up to the applications. The deliverable also investigates the behavior and performance of a medium-scale MobileMAN network made of (up to) 23 nodes.

Project funded by the European Community under the "Information Society Technologies" Programme (1998-2002)

Summary

The aim of this deliverable is to present the validation results of the architecture, protocols and services designed for the MobileMAN project. This deliverable therefore represents the complement of Deliverable D13. Specifically, D13 presents the MobileMAN architecture and protocols, while in this deliverable the solutions we devised are validated.

There are two main approaches in system evaluation: measurements on real testbeds and analytical/simulation modeling. The performance study of mobile ad hoc networks is a complex task that cannot be addressed by using a unique performance technique but requires a careful mixing of measurements on real testbeds with simulation/analytical studies. Whenever possible we constructed small- medium-scale testbeds to validate our solutions by taking into consideration real scenarios. Modeling studies have been extensively used in the protocols' design phase, and to study the MobileMAN system behavior in complex scenarios that are very difficult (if not impossible) to be studied by prototypes. In the latter case, to develop and solve our simulation models we used simulation tools.

In this deliverable we present a comprehensive overview of all the activities we performed to validate the solutions we developed for the MobileMAN environment. The deliverable follows a bottom up approach from wireless technologies up to the applications. The deliverable ends with a section reporting the experimental evaluation of a MobileMAN medium-size mobile-ad-hoc network (up to 23 nodes) which integrates the solutions we have developed.

Specifically, in Section 1, after discussing the performance modeling techniques, we present the characteristics of the simulation framework, which extends the Network Simulator NS-2 (v. 2.27) with a cross-layer interface (XL-interface) that standardizes vertical interactions among protocols according to the MobileMAN cross-layer architecture. This simulation framework has been used in successive sections to validate our cross-layer solutions (e.g., see *Reliable Forwarding* and cross-layer optimization of the *Gnutella protocol*).

In Section 2 we analyze and compare the performance (in multi-hop ad hoc networks) of 802.11 card with those of the enhanced card we designed and implemented. First we present a simulation study that shows the effectiveness of our solutions in several scenarios that (in the literature) are known as critical for 802.11 cards. Then we present experimental results obtained in a 4-node networks. In this network the nodes use either the 802.11 card or our enhanced card. Experimental results confirm previous simulation studies. In addition, they point out additional advantages of the enhanced card when used in a real environment with highly variable channel conditions.

Section 3 is devoted to analyzing MobileMAN networking protocols that use the one-hop transmission services provided by the network interface card to construct end-to-end (reliable) delivery services. Specifically, we first present our experimental results related to OLSR and AODV in small scale networks with node mobility. These results complete the study reported in Deliverable D8. Secondly, we report the performance results of our mechanism for reliable forwarding which exploits cross layer interactions (REEF). This study has been performed via simulation by exploiting our extension of the NS-2 environment. The section ends presenting

experimental results of our transport protocol, TPA. Experimental results confirm the observations obtained via simulation and reported in Deliverable D13.

Section 4 is devoted to the interconnection of MobileMAN islands to the Internet. Our solutions have been briefly described in Deliverable D14 where we presented the software developed to support the interconnection. For completeness in this deliverable we first present a refined description of our solution and then we report the experimental results that confirm the effectiveness of our approach.

Section 5 addresses the enforcement of cooperation within a MANET. Specifically, the section presents an in depth analysis of CORE, i.e., our mechanism to address cooperation issues. The features of CORE are analyzed in terms of simulation metrics that we deem relevant to assess the basic properties of a cooperation enforcement mechanism: the energetic cost beard by CORE-enabled nodes and the efficiency of the detection and punishment mechanisms used in CORE. Simulation results are used to understand if and when a mechanism to distribute reputation information could be necessary in order to improve punishment efficiency: reputation distribution is an optional feature of the CORE mechanism and constitutes to discriminate between CORE and other reputation-based cooperation enforcement mechanisms.

Section 6 deals with the MobileMAN middleware platforms. Performance studies are used to show the effectiveness of the cross-layer optimizations. Specifically, we considered two well-known p2p platforms, Gnutella and Pastry, which represent un-structured and structured overlays, respectively. In the case of Gnutella the study has been performed via simulation; while for Pastry, which is part of the MobileMAN architecture, we performed an experimental study. Specifically, in the Pastry case, we present a set of experimental results obtained by comparing in a small testbed the performance of Pastry with those of *CrossROAD*. CrossROAD is our proposal to enhance Pastry by exploiting cross-layer interactions. Both Gnutella and Pastry studies clearly pointed out that cross-layer optimization are mandatory to achieve good performance in a mobile ad hoc network.

In Section 7 we investigate the quality of service experienced by the three applications we selected to test the MobileMAN architecture: UDDI, a whiteboard application (WB) and a VoIP session. In all cases we evaluated the application performance when running on top of a small MobileMAN testbed. In the case of UDDI and WB, we tested both the legacy and cross-layer architecture.

Section 8 concludes the deliverable by investigating the behavior and performance of a medium-scale MobileMAN network made of (up to) 23 nodes.

Table of Contents

1. INTRODUCTION

The aim of this deliverable is to present the validation of the architecture, protocols and services designed for the MobileMAN project (see Deliverable D13). This validation has been performed by studying both the performance of single protocols in isolation and by investigating the performance of a MobileMAN system obtained by integrating our solutions with existing algorithms/software/hardware.

There are two main approaches in system performance evaluation: the first uses measurements on real testbeds; the second is based on a representation of the system behavior via a model [L83] [KM88]. As already pointed out in previous deliverables, models often introduce simplifications and assumptions that mask important characteristics of the real protocols behavior [ABCG04, LNT02]. To avoid these modeling approximations, simulations and analytical studies have to be complemented by measurements on real prototypes. Testbed results are very important as they are able to pointing out problems that cannot be detected by simulation/analytical studies. On the other hand, measurement techniques are applied to real systems, and thus they can be applied only when a real system, or a prototype of it, is available. This makes measurement studies very expensive and complex and (as already pointed out in previous deliverables) currently, only few measurements studies on real ad hoc testbeds can be found in the literature, see e.g., [BMJ00] [APE02]. The Uppsala University APE testbed [APE02] is one of the largest, having run tests with more than thirty nodes. In addition constructing a real ad hoc network test-bed for a given scenario is limited in terms of working scenarios, mobility models, etc. Furthermore, measurements are generally non-repeatable. For these reasons, protocols scalability, sensitiveness to users' mobility patterns and speeds are difficult to investigate on a real testbed. Using a simulation or analytic model, on the other hand, permits the study of system behavior by varying all its parameters, and considering a large spectrum of network scenarios.

To summarize, the performance evaluation of a mobile ad hoc networks is a complex task that cannot be addressed by using a unique performance technique but requires a careful mixing of measurements on real testbeds with simulation/analytical studies. Measurements on real testbed applied to small- medium-scale networks characterize the behavior of the system under study in limited but realistic conditions, and also provide important information for analytical/simulation models tuning. Simulation and analytical models can be used both in the protocol design phase (to compare and contrast alternative solutions) and to complement measurement results by investigating large scale networks and/or high-mobility scenarios. Indeed, mixing measurements with simulation/analytical studies is the approach we used for validating the MobileMAN architecture and protocols. Whenever possible we constructed small- medium-scale testbeds to validate our solutions by taking into consideration real scenarios. Modeling studies have been extensively used in the protocols' design phase and to study the MobileMAN system behavior in complex scenarios that are very difficult (if not impossible) to be studied by prototypes. In the latter case, to develop and solve our simulation models we used simulation tools. The main advantage of these tools is that they provide libraries containing pre-defined models for most communication protocols (e.g. 802.11, Ethernet, TCP, etc.). Popular network simulators used in ad hoc networks include: OPNET [OPN],

NS-2 [NS2], Glomosim [GLOM] and its commercial version QualNet [QUAL]. They all provide advanced simulation environments to test and debug different networking protocols, including collision detection modules, radio propagation and MAC protocols. In the first phases of the project we analyzed and compare all these simulation tools. As in the last two-three years the NS-2 simulation tool emerged as the reference environment for MANETs evaluation, we performed our recent simulation studies by exploiting the NS-2 platform. As pointed out in Deliverable D13, we developed both legacy architecture and more advanced architecture based on the cross layer approach. In the former case we simply integrated in the NS-2 framework the models of the protocols we designed. On the other hand, in order to validate the cross-layer solutions we had to in-depth modify the NS-2 framework to support cross-layer interactions. In Section 1.1 we briefly present the simulation framework we implemented inside NS-2 to simulate our cross-layer architecture.

1.1. Simulation Framework

As explained in Deliverable D13, we defined an enhanced MobileMAN architecture that exploits cross-layer optimizations by maintaining the benefits of a modular architecture. This is achieved by introducing a cross-layer interface (XL-interface) that standardizes vertical interactions and gets rid of tight-coupling from an architectural standpoint [CCMT05].

Engineering the XL-interface presents a great challenge. This component must be general enough to be used at each layer, providing a common set of primitives to realize local protocol interactions. To support this novel paradigm we classified cross-layer functionalities and extended standard TCP/IP protocols in order use them. The result of this effort has been implemented in the ns2 Network Simulator, realizing a simulative evaluation framework for the usability of the XL-interface at different layers.

In this Section we describe the realization of a simulation framework, which allowed us to practice the usage of the XL-interface, and evaluate the performance of the resulting solutions. Our framework targets the Network Simulator ns2 (v. 2.27), and basis on a library of objects and abstractions provided by the Naval Research Laboratory (i.e., ProtoLib) [ProtoLib], which includes an implementation of the Optimized Link-State Routing protocol (OLSR). After a concise description of ProtoLib's features, we report the enhancements implemented to make cross layering possible among protocol agents.

1.1.1. The Proto Library

The protocol Prototyping library (ProtoLib) [ProtoLib] is not so much a library as it is a toolkit. The goal of the ProtoLib is to provide a set of simple, cross-platform C++ classes that allow the development of network protocols and applications.

Although ProtoLib is principally for research purposes, the code has been constructed to provide robust, efficient performance and adaptability to real applications. Currently, ProtoLib supports most UNIX platforms (including MacOS X) and WIN32 platforms. The version used in this work also supports the ns2 simulation environment.

The approach behind this environment is to provide the programmer with abstractions of common networking objects, like network addresses, sockets and timers, which are then mapped to a target platform with specific implementations. Hence, the main idea is to

have the programmer writing networking code using the abstractions, so as to have it working on a significant set of real-platforms, as well as on simulation environments, with minimal changes.
Another important feature is that the ProtoLib package comes with an implementation of OLSR, compliant with the latest specification, and based on the above abstractions. We modified OLSR in order to realize performance optimizations at different layers.

1.1.2. Introducing cross-layer interactions

After patching the network simulator with the ProtoLib and the OLSR extensions, we introduced also a set of primitives to allow cross-layer interactions. It was a "natural" choice to place the cross-layer primitives inside the ProtoLib, realizing them as abstractions that protocols can use to share data and exchange local events. This allowed us to be compliant with the cross-platform philosophy of the ProtoLib, directly providing an implementation for the ns2 simulation environment. Additionally, it was a proof of concept of the "usability" of the XL-interface.
We implemented interfaces for XL Data and XL Event objects, respectively for sharing protocol internal data (i.e., synchronous interactions) and for subscribe/notify to internal events (i.e., asynchronous interactions). In the following descriptions, we go through the functionalities of the new components.

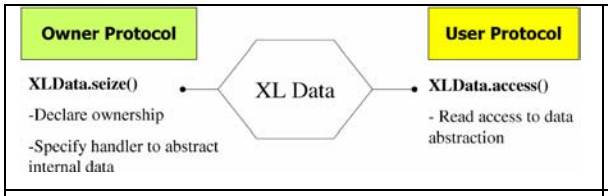


Figure 1.1. Cross-Layer Data.

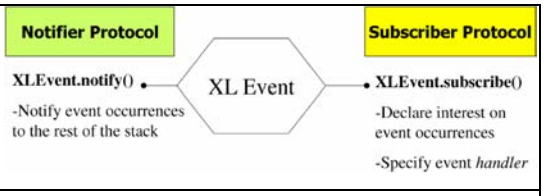


Figure 1.2. Cross-Layer Events.

- **ProtoXLData**. This is a generic class (see Figure 1.1) that identifies internal data owned by a protocol and shared to the rest of the network stack. It offers methods to declare ownership of the data and to specify a call-back function for "translating" the internal data format used by the owner, in a cross-layer ontology common to the whole stack. Other protocols access instances of this class in a read-only format.
- **ProtoXLEvent**. This is a generic class (see Figure 1.2) that identifies conditions or events detected internally to the protocol, which may result of interest for the rest of the stack. It offers methods to subscribe interest in events derived from this class, as well as to notify occurrences of them.

1.2. References

[ABCG04]	G. Anastasi, E. Borgia, M. Conti, E. Gregori, "WiFi in Ad Hoc Mode: A Measurement Study", Proc. 2nd IEEE Conference on Pervasive Computing and CVommunications (PerCom 2004), March 2004.
[APE02]	APE: Ad hoc Protocol Evaluation testbed. Department of Computer Systems at Uppsala, Sweden. http://apetestbed.sourceforge.net/
[CCMT05]	M. Conti, J. Crowcroft, G. Maselli, and G. Turi, "A Modular Cross-Layer Architecture for Ad Hoc Networks," in *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks*, C. Press, Ed., July 2005.
[C04]	M. Chiang, "To Layer or not to Layer: Balancing Transport and Physical Layers in Wireless Multihop Networks," in *Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, 2004.
[CSN02]	K. Chen, S. H. Shah, and K. Nahrstedt, "Cross-Layer Design for Data Accessibility in Mobile Ad Hoc Networks," *Wireless Personal Communications*, vol. 21, no. 1, pp. 49–76, 2002.
[GLOM]	GloMoSim, Global Mobile Information Systems Simulation Library, http://pcl.cs.ucla.edu/projects/glomosim/.
[KKT04]	U. C. Kozat, I. Koutsopoulus, and L. Tassiulas, "A Framework for Cross-layer Design of Energy-efficient Communication with QoS Provisioning in Multi-hop Wireless Netwroks," in *Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, 2004.
[KK03]	V. Kawadia and P. R. Kumar, "A Cautionary Perspective on Cross Layer Design." *IEEE Wireless Communication Magazine*. pp. 3-11, vol. 12, no. 1, February 2005.
[KM88]	J.F. Kurose, H. Mouftah, "Computer-Aided Modeling of Computer Communication Networks", IEEE Journal on Selected Areas in Communications, Vol. 6, No. 1 (January, 1988), pp. 130-145.
[L83]	S.S. Lavenberg, *Computer Performance Handbook*, Academic Press, New York, 1983.
[LNT02]	H. Lundgren, E. Nordstron, C. Tschudin, "Coping with Communication Gray Zones in IEEE 802.11 based Ad Hoc Networks", Proceedings of the *ACM Workshop on Mobile Multimedia (WoWMoM 2002)*, Atlanta (GA), September 28, 2002, pp. 49-55.
[NS2]	The Network Simulator - ns-2, http://www.isi.edu/nsnam/ns/index.html
[ProtoLib]	"PROTEAN Research Group," http://cs.itd.nrl.navy.mil/5522/.
[QUAL]	Qualnet simulator, http://www.qualnet.com/.
[SGN03]	R. Schollmeier, I. Gruber, and F. Niethammer, "Protocol for Peer-to-Peer Networking in Mobile Environments," in *Proceedings of 12th IEEE International Conference on Computer Communications and Networks*, Dallas, Texas, USA, Oct. 2003.

[YLA02]	W. H. Yuen, H. Lee, and T. D. Andersen, "A Simple and Effective Cross Layer Networking System for Mobile Ad Hoc Networks," in *Proceedings of IEEE PIMRC 2002*, Lisbon, Portugal, 2002.

2. ENHANCED 802.11 CARD

2.1. Simulations

As discussed in Deliverable D10 [D10] and D13 [D13], the AOB mechanism has been selected as the MAC protocol candidate to be adopted in the MobileMAN enhanced 802.11 card. Section 2.2 will report on the current status of this card implementation and will show the results of experimental tests conducted in various conditions to verify the card functionalities and the performance improvements over standard IEEE 802.11 cards. However, in the above-mentioned deliverables it was also motivated the need for further extending the AOB functionalities, in order to deal with the heterogeneity of the MobileMAN environment and the multi-hop communications. In the Deliverable D10 we have proposed an extension of the AOB protocol to solve the significant unfairness problems that arise in single-hop networks when *enhanced* stations using the modified 802.11MAC protocol compete with *legacy* stations adopting the standard 802.11 MAC protocol. Our approach to solve this problem is to design a component capable of estimating how long the *enhanced* stations further defer their frame transmissions with respect to the *legacy* stations, and then exploiting this information to enable the *enhanced* stations to reclaim new transmission opportunities. Specifically, in our solution enhanced stations earn *credits* when releasing transmission opportunities granted by the standard protocol. The collected credits are a *virtual currency* spent by the enhanced stations to reclaim new transmission opportunities.

In this section we want to further elaborate on these concepts to extend the AOB mechanism in order to solve the unfairness problems that arise in multi-hop networks, even if only enhanced stations are present. In this section we first discuss the major unfairness causes in 802.11-based multi-hop networks. Then, we review related work to highlight advantages and flaws of other proposals. Afterwards, we describe the extended Fair AOB protocol, hereafter simply Fair MAC, and we present numerical results from simulation tests to show that our scheme can achieve a better fairness than standard IEEE 802.11, while increasing the total network throughput.

2.1.1. Unfairness in 802.11-based Multi-hop Networks

Several unfairness and performance issues involving IEEE 802.11 in a multi-hop context have been pointed out through simulation studies and experimentations [XS02, ACG03]. Many of these problems originate from the asymmetric perception of the medium availability that nodes have in multi-hop networks. Indeed, a node perceiving no on-going transmission tries to capture the whole channel bandwidth for its own transmissions. A successful transmission prevents any other station in the neighborhoods of either the transmitter or the receiver from gaining access to the medium. However, the level of contention in those neighborhoods (i.e., the number of contending nodes in those parts of the network) could be different, leading to an unbalanced allocation of channel bandwidth [NKGB00]. Moreover, nodes failing to gain access to the medium are invisible to other emitters and bandwidth-consuming nodes cannot easily be notified of

the presence of starved nodes.

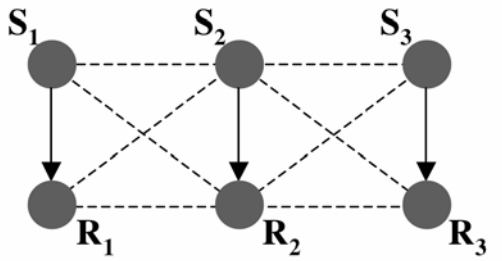


Figure 2.1. Three-pairs scenario: the central emitter contends with the two independent other

Traditionally, the most extensively investigated network scenario that introduces unfairness is the so-called *hidden terminal* situation, in which two independent emitters transmit to a common receiver, resulting in undetectable receiver-side collisions. However, recent papers have shown that several other configurations can have even more serious performance issues. In [ChaudetDG05a], a certain number of basic network configurations involving a small number of nodes, which lead to severe performance issues, have been gathered and analyzed. For instance, it is shown that long-term unfairness occur in the *three-pairs* scenario depicted in Figure 2.1 in which three couples of emitter-receiver are placed so that different pairs of nodes cannot directly communicate, but a transmission from one pair blocks the carrier sense mechanism of its direct neighbors. The two external pairs are totally independent, therefore their backoff timers evolve asynchronously and their transmissions are not synchronized. In this configuration, the central pair is disadvantaged and usually obtains a quite low bandwidth share, as both exterior pairs need to be silent simultaneously for the central pair to decrement a single backoff slot.

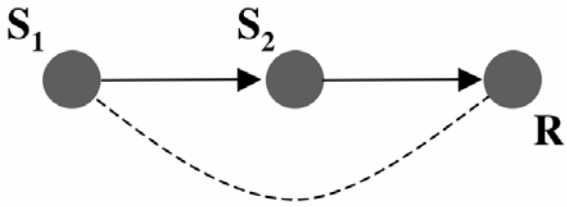


Figure 2.2. Large-EIFS problem: acknowledgments from R trigger an *EIFS* timer at the first emitter, provoking unbalance.

Another subtle phenomenon that induces substantial unfairness and throughput degradation is the disproportion between the time deferral triggered whenever a node detects a correct frame on the channel, i.e., the *DIFS* time, and the one triggered whenever a node detects an erroneous frame, i.e., the *EIFS* time. The network scenario depicted in Figure 2.2 illustrates this problem, also known as *Large-EIFS* problem [ZNG04]. In this case, an acknowledgment emitted by the receiver R triggers an *EIFS* deferral time at the leftmost emitter. The situation is unbalanced between the two

emitters, as one has to wait a *DIFS*-long idle time, while the other one needs to wait an *EIFS*-long idle time before decrementing any backoff slot. For other elementary configurations negatively affected by long-term and short-term fairness issues, the reader could refer to [ABCG04, CDG05a]. On the other hand, in more complex topologies (e.g., chain of nodes, grid topologies, random topologies, etc.); the resulting unfairness is often due to a complex combination of the above-mentioned causes.
Several different MAC protocols have been proposed to alleviate the unfairness issues in multi-hop ad hoc networks (and a comprehensive survey of them can be found in [JLB04]), but they focus primarily on solving the hidden terminal problem [FG97]. The most common approach has been to implement *floor acquisition* mechanisms based on extensions of the basic RTS/CTS handshake. However, recent experimentations [ABCG04, CDG2005] and simulation studies using realistic channel models [XGB03] have shown that the RTS/CTS handshake cannot prevent all interference as expected in theory. The reason is that the RTS/CTS handshake is effective in resolving the hidden terminal problem only under the assumption that all hidden nodes are within the transmission ranges of the receivers [XuGB03]. However, in IEEE 802.11-based multi-hop networks, the interference range is much larger than the transmission ranges [ACG03] and therefore this situation is less likely to appear. Other aspects, such as the fact that transmission ranges are in practice much shorter than usually assumed in simulations, also reduce the effectiveness of the RTS/CTS handshake in providing robust estimates of the channel occupations [ACG03]. Consequently, the design of practical solutions to eliminate the negative impact on the performance of IEEE 802.11-based multi-hop networks of the several unfairness causes discussed in this section, is still a totally open research area.

2.1.2. Related Work

In recent years, several algorithms and protocols have been developed to achieve fair channel allocation in multi-hop ad hoc networks [ONKC98, NKGB00, LLB00, BWK00, HB01, QS02, FB03]. In many of these papers, solutions have been proposed to achieve *weighted fairness*, such that different traffic flows are allocated bandwidth according to their weights. In particular, the authors of [NKGB00] proposed a mechanism for translating an arbitrary fairness model into a corresponding backoff algorithm, but they assumed that an ideal contention avoidance scheme is used to eliminate all hidden/exposed sender/receiver problems. The authors of [LLB00] proposed a packet-scheduling scheme that seeks to maximize aggregate channel allocation while providing a *minimum* fair allocation of channel bandwidth. However, this is achieved by constructing a conflict-free tree with a global exchange of topology and flow information between the nodes. The same shortcoming of requiring the exchange of information between nodes to either maintain the topology knowledge or to compute the fair allocation can be found in several other papers. For instance, the authors of [ONKC98] studied a $p_{i,j}$-persistent CSMA-based backoff algorithm that achieves an equal per-flow allocation relying on each node that regularly broadcast information on either the number of its neighbors or the average contention period of its neighbors. More recently, the authors of [HB01] developed an algorithm to calculate the fair shares that would enforce max-min fairness in an ad hoc network, which needs the knowledge of the two-hop neighbors.

An alternative approach employed in more recent papers is to solve the fairness issues by maintaining the *topology-transparency*, i.e., implementing backoff algorithms that are topology blind. The pioneering work for this type of schemes was [BWK00], in which a measurement-based backoff scheme has been proposed, which adjusts the contention windows according to the stations' fair shares. The drawback is that only frames that can be decoded contribute to the node's traffic statistics. This results in unreliable estimates of channel shares for stations that are external to the transmission ranges of the computing node, but internal to its interfering range, which is a common condition in 802.11-based multi-hop ad hoc networks, as shown in [ACG03, XGB03]. Recently, the authors of [FB03] proposed to adjust the contention windows only according to the number of successful transmissions carried out by each station, but the fairness improvement results into a significant reduction of the total network throughput.
A few papers have addressed the problem of jointly achieving a fair allocation of bandwidth and the maximization of channel utilization. The authors of [QS02] proposed a dynamically tuning backoff algorithm that enhances the overall performance while achieving a weighted fairness among stations, but it requires the estimate of the number of stations in the contention area, and it is designed to work in single-hop networks, in which there are no hidden terminals.

2.1.3. Description of the Fair-MAC Protocol

The main goal of the extensions to the AOB protocol we propose in this section is to alleviate the problem of fairness occurring in 802.11-based multi-hop networks without degrading the total network throughput. The design principles we follow are: *i)* to avoid any exchange of information between the nodes; and *ii)* to rely only on topology-blind estimates of the network status based on the standard physical carrier sensing activity.
As discussed in Deliverable D10 [D10], the AOB mechanism schedules the frame transmissions according to the IEEE 802.11 backoff algorithm, but adds an additional level of control before a transmission is enabled. Specifically, a transmission opportunity already granted by the standard backoff algorithm is rescheduled by AOB, after selecting a new backoff interval, in a probabilistic way. The probability $(1-P_T)$ of postponing a transmission attempt is computed dynamically, and it depends on the network congestion level, which is estimated using the utilization rate of the slots, the so-called s*lot utilization*. It is worth reminding that the slot utilization, also denoted as SU, is defined as the ratio between the number of busy periods (i.e., channel occupations due to transmissions) during the backoff interval and the total number of slots available for transmission during the backoff interval, i.e., the sum of idle slots and busy period [BCG04]. We have proved that the optimal slot utilization ensuring the maximization of the network throughput, called *Asymptotic Contention Limit* ($ACL(q)$), depends only on the average frame size q and it is independent of the number of competing stations. Finally, as the mechanism should prevent starvation, the probability of transmission is in addition dependent on the number of previous unsuccessful successive transmission attempts, designed by N_A in the following. The expression of the probability of transmission is then given by the following equation:

$$P_T = 1 - \left[\min\left(1, \frac{SU}{ACL(q)}\right)\right]^{N\_A} \quad (1)$$

This expression is decreasing as the medium load increases. Transmitting frames according to this probability leads to a near-optimal behavior in single-hop networks composed of only cooperating stations as shown in [BCG04].
The rationale behind the AOB scheme is that a station should refrain from transmitting whenever its transmissions could overload the network, inducing a congestion level higher than the optimal one. However, leaving time to other contending nodes for gaining access to the network could also help to address the fairness issues. Ideally, stations provoking an unfair share of the channel bandwidth should release more transmission opportunities than nodes with a lower channel share. It is obvious that a practical and realistic solution for this problem should rely on simple, interference-resilient and topology-independent procedures to evaluate the stations' channel shares, as the slot utilization. Moreover, we follow the same principle as [BWK00], in which each emitter *i* considers the whole set of its neighbors as a single contending emitter. In this case, each station has to estimate what channel share it gets and what channel share its neighbor stations as a whole get. In the following, we denote with *slot utilization internal* (SU_{int}) the channel share that is occupied by the computing node's transmissions, and with *slot utilization external* (SU_{ext}) the channel share that is occupied by its neighbor nodes' transmissions. To compute these quantities, each station observes the channel status during a given time window T, and it counts the if number n_{tx} of its transmission attempts, the number n_{rx} of the number of channel occupations (either correct frames or collided frames) observed by this node, and by n_{idle} the number of slots during which the medium has been perceived idle during this interval (including *DIFS* and *EIFS* periods). It is straightforward to note that the total number of busy periods during the observation time T is equal to $n_{tx} + n_{rx}$. From these measurements, the two slot utilization values are computed as follows:

$$SU_{\text{int}} = \frac{n_{tx}}{n_{idle} + n_{tx} + n_{rx}} \quad (2.a)$$

$$SU_{\text{ext}} = \frac{n_{rx}}{n_{idle} + n_{tx} + n_{rx}} \quad (2.b)$$

To obtain per-station fair allocation of the channel bandwidth, each station should dynamically tune its contention windows such as to equalize the SU_{int} and SU_{ext} values (this process is somehow similar to the equalization of the normalized throughout shares described in [BWK00]). To achieve this objective we modify the probability of transmission used in the original AOB (see formula (1)) as follows:

$$P_T = 1 - \left[\min\left(1, \frac{SU_{\text{int}} + SU_{\text{ext}}}{ACL(q) - SU_{\text{int}}}\right)\right] \quad (3)$$

Expression (3) is decreasing as the medium load increases, and especially as the share of medium load due to the computing station increases. Consequently, stations with a

disproportionate channel share are less aggressive in accessing the channel than stations with lower channel shares. Intuitively, decreasing the probability of transmission as medium load due to other nodes' emission increases could have the effect of accentuating fairness issues in some cases. Nevertheless, as this expression decreases faster in function of S_U_{int} than in function of S_U_{ext}, nodes responsible of unfairness are more penalized than nodes suffering from unfairness.

Preliminary results indicated that this expression of the probability of transmission results in an improved fairness in multi-hop configurations. However, refraining from transmitting could cause a global throughput decrease in certain situations. To resolve this limitation we thread the footprints of [BCG05], by introducing a *credit* and *reward* mechanism that cooperates with the backoff procedure. The basic idea behind this mechanism is that each station collects credits when it refrains from transmitting due to the probability of transmission mechanism. The credits are then spent to perform backoff-free consecutive transmissions when the emitter gains access to the medium, compensating the eventual performance loss due to releasing transmission attempts granted by the standard backoff protocol. In other words, the backoff-free transmission opportunities are a sort of reward for the emitters that participate to the good operation of the network.

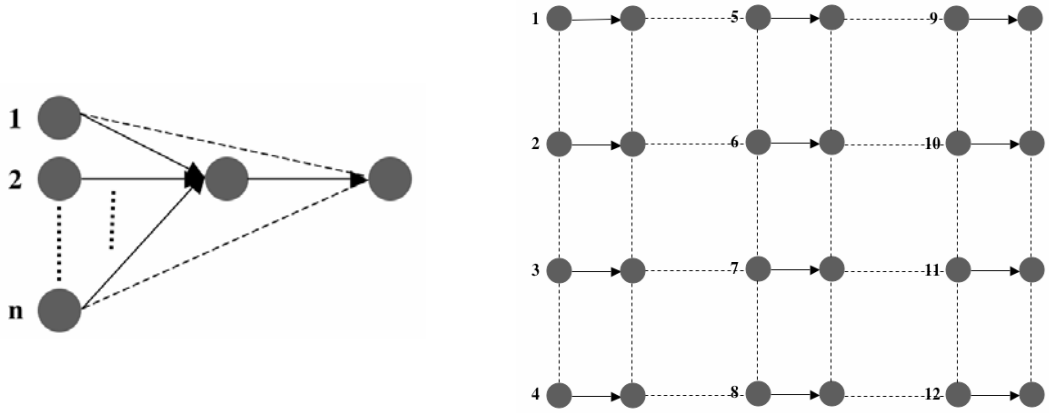
When deferring a transmission, a node earns a number of credits corresponding to the mean backoff it would have drawn, i.e., $(CW-1)/2$, CW representing the current contention window (maximum backoff) value. To better adapt this scheme to the multi-hop environment, and to deal with the asymmetric conditions in which a signal is perceived on the medium, blocking the backoff decrease process and triggering an *EIFS*-long waiting time, as in the Large-EIFS problem, a station collects credits also during the *EIFS* deferral time. In particular, when an *EIFS* timer terminates, a node earns a number of credits corresponding to the number of idle slots contained in the *EIFS* period. A question that naturally arises is: how many credits should each node spend for a backoff-free transmission? The credits are an average measure of time loss due to additional backoffs; hence, the node should consume a number of credits equal to the backoff it would have chosen to perform a regular transmission. In order to limit burst sizes, we also define a *credit cap*, the maximum number of credits a node is allowed accumulating. It is worth noting that a node transmitting a burst of frames keeps on collecting statistics, as well as its neighbors. During this period, the number of idle slots decreases for both sides, whereas the number of busy slots is increased by one. Consequently, at the end of the burst, the probability of transmission of the bursting emitter is slightly lower than the one of its neighbors.

Summarizing, the proposed extension of the AOB protocol operates as follows. When a node gain the channel access according to the probability P_T defined in (3), it is allowed transmitting multiple frames in a single burst, which size l is dynamically computed by using the credits owned by that node (the node contends only for the first frame transmission in the burst, while the other $l-1$ are backoff-free frame transmissions separated by a *SIFS* period. This is similar to the transmission process of fragmented frames). It is worth remarking that frame bursting is a feature already implemented in the 802.11*e* and 802.11*n* extensions of the IEEE standard [XR03], although for different purposes. The proposed credit scheme provides a mechanism to dynamically adjust the

burst size such as to ensure fairness while reducing the protocol overheads.
In the next section, we evaluate the proposed mechanisms in different well-known problematic topologies in order to understand the enhanced MAC protocol behavior and point out its advantages and flaws, which gives indications on further improvements of the protocol.

2.1.4. Performance Evaluation

The extension of the AOB protocol described in the previous section has been implemented and evaluated under the network simulator ns-2 in version 2.27. In order to evaluate only the medium access protocol performance, routing has been replaced by a static offline routes computation based on shortest-path calculation, and ARP mechanism has also been disabled. This way, the only traffic present in the network is due to the data flows. IEEE 802.11 parameters have been tuned to correspond to the HR-DSSS 11 Mb/s physical layer [I01]. All simulations are performed using 1000-bytes long payload frames, and the sources have an infinite amount of traffic to send. Each emitter transmits frames continuously in order to saturate the medium. In these simulations, the slot utilization values are computed by each node using an observation period T of 50ms with an exponentially smoothed moving average with a 10% smoothing percentage. This way, the average slot utilization is mainly computed over a 500ms time window. In the scenarios simulated below, in which the duration of the flows carried by the network is infinite, the duration of the observation period has a negligible impact on the scheme performance. Nevertheless, in more realistic situations, with frequent short-lived flows (*mice*) as in wired networks the T period may have an impact on the scheme performance. Future work will include a more thorough study of this parameter.



(a) Large-EIFS(n) (b) Grid Scenario

Figure 2.3. Topologies considered in the simulations.

The results presented in the following were obtained by averaging the throughputs achieved by the data flows over 10 simulations in the scenarios depicted in Figures 1.1 and 1.2 and other well-known problematic topologies (see Section 2.1.1). Specifically,

we have considered two pairs placed in mutual carrier sense range to quantify the performance enhancement achieved on a symmetric scenario in which the use of $EIFS$ deferral results in a performance decrease. We have also extended the Large-EIFS scenario presented in Figure 2.2, by adding different numbers of left-side emitters (we denote as Large-EIFS(n) the configuration with *n* left-side emitters), as depicted in Figure 2.3(a) . These extended configurations suffer from the same kind of fairness issue as the basic scenario. Finally, we have also investigated the grid scenario depicted in Figure 2.3(b), which was also studied in [FB03]. In the grid scenario, rows are separated by distance greater than the transmission range but still interfere, while the distance between nodes in a row is identical and selected such that each node can interfere their two-hop neighbors' transmissions.

Figure 2.4 shows the total network throughput achieved in the different configurations we simulated. The first bar in a group represents the total network throughput achieved by IEEE 802.11, the second bar represents the performance of Fair-MAC without the credit mechanism, and the subsequent bars show the performance of Fair-MAC with different credits cap values. Figure 2.5 shows the *min-max* ratio of the throughputs, i.e., the ratio of the minimum throughput achieved in the network over the maximum throughput, which quantifies fairness. This metric has the advantage over the usual variance to be independent of the number of considered emitters and therefore does not mask situations in which small amount of nodes are starved.

The extended AOB protocol, without using the credit mechanism, usually highly improves fairness over classical IEEE 802.11, the ratio between the lowest flow and the highest flow being improved by a factor 2 to 3 on average, this improvement reaching a factor of 10 in the grid scenario. Using different fairness criteria (e.g., Jain's criterion) leads to the same fairness improvement conclusions. However, according to the modified MAC protocol, each node refrains from transmitting in order to leave transmission opportunities for less favored flows. This usually results in an overall performance decrease that, in the simulated scenarios, does not go above 20% and is about 15% on average. Activating the credit mechanism is beneficial to increase of the overall throughput, often leading to a better use of the radio medium bandwidth than classical 802.11. In the considered scenarios, by using a credit cap of 1024 credits (corresponding to an average maximum allowed burst size of 66 frames), we experienced a throughput increase between 10% and 20% with respect to legacy IEEE 802.11. However, in the Large-EIFS scenarios, increasing the credit cap value causes a degradation of the fairness improvement, though a credit cap of 1024 credits is still capable of reducing the ratio between the lowest and the highest throughput flows by more than a factor 2.

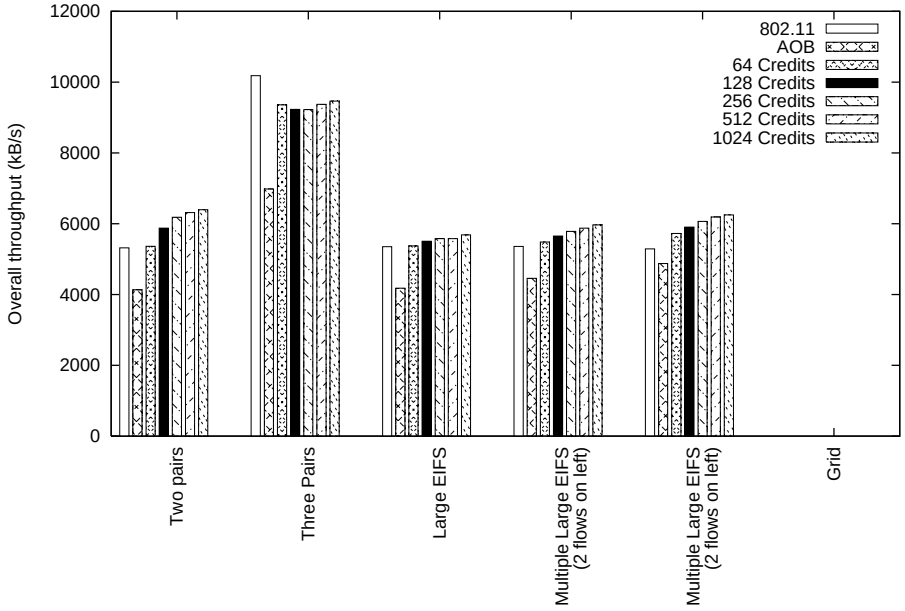


Figure 2.4. Total network throughput in the considered network scenarios.

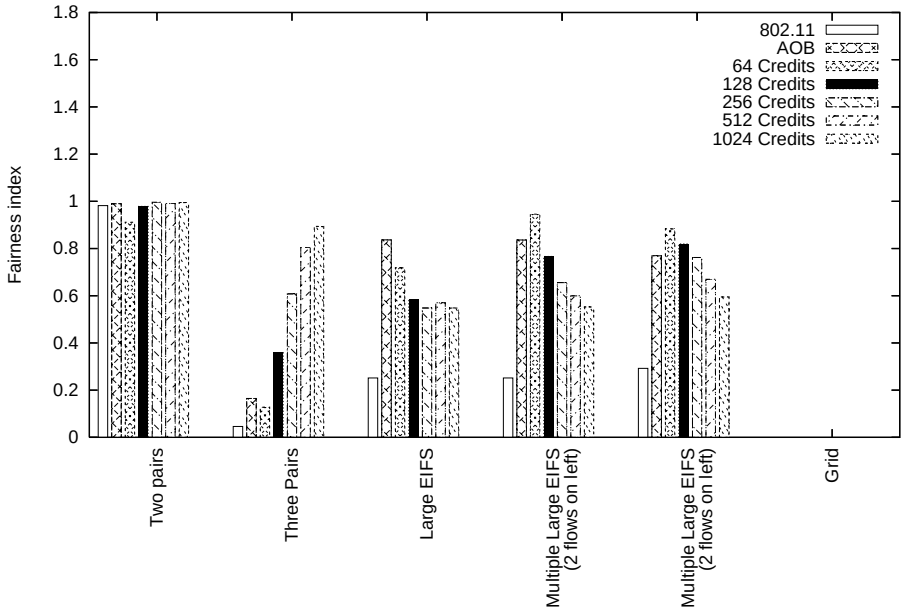


Figure 2.5. Fairness in the considered network scenarios.

Credit caps should therefore be properly tuned to reach the best compromise between throughput increase and fairness improvement. Nevertheless, there is no generic proper limitation of the burst size as the three-pairs scenario behavior illustrates. In the three-pair scenario, increasing the number of maximum credits results in both throughput and fairness improvements. This is due to the peculiarities of this scenario, in which the

exterior pairs already transmit long bursts of frames when using classical IEEE 802.11, therefore the credit mechanism does not affect much their performance. Ensuring a fairer medium access means that the central pair gains access to the medium more often than the two exterior pairs. From the overall system point of view, this results in emitting one frame instead of two in the same time interval. That's why credits enhance both fairness and performance of the mechanism. The grid scenario is somehow similar to the three-pair scenario as fairness decreases only when a too large number of credits can be accumulated (see the 1024-credit cap case).
The foregoing results indicate that it is necessary to trade performance for fairness by properly limiting the allowed burst size. The design of adaptive mechanisms to select the best credit cap value, as well as of different credit evolution schemes that dynamically adapt the credit collection to the slot utilization measures, is an ongoing research activity.

2.2. Tests

2.2.1. Hardware and Software Setup

The hardware setup of the Network Interface Card developed in the framework of the MobileMAN project has been extensively described in Deliverable D12 [D12]. Thus, in the following we will focus on the software components, outlining the different modules that have been so far implemented in the enhanced card.
As explained in Section 2.1.3, the AOB protocol and its extensions are based on the notion of slot utilization. For this reason, the first component that has been developed is the one for the run-time estimation of this important index. Specifically, we do not estimate the aggregate slot utilization, as done in [BCG00], [BCG04], but we split it into two contributions: the internal slot utilization (SU_{int}) and the external slot utilization (SU_{ext}), such as to differentiate between the contribution to the channel occupation due to the node's transmissions and to its neighbors' transmissions. This is motivated by the need to keep our implementation as much flexible as possible, such as to allow future modifications as the one described in Section 2.1.3. Another variation with respect to the original AOB is the time interval over which we compute the slot utilization. In fact, the original AOB computes the slot utilization after each backoff interval, while in our implementation we used a constant observation period T of100ms. This choice is motivated by the need to avoid frequent slot utilization computations, which could interfere with the time constraints of the atomic MAC operations (e.g., RTS/CTS exchange). Then, the SU_{int} and SU_{ext} values are computed using the formulas (2.a) and (2.b). It is evident that the SU used in formula (1) can be computed as the sum of SU_{int} and SU_{int}. Thus, our implementation and the original AOB are equivalent. It is worth remarking that: *i)* two channel occupations should be considered separated only when they are separated by an idle period longer than the $DIFS$ period. This guarantees that the MAC ACK frames are not counted as channel occupations different from the data frames they acknowledge. *ii)* To compute the T_{idle} it is necessary to count also the idle periods during which the $DIFS$ and $EIFS$ timers are active, and not only the backoff idle slots.
Using formulas (2a) and (2b) we compute a single sample of the slot utilization. However, to avoid sharp fluctuations in the slot utilization estimates we should average

these single measures. Hence, the SU index should be computed applying a moving average window estimator to the samples. Specifically, let assume that the station is observing the channel during the *i*-th observation period. Then, it follows that:

$$\overline{SU}_{\text{int}}^{(i)} = \alpha_1 \cdot \overline{SU}_{\text{int}}^{(i-1)} + (1-\alpha_1) \cdot SU_{\text{int}}^{(i)} \tag{4a}$$

$$\overline{SU}_{\text{ext}}^{(i)} = \alpha_1 \cdot \overline{SU}_{\text{ext}}^{(i-1)} + (1-\alpha_1) \cdot SU_{\text{ext}}^{(i)} \tag{4b}$$

where α_1 is the smoothing factor, $\overline{SU}_{\text{int}}^{(i)}$ ($\overline{SU}_{\text{ext}}^{(i)}$) is the average internal (external) slot utilization estimated at the end of the *i*-th observation period, and $SU_{\text{int}}^{(i)}$ ($SU_{\text{ext}}^{(i)}$) is the internal (external) slot utilization measured during the *i*-th observation period using formula (2a) ((2b)).

Exploiting the SU estimate we can easily compute the P_T using formula (1). Since the $ACL(q)$ value depends almost only on the average frame size q and it doesn't depend on the number of stations in the network, as proved in [BCG04], the $ACL(q)$ values for different frame sizes can be stored *a priori* inside the node transmitter. Similarly to the slot utilization computed in formulas (2a) and (2b), formula (1) could induce sharp fluctuations in the P_T estimate. For this reason also to compute the average P_T value it is necessary to introduce a smoothing function. In particular, let us assume that the *j*-th backoff interval is terminated (i.e., the backoff counter is zero). Then, it follows:

$$\overline{P_T}^{(j)} = \alpha_2 \cdot \overline{P_T}^{(j-1)} + (1-\alpha_2) \cdot P_T^{(j)} \tag{5}$$

where α_2 is the smoothing factor, $\overline{P_T}^{(j)}$ is the average probability of transmission to use when deciding whether performing the transmission attempt or not, and $P_T{}^{(j)}$ is the probability of transmission computed according to formula (1). It is worth noting that it should be $\alpha_2 > \alpha_1$ because the P_T value is updated after each backoff interval, therefore significantly more often than the SU, which is updated only after each observation interval T (for instance, $\alpha_1 = 0.9$ and $\alpha_2 = 0.95$).

The block diagram shown in Figure 2.6 outlines the different components that have been defined to implement the AOB MAC protocol.

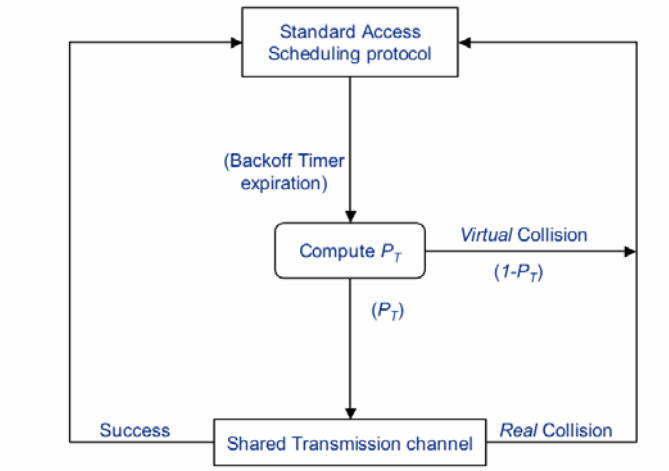


Figure 2.6. Block diagram of the implemented AOB mechanism.

In the original AOB, it is only necessary to compute the P_T value according to formula (1), and to maintain the $\overline{SU}_{\text{int}}$ and $\overline{SU}_{\text{ext}}$ estimates using formulas (4a) and (4b). For this reasons, the implementation of the original AOB scheme has been the first phase of the enhanced MAC protocol implementation. However, throughout the MobileMAN project several extensions to the basic AOB scheme have been designed and evaluated using simulations. These extensions aim at improving the MAC protocol efficiency and at solving unfairness problems that typically occur in multi-hop configurations and heterogeneous environments. The details of these additional features have been reported in previous deliverables (see Deliverables D10 [D10] and D13 [D13]) and in Section 2.1.3 of this document. However, all these modifications are based on a common idea, that is the estimation of the amount of channel time each station is releasing, such as to enable the station to recuperate its released time when new transmission opportunities are granted by the MAC protocol. The implementation of these extensions of the AOB protocol requires designing two basic components: a module for the credit computation, and a module for the average contention window estimation. In the following we describe the operations performed by these two additional modules that have been implemented in the enhanced card.

Credit Computation

The credits are the measure of how many transmission opportunities the station has released due to the probability of transmission. Specifically, each station earns a given amount of credits when it releases a transmission opportunity respect to the standard basic access mechanism, credits that can be spent to perform additional transmission attempts. Let us assume that the *j*-th backoff interval is terminated (i.e., the backoff counter is zero), and that the backoff timer was uniformly selected in the range $[0,\ldots,CW(k)-1]$. If, according to the probability of transmission, the station releases its transmission opportunity granted by the standard backoff, the new contention window

used to reschedule the frame transmission will be $CW(k+1) = \min(2^k, 2^{k_{MAX}}) \cdot CW_{MIN}$. Thus, after the *virtual* collision the number of credits CR owned by that station will be:

$$CR = CR_{old} + \min(2^k, 2^{k_{\max}}) \quad (6)$$

Average Contention Window Estimation

The credits should be used by each station to perform additional transmission attempts using a null backoff. Specifically, let assume that the station is authorized by the standard backoff rules to access the channel, and it decides to perform a real transmission attempt according to the probability P_T. In this case, our modified mechanism could authorize the station to transmit multiple frames in a burst whether it owns enough credits. It is worth pointing out that transmitting a burst of data frames is equivalent to deliver a longer frame, because we are not introducing new collisions (the concatenated frames are transmitted with null backoff and are separated by a $SIFS$, hence the collision probability is negligible). But, how many credits would be needed to perform an additional frame transmission? To answer to this question we need to have an estimate of the average backoff value that the standard backoff scheme would use in the case that no filtering of the channel access is implemented. To achieve this goal, it is necessary to remind that the collisions can be either *virtual* collisions when the station voluntarily defer a transmission attempt, or *real* collisions when the station perform the transmission attempt but it doesn't receive the MAC ACK frame. Let us assume that the total number of transmission opportunities assigned to a station before the successful transmission is K, and that K_{rc} have been the real collisions occurred. Hence, $K - K_{rc}$ have been the *virtual* collisions, i.e., the released transmission opportunities. Denoting with $\overline{CW}_{enh}^{(j)}$, the average contention window estimated after the *j*-th successful transmission, and with $\overline{CW}_{std}^{(j)}$ average contention window of the equivalent standard MAC protocol estimated after the *j*-th successful transmission, we have that:

$$\overline{CW}_{enh}^{(j)} = \alpha_2 \cdot \overline{CW}_{enh}^{(j-1)} + (1-\alpha_2) \cdot \sum_{k=1}^{K} CW(k) \quad (7)$$

$$\overline{CW}_{std}^{(j)} = \alpha_2 \cdot \overline{CW}_{std}^{(j-1)} + (1-\alpha_2) \cdot \sum_{k=1}^{K_{rc}} CW(k) \quad (8)$$

The $\overline{CW}_{std}$ value will be used as threshold to decide if the station has enough credits to perform a transmission attempt.

The second phase of the enhanced MAC implementation has required the implementation of the additional two components described above; such as to introduce the capability of frame burst transmission in the AOB protocol. Hereafter, we indicate the AOB protocol extended with the credit mechanism as AOB-CR.
The block diagram shown in Figure 2.7 outlines the different components that have been defined to implement the AOB-CR MAC protocol. In particular, when the station performs a successful transmission attempt, it should compare the available credits against the CW_{std} threshold, computed according to formula (8). If $CR > CW_{std}$, the

station should immediately perform a new transmission attempt separated by a *SIFS* interval from the previous one. It is worth pointing out that transmitting a burst of frames shouldn't affect the computation of the slot utilization. This implies that the transmitter should increment the n_{tx} value by one when transmitting a frame burst. On the other hand, all the other stations also will increment the n_{rx} value by one because the frames in the burst are separated by a *SIFS* interval, thus appearing to the receiver as a single channel occupation.

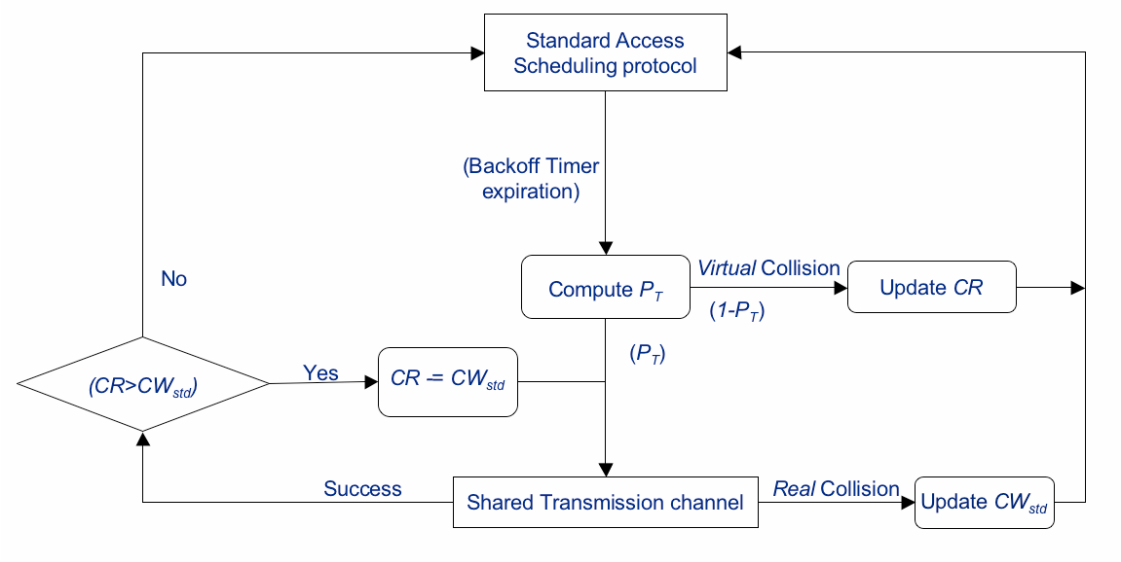


Figure 2.7. Block diagram of the implemented AOB-CR mechanism.

This AOB variant has been tested in the same conditions used for the original AOB protocol to verify that the credit mechanism can further improve the throughout performance.

2.2.2. Experimental results

In order to validate our enhanced architecture we carried out comparative tests of the performance achieved by the legacy IEEE 802.11 backoff mechanism and the enhanced one. In both sets of experiments we used our WNI implementation. All the tests were performed in a laboratory environment, considering ad hoc networks in single-hop configurations. Nodes are communicating in ad hoc mode and the traffic was artificially generated. In our scenarios we used a maximum of 4 stations, due to hardware limitations. However, this is not seen as a problem, because we were already able to demonstrate the performance of our solution and the coherence with previously performed simulations. In Section 2.2.3 we further elaborate on this point, providing simulation results to support our claim.

As discussed in Section 2.1.3, the average backoff value that maximizes the channel utilization is almost independent of the network configuration (number of competing stations), but depends only on the average packet sizes [BCG03]. Therefore, the $ACL(q)$ value for the frames size used in our experiments can be pre-computed and loaded in the

MAC firmware. The implementation in the FPGA of the algorithm defining the $ACL(q)$ value in order to compute it at run-time, is an ongoing activity.
We used different scenarios (2, 3 and 4 stations), in order to study at the same time, the performance of our implementation and the correspondence with the previous simulation results. The stations are identically programmed to continuously send 500-bytes long MSDUs (MSDU denotes the frame payload). The consecutive MSDU transmissions are separated by at least one backoff interval and we did not use the RTS/CTS handshake, or the fragmentation. The minimum contention window was set to $8 \cdot t_{slot}$ (160 μsec), and all values were computed in stationary conditions. The nodes topology is illustrated in Figure 2.8. All the experimental results we show henceforth were obtained by computing the average over ten replications of the same test.

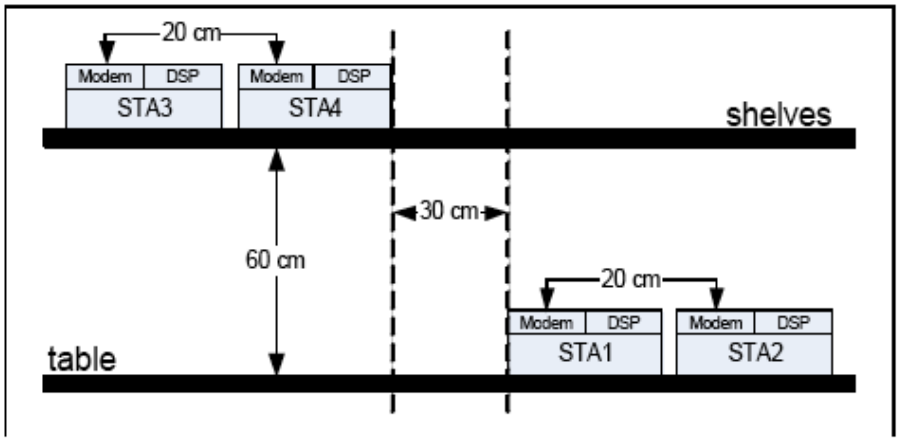


Figure 2.8. Nodes topology used in the measurements.

As already demonstrated in [BCG04] and [BCG05] the AOB mechanism introduces a minimum overhead that could negatively affect the performance of the communications between two stations. Thus, our first set of experiments was carried out to verify this performance decrease in network configurations where two stations are performing either a unidirectional or bidirectional communication, as illustrated in Figure 2.9.

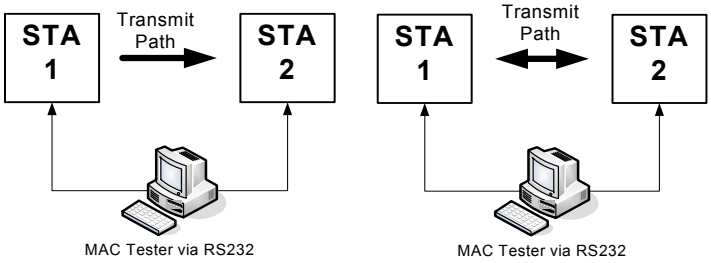


Figure 2.9. Unidirectional and Bidirectional communications with two stations.

The results we obtained in this point-to-point configuration are reported in Table 2.1. In particular, N_{tx} denotes the average number of transmission (either successes of collisions) performed during a period T, while RC denotes the average number of collisions suffered during a period T. Hence, the average throughput TP, expressed in byte/s, can be computed as:

$$TP = \frac{N_{tx} - RC}{T} \cdot MSDU . \quad (4)$$

From the listed numerical results, we can observe that the throughput decrease in the case of two competing stations is lower than 3%.

Table 2.1. Point-to-point scenario results.

	Unidirectional Flow		Bidirectional Flow	
	Standard MAC	AOB MAC	Standard MAC	AOB MAC
SUint	0.09645	0.04114	0.05709	0.02407
SUext	0.00024	0.00040	0.05939	0.02312
P_T	-	0.56149	-	0.52652
Ntx	33.60193	29.88714	16.8941	15.99531
RC	0.18143	0.38419	1.16443	0.72829
TP	**167100**	**147550 (-11.7%)**	**78650**	**76350 (-2.94%)**

In the second set of experiments we considered a network configuration with 3 stations, as depicted in Figure 2.10.

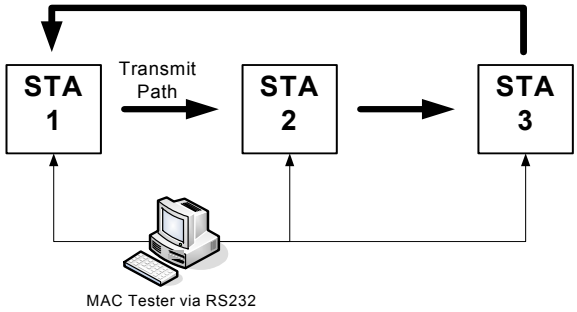


Figure 2.10. 3 stations scenario.

The experimental results we obtained in the 3 stations configuration are reported in Table 2.2. We can note that with three competing stations, the throughput improvement is about 5.7%. This is explained by observing that the number of collisions occurred during a T period is four times less for the AOB MAC protocol than for the standard one.

Table 2.2. 3-stations scenario results.		
	Standard MAC	AOB MAC
SUint	0.04585	0.01911
SUext	0.07509	0.03142
P_T	-	0.49922
Ntx	13.0545	11.4754
RC	2.74377	0.57712
TP	**51555**	**54485 (+5.7%)**

Finally, the last set of experiments was carried out in the 4 stations scenario depicted in Figure 2.11.

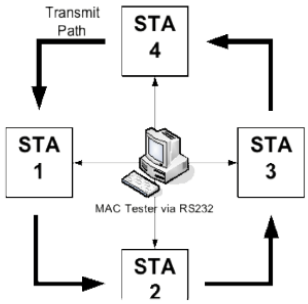


Figure 2.11. 4 stations scenario.

The experimental results obtained in the 4 stations configurations are reported in Table 2.3. These results confirm the positive trend shown in the previous experiments, since the throughput increase in the case of four stations is 9.4%.

Table 2.3. 4-stations scenario results.		
	Standard MAC	AOB MAC
SUint	0.03871	0.01517
SUext	0.09371	0.03846
P_T	-	0.44863
Ntx	9.91685	8.8006
RC	2.18539	0.33916
TP	**38655**	**42300 (+9.4%)**

To summarize, in Table 2.4 we report the aggregate throughput measured in the different network configurations we have tested, considering both the AOB MAC protocol and the AOB-CR MAC protocol.

Table 2.4. Summary of experimental results: total throughput.			
	2 STAs (bidir.)	3 STAs	4 STAs
Std MAC	**157300**	**154665**	**154620**
AOB MAC	**152700 (-2.94%)**	**163455 (+5.7%)**	**169200 (+9.4%)**
AOB-CR MAC	**169495 (+7.75%)**	**170635 (+10.32)**	**169925 (+9.89)**

The above results clearly demonstrate that the AOB MAC protocol may significantly improve the per-station throughput as the number of stations increases, and the throughput improvement is higher when the credit mechanism is employed in the original AOB scheme.

2.2.3. Comparison with the simulations

We carried out experiments with up to four stations due to hardware limitations. However, we argue that the positive trend observed in our tests will be confirmed also for large numbers. To substantiate this statement, in this subsection we show numerical results obtained through discrete-event simulations, considering the same parameter setting adopted during the real tests. In particular, Figure 2.10 shows the channel utilization of the IEEE 802:11 MAC protocol (rate 2Mbps) with and without the AOB mechanism, versus the number of wireless stations in the network for an ideal wireless channel, i.e., not affected by noise. In the same figure we also show the maximum throughput achieved when the stations adopts the *optimal* backoff interval (computed according to [CCG00]). The shown results refer to a payload size of 500 bytes, and were obtained using a minimum contention window equal to 8 time slots, as in our real experiments. From the figure, it is straightforward to note that the throughput measured in the real tests is always lower than the one obtained through simulations. The worse throughput performance is due to the negative impact of the radio interferences that are present in a real environment. However, the comparison between the experimental results reported in Table 2.4, and the simulation results shown in Figure 2.10 indicate the AOB mechanism is quite effective in reducing the throughput degradation caused by the radio interferences. In fact, while the theoretical throughput improvement achieved by the AOB mechanism in a 3 stations scenario is 2.6% in the case of ideal channel conditions, in our tests we measured an improvement of the 5.7%. Similarly, the theoretical throughput improvement achieved by the AOB mechanism in a 4 stations scenario is 4.9%, while in our experiments we obtained a 9.4% improvement. The reason of this better performance is that the efficiency of the AOB scheme increases as the contention in the network increases (as shown in Figure 2.10). Since the frame losses due to channel noise are treated by the standard MAC protocol as normal collisions, the AOB scheme could also be useful to reduce the probability of transmitting frames that may get lost due to the errors induced by channel noise

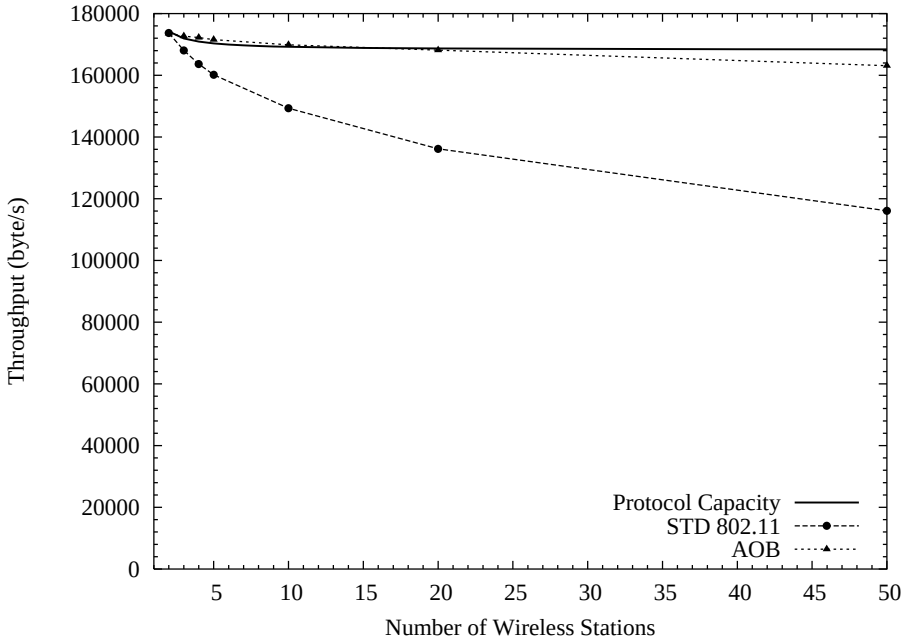


Figure 2.10. Throughput of the IEEE 802.11 protocol with and without the AOB mechanism versus optimal value.

2.2.4. Conclusions

Experiments were carried out with the implementation of an enhanced IEEE 802.11 MAC card adopting the AOB [BCG04] backoff algorithm. The card is still fully compatible with current implementations of the IEEE 802.11 technology because the radio part is compliant to the 802.11 standard. However, the presented experimental results show that the enhanced mechanism outperforms the standard 802.11 MAC protocol in real scenarios. We have also shown that the advantages of this mechanism go further than the high contention scenarios (e.g., ad hoc networks), for which it was designed, because it is also effective in lessening the negative impact of the external interferences, which traditionally decrease the performances of wireless networks in any environment.

2.3. References

[I99]	ANSI/IEEE Std. 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, August 1999
[B00]	G. Bianchi. Performance Analysis of the IEEE 802.11 Distributed Coordination Function. *IEEE Journal on Selected Areas in Communications*, 18(9): 1787-1800, 2000
[XS01]	S. Xu, T. Saadawi. Does the IEEE 802.11 MAC Protocol Work Well in Multihop Wireless Ad Hoc Networks?, *IEEE Communications Magazine*,

	39(6):130-137, June 2001.
[BCG00]	L. Bononi, M. Conti, L. Donatiello. Design and Performance Evaluation of a Distributed Contention Control (DCC) Mechanism for IEEE 802.11 Wireless Local Area Networks. *Journal of Parallel and Distributed Computing*, 60(4):407-430, April 2000.
[BCG04]	L. Bononi, M. Conti, E. Gregori. Run-Time Optimization of IEEE 802.11 Wireless LANs performance. *IEEE Trans. Parallel Distrib. Syst.*, 15(1):66-80, January 2004.
[CCG00]	F. Calì, M. Conti, E. Gregori. Dynamic Tuning of the IEEE 802.11 Protocol to Achieve a Theoretical Throughput Limit. *IEEE/ACM Trans. Networking*, 8(6):785-799, December 2000
[AMB02]	A. Acharya, A. Misra, S. Bansal. A label-switching packet forwarding architecture for multi-hop wireless LANs, in *Proc. of WoWMoM 2002*
[D10]	MobileMAN Deliverable D10, "MobileMAN architecture, protocols, and services – Intermediate Report", http://cnd.iit.cnr.it/mobileMAN/pub-deliv.html
[BCG02]	R. Bruno, M. Conti, E. Gregori. Optimization of Efficiency and Energy Consumption in p-Persistent CSMABased Wireless LANs. *IEEE Trans. Mob. Comp.*, 1(1):10-31, March 2002
[CGMT04]	M. Conti, S. Giordano, G. Maselli, G. Turi. Cross-Layering in Mobile Ad Hoc Network Design, *IEEE Computer*, 37(2):48–51, February 2004
[BCG03]	R. Bruno, M. Conti, E. Gregori. Optimal Capacity of p-Persistent CSMA Protocols. *IEEE Commun. Lett.*, 7(3):139-141, March 2003
[BCG05]	R. Bruno, M. Conti, and E. Gregori, "Distributed Contention Control in Heterogeneous 802.11b WLANs," in *Proc. of WONS 2005*, St Moritz, Switzerland, January 19–21 2005, pp. 190–199.
[BDGP03]	R. Bernasconi, I. Defilippis, S. Giordano. A. Puiatti. an enhanced MAC architecture for multi-hop wireless networks, in *Proc. of PWC2003*, Venice
[FG97]	G. Fullmer and J. Garcia-Luna-Aceves, "Solutions to Hidden Terminal Problems in Wireless Networks," in *Proc. of IEEE SIGCOMM'97*, Cannes, France, September 14–18 1997, pp. 39–49.
[ABCG04]	G. Anastasi, E. Borgia, M. Conti, and E. Gregori, "Wi-Fi in Ad Hoc Mode: A Measurement Study," in *Proc. of IEEE PerCom 2004*, Orlando, FL, March 14–17 2004, pp. 145–154.
[CFG05]	C. Chaudet, D. Dhoutaut, and I. Guérin Lassous, "Experiments of some performance issues with IEEE 802.11b in ad hoc networks," in *Proc. of WONS 2005*, St Moritz, Switzerland, January 19–21 2005, pp. 158–163
[XGB03]	K. Xu, M. Gerla, and S. Bae, "Effectiveness of RTS/CTS handshake in IEEE 802.11 based ad hoc networks," *Ad Hoc Networks*, vol. 1, pp. 107–123, 2003.
[XS02]	S. Xu and T. Saadawi, "Revealing the problems with 802.11 medium access control protocol in multi-hop wireless ad hoc networks," *Computer Networks*, vol. 38, pp. 531–548, Mar. 2002.
[ACG03]	G. Anastasi, M. Conti, and E. Gregori, "IEEE 802.11 Ad Hoc Networks: Protocols, Performance and Open Issues," in *Ad Hoc Networking*, New York, NY: IEEE Press and John Wiley&Sons, 2003
[FB03]	Z. Fang and B. Bensaou, "A Novel Topology-blind Fair Medium Access Control for Wireless LAN and Ad Hoc Networks," in *Proc. of IEEE ICC 2003*, Anchorage, AL, May 11–15 2003, pp. 1129–1134.
[BWK00]	B. Bensaou, Y. Wang, and C. Ko, "Fair Medium Access in 802.11 based

	Wireless Ad-Hoc Networks," in *Proc. of ACM MobiHoc 2000*, Boston, MA, August 11 2000, pp. 99–109.
[BH03]	L. Buttyan and J. Hubaux, "Stimulating cooperation in self-organizing mobile ad hoc networks," ACM/Kluwer Mob. Net. and Appl. (MONET), vol. 8, no. 5, pp. 579–592, October 2003.
[NKGB00]	T. Nandagopal, T. Kim, X. Gao, and V. Bharghavan, "Achieving MAC Layer Fairness in Wireless Packet Networks," in *Proc. of ACM MOBICOM 2000*, Boston, MA, August 6–11 2000, pp. 87–98.
[CDG05]	C. Chaudet, D. Dhoutaut, and I. Guérin Lassous, "Performance issues with IEEE 802.11 in ad hoc networking," *IEEE Communication Magazine*, 2005, to appear.
[ZNG04]	L. Zhifei, S. Nandi, and A. Gupta, "Improving MAC Performance in Wireless Ad-Hoc Networks Using Enhanced Carrier Sensing (ECS)," in Proc. of IFIP NETWORKING 2004, vol. 3042. Athens, Greece: Springer - LNCS Series, May 9–14 2004, pp. 600–612.
[JLB04]	R. Jurdak, V. Lopes, and P. Baldi, "A Survey, Classification and Comparative Analysis of Medium Access Control Protocols for Ad Hoc Networks," *IEEE Communications Surveys & Tutorials*, vol. 6, no. 1, pp. 2–16, First Quarter 2004.
[ONKC99]	T. Ozugur, M. Naghshineh, P. Kermani, and J. Copeland, "Fair media access for wireless LANs," in *Proc. of IEEE GLOBECOM'99*, Rio de Janeireo, Brazil, December 5–9 1999, pp. 570–579.
[LLB00]	H. Luo, S. Lu, and V. Bharghavan, "A New Model for Packet Scheduling in Multihop Wireless Networks," in *Proc. of ACM MOBICOM 2000*, Boston, MA, August 6–11 2000, pp. 76–86.
[HB01]	X. Huang and B. Bensaou, "On Max-Min Fairness and Scheduling in Wireless Ad-Hoc Networks: Analytical Framework and Implementation," in *Proc. of ACM MobiHoc 2001*, Long Beach, CA, October 4–5 2001, pp. 221–231.
[QS02]	D. Qiao and K. Shin, "Achieving Efficient Channel Utilization and Weighted Fairness for Data Communications in IEEE 802.11 WLAN under DCF," in *Proc. of IEEE IWQoS 2002*, Miami Beach, FL, May 15–17 2002, pp. 227–236.
[XR03]	Y. Xiao and J. Rosdahl, "Performance Analysis and Enhancement for the Current and Future IEEE 802.11 MAC Protocols," *ACM SIGMOBILE Mobile Comp. and Comm. Review*, vol. 7, no. 2, pp. 6–19, April 2003.
[I01]	ANSI/IEEE Std. 802.11b: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification/Amendment 2: Higher-speed Physical Layer (PHY) in the 2.4 GHz band, Nov. 2001.
[D13]	MobileMAN Deliverable D13, "MobileMAN Domain Modelling", http://cnd.iit.cnr.it/mobileMAN/pub-deliv.html

3. NETWORKING

3.1. Routing experiments in small scale test bed

In this Section we present an experimental evaluation of full ad hoc network architecture with particular attention to routing layers. In particular we set up a MANET prototype on which we performed a large set of experiments. Specifically, we evaluate performance of OLSR and AODV either in indoor and outdoor environments on networks of 2-4 hops size with up to 8 nodes; in a first phase, we analyzed performances in case of static networks, then we considered scenarios with low mobility. These scenarios could seem not meaningful if compared to those simulations scenarios using hundreds of mobile nodes, but indeed they represent realistic scenarios of few people exploiting the Ad Hoc network to share documents. In fact, as pointed out in [GLNT] with current technology, benefits of Ad Hoc network will vanish beyond the Ad Hoc horizon of 2-3 hops and 10-20 nodes. Our analysis shows that with semi-static topology the proactive approach performs much better than the reactive from the efficiency and QoS standpoint, and it introduces a limited overhead. On the other hand, even in this simple scenario, AODV performances are often poor introducing delays of seconds in order to ping a node few hops away.

3.1.1. Experimental Environment

In order to compare the selected protocols we used two robust implementations (among those available): UNIK-OLSR [OLSR] by University of Oslo-Norway, and UU-AODV [AODV] by Uppsala University-Sweden. The measurement test-bed was based on an Ad Hoc network made up of laptops with different capabilities running Linux and equipped with two different wireless cards compliant to IEEE 802.11b standard working at a constant data rate (11 Mbps). Moreover, we considered a static network where all stations do not change their position during the experiments, introducing also scenarios with topology changes due to nodes' connection/disconnection; in addition we performed experiments with low mobility consisting of few nodes moving in the network during the last of the experiment. All the indoor experiments took place at the ground floor of CNR campus in Pisa [MMD]. The structural characteristics of the building (e.g., the variety of materials used for walls), strictly determine the transmission coverage for nodes of a wireless network situated within, making hence the wireless links quality varying in a continuous and unpredictable manner. As a result the whole place can be considered a realistic environment for testing an ad hoc network. In order to perform experiments on string topology we also set up the network in an open field of about 300 meters long. There were no physical obstacles (e.g., buildings, trees) among nodes, thus each couple of adjacent stations was in line-of-sight and in their respective transmission ranges.

3.1.2. Experimental Analysis

In this section we present the performance of the two routing protocols in the indoor and outdoor scenarios, and their behavior in static and mobility conditions.

Indoor Experiments

The reference scenario for the experiments is presented in Figure 3.1. It shows a multi-hop network of 8 nodes in which only nodes connected with lines are in direct communication. The performance comparison is based on the following performance indices:

- Overhead introduced in the network due to routing messages;
- Delay introduced in data transfer.

In addition to the routing protocol, we introduced some traffic at the application layer using the ping utility. This guarantees that AODV runs in a complete manner; otherwise, without any application-level traffic, its routing information is reduced only to Hello packets exchanges. In our scenario, a selected node generates ping traffic towards the remaining nodes of the network according to a random selected sequence, and precisely it pings a node for 1 minute, and then starts pinging the next node in the sequence. We performed two sets of experiments changing the "pinger", e.g. the node selected as the source of ping traffic, in order to evaluate the impact of node position on the routing load. We repeated the same set of experiments several times producing similar results, so we present just one of them.

In the first set, the pinger was the central node E and the sequence was: A, H, D, F, G, B, and C. The resulting behavior of the network in terms of the overhead introduced by OLSR and AODV is presented in Figures 3.2 and 3.3, respectively. The curves show the amount of control traffic observed by each node of the network as the sum of routing traffic generated locally by the node and the one received from other nodes and forwarded by it.

As it clearly appears in the graphs, the position of the node and how it is connected to the other nodes strictly determine the control traffic observed by it. If we look at Figure 3.2 for example we can notice that curves seem to form four clusters.

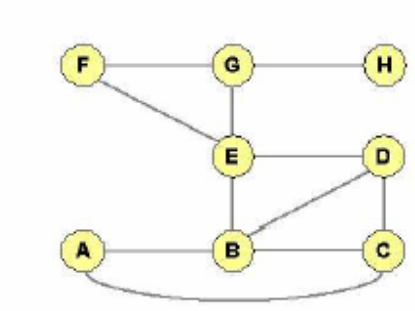


Figure 3.1 Network Topology

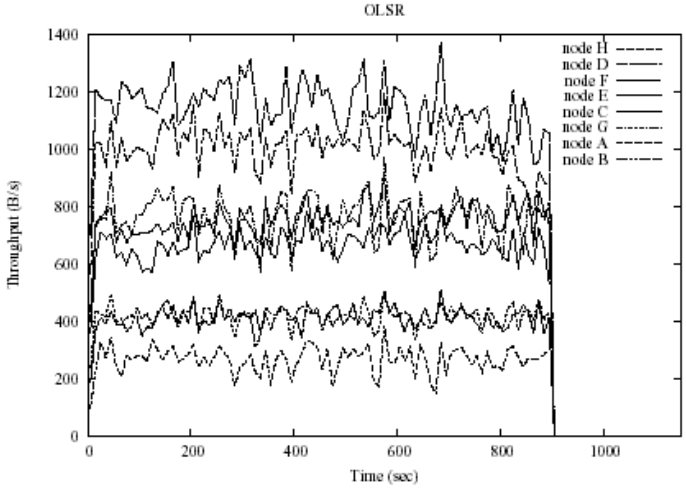


Figure 3.2. Pinger E: OLSR Overhead

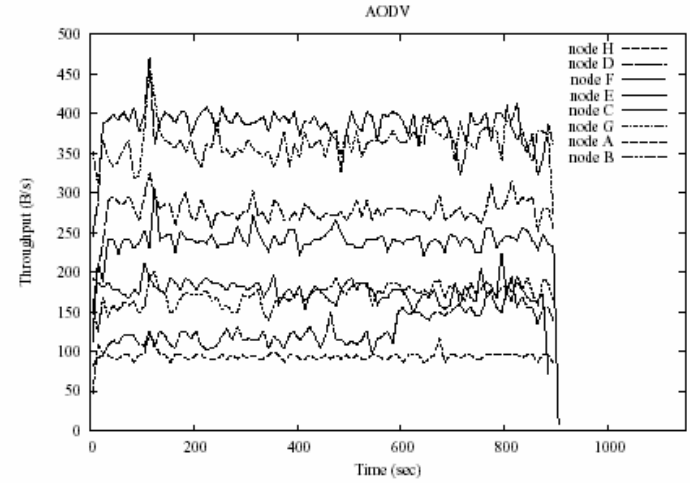


Figure 3.3. Pinger E: AODV Overhead

Specifically, node B and D observe the highest traffic of about 1.1 KBps, nodes C, E and G have an intermediate load around 800 Bps, nodes A and F observe traffic of about 400 Bps and at last node H obtains the lowest load (300 Bps) that represents 1/4 of the traffic load performed by B and D. Thus, we can conclude that there is a connection between the obtained load and the role in the network graph and, more precisely, the traffic load scales with the node's degree. Since node H is a leaf and it is connected to the network with one link it observes the lowest load; while increasing the number of neighbors the introduced overhead is higher. For AODV protocol (see Figure 3.3) we do not observe

the same regular relationship as pointed out previously for OLSR. For example, nodes with the highest degree (B and E) experience an intermediate load. An explanation of this behavior is the reactive nature of AODV protocol that makes routing overhead dependent on the traffic flows at the application level. From the quantitative standpoint, obviously the overhead introduced by OLSR is significantly higher than the one produced by AODV due to the different policy to create and maintain routes. Specifically, OLSR overhead falls in a range of [200-1200] Bps, while using AODV it is around [200-400] Bps. However, it is important to highlight that these values reduce the available 802.11 bandwidth only of a negligible percentage, in the worst case of a quantity of 1.2 KBps. To evaluate the delay introduced by the selected routing protocols we measured the end-to-end latency for completing a simple ping operation between couples of nodes. In the following analysis, we refer to results for nodes at 2 hops distance[1]. At the start-up, when both protocols aren't yet stabilized and all data structures are empty, AODV suffers a delay of 19-20 seconds to find the path toward the destination A, while OLSR requires about 8 seconds completing the same operation. The subsequent ping operations take about 200 msec (or less) when using OLSR (because of its frequent updates of routing tables) and about 1 sec in case of AODV. In the last case, those performances happen when the route has just been stored in some neighbor's cache, and hence the RREQ doesn't need to reach the destination. Observing that the first path discovery requires many seconds, we decided to investigate AODV's performance when routes' entries expire in the cache, thus we introduced a sleep time of 20 seconds between two consecutive ping operations. The measured delay is about 2 sec in almost all cases except the node A case, where we still measure 20 sec. This difference can be explained taking into account that when H pings nodes in the network (excluding A) AODV protocol has already achieved a steady state since each node knows at least 1-hop neighbors due to the Hello's exchange.

To summarize, in this experiment delays introduced by AODV are significantly longer compared to those obtained using OLSR. Furthermore numerical results indicate that QoS problems may occur when using the AODV protocol; applications with time constrains may suffer a long latency to discover paths in a reactive way.

In the second set of experiments we chose the external node H as pinger. It pings continuously (for 400 sec) the same destination A following the shortest path available in the network (H-G-E-B-A). After x seconds from the beginning of the experiment (x equals 250 and 180 sec in OLSR and AODV experiments, respectively), node B disconnects itself from the network. This topology change forces the network to react, searching for a new route in order to deliver packets to node A. After B disconnection, packets start to follow the unique available path through nodes D and C. The results for OLSR and AODV are summarized in Figures 3.4 and 3.5, respectively.

Referring to OLSR (see Figure 3.4), we can note a load distribution similar to the one observed in the first set of experiments, e.g., node B and node H experience the highest and lowest load, respectively. After node B disconnection, there is a transient phase in which the nodes' traffic decreases (due to some missing routes); after this period, a new

[1] As expected there is no difference among protocols when pinging 1-hop neighbors

steady state is achieved. In this new state, we can observe a significant decrease of the traffic in the nodes that are connected with node B (A, E, D, C), while nodes far from the \dead" node perform almost the previous overhead. Once again, the position of a node in the network determines its load.

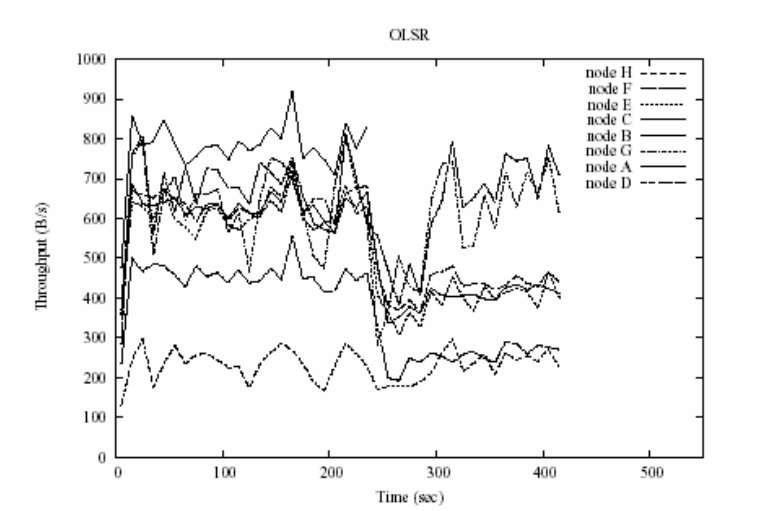


Figure 3.4. Disconnection's event: OLSR overhead

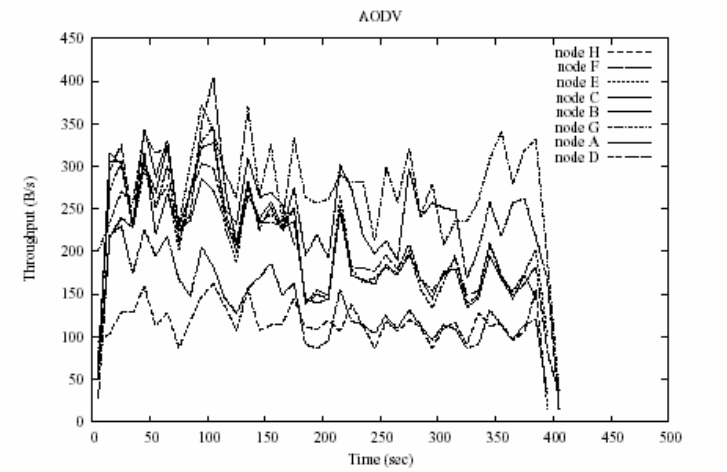


Figure 3.5. Disconnection's event: AODV overhead

Looking at AODV results (see Figure 3.5) we observe less marked differences in the entire duration of the experiment. After the transient state following the B shut down, the active nodes almost observe the same load: the traffic has a range variability of about 100 Bps. This confirms that in this case protocol overhead is correlated to the application flow. As far as the delay is concerned, we got results similar to those observed during the start-up phase of the first set of experiments (i.e., larger delays with AODV). Moreover, referring to the disconnection's event, the ping operation performed while OLSR is

updating its routing tables experience a delay of about 6 sec to be completed; while AODV introduces a delay of [9-14] sec to discover a new route to the same destination A.

String Topology

In a third set of experiments, we compared OLSR and AODV performances in a string topology (see Figure 3.6), both in indoor and outdoor environments. The main performance index used in this case is the Packet Delivery Ratio, calculated as the total number of packets received at the intended destinations and divided by the total number of generated packets. In order to obtain an open environment aligned with the indoor scenario (i.e., a string topology where only adjacent nodes are in the transmission range of each other) we had to increase the distances between stations up to 70 meters, while in indoor, due to walls, doors obstacles, they were at a distance of about 15m. The characteristics of open spaces are quite different from indoor spaces. In both environments, wireless links can vary frequently and rapidly in time and space due to several factors but in the open space the node-distance increase makes the wireless links more unstable. For example, [GKNLYE] shows with an extensive test-bed that wave's propagation in a real environment is very complex depending on phenomena such as background noise, obstacles' presence and orientation between sender and receiver antennas.

Figure 3.6. String Topology

In the outdoor systems, the longer distances cause higher links variability. In addition, it is possible that not all nodes are in the same carrier sense range [ABCG], thus the coordination may result very complex. The outdoor testing methodology is similar to the one adopted in the previous scenario: the sender A pings continuously each node in the network with the sequence B, C, D. In this case the duration of each ping operation is variable and in particularly it scales with the distance to the intended destination (i.e., 1 minute for 1-hop node, 2 min for 2-hops node and so on). Looking at the introduced delay, the outdoor results add no new qualitative information respect of the previous discussion; however, in outdoor, times required to complete a ping operation may further increase due to the links variability; for example, OLSR introduces a 1 sec delay in a 3-hops connection. In addition, in this set of experiments we introduce an evaluation of the packet delivery ratio (PDR) for the two protocols, averaged over several repeated test-runs (see Table 2.2.1).
Looking at the indoor results, we notice that OLSR delivers packets with high probability to all nodes in the string topology; AODV works properly with node in 2-hops neighborhood, but its performance decreases up to 50% of packets delivery when the distance sender-receiver grows up to 3 hops. Currently we are investigating which

phenomena cause the enormous packet loss on AODV. Nevertheless, examining the log files, we notice that sometimes unidirectional links between not adjacent nodes may appear in the network. Since AODV exploits also unidirectional links, it is possible that ICMP packets follow different paths in the end-to-end communication.

	INDOOR			OUTDOOR		
	B	C	D	B	C	D
OLSR	1	0.97	0.98	0.98	0.65	0.47
AODV	0.85	0.9	0.49	0.95	0.01	0

Table 3.1. Overall Packet Delivery Ratio (PDR)

Furthermore, since these links vary with high frequency, every time they disappear a RERR packet is generated and a new path discovery starts (we observe that several times). If no route is found before the timeout expiration all buffered application packets are lost. OLSR doesn't suffer this problem because only symmetrical links are considered resulting in a more stable network. The outdoor results show a good behavior of the two protocols only in the nearby; in fact when the sender-receiver distance increases, both algorithms suffer significant packets' losses. The packet delivery ratio of OLSR decreases up to 50% when it pings the farthest node D. Performances of AODV drastically degenerate when running in outdoor environment: almost all ping operations to nodes distant more than 1 hop failed. In addition, we run the same set of experiments varying the data rate to 2 Mbps. In this case AODV reaches a better performance increasing its PDR up to 0.8 when pinging node C (OLSR result is aligned with AODV), but no improvement is obtained towards node D. It seems that in open space the reactive nature of AODV is more penalized than the OLSR proactive one. Due to walls etc. one-hop distances involved in indoor environment are much shorter than those used in open space, thus a better coordination at the MAC layer guarantees a higher packets delivery. As previously explained, in outdoor it is possible that not all nodes are in the same carrier sense range, affecting the overall performance. Having in advance redundant routing information, as with proactive protocols, guarantees that each node is able to create and maintain its own view of the entire network, even in bad conditions, and consequently delivering at least a percentage of packets successfully. Hence OLSR results more robust than AODV in outdoor environments.

Mobility Scenarios

In this set of experiments we introduced mobile nodes in order to evaluate the impact of the mobility on routing protocols. To this end, we used the same 4-node string topology network as before and we performed three sets of experiments increasing the number of mobile nodes (all the scenarios are shown in Fig.2.2.7). In this scenario the connectivity between the sender and the receiver changes from 1 hop to 3 hops and vice-versa during the experiments. To have a comparison between OLSR and AODV, we studied the PDR (Packet Delivery Ratio) and the delay needed for the network's reconfiguration due to the movements of nodes. In particular, in our scenarios, all the nodes start running the routing protocol and, after an initial period necessary for the network topology stabilization, node A pings continuously node D until the end of the experiment. We

repeated the same set of experiments several times; obtained results were similar, so we present an average of them.

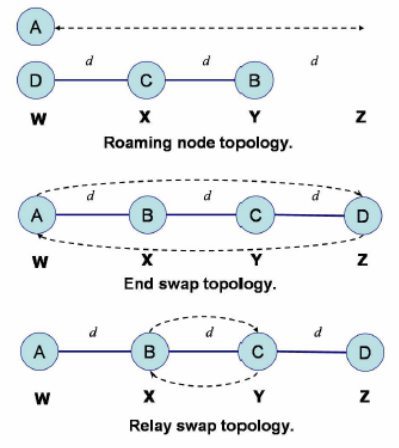


Figure 3.7. Mobility scenarios

One may argue that similar set of experiments were already available in literature. On the other hand we think that there are several main reasons to perform these experiments in our environment:

1. Our cross layer architecture assumes an underlying proactive routing protocol. Comparing AODV and OLSR performance enable us to better understand if and how the proactive assumption impacts on the overall system performance. Previous results [BCDP] and those presented here indicate that in small-medium scale networks and low mobility scenarios OLSR does not penalize the system performance;
2. Measurements related to the topology management provide a reference to understand the behavior of the p2p protocols and, in the specific case of CrossROAD, also give a direct measurement of the expected delays in the overlay construction and reconfiguration. Therefore, a better understanding of the routing protocol performance will be useful when analyzing the behavior of the p2p platforms.

The configuration and the methodology used for the experiments follow those published in [L05], and can be taken as a reference for our performance evaluation. In the first set of experiments, called "Roaming node", there are 3 static nodes (B, C, D) and the "roaming" node A. The experiment lasts 2 minutes: from the initial position W, node A starts moving and every 20sec it reaches the next position in the line (X, Y, Z); once it has reached the last position Z, it immediately moves in the opposite direction following the reverse path and reaches the starting position near node D after another minute. The second set of experiments is referred as "End node swap" due to the movement of the two communicating nodes (A and D), while the rest of the network remains in the same configuration. More specifically, the two end nodes maintain their initial position for the first 20sec of the ping operation, then they start moving reaching the next position in the

PDR	Roaming node	End node swap	Relay swap
AODV	0.87	0.67	0.60
OLSR	0.83	0.77	0.66

Table 3.2. PDR in Mobility scenarios

line every 20sec. The experiment lasts other 20sec after the end nodes have swapped their positions. The last set of experiments, named "Relay swap", is similar to the previous one: there are 2 mobile nodes in the network that change positions during the test. In this case after 20sec from the beginning of the ping operation, central nodes start moving and swap their positions after 20sec, then they remain in this new configuration until the end of the experiment (it lasts 60sec). In all the performed experiments each mobile node moves along the line with a speed of about 1m/s, since we are interested in investigating low mobility scenarios.

Looking at the Packet Delivery Ratio index, as shown in Table 3.2, we notice that increasing the complexity of the proposed scenarios, the performance of the two routing protocols decreases up to about 60% of packets delivery in case of Relay swap scenario. Specifically, in the Roaming node scenario we can note that both protocols have similar behaviors: there is a packet loss of about 25%. Examining the log files, we observe that, for both protocols, packet losses mainly occur when node A goes beyond position Y and reaches the string's end; specifically this represents the time in which the connection A-D changes from PDR Roaming node End node swap Relay swap 2-hop to 3-hop connection, due to the loss of the direct link A-C. In the End swap scenario, the proactive protocol performs better than the reactive protocol: delivered packets increase of 10%. OLSR introduces the high percentage of its packet loss in the last 40sec of the test-run when the connection becomes again a 3-hop connection; on the other hand at the beginning of the experiment all packets were correctly received since the network was already stabilized when data transfer started. In contrast AODV distributes uniformly its packet loss during the entire test-run. As previously said, in the third set of experiments the packet delivery ratio of OLSR and AODV decreases up to 66% and 60%, respectively. In particular, from the log files we notice that packet losses occur during the relay swap phase (i.e., from 20 to 40sec), in which only half of the number of packets generated by node A reaches the destination successfully.

To evaluate the delay introduced by the two routing protocols due to nodes' movements, we measured the time needed to update the routing table for OLSR and to discover new paths to the destination for AODV. In the first scenario, when node A moves toward position Z, OLSR requires 5sec to discover a 2-hop path to D after the direct link A-D is lost; while it needs 10sec when the path in the connection increases from 2 to 3 hops. AODV introduces a delay of 2sec for the first topology change, and 7sec for the second one. Both protocols do not introduce any additional delay in the reverse path (from Z to W position). In the End swap scenario, OLSR introduces a delay of 15sec when the topology changes from a fully connected (each node see all the others) to a topology of three hops. In the same topology change, AODV experiences a delay of 10sec but it also introduces a similar delay to move from the starting configuration to a fully connected

topology. In the last scenario, during the relay movement, OLSR introduces a delay of 15sec for the routing table reconfiguration, while AODV requires 11sec to discover a new route to the destination.

3.1.3. Conclusions

During this study a large number of experiments have been set up in order to obtain real measurements on an ad-hoc-network architecture. In particular we describe the performances of two routing protocols for ad hoc networks comparing them in a real Ad Hoc network with different scenarios and environments. Our results point out severe QoS problems, mainly when using AODV due to the reactive nature of the protocol, and indicate that, with a proactive protocol: i) the response times are much better (200 ms vs. 2 sec when pinging 2-hops neighbors), ii) the protocol overheads, at least inside our small network, are not heavy (i.e., in the worst case 1.2 KBps), and iii) the success of packets delivery is higher. Furthermore, when considering higher level protocols on top of Ad Hoc test-bed, e.g. FreePastry [BCDP], benefits in using a proactive approach are more evident. In conclusion the use of the proactive OLSR does not penalize the system performance either in terms of PDR and reconfiguration delays in static and low mobility scenarios.

3.1.4. References

[LXG03]	H. Lim, K. Xu, and M. Gerla, "TCP Performance over multipath routing in mobile ad hoc networks," in Proceedings of the IEEE International Conference on Communications (ICC), May 2003.
[M95]	J. Meyer, "Performability evaluation: where it is and what lies ahead," in IEEE Internation Computer Performance and Dependability Symposium (IPDS'95), Erlangen, Germany, 1995.
[CGM]	M. Conti, E. Gregori, and G. Maselli, "Reliable and Efficient Forwarding in Ad Hoc Networks," Ad Hoc Networks Journal, Elsevier, to appear.
[CMTG04]	M. Conti, G. Maselli, G. Turi, and S. Giordano, "Cross-Layering in Mobile Ad Hoc Network Design," IEEE Computer, special issue on Ad Hoc Networks, vol. 37, no. 2, pp. 48–51, 2004.
[CCMT05]	M. Conti, J. Crowcroft, G. Maselli, and G. Turi, "A Modular Cross-Layer Architecture for Ad Hoc Networks," in Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks, CRC, Ed., July 2005.
[NS2]	"The Network Simulator - ns-2," http://www.isi.edu/nsnam/ns/.
[NCD]	A. Nasipuri, R. Castaneda, and S. Das, "Performance of Multipath Routing for On-Demand Protocols in Mobile Ad Hoc Networks," ACM/Kluwer Mobile Networks and Applications (MONET), vol. 6, no. 4, pp. 339–349, 2001.
[MD]	M. Marina and S. Das, "On-demand multipath distance vector routing in ad hoc networks." in Proceedings of IEEE International Conference on Network Protocols (ICNP), 2001, pp. 14–23.

[ABCG]	G. Anastasi, E. Borgia, M. Conti, and E. Gregori. "Wi-Fi in Ad Hoc Mode: A Measurement Study". In Proc. of PerCom 2004, Orlando, Florida, March 2004.
[AODV]	AODV Implementation. Dep. of Information Technology, Uppsala University (Sweden). http://user.it.uu.se/~henrikl/aodv.
[BCDP]	E. Borgia, M. Conti, F. Delmastro, and L. Pelusi. "Lessons from an Ad-Hoc Network Test-Bed: Middleware and Routing Issues". In Ad Hoc & Sensor Wireless Networks, An International Journal, Vol.1, Numbers 1-2, 2005.
[GKNLYE]	R. S. Gray, D. Kotz, C. Newport, J. Liu, Y. Yuan, and C. Elliott. "Experimental Evaluation of Wireless Simulation Assumptions". In Proc. of MSWiM 2004, Venice, Italy, October 2004.
[GLNT]	P. Gunningberg, H. Lundgren, E. Nordstrom, and C. Tschudin. "Lessons from Experimental MANET Research". To appear in Ad Hoc Networks Journal, special issue on "Ad Hoc Networking for Pervasive Systems". M.Conti, E.Gregori (Editors).
[L04]	H. Lundgren. "Implementation and Experimental evaluation of Wireless Ad hoc Routing protocols". PhD thesis, http://publications.uu.se/theses/abstract.xsql?dbid=4806.
[MMD]	MobileMAN Deliverables (D8, D10). http://cnd.iit.cnr.it/mobileMAN.
[OLSR]	OLSR Implementation. Institute for Informatics, Oslo University (Norway). http://www.olsr.org.

3.2. Reliable forwarding

This Section copes with performability issues of nodes communication, considering the different causes behind service degradation (e.g. selfish/malicious nodes, network faults). For the concept of performability we point at the definition given in [M95], where performance refers to how effectively (i.e. throughput), or efficiently (i.e. resource utilization) a system delivers a specified service, presuming it is delivered correctly. On the other hand, reliability reflects the dependability (i.e. continuity) of service delivery. We aim at optimizing both performance and reliability measures by improving i) the throughput of data transfer (i.e. service effectiveness) through a lightweight mechanism (i.e. system efficiency); ii) the quality of data transfer, so as to provide continuous network connectivity (i.e. service dependability). To this end, we adopt a simple forwarding scheme, based on multi-path routing, which estimates neighbors' reliability and forwards traffic on most reliable routes. The basic mechanism, called REEF (REliable and Efficient Forwarding) [CGM], is composed by a reliability estimator and a forwarding policy. Every node keeps a reliability index for each neighbor. This measure is affected by all paths rooted at the pointed neighbor and is updated every time the node sends a packet through it. The updating is positive whenever the packet delivery is successful, negative otherwise. In order to understand whether packets get delivered, we use end-to-end acknowledgments. If data packets are sent relying on the UDP protocol, REEF requires the introduction of a notification system that entails the destination node

to send acknowledgments. In case data transfer relies on the TCP protocol (as considered in this work), REEF uses TCP ACKs as delivery notifications. Although REEF works at the network level, it can efficiently retrieve transport layer acknowledgments through a cross-layer interaction with the transport agent. Specifically, REEF is supported by the cross-layer architecture described in ([CMTG04] and [CCMT05]), which allows protocols to exchange information beyond that provided by standard interfaces, and maintaining a clean architectural modularity. After sending a packet, the sender node waits for an ACK from the destination node, and then updates the neighbor's reliability. With a set of routes at hand, REEF can select the best route to forward a packet, according to reliability estimates, which reflect the behavior so far observed.

This mechanism has shown significant improvement on network throughput when a simplified transport protocol with acknowledgment is used [CGM]. However, the impact on transport protocols with congestion control mechanisms, such as TCP, has not been investigated. This is an important aspect because multi-path routing is not always convenient in the ad hoc environment. As studied in [LXG03], and also confirmed in this work, the plain use of multiple routes may degrade the TCP performance. This is due to intrinsic mechanisms of the TCP protocol. When a TCP sender does not get acknowledgment of sent packets, it reduces the congestion window, causing the retransmission timeout to progressively enlarge, leading to high restart latency and very poor efficiency. Another problem is that the round trip time estimation is not accurate under multi-path routing. TCP senders may prematurely timeout packets which happen to take the longest path. Packets going through different paths may arrive at destination out of order and trigger duplicate ACKs, which in turn may trigger unnecessary TCP congestion window reductions.

The first contribution of this work is a novel forwarding policy for the REEF mechanism, which improves the network performability, taking into account the several causes of packet dropping, as well as the above mentioned TCP limitations. The main idea is to combine the reliability of a route with its length, so as to keep the advantages of multi-path forwarding, necessary to tolerate network faults or intentional misbehavior, and limit, at the same time, the drawbacks of using TCP over multiple routes.

In addition, this work provides an accurate performability analysis, which has been carried out with the *ns-2* Network Simulator, to demonstrate the effectiveness of the forwarding mechanism, in realistic environments. To this end, we implemented a multi-path routing protocol, and modified the forwarding agent at the network layer to include the REEF mechanism.

Furthermore, we realized a cross-layer interaction between the forwarding and the transport agents to make them exchange information on TCP acknowledgments. We simulated practical situations where nodes execute file transfers (e.g. FTP) over TCP connections. Besides totally cooperative environments, where only network faults, such as congestion and lossy links, may affect the network performance, the mechanism has proved effective also in presence of a variable percentage of misbehaving nodes. Furthermore, nodes mobility is nicely tolerated.

3.2.1. Overview of REEF

The forwarding mechanism is composed by a reliability estimator and a policy to forward traffic. In the following, we first give a brief overview of the reliability estimator. For a detailed description we point the reader at [CGM]. Then, we describe the new forwarding policy, which is later evaluated with an extensive simulation study. Each node keeps track of neighbors' reliability according to its "personal" experience while transferring data. Whenever a node communicates with another node in the network, it estimates the reliability of the neighbor node involved in relaying its packets. Specifically, it maintains a table of sent packets, storing also the identity of the next hop that has been charged with forwarding the packet toward the destination. Then its reliability is estimated according to the delivery result. If the source node receives a TCP acknowledgment, then all the intermediate nodes have correctly forwarded the packet, and hence the reliability of the neighbor node is positively updated. Otherwise, some node on the path misbehaved, and the neighbor's reliability decreases. This means that the reliability of a neighbor node may be affected by a packet loss that was caused by another node on the path, and quantifies the cooperation/performance/reliability so far observed for paths rooted at that neighbor. The simplest way to estimate the reliability of a neighbor J is

$$R_J \leftarrow \alpha R_J + (1-\alpha)M$$

where α, $0 \leq \alpha \leq 1$, is the percentage of the previous estimate that we consider in the current update, and M represents the present delivery outcome and may assume the following values:

$$M = \begin{cases} 0 & \text{if } s \text{ does not receive ack from } d \\ 1 & \text{if } s \text{ receives ack from } d \end{cases}$$

As previously stated, the reliability estimator works at the network layer and uses information coming from the transport layer. This is possible through a cross-layer interaction between the two protocols, which bases on an innovative architecture that standardizes vertical communication between protocols, and allows for several performance optimizations [CMTG04][CCMT05]. In this specific case, the TCP notifies packet acknowledgments to the forwarding agent through cross-layer events, which in turn trigger the update of the relative reliability index. In this way, the network layer can easily understand the packet delivery status without producing any additional overhead (packet sniffing would result heavy).

Forwarding policy

Reliability estimates are useful to choose the best route for packet forwarding. Whenever multiple paths are available, the route with the highest success probability is desired. However, a forwarding policy must be defined keeping in mind also the limitations of TCP over multi-path routing. Sending packets on routes that highly differ for the number of hops toward the destination has proved to be detrimental for network throughput [LXG03]. For this reason, we propose to select paths according to reliability indexes as well as distances to the destination. In order to combine these two factors, we estimate the average number of transmissions, in terms of hops number, to successfully deliver a packet to its destination. Let n be the number of hops between the source and destination

nodes on a given route, and R_i the reliability index of the first node I on that route. Let us approximate $p = R_i$ the probability to succeed while sending a packet through node I, and suppose to obtain a successful delivery (represented by the p factor) after k-1 failures. Then, we estimate the average number of times the sender transmits a packet to successfully reach the destination node as:

$$E[N_T] = \sum_{k=1}^{\infty} kp(1-p)^{k-1}$$

As the packet delivery may fail at any hop on the path, we consider the distance n between the source and the destination nodes as an upper bound for the number of transmissions along the path, and estimate the average cost E[C] of transmitting through a specified neighbor as:

$$E[C] \le n\sum_{k=1}^{\infty} kp(1-p)^{k-1}$$

If we consider that the mean of the geometric distribution is $1/p$, we can re-write the average cost as,

$$E[C] \approx \frac{n}{p}$$

Thus, we define a new forwarding policy, namely *performability-route* (*p-route*), in the following way. Given $p_i = R_i$ for each neighbor i, and n_i the number of hops to reach the destination through neighbor i, then we choose the route with the minimum cost, in terms of number of transmissions, $\min_i \{n_i / p_i\}$, and if multiple routes have the same value, we randomly choose one of them. This choice is made on a per-packet base (as provided for the route selection by standard forwarding agents), and is repeated by each intermediate hop between the source and the destination nodes, whenever the routing protocols provides only the next hop toward the destination. Furthermore, with this policy, we consider routes that require the minimum (estimated) number of transmissions to successfully reach the destination, and avoid next-hops that are far from the destination.

Performability requirements and goals

As previously stated, performability of data transfer involves service efficiency and effectiveness for measuring performance, and service dependability or continuity for judging the level of reliability. We identify the efficiency as a design requirement for the forwarding mechanism, while service effectiveness and dependability are goals to achieve. Service efficiency focuses on resource utilization and asks for computational lightness. Our forwarding mechanism is based on a cross-layer architecture [CMTG04] [CCMT05] that allows to easily retrieve transport layer acknowledgments, without any additional effort. Furthermore, resources employed to keep and update reliability indexes are minimal: a single value is stored for each neighbor. However, it is worth noting that the hypothesis on multi-path routing brings some overhead in terms of network traffic. Hence, in order to satisfy the need for service efficiency, an evaluation of resource utilization (in terms of network resources) is necessary. To this end, we perform a

simulation study, showing that the overhead produced by multi-path and single path routing protocols are comparable.

Service effectiveness and dependability are achieved through the multi-path forwarding policy that distributes traffic among most successful routes and avoids misbehaving nodes. The idea is to choose the route with a low number of hops and high reliability, so as to minimize the number of transmissions needed to reach the destination. The policy is such that as soon as the reliability of a path decreases, because the sender observes packets losses, and the ratio of its distance to the destination over the reliability of the next node is no more the most convenient, a different (more reliable) route is used, even if it is longer. This allows going round misbehaving points, providing continuous network connectivity. In fact, in case of nodes which are congested, selfish, or malicious, packet dropping may lasts for a while, paralyzing the data transfer. Furthermore, spreading traffic among routes, with the same level of success, operates a load balancing that, in turn, reduces the possibility of congestion, as well as the motivations for selfish behavior. As a consequence, the throughput and the quality of data transfer are increased, thanks to a fairly distribution of traffic on different paths, and the avoidance of misbehaving nodes that cause service interruption.

3.2.2. Simulation Framework

To realize a simulation framework, suitable for a complete evaluation of the reliable forwarding components, we used the Network Simulator *ns-2* (v. 2.27) [NS2], and a library of objects and abstractions provided by the Naval Research Laboratory (i.e., ProtoLib), which includes an implementation of the Optimized Link-State Routing protocol (OLSR). OLSR is a well-established proactive protocol of the MANET IETF working group, which suites our cross-layer architecture and supports several cross-layer optimizations ([CMTG04],[CCMT05]).

Starting from the *ns-2* + ProtoLib basement, we were able to first introduce the cross-layering concepts described in [CCMT05] and, therefore, use them to develop a reliable forwarding agent at the network layer. As detailed in the following section, we extended OLSR in order to have a multi-path version of it, and also re-programmed agents at transport layer (i.e., TCP agents) so as to send cross-layer events for positive and negative acknowledgments. We used TCP-Reno with Delayed Ack.

Furthermore, we modified the forwarding agent at the network layer in order to:

1) catch cross-layer events coming from transport agents;
2) maintain reliability tables according to the notified events, and the routes discovered by the OLSR routing agent;
3) implement the REEF forwarding policy on the reliability tables.

Multi-Path OLSR. Performance of packet forwarding is dependent on the ability of the REEF mechanism to utilize alternative routes when it detects non-operational ones. The availability of redundant routes is usually not provided by common routing protocols that, typically, build shortest-path routing tables. For this reason, we implemented a multi-path extension of the OLSR protocol provided by the ProtoLib package.

OLSR is a proactive routing protocol, based on a link state algorithm and optimized for mobile ad hoc networks. It minimizes the overhead from flooding of control traffic by using only selected nodes, called Multipoint Relays (MPRs), to send and retransmit topology control (TC) messages. When a node broadcasts a packet, only its MPR set rebroadcasts the packet, while other neighbors simply process the packet. TC messages are sent by a node in the network to declare a set of links, called advertised link set, which must include at least the links to all nodes of its MPR Selector set (i.e. the neighbors which have selected the sender node as a MPR). This is sufficient information to ensure the computation of a routing table based on a shortest path algorithm. By increasing the amount of information included in the TC messages, and the number of node sending them, it is possible to build a multi-path routing table. In particular, the requirements for multi-path routing are: i) the advertised link set of the node is the full neighbor link set; ii) besides MPRs each node having at least one neighbor must send TC messages. In practice, these two requirements are easily satisfied by the "all links" feature implemented in the ProtoLib's OLSR agent.

Our contribution to realize multi-path routing was to enhance the OLSR agent implementation with a new procedure, *MakeNewMultipathRoutingTable*, which is based on breadth-first logic. The first step is to add to the routing table all symmetric neighbors (with hop distance h=1) as destination nodes. Then, for each added route, we go through the topology set to build all routes of length 2 (h=2). Again, starting from this new set of routes, all routes 3-hops long are built, and so on. The procedure is repeated starting from the set of routes just added (h=n) and building the set of routes 1-hop longer (h=n+1), until there are no more routes to build. Obviously, in case of multiple routes to a destination passing through the same neighbor we consider only the shortest one.

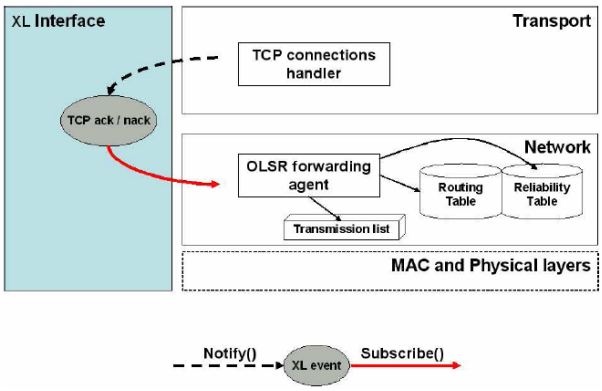


Figure 3.8 Cross-layer interactions between the network and the transport layers.

Introducing cross-layer interactions. After patching the network simulator with the ProtoLib and the multi-path OLSR implementation, we introduced also a set of primitives to allow cross-layer interactions. Specifically, the realization of the forwarding mechanism in our evaluation framework involves the introduction of a class of cross-layer events of type Recv TCP-ack/nack, to which the forwarding agent subscribes for notifications coming from a local TCP agent. These events notify the forwarding agent about delivery outcomes of packets related to connections between the local host and a

foreign party. In particular, the TCP agent sends a TCP-ack event to the forwarding agent whenever it receives a valid acknowledgment. Instead, TCP-nack events are caused by packets retransmissions and generated when: 1) a packet timeout expires; 2) three duplicate acknowledgments on the same packet are received. An event notification causes the forwarding agent to update the reliability index associated to the neighbor through which the packet passed. The update is positive for TCP-ack and negative for TCP-nack.
In order to easily relate notified events with neighbor nodes, the forwarding agent keeps a transmission list containing for each sent packet, the TCP sequence number and flow identity, plus the neighbor through which the packet was sent. When an acknowledgment is notified, the forwarding agent looks for the corresponding packet in the transmission list, in order to retrieve the neighbor that relayed it. Then, it updates the reliability index of such neighbor according to the entity of the event. Packets are stored in the transmission list before being sent, and removed after the reception of a Recv-ack/-nack event. If the received ack is cumulative (i.e., it acknowledges the reception of multiple consecutive packets), then the forwarding agent makes an update for each entry in the transmission list with the sequence number lower or equal to the received ack.

Reliability indexes are maintained in a table containing an entry for each neighbor. In order to keep the reliability table constantly up-to-date, we trigger an update every time there is a change in the neighbor list. Figure 3.8 shows the resulting system architecture, with the cross-layer interaction between the forwarding and TCP agents.

3.2.3. Performability Evaluation

To evaluate the performability of data transfer on top of REEF, we consider the following metrics.
Overhead. Since the cost of internal computation in terms of space and energy consumption is negligible compared to the cost of transmission, we look at the overhead caused by extra routing messages, measured in Bytes/sec. Hence, routing overhead represents a measure of service efficiency.
TCP sequence number. The sequence number of TCP packets acknowledged by the destination, as function of time, is a measure of both performance and reliability. In the first case, a comparison of TCP sequence numbers between different forwarding policies allows to understand which one performs better (i.e. effectiveness). In the second case, a constant increase of TCP sequence numbers shows continuous network connectivity, while a flat line on the plot indicates an interruption of packet delivery (i.e. reliability).
Throughput. We refer to the TCP throughput as the amount of the data correctly received by TCP destination nodes, in a time unit. This metric is useful to quantify the effectiveness of the forwarding mechanism in presence of TCP data transfer.
In the following, we first evaluate the routing overhead produced by our multi-path OLSR. This study is important to demonstrate that it is possible to improve TCP performance at the expense of very low routing overhead. After demonstrating that multi-path routing adds a reasonable amount of overhead, we investigate the impact of REEF on TCP traffic. Performability improvements are analyzed from two perspectives. With a transient analysis, we observe TCP connections on time intervals, in order to check whether our policy improves the quality of TCP data transfer. On the other hand, with a

steady-state analysis, we show statistic results on general scenarios, which measure, for example, the total network throughput by varying the percentage of misbehaving nodes, and the nodes mobility. In order to simulate nodes misbehavior, we used the *SelectErrorModel* class, provided by *ns-2*, which allows to selectively discarding packets, by indicating the packet type (e.g., tcp) and the dropping frequency. Simulations are based on TCP-Reno agents with Delayed Ack.

A. Single vs. multi-path routing

Instructing the routing agent to calculate multiple routes may cause additional overhead, depending on the nature (proactive or reactive) of the protocol. For example, simulation studies on reactive protocols show that there are significant advantages with multi-path routing [NCD] [MD]. Their findings demonstrate that the number of route discoveries and hence of routing load decreases, even though end-to-end delay of data packets slightly increases. Further results show that multi-path routing allows achieving faster and efficient recovery from route failures in highly dynamic networks.

In this work, we consider OLSR, a proactive routing protocol, and evaluate the additional overhead induced by a multi-path version of it. In the implementation of the multi-path OLSR we identified the following as the main requirements: i) the advertised link set of the node must be the full neighbor link set; ii) besides MPRs each node having at least one neighbor must send TC messages. Consequently, the additional overhead produced by the multi-path version is mainly determined by the amount of increased topology information traveling through the network, and the number of generated TC messages. Hereafter, we go through a measurement study to quantify this overhead.

To evaluate the performance of multi-path OLSR with respect to its legacy version, we simulated a range of network scenarios. Figure 3.9 shows the mean overhead (Bytes/sec.) produced by the two protocols, varying the network size. This metric is measured as the mean total number of bytes sent by all network nodes. In particular, we evaluated routing overhead for different network sizes, respectively 10, 20, and 40 nodes. For each network size, we created three different scenarios, with increasing routes length. To this end, nodes are randomly placed in an area of different shapes: from a square to thin rectangles. Results are averaged on the three scenarios. Simulation time is 900 seconds, and Hello and TC message intervals are respectively 2 and 5 seconds. Figure 2 shows that in a static environment, the additional overhead produced by multi-path OLSR is almost independent of the number of nodes, and is around 15%. Considering that in the case of 40 nodes we have less than 2000 Bytes/sec of added routing load, we can state that multi-path routing adds a reasonable amount of overhead, and hence we can have a good trade-off between costs and benefits of our multi-path forwarding.

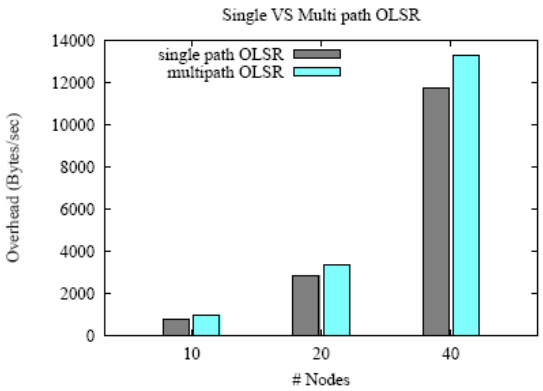


Figure 3.9 Routing overhead in a static network.

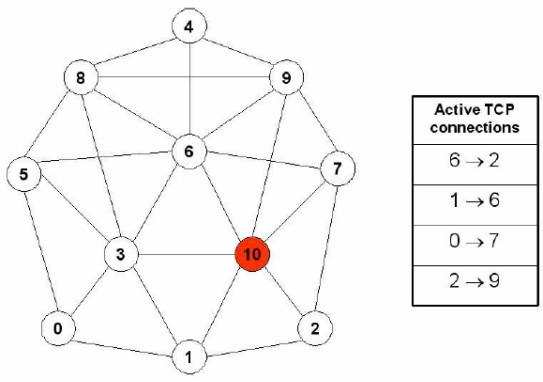


Figure 3.10 Simulated network.

B. The impact of REEF on TCP: A transient analysis

The objective of this evaluation is to find out whether REEF may provide higher throughput and better network connectivity to TCP traffic, with respect to the conventional case in which packet forwarding is based on single path routing (OLSR), where the shortest route is always chosen. In particular, we focus on networks affected by fault conditions and misbehaving nodes that forward traffic in an intermittent fashion.

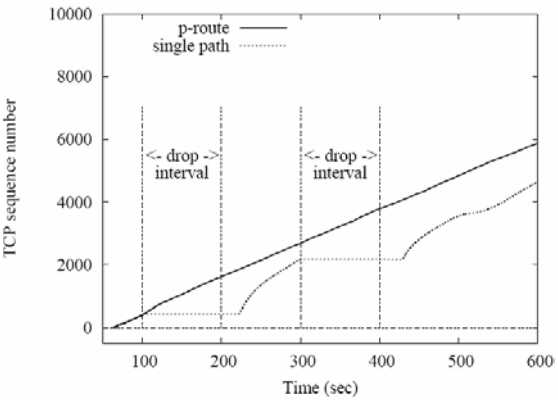
Partially misbehaving network. With this study, we aim at evaluating REEF's effectiveness, investigating the quality of single TCP connections, on a time interval. To this end, we consider a small network, composed of a dozen of nodes, with routes that are 3-4 hops long. The small size of the network allowed us to analyze in details the behavior of single nodes and connections.

The simulated network is composed of 11 nodes, on a 600 by 600 square meters area (see Figure 3.10). One node in the network (i.e. node 10) behaves as on/off forwarder: it does not relay traffic, from second 100 to 200, and from 300 to 400.

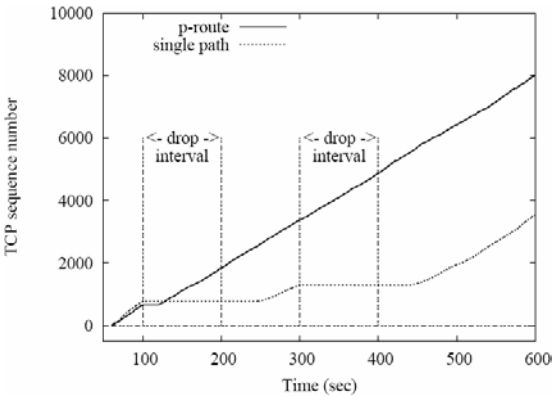
This behavior can be typical of a selfish or a malicious node, as well as a fault condition. We configured 4 TCP connections with an FTP application on top of each as traffic generator. As FTP produces bulk data to send, it may cause situations of congestion. In the simulation, all FTP agents start around time 60 and last for the whole simulation run, which is 600 seconds. In particular, FTP transfers are active between nodes 6 and 2, 1 and 6, 0 and 7, and 2 and 9. To check TCP behavior, we analyzed the sequence number of packets received by the destination nodes (i.e. FTP clients), as function of the application lifetime. In this way, it is possible to understand how the forwarding policies react to nodes misbehavior, and what their impact is on active connections.

The results from this simulation study have validated the performance improvement achieved by the p-route policy. Besides Figure 3.11(a), which deserves to be discussed apart, Figures 3.12(a), 3.12(b), and 3.12(b) illustrates how the p-route policy outperforms single path forwarding. As shown by diagrams, not only p-route achieves the

higher sequence number, increasing it up to 100% (Figures 3.12(b) and 3.13(b)), but it also provides better service delivery.

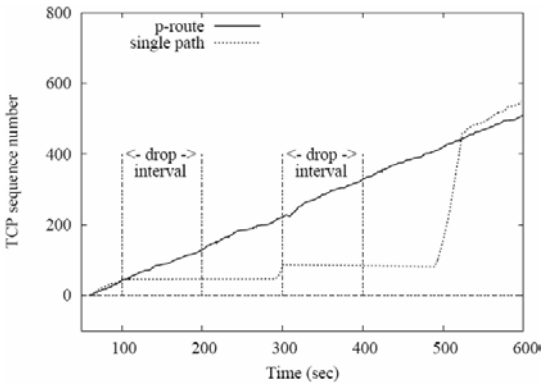


(a) Node 2 is FTP client of node 6.

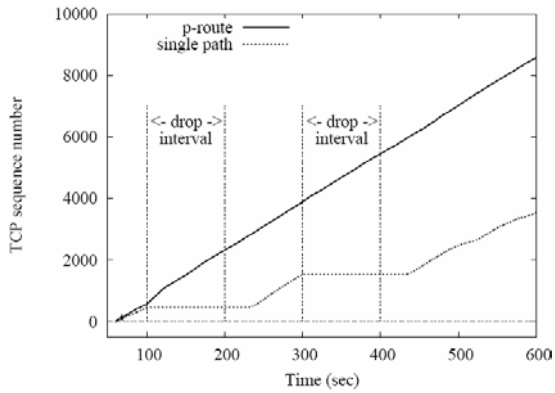


(b) Node 6 is FTP client of node 1.

Figure 3.11. Sequence number of TCP packets received by FTP clients as function of time, in presence of a misbehaving node.



(a) Node 7 is FTP client of node 0.



(b) Node 9 is FTP client of node 2.

Figure 3.12. Sequence number of TCP packets received by FTP clients as function of time, in presence of a misbehaving node.

In fact, p-route offers continuous network connectivity regardless of nodes misbehavior. It uses alternative paths, and hence avoids the delivery interruption, while the other policy shows an evident discontinuous behavior when the selfish node discards packets: flat lines on misbehavior intervals (i.e. 100-200s and 300-400s) indicate a complete inability of nodes to exchange data.

Looking at the TCP connection between nodes 0 and 7 (see Figure 3.13(a)); it is evident that both policies obtain similar results, even if in different ways. Performance on this connection is very poor as the maximum sequence number obtained is around 600, while

the other connections get up to 8000. The reason for this behavior does not directly depend on the applied forwarding policy. Instead, it is due to a combination of factors such as network topology, active connections, and nodes congestion. First of all, we remark that packet loss has detrimental effects on TCP performance, as the congestion window is reduced and the TCP retransmission timeout becomes progressively larger leading to high restart latency and very poor efficiency. Hence, the more packets get lost, the higher is the degradation of the involved TCP connections. Furthermore, the connection between nodes 0 and 7 relies on a route that is longer than the others. Packets have to traverse more intermediate hops, increasing chances of getting into congested nodes. In fact, most of nodes between 0 and 7 are overloaded because of the other active FTP transfers, and hence there is no way to go around the problem. With single path forwarding, the connection experiences long pauses that correspond to misbehavior intervals. With p-route, connections get the best service, without transfer interruptions. However, the last TCP sequence number is quite low. From an analysis of trace files, we observed that the connection is affected mainly by congestion events. This causes poor performance because the neighbors of the sender node do not have knowledge of current network conditions. As they are not involved in end-to-end communications, they do not update their reliability, and hence unconditionally use shorter (but congested) routes. Analyzing more in details the TCP connection between node 0 and 7, the sender can communicate with the receiver through neighbor nodes 3, 5, and 1. In case the delivery relies on neighbor 3, this must choose one route toward the destination. As all neighbors of node 3 have the same reliability, node 3 chooses 6 or 10 as they provide a shorter route. Unfortunately, both of them are not reliable, because the former is overloaded and the latter is misbehaving. Node 6 would be chosen even in the case the communication occurs through neighbor 5. Packets find similar obstacles through neighbor 1, because node 10 is misbehaving and node 2 is involved in the other two connections. The only way to successfully deliver packets to the destination would be through the nodes on the border of the network (i.e. 5, 8, and 9), but this path is longer and intermediate nodes do not have knowledge in order to deviate traffic there. Hence, the protraction of intermittent losses (experienced on almost all used routes) slows down the TCP sender that stabilizes on low sending rate. Figure 3.14 shows the total TCP throughput of FTP clients, as a function of time. In this case, we also report the results obtained with a load-balancing policy, which equally spreads traffic among all possible routes to the destination. We remark that this policy, as well as our p-route policy, makes a per-packet and per-hop choice. P-route achieves the higher throughput and keeps it constant for the whole time, showing high tolerance to misbehaving nodes. Instead, the other forwarding policies are more sensitive to packets dropping; as soon as the misbehaving node starts discarding packets, both single-path and load-balancing experience a sharp drop of network throughput. This result confirms the ineffectiveness of a plain multi-path forwarding, as studied in [LXG03].

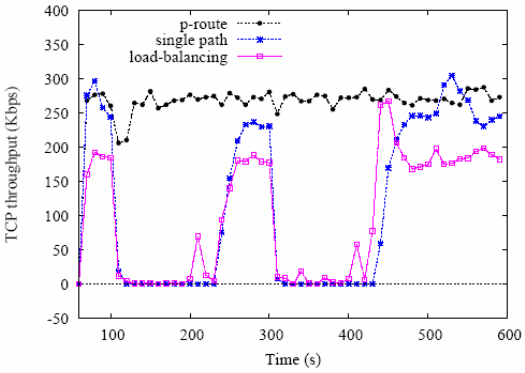
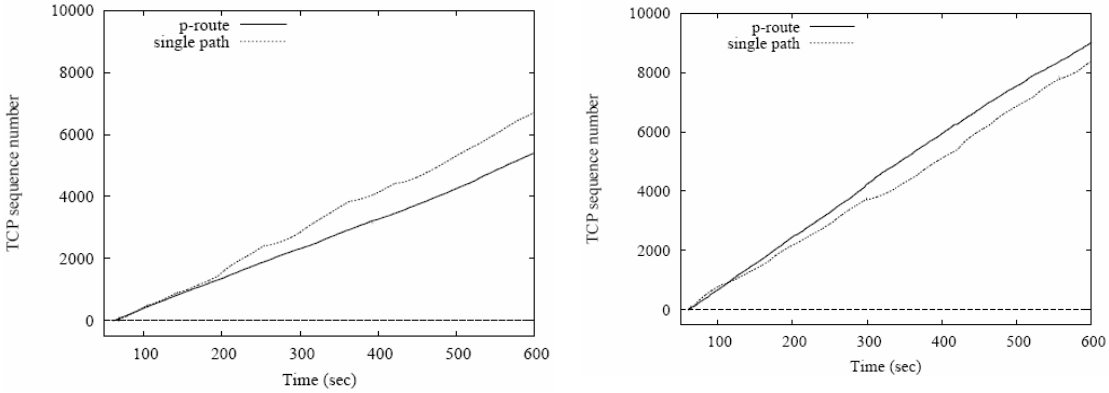


Figure 3.13. TCP throughput on FTP clients obtained by applying the different forwarding policies, in presence of a misbehaving node.

In conclusion, in presence of misbehaving nodes, the p-route policy significantly improves the performance and reliability of TCP connections, achieving a twofold advantage: 1) the amount of packets successfully delivered at destination is increased up to 100%; 2) TCP connections benefit from a delivery service of higher quality, which provides continuous connectivity to the communicating end-points, hiding the effects of misbehaving nodes.



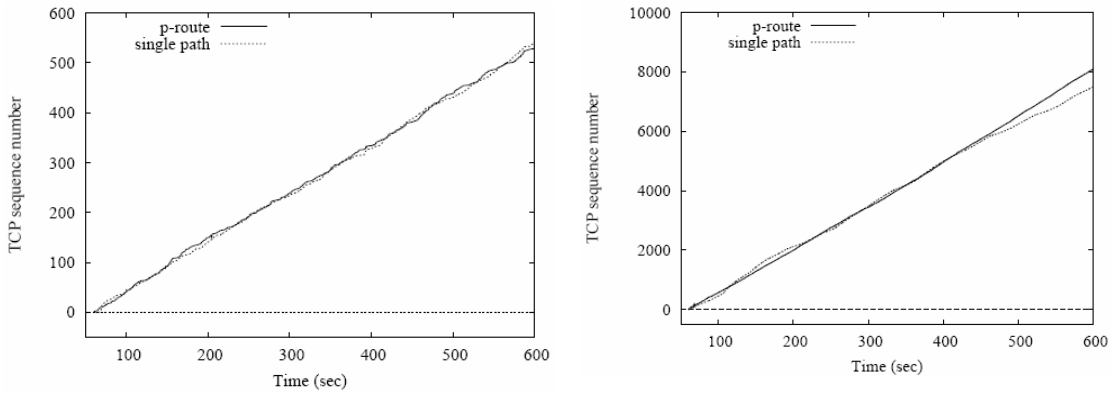
(a) Node 2 is FTP client of node 6.

(b) Node 6 is FTP client of node 1.

Figure 3.14. Sequence number of TCP packets received by FTP clients as function of time, in a network without misbehaving nodes.

Totally cooperative network. To show that REEF yields better performance even in absence of misbehaving nodes, we repeated the simulation on the same network scenario, with the difference that all nodes cooperate to packet forwarding. In this case, packet loss can be caused only by temporary fault conditions. Behavior of TCP connections is depicted in Figures 3.14 and 3.15 Plots show that the p-route policy is globally better than single path forwarding, as it almost always achieves the highest sequence number; the only exception is for the connection between nodes 6 and 2 (see Figure 3.14(a)).

The connection between nodes 0 and 7 shows results similar to the case with the misbehaving node. The low performance for both policies confirms the motivations previously stated. This connection suffers from congestion events on intermediate nodes, and the higher distance between source and destination increases the negative effects.
Finally, Figure 3.16 shows the predominance of p-route over single path. The achieved throughput is on average higher and more uniform for p-route, while single path forwarding presents wide fluctuations, staying often below the p-route curve. The load-balancing policy again performs worse than single path.



(a) Node 7 is FTP client of node 0.

(b) Node 9 is FTP client of node 2.

Figure 3.15 Sequence numbers of TCP packets received by FTP clients as function of time, in a network without misbehaving nodes.

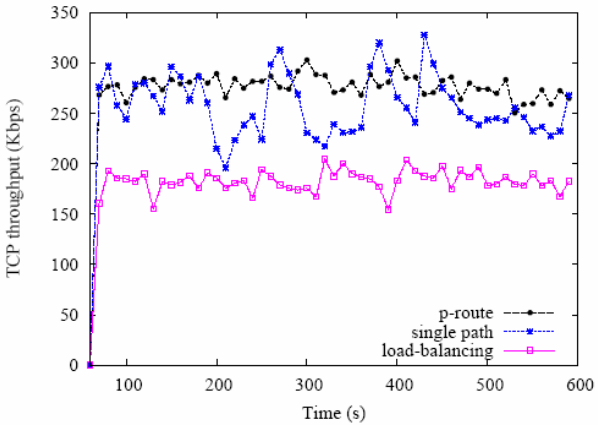


Figure 3.16 TCP throughput on FTP clients obtained by applying *p-route* and single path forwarding, in absence on intentional nodes misbehavior.

C. The impact of REEF on TCP: A steady state analysis

With a steady-state analysis, we show aggregate results on general scenarios, to evaluate the scalability of the REEF mechanism. Specifically, we measure the average total network throughput as a function of the percentage of misbehaving nodes, and mobility.
The experiments have been conducted in various network scenarios. We fixed the network size to 20 nodes, placed in a 1000x700 area, and configured 5 TCP connections. We used Telnet sessions as traffic generators. Packets inter-arrival times are chosen from an exponential distribution with average 0.2 seconds. All TCP connections are established at time 60 (to allow the routing table construction), and last for the whole simulation time that is 15 minutes (900 s). Connection end-points are generated randomly and, for each scenario, 10 runs are performed to decrease the impact of randomness. We then introduced misbehaving nodes that cooperate to routing (and hence they appear in routing tables of the other nodes) but do not forward TCP traffic. The experiments were repeated for the different forwarding policies and the presented results are the average of the 10 runs. Hereafter, we present the performance analysis of our forwarding policy according to the aforementioned parameters.

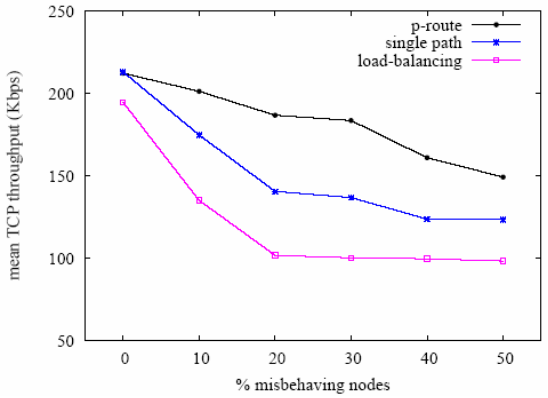


Figure 3.17 Mean TCP throughput in a network of 20 nodes with 5 TCP connections as function of the percentage of misbehaving nodes.

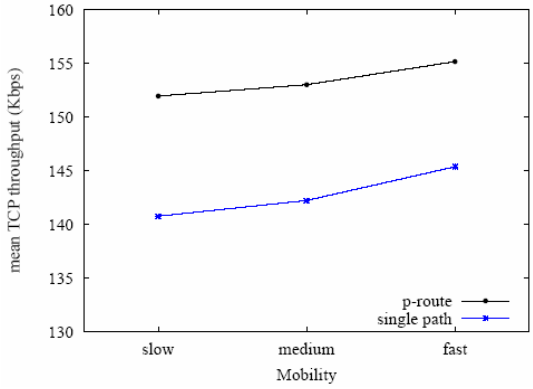


Figure 3.18 Mean TCP throughput as function of nodes mobility in a network of 20 nodes with 5 TCP connections and 30% of misbehaving nodes.

Percentage of misbehaving nodes. The first set of experiments aimed at evaluating the effect of an increasing percentage of misbehaving nodes on the network throughput. To this end, we produced random scenarios with increasing number of misbehaving nodes, which are not endpoints of TCP connections. We created 6 different scenarios, with respectively 0, 2, 4, 6, 8, 10 misbehaving nodes (i.e., 0% to 50%). Figure 3.17 shows that when TCP works on top of load-balancing, it always behaves worse than using single path, regardless of the percentage of misbehaving nodes. This negative result is caused by the combination of two factors: i) spreading traffic among multiple routes increases the possibility to run into unreliable routes (with consequent packet loss); ii) TCP sensitively reacts to packet loss, decreasing the transmission rate.

Hence, some communications encounter misbehaving nodes, even if the shortest path between the endpoints is free of them. This causes TCP to slow down the sending rate, with consequent performance degradation. On the other hand, p-route outperforms single path forwarding whenever misbehaving nodes are present, while the two methods are comparable in cooperative networks (i.e. the percentage of misbehaving nodes is 0). Specifically, the performance gain grows up to 50% in the case of 20% and 30% of misbehaving nodes. This is a visible improvement.

Nodes mobility. In the following set of experiments we wanted to study the effects of nodes mobility on the total network throughput. To this end, we generated random way-point mobility scenarios using the *set-dest* utility shipped with *ns-2*. Considering a fixed population of 20 nodes moving over a rectangular area of 1000 by 700 square meters, with nodes speed uniformly ranging inside [1, 5] m/s, we created three sets of mobility scenarios: 1) a slow scenario with pause times up to 10 seconds; 2) a medium scenario with pause times up to 5 seconds; 3) a fast scenario where nodes continuously move as the pause time is set to 0. The percentage of misbehaving nodes is set to 30% (i.e., 6 nodes discard TCP traffic). Figure 3.18 shows that even in presence of nodes mobility the p-route policy globally achieves better performance than single path forwarding, with a constant increase in network throughput around 10%. This is a significant result because p-route is able to distribute traffic among multiple routes even a dynamic environment.
It is worth noting that throughput increases while mobility goes up. This effect is caused by the random way-point mobility model that tends to group nodes in the middle of the simulation area, making nodes closer and routes shorter. A denser network allows nodes to easily reach each other, increasing their ability to communicate.
Intuitively, we believe that p-route performance can be further improved by providing the REEF mechanism with a cache on reliability indexes. Instead of deleting a node from the reliability table as soon as it is not anymore a neighbor, the idea it to keep it for some time, so as to remember its reliability value in the case it appears again as neighbor node. Ongoing work is evaluating this caching mechanism on different mobility models, such as Group and Manhattan mobility.

3.2.4. Conclusions

The performability of data transfer in ad hoc environments is highly sensitive to packets loss, which may be caused by several factors, such as congestion and lossy links, as well as selfish and malicious nodes. When designing protocols for nodes communications, special care has to be taken to consider all the causes that may degrade the system performance. In the case of TCP traffic, even the use of multiple paths may have negative effects on the network throughput.
This work proposes a new multi-path forwarding policy, performability-route (p-route), which address nodes misbehavior and network faults. Focusing on TCP traffic, we show how the p-route policy tolerates the negative effects that packet loss has on the protocol behavior, and overcomes the limitations of using multiple paths for TCP packets. To show such improvement, we carried on a simulation analysis in realistic environments, with a multi-path routing protocol, and TCP file transfers. We also simulated nodes

misbehavior, so as to investigate the effectiveness of our forwarding policy in both cooperative and partially misbehaving networks. The performability achieved with the p-route policy is compared with the standard single path forwarding, which chooses always the shortest route. Simulation outcomes elect the p-route policy as the more efficient forwarding method, as it shows good tolerance to packets loss, maintaining a satisfactory level of connectivity among nodes, and avoiding delivery interruptions typical of single path forwarding. By increasing the percentage of misbehaving nodes (from 0% to 50%), p-route maintains a high level of efficiency, with a TCP throughput improvement up to 50%. The predominance of p-route is visible even in comparison with a plain multi-path forwarding, or in presence of nodes' mobility.
One may argue that the p-route policy may not make the correct decision in choosing alternative paths, as in some cases the one-link reliability does not correctly capture the reliability of the whole path from a sender to a specific destination. This is possible when some nodes do not communicate spontaneously (open TCP connections) with other parties, and hence have no mean for updating their reliability indexes. In REEF jargon, this is equivalent to have "blind" nodes, where reliability indexes are set to the initial value, and nodes are unable to distinguish the proximity of misbehaving nodes. This behavior has been observed in the simulation study. Obtained results show that choices made by blind intermediate nodes do not cause inefficiency, in comparison with single path forwarding, and result in a multi-path forwarding mechanism that favors shorter routes.

3.3. TPA Implementation and Preliminary Experimental Analysis

This section describes the implementation and preliminary experimental evaluation of the TPA (Transport Protocol for Ad hoc networks) protocol. TPA is a novel transport protocol for Mobile Ad hoc Networks (MANETs) developed in the framework of the MobileMAN project as part of the of the cross-layer network architecture (see Deliverable D13). It was conceived as an alternative to the TCP protocol in MANETs. TCP was originally designed under the assumption that nodes are static and communication links are characterized by a relatively small Bit Error Rate (BER). Under these assumptions packet losses are mainly due to buffer overflows at intermediate routers. However, these assumptions are no longer valid in MANETs since (**i**) nodes may be mobile, (**ii**) wireless links have a BER higher than wired links typically used in traditional networks, and (**iii**) congestion phenomena are mainly caused by contention at the link layer (while buffer overflow at intermediate nodes are rare). Therefore, TCP exhibits poor performance in MANETs, especially when nodes are mobile. Unlike TCP, TPA is tailored to the characteristics of MANETs. In particular, it includes mechanisms for managing route changes and route failures that can occur as a consequence of node mobility. In addition, it uses a different congestion control mechanism with respect to TCP. Finally, the protocol is targeted to minimize the number of useless (re-) transmissions in order to save energy.

A detailed description of the protocol, with a preliminary simulation analysis, was included in Deliverable D13, see also [AACP05]. In this report we present a prototype implementation of the TPA protocol in a UNIX environment. This prototype was used to evaluate the performance of the TPA protocol in a real environment. To this end we set up a testbed based on laptops running the UNIX operating system, and measured the performance of the TPA and TCP, respectively. The results obtained have shown that TPA outperforms TCP.

Hereafter we first describe the protocol implementation in a UNIX environment. Then we introduce the experimental environment used for performance evaluation, and we discuss the results obtained in different operating scenarios.

3.3.1. TPA protocol implementation

Design Guidelines

Network protocols are usually implemented in the kernel space and can be accessed by network application through an interface consisting of a set of system calls (e.g., the socket-based interface). Security and performance are the main motivations behind this approach. We refer to such an organization as monolithic [TNML93] because all protocol stacks supported by the system are implemented within a single address space. However, there are several factors that motivate a non-monolithic organization, i.e., implementing network protocols out of the kernel space. The most obvious of these factors is ease of prototyping, debugging and maintenance. A user-level implementation is appealing especially when developing a novel network protocol as it allows the following benefits.

- *Shortest revise/test cycle.* Kernel development includes an additional step in the revise/test cycle: a system reboot. This inconvenience increases the turnaround between revisions from a few seconds to a few minutes.
- *Easier debugging.* User-level development allows for easier source-level debugging.
- *Improved stability*. When developing protocols in a user-level environment, an unstable stack affects only the application using it and does not cause a system crash.

Therefore, we decided to implement TPA in the user space. Specifically, we implemented it as a user-level library that can be used by software developers for programming network applications for ad hoc networks. Figure 3.19 shows the TPA location in the network protocol stack. Since TPA only requires a datagram service it is implemented on top of the UDP/IP protocols that are accessed through the socket-based system calls.

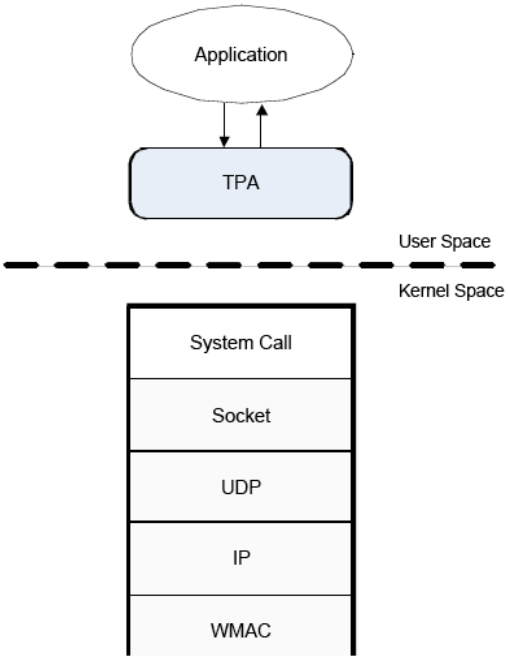


Figure 3.19. Protocol stack

When using a traditional transport protocol implemented in the kernel space, a user application can access protocol services through the socket interface. Therefore, messages generated by the application are passed down from the user space to the kernel space. On the other hand, when using the TPA protocol, data generated by the application are accumulated by the TPA protocol to form a block. Blocks are then segmented and each single segment, or packet, is sent through the network. In our implementation the segmentation process and the processing of each single segment, according to the protocol specifications, are obviously performed by the user-level library implementing the TPA protocol. Therefore, in our implementation, each single segment (not the entire message generated by the application) is passed down from the user space to the kernel space.
In the design of the TPA software architecture our main concern was to allow a transparent and easy integration of legacy applications with the TPA protocol. Therefore, we followed the design principles.

- *Transparent integration with applications*. The original semantics of applications must be preserved without changing their source code.
- *Socket API exportation*. The TPA must provide the same socket application programming interface (API) provided by TCP protocol. This allows the re-use of a legacy application on top of TPA with only minor changes. This requirement is a consequence of the previous one.

The user-level library has been implemented by using the C programming language. This allows the following benefits.

1. it permits to manage bits very quickly and efficiently;
2. it is fully compatible with all Unix/Linux systems.

Application Programming Interface

The application programming interface provided by the TPA protocol is similar to the interface available in a UNIX system when using the TCP protocol. Specifically it consists of a set of functions, each corresponding to a TCP system call. The list of functions provided by the user-level library implementing the TPA protocol is shown in Figure 3.20, whereas the meaning of each function is explained in Table 3.3.

```
int tpa_socket();
int tpa_connect(int sockfd, const char* ip, int port, struct sockaddr_in *from, socklen_t
addrlenfrom, struct sockaddr_in *to, socklen_t addrlento);
int tpa_bind(int sockfd, struct sockaddr_in *from, socklen_t addrlen, int port, const char*
ip);
int tpa_listen(int sockfd, struct sockaddr_in *from, socklen_t *addrlen);
int tpa_send(int socketid, const void* packetbuf, int packetlen);
int tpa_recv(int socketid, void *buff, int len);
void tpa_close(int sockfd);
```

Figure 3.20. List of functions provided by the TPA protocol

Function name	Meaning
tpa_socket()	Creates a TPA socket. It returns an integer value that represents the socket ID
tpa_connect()	Binds a socket with a server identified by ip parameter passed as argument in the TPA connection opening phase.
tpa_bind()	Binds the socket descriptor sockfd to the address specified into from structure.
tpa_listen()	Accepts TPA connection requests.
tpa_close()	Closes a TPA connection opened with the socket ID passed as argument.
tpa_send()	Sends data over a TPA connection identified by the socket ID passed as argument.
tpa_recv()	Receives data over TPA connection identified by the socket ID passed as argument.

Table 3.3. Functions provided by the TPA library.

In addition to the functions listed in Figure 3.20, the TPA library also includes internal functions that are used by functions in Figure 3.20. The fundamental idea behind the TPA software design is that a TCP legacy application can be on top of the TPA protocol by simply introducing very minor modifications. Specifically, each system call must be modified by introducing the tpa_ prefix. To better clarify this issue Figure 3.21 shows changes that must be introduced in a client program when passing from TCP to TPA.

```
/* ... */
s = socket(...);
connect(s, ...);
send(s, ...);
recv(s, ...);
/* ... */
close(s);
```

Standard Library

```
/* ... */
s = tpa_socket(...);
tpa_connect(s, ...);
tpa_send(s, ...);
tpa_recv(s, ...);
/* ... */
tpa_close(s);
```

TPA Library

Figure 3.21. A simple example showing changes to be introduced in a legacy application when using the TPA protocol.

Software organization

The TPA protocol has been implemented with distinct execution flows that interact according to the *client/server* and *producer/consumer* models. Specifically, we structured the TPA protocol by means of three processes: a *data-processing* process, a *sender* process, and a *receiver* process. The *data-processing* process gathers data passed by the application process (by using the `tpa_send()` function in a buffer to form blocks. Data blocks are managed by the sender process according to the TPA specification. Finally the receiver process is in charge of processing ACKs and data coming from the network.

For the sake of space we omit here the detailed description of the above processes. We only provide some details about the timer implementation in the next section.

Timer implementation

The TPA protocol requires starting a timer for each packet sent. Therefore, we used software timers managed by a timer scheduler [L90]. Specifically, we implemented a list of data structures, each representing a timer. The timer whose data structure is at the head of the list is the timer active at the current time. All the other timers are updated in order to timeout and/or removal of any timer.

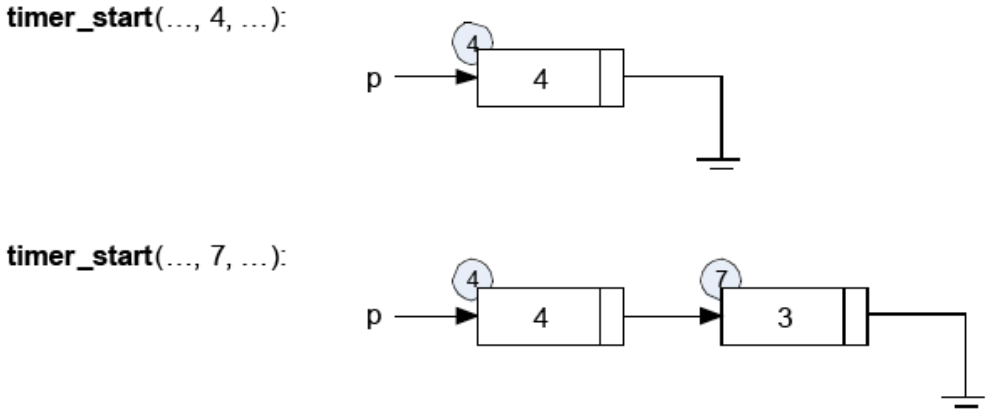


Figure 3.22. Implementation of software timers

Figure 3.22 shows a simple example. A timer that will expire in 4 second is started. Therefore, an item with value 4 in inserted in the list by the timer scheduler. Then, a new timer that will expire in 7 second is started. A second item is thus inserted in the list

whose value (3) is given by the difference between the timer value (7) and value of the active timer (4).

To achieve a finer time granularity, in the timer scheduler implementation we used the system call setitimer() instead of alarm(). This allows to achieve an accuracy in the order of milliseconds (alarm () only provides an accuracy in the order of seconds), which is very important in a MANET environment.

3.3.2. Testbed description

The TPA protocol was evaluated in a Linux environment using the Mandriva Linux LE 2005 distribution (version 10.1) that implements the kernel version 2.6.11. The wide variety of commercial wireless cards -- whose drivers are not always fully compatible -- suggested us to use the same wireless cards in all the machines involved in the experiments. Specifically, we used IBM ThinkPad R50e with bundled Intel PRO/Wireless 2200BG wireless cards. We considered the 802.11b version of wireless cards, and set the data rate to 2 Mbps so as to compare the experimental results with the simulation results provided by the ns-2 simulation tool in the same operating conditions. The routing protocol used in our experiments is DSR. Specifically, we used the Uppsala University implementation of DSR (Dynamic Source Routing), throughout referred to as DSR-UU. Finally, in our tests we set the transmission power (*txpower*) to the minimum value (12dB) in order to minimize the transmission range. This allows the deployment of laptops at reasonable distances, and makes easier to experiment networks with a relatively large number of hops (e.g. 3 hops). Using the minimum transmission power we measured a transmission range of approximately 8.5m and a Carrier Sensing Range of about 16.2m. Based on these values we set the distance between consecutive nodes to 6.5 m. The above parameters are summarized in Table 3.4.

Node distance	6.5 m
TX_range	8.5 m
CS_range	16.2 m

Table 3.4. Network parameters.

3.3.3. Experimental results

We compared the performance of the TPA protocol with those of the TCP NewReno with SACK extension and an MSS value of 1024 bytes. Since the TPA protocol implements a congestion window of 2 and 3 packets (the packet size is equal to the TCP MSS value), in our experiments we limited the TCP congestion window size to 2 and 3 MSSs, respectively. However, for comparison we also considered the default value for the TCP congestion window (W_{max}=85 Kbytes). We considered a string topology where consecutive nodes are at a distance of 6.5m, as anticipated above. In particular, we investigated three specific scenarios, where the distance between the sender and receiver is 1, 2, and 3 hops, respectively. For each scenario we measured the throughput achieved by both protocols for different values of the congestion window size.

Single-hop network

We started our experimental analysis by considering a single-hop network consisting in only two static nodes: a sender and a receiver (Figure 3.23). The results obtained in this scenario are shown in Figure 3.24-left, and summarized in Table 3.5. Figure 3.24-right shows the simulation results provided by the ns-2 tool in the same operating conditions.

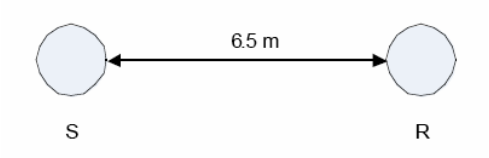


Figure 3.23. Single-hop network.

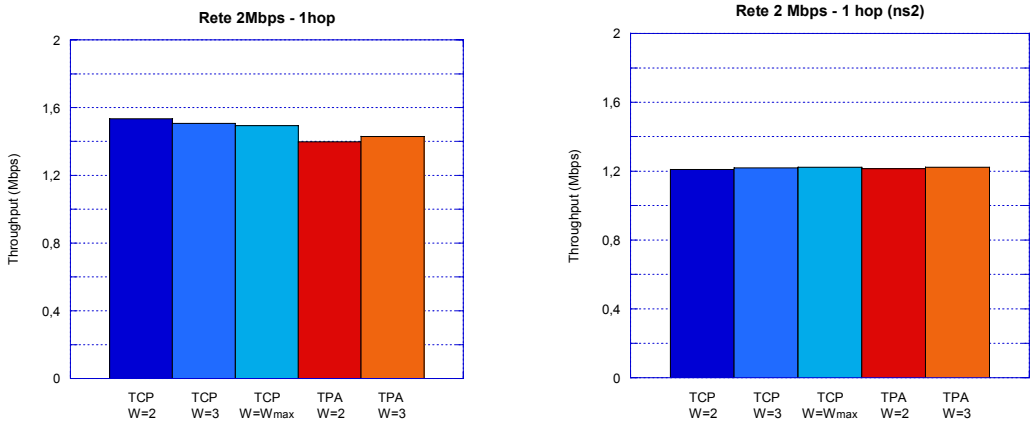


Figure 3.24. Throughput achieved by TCP and TPA in a single-hop scenario with different congestion window sizes: experimental measurements (left) and simulation results (right).

$TCP_{1\ hop}$			
W_{size}	**TH_{avg} (Mbps)**	**TH_{min} (Mbps)**	**TH_{max} (Mbps)**
2	1,53	1,50	1,55
3	1,50	1,48	1,53
W_{max}	1,49	1,46	1,51

$TPA_{1\ hop}$			
W_{size}	**TH_{avg} (Mbps)**	**TH_{min} (Mbps)**	**TH_{max} (Mbps)**
2	1,39	1,38	1,40
3	1,42	1,42	1,43

Table 3.5 Throughput achieved by TCP and TPA for different congestion window sizes.

The simulation results do not show a significant difference between TCP and TPA. On the other hand, from the experimental analysis it comes out that the throughput provided experienced by the receiver is better when using the TCP protocol (in the order of 5-10%). This apparently strange behaviour can be explained by observing that TPA is implemented in the user space while TCP is implemented in the kernel space. Therefore, TPA protocol accounts for a higher overhead due to process synchronization, timer management and so on. All these issue contribute to decrease the protocol efficiency. In addition, in a single-hop environment packet losses have not a sever impact on the performance of TCP.

2-hop network

We now considered a 2-hop network of static nodes (Figure 3.25). We still have a string topology, but the sender is now transmitting to the receiver through an intermediate node. The results obtained in this scenario are shown in Figure 3.26-left, and summarized in Table 3.6. As above, Figure 3.26-right shows the simulation results provided by the ns-2 tool in the same operating conditions.

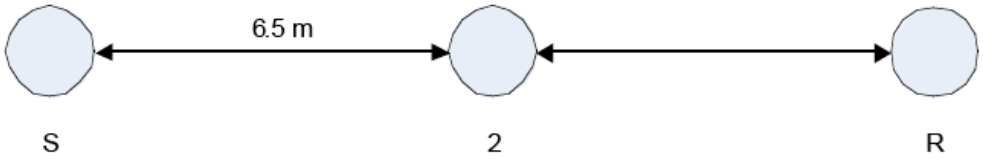


Figure 3.25. 2-hop network.

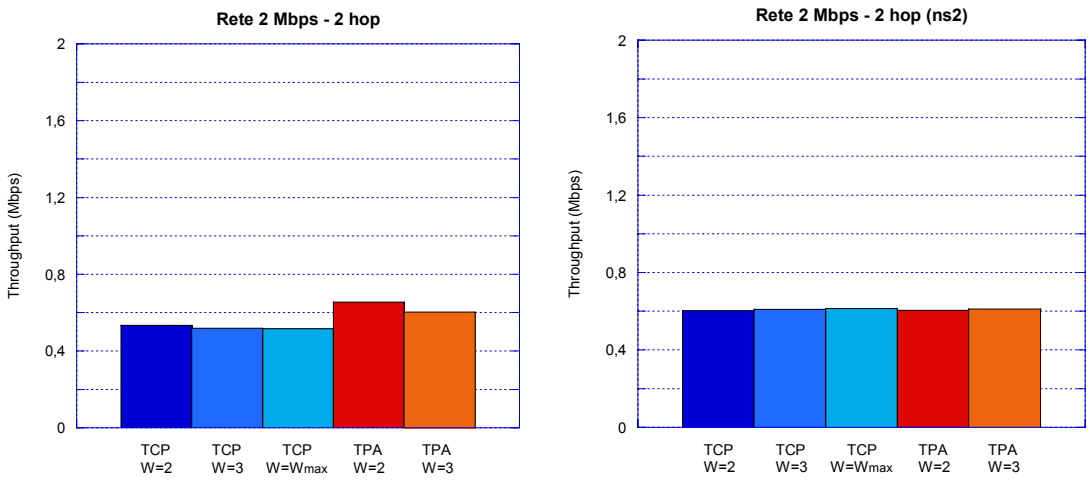


Figure 3.26. Throughput achieved by TCP and TPA in a 2-hop scenario with different congestion window sizes: experimental measurements (left) and simulation results (right).

$TCP_{2\ hop}$			
W_{size}	TH_{avg} (Mbps)	TH_{min} (Mbps)	TH_{max} (Mbps)
2	0,53	0,43	0,57
3	0,52	0,43	0,58
W_{max}	0,52	0,38	0,58

$TPA_{2\ hop}$			
W_{size}	TH_{avg} (Mbps)	TH_{min} (Mbps)	TH_{max} (Mbps)
2	0,65	0,59	0,67
3	0,60	0,58	0,62

Table 3.6. Throughput achieved by TCP and TPA for different congestion window sizes.

As above, the simulation results show that there is no significant different between TPA and TCP. However, from the experiments in real conditions it clearly emerges that the throughput achieved by the receiver node when using TPA is significantly higher. In detail, TPA with congestion window equal to 2 provides a throughput 20% higher than that provided by TCP in the same conditions (16% when the congestion window size is set to 3). TPA performs better even if it is implemented at the at the user level.

The discrepancy between the simulation and experimental results can be explained because in the simulation analysis the wireless channel is assumed to be ideal, i.e., packet losses are only due to contentions. Obviously, in a real environment there are interferences and other problems that increase the packet loss rate. In addition, in a 2-hop scenario the impact of packet losses on TCP is much more severe.

3-hop network

We now consider a (static) string topology with for nodes. The sender and receiver are thus 3 hops apart. The results obtained in this scenario are shown in Figure 3.27-left, and summarized in Table 3.7. As above, Figure 3.27-right shows the simulation results provided by the ns-2 tool in the same operating conditions.

Figure 3.27. 2-hop network.

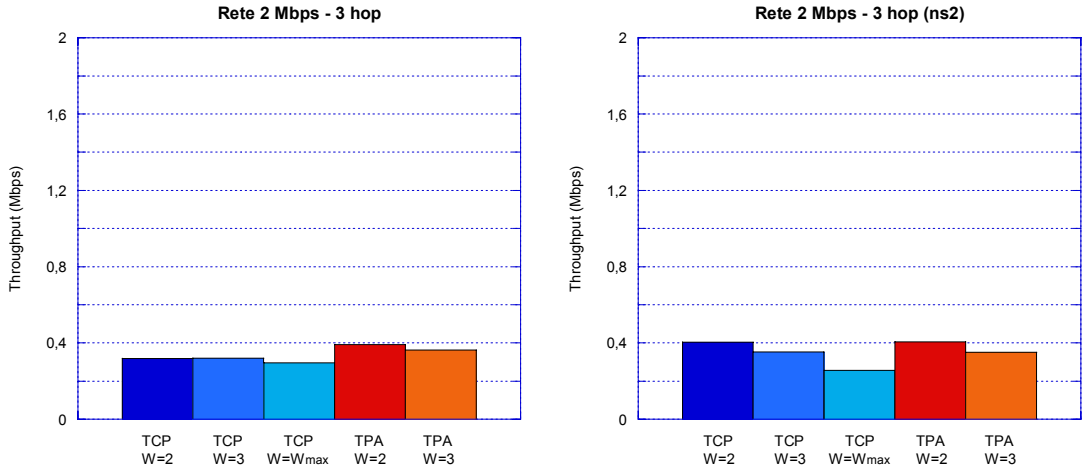


Figure 3.27. Throughput achieved by TCP and TPA in a 3-hop scenario with different congestion window sizes: experimental measurements (left) and simulation results (right).

$TCP_{3\ hop}$			
W_{size}	TH_{avg} (Mbps)	TH_{min} (Mbps)	TH_{max} (Mbps)
2	0,32	0,26	0,42
3	0,32	0,24	0,36
W_{max}	0,30	0,24	0,40

$TPA_{3\ hop}$			
W_{size}	TH_{avg} (Mbps)	TH_{min} (Mbps)	TH_{max} (Mbps)
2	0,39	0,36	0,42
3	0,36	0,33	0,41

Table 3.7. Throughput achieved by TCP and TPA for different congestion window sizes.

Also in this scenario TPA outperforms TCP. With a congestion window size equal to 2 the difference in terms of throughput experienced by the final receiver is about 23% (14% for a congestion window of 3 packets).

3.3.4. Conclusions

In this section we have described an implementation of the TPA protocol, a novel transport protocol specifically tailored to MANETs. The protocol has been implemented at the user level mainly for decreasing the time needed by software development. In fact, this approach shortens the revise/test cycle, and makes the debugging process easier and faster. Our implementation of TPA protocol offers an Application Programming Interface (API) similar to the one provided by the TCP in a UNIX environment, thus allowing legacy application to be used with very minor changes.

We used the user-level TPA implementation to set up an experimental testbed and compare the performance of TCP and TPA in a real environment. The results obtained have shown that in a real environment TPA outperforms TCP even in a static scenario, especially when the number of hops increases. We are currently extending the experimental analysis by considering additional scenarios and performance indices

3.3.5. References

[AACP05] G. Anastasi, E. Ancillotti, M. Conti, A. Passarella, "TPA: A Transport Protocol for Ad hoc Networks: Extended Version", http://www.iet.unipi.it/~anastasi/papers/tpa.pdf/.

[CRVP98] K. Chandran, S. Raghunathan, S. Venkatesan, R. Prakash, "A Feedback Based Scheme for Improving TCP Performance in Ad-Hoc Wireless Networks", *Proceedings of ICDCS '98*, pp. 472-479.

[HV02] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks", *Wireless Networks*, Vol.8, pp. 275-288, 2002.

[L90] D. Libes, "Implementing Software Timers", C User's Journal, November 1990.

[LS01] J. Liu and S. Singh, "ATCP: TCP for Mobile Ad Hoc Networks", *IEEE J-SAC*, Vol. 10, No. 7, July 2001.

[SM01] D. Sun and H. Man, "ENIC - An Improved Reliable Transport Scheme for Mobile Ad Hoc Networks", *Proceedings of the IEEE Globecom Conference*, 2001.

[S94] W.R. Stevens, "TCP/IP Illustrated", Vol. 1, Addison Wesley, 1994.

[TNML93] C.A. Thekkath, T.D. Nguyen, E. Moy, E.D. Lazowska, "Implementing Network Protocols at User Level", IEEE/ACM Transactions on Networking, vol. 1(5), pp. 554-565, October 1993.

4. INTERCONNECTION AD HOC - INTERNET

4.1. Architecture

In this Section propose and evaluate a practical architecture to build multi-hop hybrid ad hoc networks used to extend the coverage of traditional wired LANs, providing mobility support for mobile/portables devices in the local area environment. Traditionally, MANETs have been considered as *stand-alone* networks, i.e., self-organized groups of nodes that operate in isolation in an area where deploying a networking infrastructure is not feasible due to practical or cost constraints (e.g., disaster recovery, battlefield environments). However, it is now recognized that the commercial penetration of the ad hoc networking technologies requires the support of an easy access to the Internet and its services. In addition, the recent advances in mobile and ubiquitous computing, and inexpensive, portable devices are further extending the application fields of ad hoc networking. As a consequence, nowadays, multi-hop ad hoc networks do not appear as isolate self-configured networks, but rather emerge as a flexible and low-cost extension of wired infrastructure networks, coexisting with them. Indeed, a new class of networks is emerging from this view, in which a mix of fixed and mobile nodes interconnected via heterogeneous (wireless and wired) links forms a multi-hop hybrid ad hoc network integrated into classical wired/wireless infrastructure-based networks [BCG05].

More precisely, we envisage a hybrid network environment in which wired and multi-hop wireless technologies transparently coexist and interoperate. In this network, separated group of nodes, without a direct access to the networking infrastructure, form *ad hoc "islands"*. Special nodes, hereafter indicated as *gateways*, having both wired and wireless interfaces, are used to build a wired backbone interconnecting separated ad hoc components. To ensure routing between these ad hoc parts, a proactive ad hoc routing protocol is implemented on both gateways' interfaces. In addition, the gateways use their wired interfaces also to communicate with static hosts belonging to a wired LAN. The network resulting from the integration of the hybrid ad hoc network with the wired LAN is an *extended* LAN, in which static and mobile hosts transparently communicate using traditional wired technologies or ad hoc networking technologies.

In this Section we specifically address several architectural issues that arise to offer IP basic services, such as routing and Internet connectivity, in the extended LAN. First, we propose a dynamic protocol for the self-configuration of the ad hoc nodes, which relies on DHCP servers located in the wired part of the network, and it does not require that the ad hoc node to be configured has a direct access to the DHCP server. In addition, we design innovative solutions, which exploit only *layer-2* mechanisms as the ARP protocol, to logically extend the wired LAN to the ad hoc nodes in a way that is transparent for the wired nodes. More precisely, in our architecture the extended LAN appears to the external world, i.e., the Internet network, as a single IP subnet. In this way, the hosts located in the Internet can communicate with ad hoc nodes inside the extended LAN as they do with traditional wired networks. Previous solutions to connect ad hoc networks to the Internet have proposed to use access gateways that implement Network Address Translator (NAT) [RFC2663] or a Mobile IP Foreign Agent (MIP-FA) [RFC3344]. However, such approaches are based on complex IP-based mechanisms originally

defined for the wired Internet, like IP-in-IP encapsulation and IP tunneling, which may introduce significant overheads and limitations, as discussed in depth in the following sections. On the other hand, the architecture we propose in this paper is a lightweight and efficient solution that avoids these overheads operating below the IP level. By positioning our architecture at the layer 2 (data link layer), we may avoid undesired and complex interactions with the IP protocol and provide global Internet connectivity and node self-configuration in a very straightforward way.
In the past, other architectures have been proposed to provide ad hoc support below IP. For example, in [AMB02] label switching was employed to put routing logic inside the wireless network card. More recently, the LUNAR [TGRW04] ad hoc routing framework and the Mesh Connectivity Layer (MCL) [DPZ04] have been proposed. These solutions locate the ad hoc support between the layer *2* (data link layer) and layer *3* (network layer). This "layer *2.5*" is based on *virtual* interfaces that allow abstracting the ad hoc protocols from both the specific hardware components and network protocols. However, this interconnection layer requires its own naming and addressing functionalities distinct from the layer-2 addresses of the underlying physical devices. This may significantly increase the packet header overheads. On the contrary, our proposed architecture is totally located inside layer 2, reducing implementation complexity and ensuring minimal additional overheads.
We have prototyped the main components of our architecture in a general and realistic test-bed, in which we have carried out various performance measurements. The experimental results show the performance constraints with mobility and Internet access, and indicate that an appropriate tuning of the routing protocol parameter may significantly improve the network performance.

4.1.1. Network Model

Figure 4.1 illustrates the reference network model we assume in our architecture. We consider a full-IP network in which all the traffic is transported in IP packets. In this network, mobile/portable nodes far away from the fixed networking infrastructure establish multi-hop wireless links to communicate (e.g., using IEEE 802.11 technology). As shown in the figure, gateways, i.e., nodes with two interfaces - both wired and wireless - are used to connect the ad hoc components to a wired LAN (e.g., an Ethernet-based LAN). In our architecture, it is allowed the multi-homing, i.e., the presence of multiple gateways within the same ad hoc component. Consequently, specific mechanisms are required to support the handoff between gateways without TCP-connection breaks. In general, between pairs of gateways in radio visibility of each other, two direct links can be established, both wired and wireless. However, in our model we assume that the gateways always use the wired link to communicate. The motivations behind this requirement will be clearly discussed in Section 4.1.4. However, this is a quite reasonable assumption, since wired links have higher bandwidth than wireless links, and the routing protocol should assign them a lower link cost.

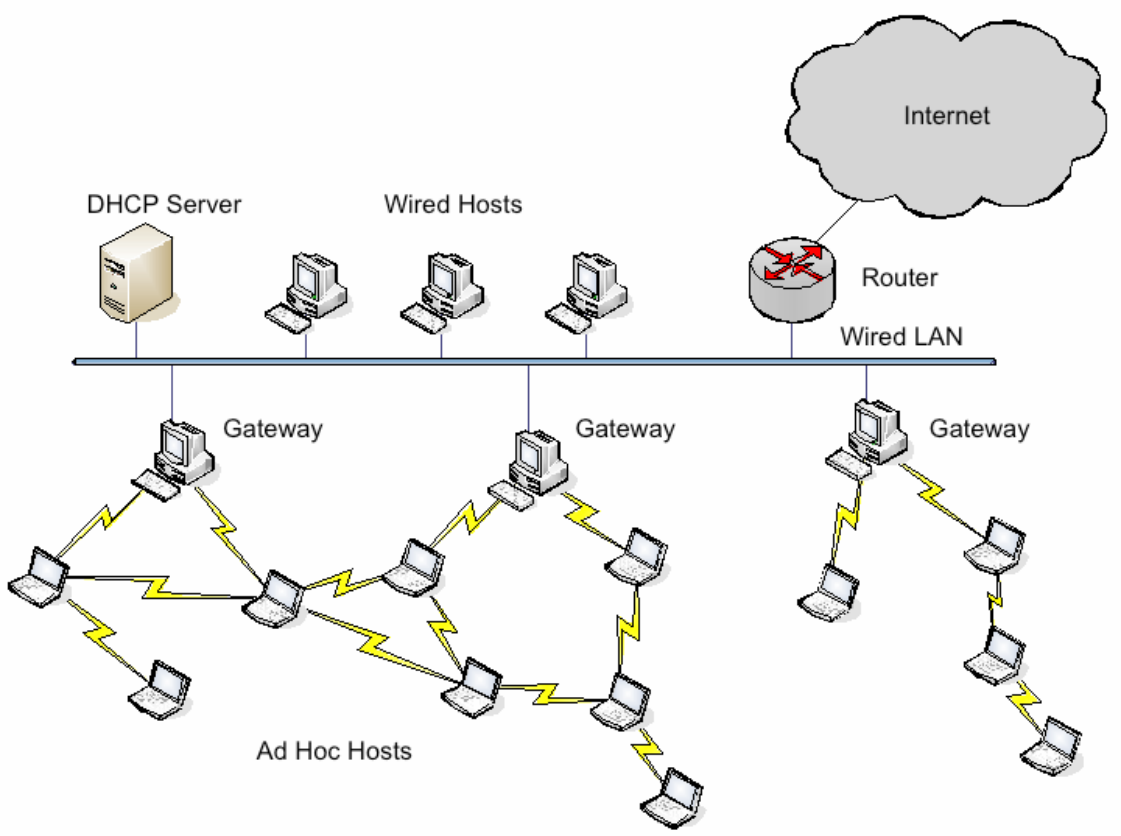


Figure 4.1. 3 Reference network model.

The wired LAN is interconnected to the external Internet through a default router *R*. In addition, one or more DHCP servers are located in the wired LAN to allocate network addresses to hosts. In the following sections, we will explain how these DHCP servers could be used to assign IP configuration parameters also to the ad hoc nodes. For the purpose of simplicity, we assume that all the IP addresses are allocated from the same IP address block IP_S/L. According to standard notation, IP_S indicates the network prefix, and L is the network mask length, expressed in bits (e.g., IP_S/L = X.Y.96.0/22). Assuming that the extended LAN adopts a unique network address implies that the extended LAN appears to the external world, i.e., the Internet network, as a single IP subnet.

Standard IP routing is used to connect the extended LAN to the Internet. However, a specific ad hoc routing protocol is needed to allow multi-hop communications among the ad hoc nodes. In this work we decided to use a proactive routing protocol as the ad hoc routing algorithm (such as the Optimized Link State Routing (OLSR) protocol [RFC3636] or the Topology Dissemination Based on Reverse-Path Forwarding (TBRF) routing protocol [RFC3684]). The motivation behind this design choice is that proactive routing protocols usually support gateways, allowing these nodes to use special routing messages to set up default routes in the ad hoc network. Indeed, default routes are an efficient mechanism to forward traffic that does not have an IP destination locally known to the ad hoc network. In addition, proactive routing protocols, adopting classical link

state approaches, build the complete network-topology knowledge in each ad hoc node. This topology information could significantly simplify the operations needed to acquire Internet connectivity. In this work, the reference ad hoc routing algorithm is OLSR, but our architecture is general and it is equally applicable to other proactive routing protocols.

4.1.2. Related Work

The implemented solutions to provide Internet connectivity in MANETs are mainly based on two different mechanisms.

One approach is to set up a Mobile IP Foreign Agent (MIP-FA) in the gateway and to run Mobile IP [RFC3344] in the MANET. In this way, the ad hoc node may register the foreign agent care-of-address with its Home Agent (HA). Whenever an ad hoc node MN wants to contact an external host X, it uses its home address (i.e., a static IP address belonging to its home network) as source address. As a consequence, the return traffic is routed to the home network through standard IP routing. The HA intercepts the traffic, encapsulating it using the care-of-address. Then, it tunnels the encapsulated packets to the FA. The FA removes the outer IP header and delivers the original packets to the visiting host MN. Different versions of this approach have been proposed and implemented for proactive [BMAAA04] and reactive [JALJM00] ad hoc networks. A drawback of these solutions is that they require significant changes in the Mobile IP implementation since the FA and the mobile node cannot be considered on the same link. Moreover, the mobile node has to be pre-configured with a globally routable IP address as its home address, limiting both the ability of forming totally self-configuring and truly spontaneous networks, and the applicability of these schemes.

An alternative solution to interconnect MANETs to the Internet is to implement a Network Address Translation (NAT) [RFC2663] on the gateway. In this way, the gateway may translate the source IP address of outgoing packets from the ad hoc nodes with an address of the NAT gateway, which is routable on the external network. The return traffic is managed similarly, with the destination IP address (i.e., the NAT-gateway address) replaced with the IP address of the ad hoc node. NAT-based solutions have been designed for both proactive [ETHE04] and reactive [EE04] ad hoc networks. NAT-based mechanisms appear as easier solutions than MIP-FA-based schemes to provide Internet access to MANETs. However, a problem that arises with NAT-based solutions is multi-homing, i.e., the support of multiple gateways in the same MANET. Indeed, to avoid session breakages it is necessary to ensure that all the packets from the same session are routed over a specific gateway. A proposed solution to this issue is to explicitly tunnel all the outgoing traffic from the same communication session destined to the external network to one of the available gateways, instead of using default routes. A limitation of this strategy is the additional overhead introduced by the IP-in-IP encapsulation. Moreover, the ad hoc nodes should be provided with the additional capability of explicitly discovering the available gateways. This would eventually require extensions to the ad hoc routing protocols.

Both the two classes of solutions discusses above implicitly assume that either there is a dynamic host configuration protocol designed to configure the nodes such as to properly working in the MANET, or the ad hoc nodes are configured *a priori*. Indeed, a node in an

IP-based network requires a unique IP-based address, a common netmask and, eventually, a default gateway. In traditional networks, hosts rely on centralized servers like DHCP [D97] for configuration, but this cannot be easily extended to MANETs because of their distributed and dynamic nature. However, various protocols have been proposed recently in literature for the purpose of address self-configuration in MANETs. In general, with protocols using stateless approaches nodes arbitrarily select their own address, and a Duplicate Address Detection (DAD) procedure is executed to verify its uniqueness and resolve conflicts. On the other hand, protocols based of stateful approaches execute distributed algorithms to establish a consensus among all the nodes in the network on the new IP address, before assigning it. The protocols proposed in [V02] and [NP02] are examples of the latter and former approach, respectively, while [WZ04] presents a general overview of the several solutions currently available. Generally, all these protocols assume reliable flooding in order to synchronize nodes' operations and resolve inconsistencies in the MANET, but this is difficult to be guaranteed in ad hoc networks. Another main limitation of these solutions is that they are designed to work in *stand-alone* MANET, while no protocols have been devised to take fully advantage of the access to external networks. In addition, the problems of selecting a unique node address, routing the packets and accessing the Internet are still separately addressed, while a unified strategy may be beneficial, reducing complexities and overheads.

4.1.3. Protocol Descriptions

This Section gives a short description of the protocols, which our architecture is based on.

OLSR

The OLSR protocol [RFC3636], being a link-state proactive routing protocol, periodically floods the network with route information, so that each node can locally build a routing table containing the complete information of routes to all the nodes in the ad hoc network running on their interfaces the OLSR protocol. The OLSR routing algorithm employs an efficient dissemination of the network topology information by selecting special nodes, the multipoint relays (MPRs), to forward broadcast messages during the flooding process. The link state reports, which are generated periodically by MPRs, are called Topology Control (*TC*) messages. MPRs grant that TC messages will reach all 2-hop neighbors of a node. In order to allow the injection of external routing information into the ad hoc network, the OLSR protocol defines the Host and Network Association (*HNA*) message. The *HNA* message binds a set of network prefixes to the IP address of the node attached to the external networks, i.e., the gateway node. In this way, each ad hoc node is informed about the network address and netmask of the network that is reachable through each gateway. In other words, the OLSR protocol exploits the mechanism of *default routes* to advertise Internet connectivity. For instance, a gateway that advertises the *0.0.0.0/0* default route, will receive all the packets destined to IP addresses without a known route on the local ad hoc network.

ARP Protocol

IP-based applications address a destination host using its IP address. On the other hand, on a physical network individual hosts are known only by their physical address, i.e., MAC address. The ARP protocol [RFC826] is then used to translate, inside a physical

network, an IP address into the related MAC address. More precisely, the ARP protocol broadcasts the *ARP Request* message to all hosts attached to the same physical network. This packet contains the IP address the sender is interested in communicating with. The target host, recognizing that the IP address in the packet matches its own, returns its MAC address to the requester using a unicast *ARP Reply* message. To avoid continuous requests, the hosts keep a cache of ARP responses.
In addition to these basic functionalities, the ARP protocol has been enhanced with more advanced features. For instance, in [RFC1027] it has been proposed the *Proxy-ARP* mechanism, which allows constructing local subnets. Basically, the Proxy ARP technique allows one host to answer the ARP requests intended for another host. This mechanism is particularly useful when a router connects two different physical networks, say *NetA* and *NetB*, belonging to the same IP subnet. By enabling the Proxy ARP on the router's interface attached to *NetB*, any host A in *NetA* sending an ARP request for a host B in *NetB*, will receive as response the router's MAC address. In this way, when host A sends IP packets for host B, they arrive to the router, which will forward such packets to host B.

4.1.4. Proposed Architecture

Our design goal in the definition of the rules and operations of the proposed architecture is to provide transparent communications between static nodes (using traditional wired technologies) and mobile nodes (using ad hoc networking technologies), employing mechanisms that run below the IP layer. As discussed previously, in this Section we address two relevant issues: node self-configuration and global Internet connectivity.

Ad Hoc Node Self-configuration

The main obstacle to use a DHCP server for self-configuration of ad hoc nodes is that the DHCP server may be not reachable to the new node, due to mobility or channel impairments. In addition, the ad hoc nodes may need multi-hop communications to reach the DHCP server, but a unique address is necessary to execute ad hoc routing algorithms capable of establishing such communications. To solve these problems, we assume that the DHCP servers are located only in the wired part of the network, while in the ad hoc part of the network we implement dynamic *DHCP Relay* agents. These are special relay nodes passing DHCP messages between DHCP clients and DCHP servers that are on different networks.

As illustrated in Figure 4.2, when a new mobile host *i* not yet configured attempts joining the ad hoc part of the extended LAN, it broadcasts a special message, the *Neighbor Req* message. At least one neighbor that is already configured, i.e., it has joined the ad hoc network, will respond with a *Neighbor Reply* message. Node *i* selects one of the responders *j* as intermediary in the process of address resolution. Then, node *i* sends a *Conf Req* message to the chosen node j that replies with a *Conf Ack* message to inform node *i* that it will execute on its behalf the process of acquiring the needed IP configuration parameters (i.e., node *j* acts as a *proxy* for the node *i*). In fact, on receiving the *Conf Req* message from node *i*, node *j* activates its internal DCHP Relay agent, which issues an unicast *DHCP Request* to one of the available DHCP servers. The DHCP server receiving the requests will answer to the DHCP Relay with a *DHCP Ack*, containing the

IP configuration parameters. The configuration process is concluded when the DHCP Relay forwards the DHCP Ack message to the initial node *i* that is now configured, and can join the network. After joining the network, node *i* may also turn itself into a DHCP Relay for the DHCP server from which it received the IP configuration parameters, letting other nodes to subsequently joining the ad hoc component. Finally, it is worth noting that it is not needed any initialization procedure for the ad hoc network, because the gateways are directly connected to the wired LAN and can broadcast a *DHCP Discover* message to locate available servers. In this way, the first mobile node to enter the ad hoc network may find at least one gateway capable of initiating the illustrated configuration process.

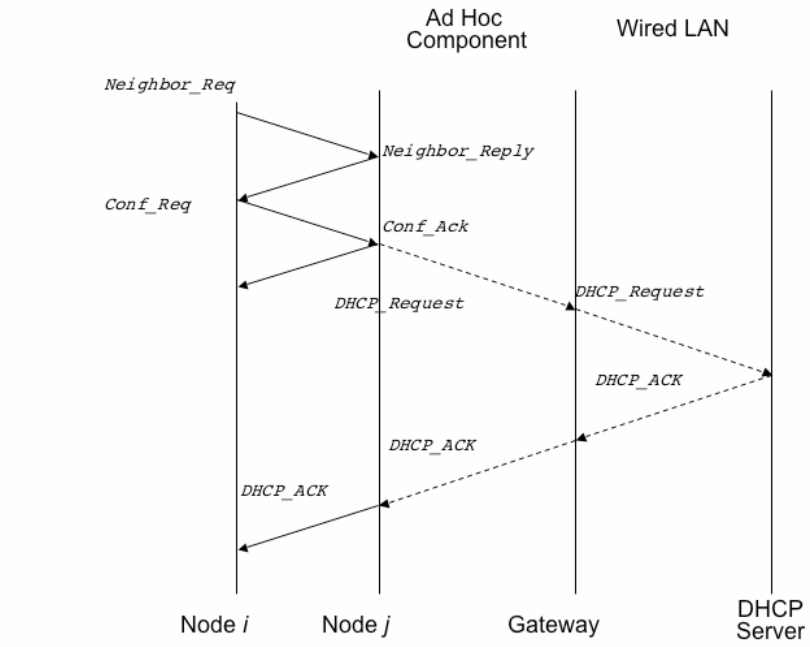


Figure 4.2. Message exchanges during the ad hoc node self-configuration.

Our proposed node self-configuration mechanism is somehow similar to the one described in [NP02]. In that paper, a preliminary message handshake was used to discover a reachable MANET node that could act as initiator of the configuration process. On the contrary, in our solution the initiator node exploits the resources of the external wired network to which the ad hoc component is connected, to perform the IP address resolution.

Global Internet Connectivity

Our design goal is to support Intranet connectivity (i.e., communications with nodes inside the same IP subnet) and Internet connectivity (i.e., communications with nodes of external IP networks) for the mobile nodes, without any configuration change in the pre-existing wired LAN. The assumption that we take as starting point in our proposal is that

the mobile nodes are configured with an IP address belonging to the same IP subnet of the wired LAN. This is achieved using the mechanism described above.
In the following we will separately explain how the proposed architecture ensures connectivity for outgoing and incoming traffic.

Connectivity for Outgoing Traffic

As outlined in Section 4.1.3, the OLSR protocol builds the routing tables with entries that specify the IP address of the next-hop neighbor to contact to send a packet destined to either another host or subnetwork. More precisely, to send a packet to a destination IP address, the mobile host searches for the longest IP prefix in the routing table matching the destination IP address. The matching routing table entry provides the next hop to send the packet. Since the gateways advertise *0.0.0.0/0* as default route, all packets destined for IP addresses without a specific route on the ad hoc network, will be routed on a shortest-hop basis to the nearest gateway and forwarded to the Internet. However, using *0.0.0.0/0* as default route for outgoing packets, introduces an inconsistency when a mobile host sends IP packets to a wired host inside the LAN.

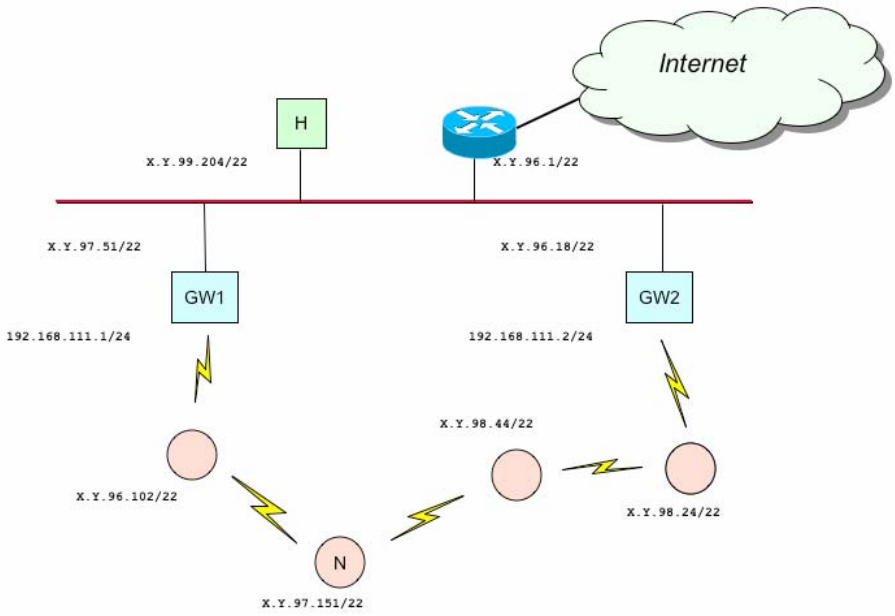


Figure 4.3. Illustrative network configuration.

To explain this problem let us consider the simple network configuration depicted in Figure 4.3. For illustrative purposes we assume that the IP subnet of the extended LAN is $IP_S/L = X.Y.96.0/22$[2]. If the mobile node N ($IP_N = X.Y.97.151/22$) wants to deliver packets to the wired node H ($IP_H = X.Y.99.204/22$), the routing table lookup on node N

[2] On the gateways' wireless interfaces we set up private IP addresses to save address space. In this way, the gateways are globally reachable using the IP address on their wired interfaces.

will indicate that the node H is connected to the same physical network of node N's wireless interface. This will result in a failed ARP request for the IP_H address. To resolve this inconsistency, we will exploit the properties of the IP longest-matching rules. More precisely, we split the original IP subnet into two consecutive smaller subnets $IP_{SL}/(L+1)$ and $IP_{SU}/(L+1)$, such as to have that the union of these two sets is equal to IP_S/L. In the considered case $IP_{SL}/(L+1) = X.Y.96.0/23$ and $IP_{SU}/(L+1) = X.Y.98.0/23$[3]. Then, we configure all the gateways in such a way that they announce, through the HNA messages, also the connectivity to these two subnetworks. In this way, each mobile host will have, for any host on the local wired LAN, a routing table entry with a more specific network/mask than the one related to its wireless interface. To better clarify this point, let us consider the node N's routing table as shown in Table 4.1. The entries 8, 9, and 11 are the ones induced by the HNA messages arrived from GW1. The entry 10 is automatically set up by the operating system when the wireless interface is configured with the IP parameters. However, when searching the routing table for matching the IP_H address, node N will found the routing entry 9 more specific than entry 10. Consequently, the longest-match criterion applied to the routing table lookup, will result in node N correctly forwarding traffic to gateway GW1 (i.e., the nearest one) to reach node H.

Table 4.1: Node N's routing table.

Entry	*Destination*	*Next hop*	*Metric*	*interface*
1	*X.Y*.97.51/32	*X.Y*.96.102	2	*eth*0
2	*X.Y*.96.102/32	0.0.0.0	1	*eth*0
3	*X.Y*.98/44/32	0.0.0.0	1	*eth*0
4	*X.Y*.98.24/32	*X.Y*.98.44	2	*eth*0
5	*X.Y*.96.18/32	*X.Y*.96.102	3	*eth*0
6	192.168.111.1/24	*X.Y*.96.102	2	*eth*0
7	192.168.111.2/24	*X.Y*.96.102	3	*eth*0
8	*X.Y*.96.0/23	*X.Y*.96.102	2	*eth*0
9	*X.Y*.98.0/23	*X.Y*.96.102	2	*eth*0
10	*X.Y*.96.0/22	0.0.0.0	0	*eth*0
11	0.0.0.0/0	*X.Y*.96.102	2	*eth*0
12	127.0.0.0/8	127.0.0.1	0	*eth*0

The mechanism described above resolves any eventual IP inconsistency that could occur in the mobile hosts, but it may cause problems for the gateways. In fact, being part of the ad hoc component, the gateways will receive HNA messages sent by other gateways, setting up the additional routing entries advertised in these messages. However, when a gateway wants to send packets to a wired host on the local wired LAN (e.g., node H), the routing table lookup will choose one of these two entries, instead of the entry related to its wired interface (i.e., *X.Y*.96.0/22). The effect is that the IP packet will loop among the GW nodes until the TTL expires, without reaching the correct destination H. To resolve this problem, we statically add in each gateway two further routing entries in addition to the one related to the default router *X.Y*.96.1. These two additional entries have the same

[3] It is straightforward to observe that this operation is always feasible, at least for L<32.

network/mask as the two announced in the HNA messages, but with lower metric. Again, to better clarify the routing operations, let us consider the illustrative example shown in Figure 4.3. In Table 4.2 we have reported the GW1's routing table. In this example, *eth*0 is the GW1's wireless interface and *eth*1 is the GW1's wired interface. When gateway GW1 wants to send packets to node H, it will found two routing table entries matching the same number of bits of node H's IP address. These are entry 9 (derived from HNA messages received from GW2) and entry 11 (statically configured on the gateway). However, entry 11 has a lower metric than entry 9 (i.e., metric 0 against metric 1). As a consequence, the packets destined to host H can be correctly forwarded to the host H on the local wired LAN through the GW1's wired interface.

Table 4.2: Gateway GW1's routing table.

Entry	*Destination*	*Next hop*	*Metric*	*interface*
1	*X.Y*.96.102/32	0.0.0.0/0	1	*eth*0
2	*X.Y*.97.151/32	*X.Y*.96.102	2	*eth*0
3	*X.Y*.98.44/32	*X.Y*.96.18	3	*eth*1
4	*X.Y*.98.24/32	*X.Y*.96.18	2	*eth*1
5	*X.Y*.96.18/32	0.0.0.0	1	*eth*1
6	191.168.111.2/24	*X.Y*.96.18	1	*eth*1
7	192.168.111.0/24	0.0.0.0	0	*eth*0
8	*X.Y*.96.0/23	*X.Y*.96.18	1	*eth*1
9	*X.Y*.98.0/23	*X.Y*.96.18	1	*eth*1
10	*X.Y*.96.0/23	0.0.0.0	0	*eth*1
11	*X.Y*.98.0/23	0.0.0.0	0	*eth*1
12	*X.Y*.96.0/22	0.0.0.0	0	*eth*1
13	0.0.0.0/0	*X.Y*.96.1	0	*eth*1
14	0.0.0.0/0	*X.Y*.96.18	1	*eth*0
15	127.0.0.0/8	127.0.0.1	0	*l*0

Connectivity for Incoming Traffic

A mechanism is required to ensure that the return traffic coming from hosts on the local wired LAN or from the Internet (through the default LAN router, as shown in Figure 4.1), gets correctly routed to the mobile hosts. Our basic idea is to introduce specific Proxy ARP functionalities into each gateway, in such a way that the gateways can hide the ad-hoc node identity on the wired physical network, which the gateways are connected to. Thus, all mobile nodes located in the ad hoc component will appear to wired hosts as being one IP-hop away. Internally to the ad hoc component, the ad hoc routing protocol will transparently provide the multi-hop connectivity and the mobility support. This is somehow similar to what is implemented in the LUNAR framework [TGRW04], in which the entire ad hoc network appears as a single virtual Ethernet interface.

In our proposed solution, a Proxy ARP server runs on the wired interfaces of each gateway. The Proxy ARP server periodically checks the gateway's routing table and ARP table, such as to publish the MAC address of the gateway's wired interface for each IP address having an entry in the routing table with a netmask 255.255.255.255, and the next

hop on the gateway's wireless interface. The former condition is verified only by mobile hosts that have joined the ad hoc network. The latter condition implies that the gateway can deliver traffic to that node only over multi-hop paths not traversing other gateways[4]. Thus, it is highly probable that the considered gateway is the default gateway selected by that ad hoc node. To illustrate how the proposed mechanism works, let us consider the network in Figure 4.3. When a node on the wired local LAN (e.g., node H) wants to send packets to an ad hoc node (e.g., node N), it assumes that the ad hoc node is on the same physical network. Hence, node H checks its ARP table for IP-MAC mapping and, if it is not present, it sends an ARP request. The gateway GW1 fulfills the previously defined conditions (i.e., node N's IP address has an entry in the GW1's routing table with a netmask 255.255.255.255, which is related to its wireless interface), while GW2 does not. Consequently, only GW1 is allowed by the Proxy ARP server to answer with an ARP reply. This ARP reply will insert the mapping [node N's IP address - MAC address of GW1's wired interface] into the node H's ARP table. Thus, the packets sent from node H to node N will be delivered to GW1, which will forward them to node N. On the other hand, node N will reply to node H using GW1, as indicated by its routing table (see Table 4.2).

There are some network configurations where asymmetric routing may occur, i.e., the forward path is different from the return path. For instance, let us consider the case in which node N is in radio visibility of two gateways GW1 and GW2. In this situation, the OLSR routing algorithm will randomly select one of these gateways as default gateway for node N. However, both gateways are allowed to send ARP replies for ARP requests issued by node H for the node N's IP address. In this case, the wired node H will update its ARP table using the information delivered in the last received ARP reply. Let us assume that GW1 is the default gateway for node N, but GW2 has sent the last ARP reply to node H. In this case, node H sends the traffic destined to node N to GW2, which routes it to node N. On the other hand, node N sends packets destined to node H to GW1, which forwards them to node H. It is important to note that asymmetric paths are not by themselves a problem. Indeed, both node N and H correctly receive and send their packets. In addition the asymmetric routing occurs only in symmetric topologies. Thus, it is reasonable to assume, in this local environment, that both paths are characterized by similar delays.

Mobility Support

In general, solutions to support Internet connectivity for ad hoc networks, which are based on gateways, experience TCP-session breaks when the default route changes, depending on dynamics and mobility in the network. To avoid that TCP sessions break, in [EE04] it was proposed to replace default routes with explicit tunneling between the mobile nodes and the gateways. However, this complicates significantly the implementation and introduces relevant overheads. On the contrary, in our architecture the mobility is supported in a transparent way for the higher protocol layers. Indeed, the

[4] It is worth reminding that gateways are always interconnected using their wired interfaces. Hence, a route to reach a mobile node can traverse two gateways only if one of the links along the path is a wired link. In this case the farthest gateway will have the next-hop routing entry for that mobile node on its wired interface.

only effect of changing the default gateway for node N, is that the node N's outgoing traffic is routed towards the new gateway (e.g., GW2), while the initial gateway (e.g., GW1) continues to receive the incoming traffic and to forward it to node N. This results into asymmetric routing. However, this asymmetry can be easily removed by using an advanced feature of the ARP protocol. More precisely, when GW2 becomes aware that the next hop for the node N switches from its wired interface to its wireless interface, it generates a *Gratuitous ARP* on the wired interface for node N's IP address. This will update the ARP table in all of the wired hosts that have an old entry for the node N's IP address, which was mapped with the MAC address of GW1's wired interface. This action restores a symmetric path for the active packet flows destined to and/or originated from node N.

4.2. Evaluation

We have prototyped the core functionalities of our architecture. In particular, we have developed the software components described in Section 4.1.4, concerning the support of Internet and Intranet connectivity for the ad hoc nodes. Currently, we are completing the implementation of the modifications to the DHCP Relay agents for testing the auto-configuration functionality. For these reasons, in the following we will show experimental results measuring the network performance with mobility and Internet access, while we left for further work the testing of the performance (such as address allocation latency and communication overheads) of the proposed node self-configuration scheme.

In our test-beds we have used *IBM R-50* laptops with *Intel Pro-Wireless 2200* as integrated wireless card. We have also used the *OLSR UniK* implementation for Linux in version 0.4.8 [T04]. The installed Linux kernel distribution was 2.6.9. The ad hoc nodes are connected via IEEE 802.11*b* wireless links, transmitting at the maximum rate of 11 Mbps. To generate the asymptotic UDP and TCP traffic during the experiments we used the *iperf* tool (online available at http://dast.nlanr.net/Projects/Iperf/). More precisely, the iperf server (termination of the traffic sessions) runs in a static host in the wired LAN, while iperf clients (originators of traffic sessions) have been set up on the mobile nodes. If not otherwise specified, the packet size is constant in all the experiments and the transport layer payload is equal to 1448 bytes. Differently from other studies [ETHE04], in which the network topology was only emulated by using the *IP-tables* feature of Linux, our experiments were conducted in realistic scenarios, with hosts located at the ground floor of the CNR building.

4.2.1. Performance Constraints of Internet Access

To measure the performance constraints in case of Internet access, we executed several experiments in the test-bed shown in Figure 4.4. The distances between the ad hoc nodes were set up in such a way to form a 4-hop chain topology with high-quality wireless links.

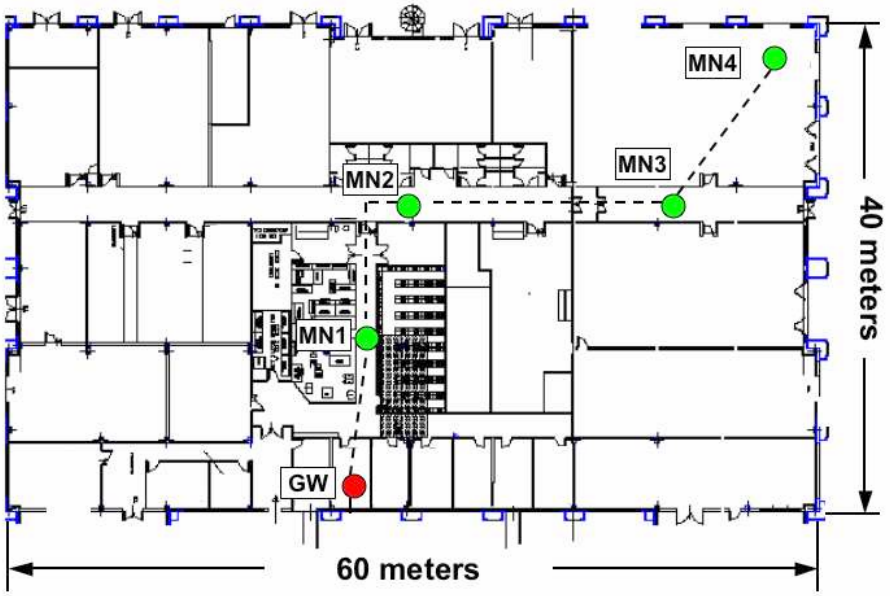


Figure 4.4. Trial scenario for testing Internet access using a chain network.

The first set of experiments was conducted to evaluate the impact on the UDP and TCP throughput of the number of wireless hops traversed in the ad hoc network to reach the gateway. During these tests all the OLSR configuration parameters have been set up according to the default values indicated in the RFC specification [RFC3636]. Figure 4.5 and Figure 4.6 show the UDP and TCP throughput, respectively, obtained during a single experiment, as a function of the time and for different chain lengths. Several observations can be derived from the shown experimental results. First, we can note that the maximum UDP throughput is always greater than the maximum TCP throughput, for every network configuration. This is obviously due to the additional overheads introduced by the TCP return traffic, which consists of TCP ACK packets. In addition, as expected, the longer the route, the lower is the peak throughput achieved by the session flow (both TCP and UDP). The figures show also that, although the nodes are static, the throughput is not stable, but both UDP and TCP flows could be in a stalled condition for several seconds. An initial explanation of this route instability is that losses of routing control frames can induce the loss of valid routes. Indeed, the routing control frames are broadcast frames, which are neither acknowledged nor re-transmitted; hence they are more vulnerable to collisions and channel errors than unicast frames. However, a careful analysis of the routing log files has pointed out another relevant condition that contributes to the route instability in our static network. Indeed, we discovered that the OLSR protocol implements an over pessimistic estimation of the link quality that may cause to consider as lost a link that is overloaded.

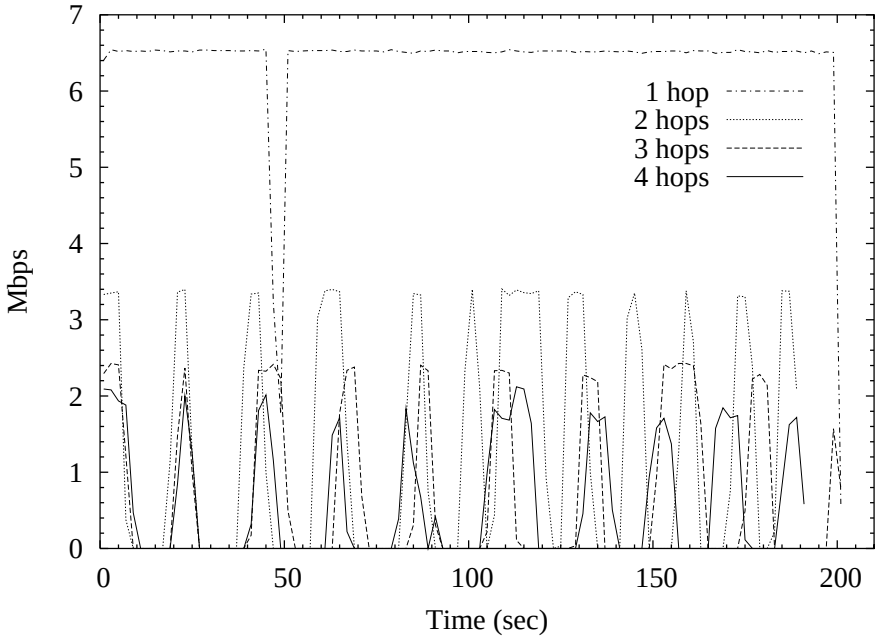


Figure 4.5. Throughput of a single UDP flow for different chain lengths.

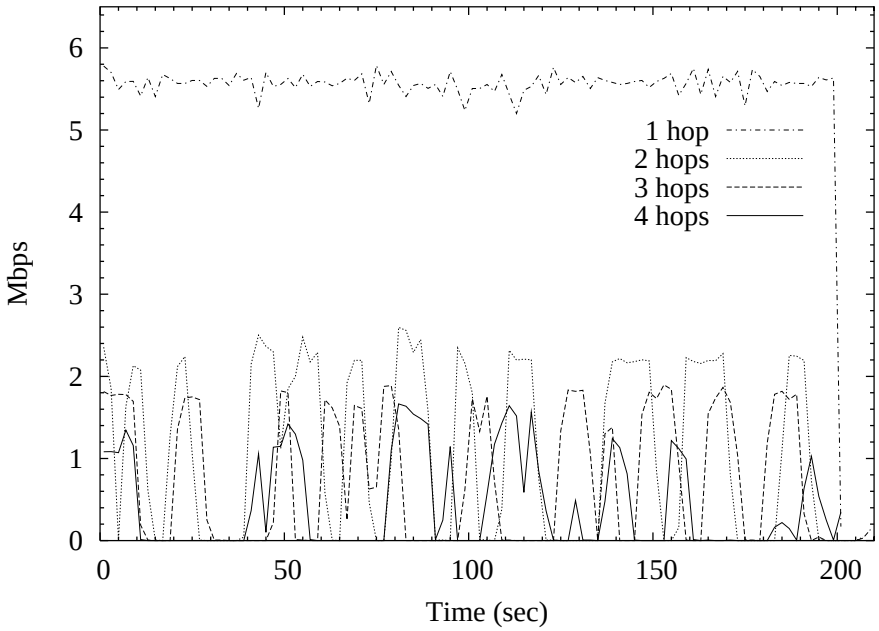


Figure 4.6. Throughput of a single TCP flow for different chain lengths.

More precisely, each node keeps updating a *link quality* value for each neighbor interface. Every time an OLSR packet is lost *link quality = (1-α) link quality*[5]. While every time an OLSR packet is correctly received *link quality = (1-α) link quality+α*, where the α value is the smoothing factor of the estimator. The OLSR specification suggests as default configuration $\alpha = 0.5$. This implies that the *link quality* value is halved after each OLSR packet loss. The *link quality* parameter is used to estimate the link reliability, according to a procedure denoted as *link hysteresis* [RFC3636]. More precisely, the value of the *link quality* is compared with two thresholds, called *HYST_THRESHOLD_LOW* and *HYST_THRESHOLD_UP*. When the *link quality* value is lower than *HYST_THRESHOLD_LOW*, the link is considered as *pending*, i.e., not established. A pending link is not completely dropped because the link information is still updated for each *HELLO* message received. However, a pending link is not a valid link when computing routing tables. In addition, a pending link can be considered again as established only when *link quality* value becomes bigger than *HYST_THRESHOLD_UP*. The OLSR specification suggests as default configuration *HYST_THRESHOLD_LOW* = 0.3 and *HYST_THRESHOLD_UP* = 0.8. According to these values and to the scaling factor α, even a perfect link (i.e., a link with *link quality= 1*) will be purged from the routing tables when two consecutive OLSR packets are lost. We argue that the standard setting of the hysteresis parameters introduces a critical instability in the routing tables, because it is not infrequent to loose broadcast packets (as the OLSR packets are) when the channel is overloaded.

Table 4.3: UDP Throughput in a chain network, with and without link hysteresis.

	UDP		
	HYST	NO HYST	
1 *hop*	6.124 Mbps (304 Kbps)	6.363Mbps (393 Kbps)	+4%
2 *hop*	1.252 Mbps (55 Kbps)	2.501 Mbps (57 Kbps)	+100%
3 *hop*	700.4 Kbps (60 Kbps)	1.206 Mbps (87 Kbps)	+86%
4 *hop*	520.6 Kbps (54 Kbps)	1.141 Mbps (56 Kbps)	+119%

To verify our claim we have carried out a second set of experiments in the same network configuration depicted in Figure 4.4, disabling the OLSR hysteresis process. To provide statistically correct results, we have replicated each experiment five times. Tables 4.3 and Table 4.4 show the average and standard deviation (in parenthesis) values of the measured throughputs for the UDP and TCP case, respectively. From the results we observe that the throughput performances are significantly improved, with the improvement for a 4-hop chain reaching 119% in the UDP case and 82% in the TCP case. The study of routing table logs clearly indicates that these throughput increases are due to an improvement in the route stability with less frequent declarations of link drops due to erroneous estimations of links' reliability. It is worth pointing out that this issue

[5] To identify the loss of an OLSR packet two mechanisms are used: *1)* tracking the sequence numbers of the received OLSR packets, or *2)* monitoring OLSR packet receptions during an *HELLO* emission interval [RFC3636],

has not been identified in previous experimental studies because either the multi-hop communications where only emulated [EE04], or the channel was loaded with low-intensity *ping* traffic [B05].

Table 4.4: TCP Throughput in a chain network, with and without link hysteresis.

	TCP		
	HYST	NO HYST	
1 *hop*	5.184 Mbps (335 Kbps)	5.172Mbps (393 Kbps)	≈=
2 *hop*	956.1 Kbps (123 Kbps)	1.517 Mbps (57 Kbps)	+58%
3 *hop*	638.1 Kbps (149 Kbps)	891.7 Kbps (77 Kbps)	+39%
4 *hop*	345.9 Kbps (47 Kbps)	631.2 Kbps (74 Kbps)	+82%

In addition to the hysteresis process, the OLSR protocol employs several other mechanisms, as the link sensing, neighbor detection and topology discovery, which significantly affect the route stability. Indeed, recent works [B05, GGP05] have investigated how the setting of the classical OLSR routing parameters may affect the network performances. However, these works have specifically focused on the time required for route recalculation after a link drop due to node mobility. On the contrary, to conclude this section we will analyze the impact of different OLSR parameter settings on the performance limits of Internet access in static network configurations. More precisely, each OLSR packet, and the information it delivers, has a fixed validity time. For instance, the information provided in a *HELLO* message is considered valid for a *NEIGHB_HOLD_TIME*. This implies that a node detects a link loss with a neighbor from the lack of *HELLO* messages during a *NEIGHB_HOLD_TIME*. A similar check is performed for the *TC* messages, whose validity time is *TOP_HOLD_TIME*, and for the *HNA* messages, whose validity time is *HNA_HOLD_TIME*. A possible strategy to avoid that links and routes are dropped from the routing tables because the related information has not been refreshed within the corresponding timeout, is to increase the frequency used to generate OLSR packets. This may increase the probability that at least one new OLSR packet is received before its validity time expires. The drawback of this approach is that the more frequent the OLSR protocol generates control messages, the higher is the routing overheads. To quantify the trade-off between routing overhead increases and route stability improvements, and how this impacts network performance, we have carried out a set of experiments in a 3-hop chain using the OLSR parameter settings shown in Table 4.5. As listed in the table, we compare the default parameter setting with disabled hysteresis (*set1*) with the cases in which the frequency of OLSR packet generations is two times (*set2*) and four times (*set3*) higher, while the validity times are kept constant.

Table 4.5: OLSR parameter configurations.

OLSR parameters	*set1*	*set2*	*set3*	*set4*
HELLO_INTERVAL(s)	2	1	0.5	2
NEIGH_HOLD_TIME(s)	6	6	6	6
TC_INTERVAL(s)	5	2.5	1.25	5
TOP_HOLD_TIME(s)	15	15	15	15
HNA_INTERVAL(s)	5	2.5	1.25	5
HNA_HOLD_TIME(s)	15	15	15	15
Hysteresis	no	no	no	yes

Table 4.6: UDP and TCP throughputs in a 3-hop chain network for different OLSR parameter settings.

Parameter Setting	UDP	TCP
set1	700.4 Kbps (60 Kbps)	638.1Kbps (149 Kbps)
set2	1.306 Mbps (87 Kbps)	838.5 Mbps (79 Kbps)
set3	1.605 Mbps (76 Kbps)	1.020 Kbps (105Kbps)
set4	1.84 Mbps (106 Kbps)	1.306 Kbps (56 Kbps)

The experimental results obtained by replicating five times the throughout measurements for UDP and TCP traffic are listed in Table 4.6, in which the average throughput and its standard deviation (in parenthesis) are reported. The shown results indicate that increasing the frequency the OLSR packets are generated by a factor of four, and maintaining the default validity times, it is possible to improve the average throughput of 40% in the UDP case, and of 55% in the TCP case. We have analyzed the routing table logs generated during the trials, and again we have observed that the throughput increases are due to an improvement in route stability. On the other hand, the increase of routing overheads has a negligible impact on the throughput performance.
In summary, our experimental study indicates that the network performance of Internet access in static configurations can be significantly enhanced (in some cases we have more than doubled the measured throughputs) by properly setting the OLSR parameters such as to improve route stability.

4.2.2. Performance Constraints with Mobility

To test the mobility support in a multi-homed network configuration we considered the network layout illustrated in Figure 4.7. In our experiments, node MN2 alternates between position P1 and position P2. More precisely, it starts in position P1, where it is in radio visibility of node MN1. After 50 seconds it moves in position P2, where it is in radio visibility of node MN3. The time needed for moving from P1 to P2 is 20 seconds. After other 50 seconds, host MN2 goes back to position P1. This mobility patterns is periodically repeated throughout the test. The *HNA* messages from GW1 and GW2 form default routes to the external network on a short-hop basis. Hence, while connected to MN1, the node MN2 uses GW1 as default gateway. On the contrary, when connected to node MN3, the routes are recalculated and MN2 uses GW2 as default gateway. The new default gateway GW2 will also begin to act as Proxy ARP for the mobile node. The

return traffic will be consistently routed through the new gateway as soon as either a new ARP request for the MN2's IP address is issued by the external host, or the gateway GW2 sends a Gratuitous ARP; otherwise it will continue to arrive at the GW1 (see Section 4.1.4 for the details).

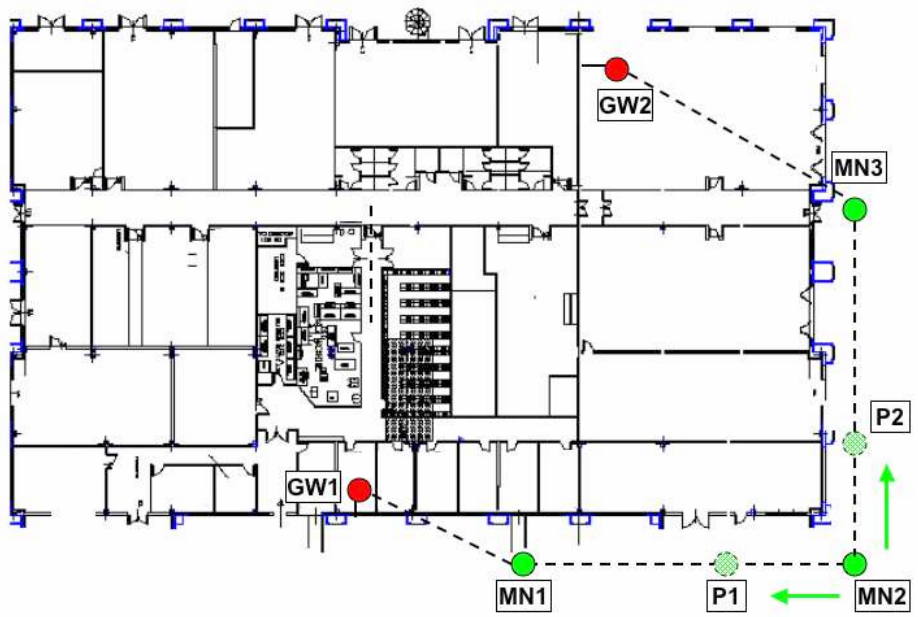


Figure 4.7. Trial scenario for testing mobility support.

Figure 4.8 shows the TCP throughput achieved by MN2 during a mobility test. We compare these results against the throughput measured when node MN2 is fixed in position P1. During both experiments, the hysteresis process was disabled and the other OLSR parameters were set up according to the default values indicated in the RFC specification [RFC3636]. The shown results confirm that the TCP session does not break when node moves. The major effect of node mobility is to introduce holes in the TCP traffic due to the time needed to recalculate the new routes to reach the default gateway. In the considered case of "soft" handoff, i.e., the mobile nodes is in radio visibility of both node MN1 and node MN3 when changing position, we measured up to 20 seconds for re-computing a consistent routing table in node MN2. It is worth noting that in similar experiments conducted in [ETHE04], the throughput of mobile node was approximately 30% lower when mobile node changed position. This was expected because the TCP-session continuity was ensured at the cost of using IP tunneling that introduces significant additional overheads. On the contrary our solution is very efficient and lightweight, because it operates directly at the data link layer.

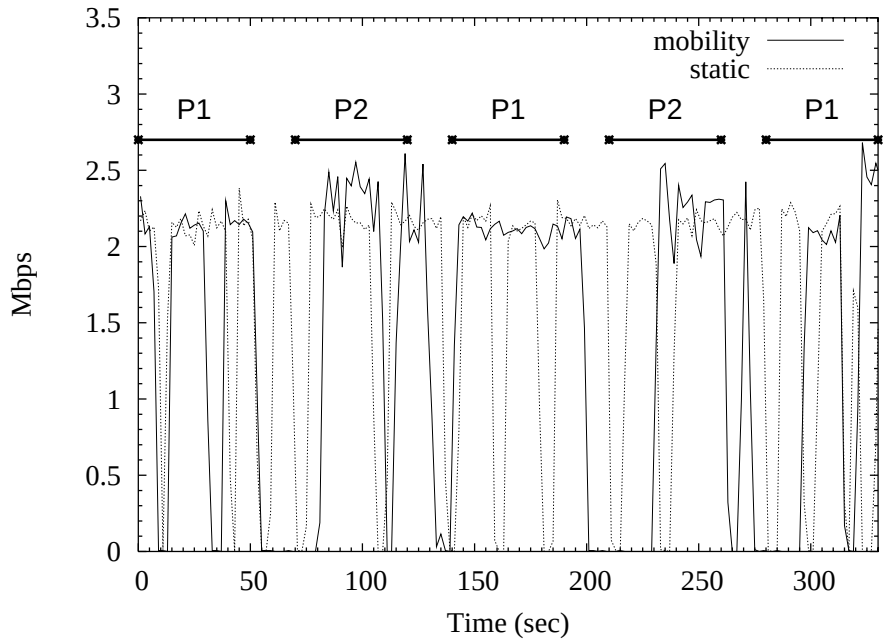


Figure 4.8. Throughput of a single TCP flow with node mobility.

4.3. References

[AMB02]	A. Acharya, A. Misra, and S. Bansal, "A Label-switching Packet Forwarding Architecture for Multi-hop Wireless LANs," in Proc. of ACM WoWMoM 2002, Atlanta, Georgia, USA, September, 28 2002, pp. 33–40.
[B05]	E. Borgia, "Experimental evaluation of ad hoc routing protocols," in Proc. of IEEE PerCom 2005 Workshops, Kauai Island, Hawaii, March, 8–12 2005.
[BCG05]	R. Bruno, M. Conti, and E. Gregori, "Mesh Networks: Commodity Multihop Ad Hoc Networks," *IEEE Communications Magazine*, vol. 43, no. 3, pp. 123–131, March 2005.
[BMAAA04]	M. Benzaid, P. Minet, K. Al Agha, C. Adjih, and G. Allard, "Integration of Mobile-IP and OLSR for a Universal Mobility," *Wireless Networks*, vol. 10, no. 4, pp. 377–388, July 2004.
[DPZ04]	R. Draves, J. Padhye, and B. Zill, "The architecture of the Link Quality Source Routing Protocol." Microsoft Research, Tech. Rep. MSR-TR- 2004-57, 2004.
[EE04]	P. Engelstad and G. Egeland, "NAT-based Internet Connectivity for On Demand MANETs," in Proc. of WONS 2004, Madonna di Campiglio, Italy, January, 18–23 2004, pp. 4050–4056.
[ETHE04]	P. Engelstad, A. Tønnesen, A. Hafslund, and G. Egeland, "Internet Connectivity for Multi-Homed Proactive Ad Hoc Networks," in Proc. of IEEE ICC'2004, vol. 7, Paris, France, June, 20–24 2004, pp. 4050–4056.
[GGP05]	C. Gomez, D. Garcia, and J. Paradells, "Improving Performance of a Real Ad-hoc Network by Tuning OLSR Parameters," in Proc. of IEEE ISCC 2005, Cartagena, Spain, June, 27–30 2005, pp. 16–21.
[JALJM00]	U. Jonsson, F. Alriksson, T. Larsson, P. Johansson, and G. Maguire Jr.,

	"MIPMANET - Mobile IP for Mobile Ad Hoc Networks," in Proc. of MobiHoc 2000, Boston, MA, USA, August, 11 2000, pp. 75–85.
[NP02]	S. Nesargi and R. Prakash, "MANETconf: Configuration of Hosts in a Mobile Ad Hoc NEtwork," in Proc. of INFOCOM 2002, vol. 2, New York, NY, June, 23–27 2002, pp. 1059–1068.
[RFC1027]	S. Carl-Mitchell and J. Quarterman, "Using ARP to Implement Transparent Subnet Gateways," RFC 1027, October 1987. [Online]. Available: http://www.ietf.org/rfc/rfc1027.txt
[RFC2131]	R. Droms, "Dynamic Host Configuration Protocol," RFC 2131, March 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2131.txt
[RFC2663]	P. Srisuresh and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations," RFC 2663, August 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2663.txt
[RFC3344]	C. Perkins, "IP Mobility Support for IPv4," RFC 3344, August 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3344.txt
[RFC3636]	T. Clausen and P. Jaquet, "Optimized Link State Routing Protocol (OLSR)," RFC 3626, October 2003. [Online]. Available: http: //www.ietf.org/rfc/rfc3626.txt
[RFC3684]	R. Ogier, F. Templin, and M. Lewis, "Topology Dissemination Based on Reverse-Path Forwarding (TBRPF)," RFC 3684, February 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3684.txt
[RFC826]	D. Plummer, "An Ethernet Address Resolution Protocol," RFC 826, November 1982. [Online]. Available: http://www.ietf.org/rfc/rfc0826.txt
[T04]	A. Tønnesen. (2004, December) Implementation of the OLSR specification (OLSR UniK). Version 0.4.8. University of Oslo. [Online]. Available: http://www.olsr.org/
[TGRW04]	C. Tschuding, R. Gold, O. Rensfelt, and O. Wibling, "LUNAR: a Lightweight Underlay Network Ad-hoc Routing Protocol and Implementation," in Proc. of NEW2AN'04, St. Petersburg, Russia, February, 2–6 2004.
[V02]	N. Vaidya, "Weak Duplicate Address Detection in Mobile Ad Hoc Networks," in Proc. of ACM MobiHoc 2002, Lausanne, Switzerland, June, 9–11 2002, pp. 206–216.
[WZ04]	K. Weniger and M. Zitterbart, "Address Autoconfiguration on Mobile Ad Hoc Networks: Current Approaches and Future Directions," *IEEE Network*, vol. 18, no. 4, pp. 6–11, July/August 2004.

5. COOPERATION MECHANISM: CORE

In this Section we carry out a simulation-based study of the CORE mechanism, implemented as an add-on component for the Glomosim [1] network simulation suite. The features of CORE are analyzed in terms of simulation metrics that we deem relevant to assess the basic properties of a cooperation enforcement mechanism: the energetic cost beard by CORE-enabled nodes and the efficiency of the detection and punishment mechanisms used in CORE. Similarly to the simulation approaches available in the literature (refer to Chapter 3 in [2]) we run our experiments for various type of scenarios by taking into account both static and dynamic networks as well as different traffic patterns. In our simulation study, we use a simple model of node selfishness (refer to Chapter 2 in [2]) whereby misbehaving entities are defined at the beginning of the simulation and node behavior is independent of simulation time.

Simulation results are used to understand if and when a mechanism to distribute reputation information could be necessary in order to improve punishment efficiency: reputation distribution is an optional feature of the CORE mechanism and constitutes the main element to discriminate between CORE and other reputation-based cooperation enforcement mechanisms.

Even if interesting results can be obtained through an accurate simulation-based evaluation of cooperation schemes, we claim that the node selfishness model used in most work available in the literature is not sufficient to grasp the salient features of mechanisms intended to stimulate cooperation among self-interested entities. By assuming a static node misbehavior whereby nodes are defined as selfish for the whole network lifetime, it is arguable that the incentive properties of cooperation schemes can be properly shown. A selfishness model that does not take into account eventual variations in the behavior of the nodes is not appropriate for the validation of a mechanism that is intended to guide selfish nodes (or end-users operating the nodes) towards a more cooperative behavior.

Since a large fraction of existing cooperation enforcement schemes are based on principles akin to decision making and economic modeling, a natural tool that emerged to be suitable for the validation of such mechanisms is game theory. In [2] we define two analytical models that describe the MANET environment and the nodes participating to the network operation in game theoretical terms. Our research has been presented in a preliminary work [3], and has been extended in [4-7]. Using game theoretical models to validate a cooperation mechanism allows the definition of a dynamic node behavior that follows a "rational" strategy imposed by the end-user operating the node. A rational strategy represents the behavior of a self-interest user that tries to maximize her profits (in terms of energy consumption) while knowing that other users in the system could do the same or could use a cooperation strategy to enforce cooperation.

Game theoretical models of MANETs are however limited in that they do not realistically represent the underlying mechanisms (and their inherent limitations) that are used to operate the ad hoc network, such as medium access control and routing protocols. In the approaches presented in this Section we overcome to some of the limitations dictated by a high level representation of nodes' interactions within the network by including in our models the salient features of the mechanisms that are analyzed: for example, in the first

model presented in the sequel of this Section we take into account the issues related to the watchdog technique that have been exposed in Chapter 4 in [2].
By combining the results obtained through the simulation-based analysis and through the game-theoretical analysis of CORE we conclude that our scheme not only meets all the requirements that have been presented in Chapter 2 in [2] but also performs better than other cooperation strategies evaluated in the literature when a realistic network model is assumed.

5.1. MANET simulation with CORE-enabled nodes

In this section we present a simulation-based analysis of the CORE mechanism. We provide a description of CORE implementation choices, with an emphasis on the determination of the system parameters that have been presented in Chapter 4 in [2]. We then describe the simulation set-up by specifying the scenarios that have been chosen to test the features of CORE and the metrics used to judge the efficiency of the detection and punishment of selfish nodes eventually present in the network. We further concentrate on the consequences in terms of energetic consumption when CORE is used by the nodes. A thoughtful discussion on what is it possible to show through our simulations and what cannot be studied because of the inherent limitations of selfishness model is provided.
The validation of the CORE cooperation mechanism is presented through the analysis of simulations results and by further examination of reputation distribution requirements as well as efficiency of the punishment mechanism.

5.1.1. CORE implementation

The CORE mechanism has been implemented as a plug-in mechanism for the Glomosim network simulator: by referring to Figure 3.1 that describes the CORE components, we implemented the detection mechanism that relies on the promiscuous mode operation of 802.11 based radio cards, the reputation management component in which only **local** reputation information is used, and the punishment mechanism that modifies the forwarding behavior of a legitimate node that detected a selfish neighbor.
The *detection mechanism* implemented in each node monitors the behavior of neighboring nodes with respect to the (data) **packet forwarding function** and relies on a FIFO buffer that can store up to *B* past observations. We recall here that by **observation** we mean the result of the comparison between an expected packet stored in the expectation table (refer to Chapter 4 in [2]) and the observed packets overheard by the promiscuous listening of neighborhood operations. We give the value $\sigma_k = +1$ for the successful observation (made at time *k*) in which the expected packet equals the observed packet, and the value $\sigma_k = -1$ for an unsuccessful observation. The monitoring mechanism is based on the watchdog technique, for which we set a timeout value $WD_{TO} = 50ms$ and a sampling frequency $WD_{freq} = 1$ [observation per packet]. The choice of the timeout value has been tailored to meet memory allocation requirements and detection capabilities: a too high value would result in a huge amount of temporary

memory wasted to store expected packets that are accumulated in the expectation table at the rate of one expected packet per data packet generated.

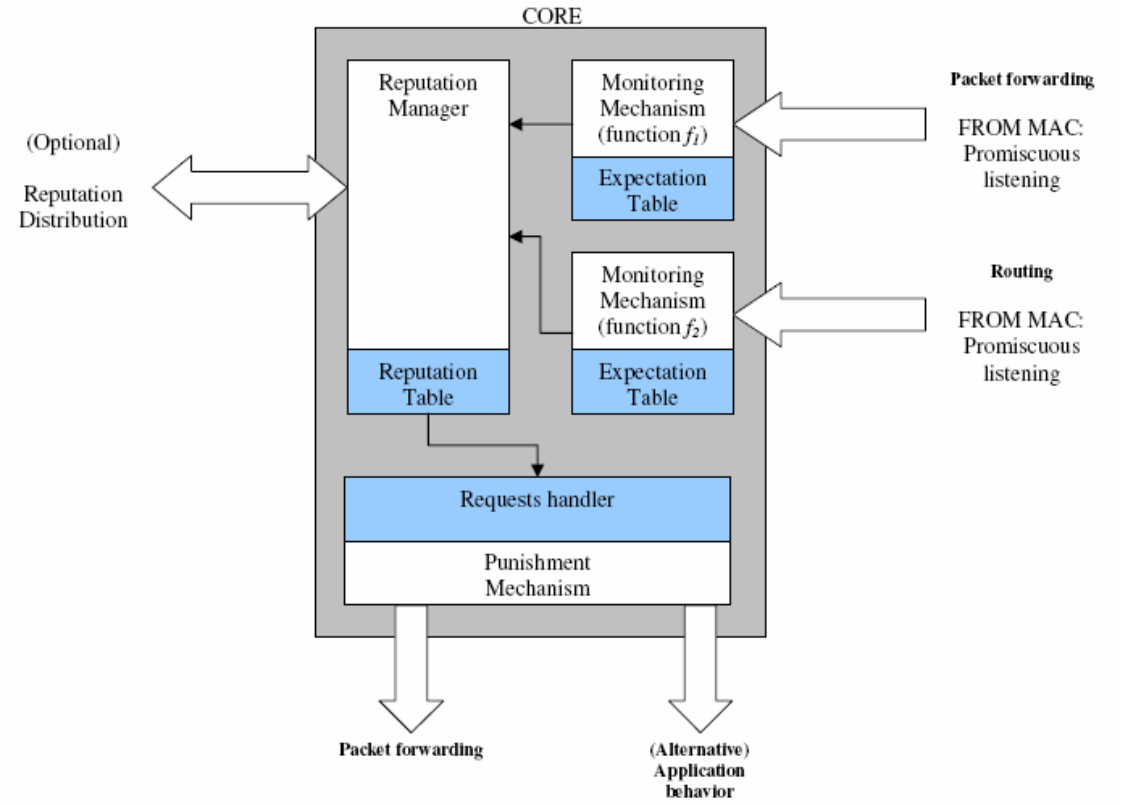


Figure 5.1. Core modular architecture.

On the other side, a too low value for the timeout would negatively impact the detection capabilities in a heavily loaded network by triggering false negative observations since eventual collisions at the MAC level or full queuing buffers due to a high number of data packets in transit slow down the forwarding response time of neighboring nodes.
The *reputation management component* has been implemented taking into account only local observations provided by the monitoring mechanism, while reputation information is not distributed in the network. In Chapter 4 in [2] we argued that the distribution of reputation information is utterly insecure and through our simulation study we want to study in which situations the detection capabilities of CORE would be improved if indirect reputation ratings would be used. For the sake of simplicity, we implemented the simplest reputation evaluation filtering function, which is a **moving average low-pass filter** with a window size of *B*. A reputation value is calculated for every neighboring node with respect to the packet forwarding function, thus the functional reputation method presented in Chapter 4 in [2] is not used in our simulation implementation.
Finally, the *punishment mechanism* has been implemented by temporarily disabling the packet forwarding function for neighboring nodes that have a reputation value that falls below the punishment threshold $P_{th} = 0$, that is, a node n_j is punished at time *k* by node n_i

because considered to be selfish if her reputation $r_{n_i}^k(n_j) < 0$. This implies that selfish nodes cannot send but can eventually receive data packets.

In the results section we analyze the effectiveness of the punishment mechanism when constant bit rate (CBR) data flows are defined between pairs of communicating nodes: we show that CBR sessions based on the UDP transport mechanism are blocked for selfish nodes because data packets originated by selfish sources are not forwarded, even in the case that a valid route has been provided to the selfish node. On the other hand, also HTTP Client-Server sessions based on the TCP transport protocol would be blocked using our punishment mechanism since a selfish node would not be able to send HTTP_Requests to the corresponding servers.
Table 5.1 summarizes the CORE parameters choices we made in our implementation.

System parameter	*Value*	*Description*
B	5	Window size of the moving average filter, *i.e.* number of past observations used to evaluate *subjective reputation*
WD_{TO}	50ms	Watchdog timeout value, *i.e.* timeout used to validate an expectation
WD_{freq}	$1 \frac{\text{observation}}{\text{packet}}$	Watchdog sampling frequency, *i.e.* number of observations per data packet generated
σ_k	$\{+1, -1\}$	Values associated to a positive or negative observation
P_{th}	0	Punishment threshold, reputation values below the threshold denote selfish nodes
Reputation	Only Local	Reputation value evaluated through the reputation management component

Table 5.1. CORE system parameters.

5.1.2. Simulation set-up

In this section we describe the simulation parameters we used in our study and the different scenarios that have been considered for the evaluation of CORE.
In our simulations we consider an ad hoc network formed by *N*=16 nodes that use the DSR routing protocol for a simulation time equals to 2000s[6].

[6] The size of the network has been chosen to simplify the discussion of the results, however we also studied the properties of CORE in a larger network obtaining similar results as the one depicted in the plots in the following results section.

Node placement have been manually selected to follow a grid pattern as depicted in Figure 5.2 and the distance between two nodes on the grid have been set in order to allow only communications that are on the rows or columns of the gird, whereas diagonal links cannot be established due to the wireless radio range limits.

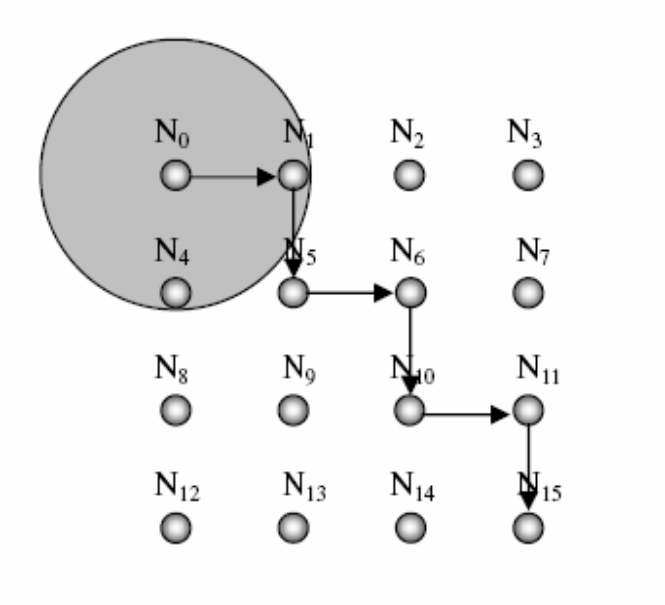


Figure 5.2. 4x4 grid network used in our simulations, with radio range and route example from source node 0 to destination node 15

Nodes are free to move (if specified in the scenario) in a 1000x1000 square meters area: the mobility model we chose is the *Random Waypoint Model*, in which nodes move to a random destination at a speed uniformly distributed between 1m/s and a maximum of 20m/s. Once they reach this destination, they stay there for as long as specified in the *pause time*, which we will use as a simulation parameter as defined later in this section.
The radio propagation model used in our scenarios is the realistic *two-ray ground reflection model* and we take into account physical phenomena such as signal strength, propagation delay and interference. The radio range has been set to the usual value of 250m.
We have defined two families of simulation scenarios: in the first set of simulations we consider a **static network** while in the second family we use the Random Waypoint mobility model to simulate a **dynamic network** in which we vary the pause time parameter.
To infer the salient characteristics of the CORE mechanism, we completed our simulation scenarios with a further parameter that we call **path diversity**, which is defined for node n_i as:

$$p_{D_i}^t = \frac{l_i}{N_{neighbors_i}}$$

where l_i is the number of incoming (or outgoing) routes to node n_i and $N_{neighbors_i}$ is the number of node n_i's neighboring nodes at simulation time *t*.

Now, the DSR routing protocol is not a multi-path routing protocol since it does not use multiple paths to reach a destination: however it stores multiple route replies that correspond to different routes from a source to a destination in order to improve responsiveness in the case of link outage. Thus, in our simulations we "*manually*" introduce the path diversity parameter by defining the following traffic patterns that use constant bit rate (CBR) communications based on UDP, and in which 1000 packets of size equals to 64 bytes are sent at a rate of 1 packet per second:

- *High path diversity traffic pattern:* sources and destinations of the CBR traffic are chosen to produce a **fully connected** network in which every possible source has to send data to every possible destination (*excluding all one-hop communications*) in the network, while the beginning time of CBR sessions is uniformly distributed between 0s and 800s.

- *Low path diversity traffic pattern:* sources and destinations of the CBR traffic are chosen randomly (*excluding all one-hop communications*), as well as the beginning time and the number of distinct CBR sessions.

As we will see in the results section, the path diversity parameter has an important influence on the performance of CORE: however, the "manual" configuration of the path diversity parameter revealed to be difficult to control in the dynamic network scenario. Indeed, when node mobility is high, it is hard to predict if the properties offered by the communication patterns defined at the beginning of the simulation will hold through time. However, path diversity can still be used in the dynamic case to explain some of the phenomena that appears in the result plots.
Furthermore, excluding one-hop communications is important in order to avoid the particular case in which the punishment mechanism implemented in CORE cannot be used. Again, when node mobility is introduced in the network, it is difficult to predict if a route counting more than one hop will hold through time: the metric used to measure the performance of CORE and described in the following section excludes one-hop communications eventually formed through the simulation run.
We additionally defined a simulation parameter that takes into account the dimension of the FIFO buffer used to store up to *B* past observations on neighborhood behavior: this parameter has been used in one specific set of simulations where we study the eventual errors in the punishment mechanism, that we call **false positives**.
Taking as a reference the selfishness model described hereafter, in the ideal scenario in which nodes cannot temporarily fail, that there are no obstacles and that the monitoring mechanism based on the promiscuous listening produces no errors, one observation would be sufficient to detect and exclude eventual selfish nodes. However, in the more realistic scenario in which temporary misbehavior is due to the presence of detection errors, a more sophisticated technique such as the one proposed in CORE has to be used.

Throughout our simulation study we analyze the impact of the buffer size *B* on the punishment errors due to failures in the detection mechanism.
For every scenario that is possible to define using the parameters described in this section, we run 20 experiments and took the average of the metrics used in the simulation and described in the following section. Further, we enriched the simulation scenarios by considering three additional experiment configurations: we consider and ad hoc network in which there are no selfish nodes, and ad hoc network that has a pre-defined percentage of selfish nodes and a CORE-enabled network that is populated with selfish nodes. We took as a reference a network with a low percentage of selfish nodes (6%) and a network in which a rather high percentage (25%) of nodes are selfish.

Selfishness model

The selfishness model used in our simulation study follows the definition that has been given in Chapter 2 in [2].
A selfish node is defined as a node that participate to routing operations but that systematically fails in forwarding data packets, by disabling the packet forwarding function at the network layer. As a practical example, a misbehaving end-user that operates a node can easily disable the packet forwarding function by using the `IPTABLES` command in a Linux powered ad hoc node.
In our simulations, during the simulation setup, we define the total number of selfish nodes: selfish nodes drop data packets for the whole duration of the simulation.
In the discussion section we argue that this simple model is not adequate to show how the CORE mechanism is able to promote cooperation of nodes: the necessity for a more complex model that takes into account node "*rationality*" is discussed, while in [2] we explain our proposed solution to overcome the limitations imposed by a simplistic selfishness model.

5.1.3. Simulation metrics

This section describes the metrics we used to evaluate CORE: the results presented in the following section refer to the metrics defined hereafter.

- **Energy consumption**

The GloMoSim network simulator has a built-in energetic model that we use to evaluate the energetic consumption that nodes operating the ad hoc network have to bear. By default, in GloMoSim a node in an IDLE status consume 500mW per hour. We used this value as a basis to infer the further consumption that derives from the network operation and from the application running on the nodes: power consumption statistics are collected through the whole simulation run, whereby a wireless channel-dependent energetic cost is associated to every transmitted or received packet.
In the following graphs, we present the energetic consumption for three set of experiments: an ad hoc network with no selfish nodes, a defenseless ad hoc network with selfish nodes, and a CORE-enabled network with selfish nodes.
Further, we show the **average gain** of a CORE-enabled network with respect to a defenseless network with selfish nodes. The average gain is defined as follows: we take the aggregate average energetic consumption for all the *legitimate nodes* in the

defenseless network and compare it to the aggregate average energetic consumption of the *legitimate nodes* in the CORE-enabled network and express the difference in percentage: as an example, an average gain of 10% indicates that, in average, the nodes of a CORE-enabled network saves up to 10% of energy with respect to a defenseless network.
We estimate energetic consumption to be a relevant metric since it represents the additional energetic cost beard by nodes that use a cooperation enforcement mechanism. This additional cost has to be limited since it could be a further source of node selfishness.

- **Punishment efficiency**

The punishment efficiency metric is defined as follows:

$$p_E = \frac{\sum_{N'} d_p}{\sum_{S'} s - h} \cdot 100\%$$

where N' is the subset of legitimate nodes in the network and S' is the subset of selfish nodes in the network;
d_p is the number of data packets originated by selfish nodes and discarded by the punishment mechanism implemented in *all* legitimate nodes;
s is the number of packets originated by each selfish node in the set S' that have a valid route to the intended destination and h represents the number of packets eventually originated by selfish nodes that are on a one-hop route.
For example, suppose that N'=15, S'=1, d_p=800, s=1000 and h=100:
p_E = 88.89%.

The punishment efficiency metric provides an overall metric to judge the effectiveness of the CORE mechanism that takes into account both the detection and the punishment mechanism: indeed, a legitimate node punishes a selfish node when a predefined number of observations have been collected through the monitoring mechanism and processed by the reputation manager component.

- **False positives**

The false positive metric is similar in its definition to the punishment efficiency:

$$\text{false positives} = \frac{\sum_{N'} d_p}{\sum_{N'} s - h} \cdot 100\%$$

With this metric we analyze the percentage of packets that have been erroneously dropped by legitimate nodes and that were originated by other legitimate nodes. Again, we do not take into account one-hop communication patterns.

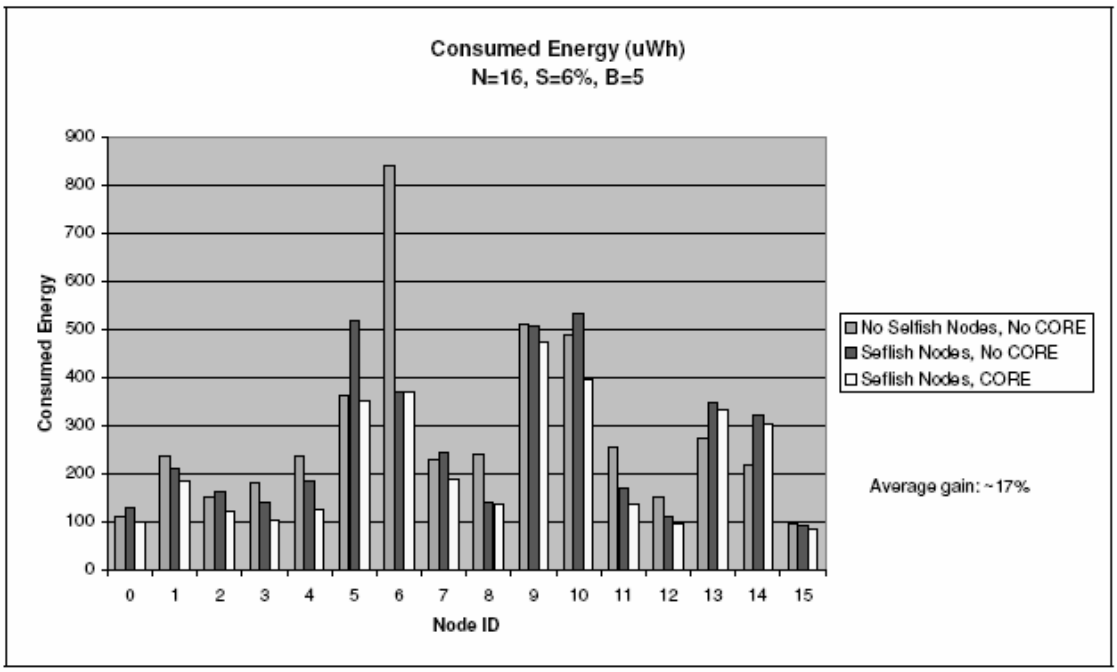
NOTE: in our simulation-based analysis of CORE we do not show packet delivery ratio variations for legitimate nodes. As opposed to the other cooperation enforcement mechanism available in the literature, CORE does not punish selfish nodes by using the *path rater technique* described in [2]. The consequence is that we cannot show through our simulations that the packet delivery ratio or alternatively the aggregate throughput of the network **increases** when the CORE mechanism is used by the nodes. We already discussed the drawbacks of the path rater technique in [2]: furthermore we do not believe that a performance metric based on throughput is meaningful to show the salient aspects of a cooperation enforcement mechanism.
Through our simulation study we are able to show the impact of CORE in terms of energetic consumption (which we recall, is the main source of a selfish behavior) and in terms of punishment efficiency, but we claim that the ability of CORE (and other cooperation enforcement mechanisms) to promote cooperation requires the definition of a different selfishness model that allows a selfish node to act *rationally*, i.e. minimizing energy consumption while knowing that other nodes will undertake the appropriate countermeasures whenever a selfish behavior is detected.

5.2. Simulation results

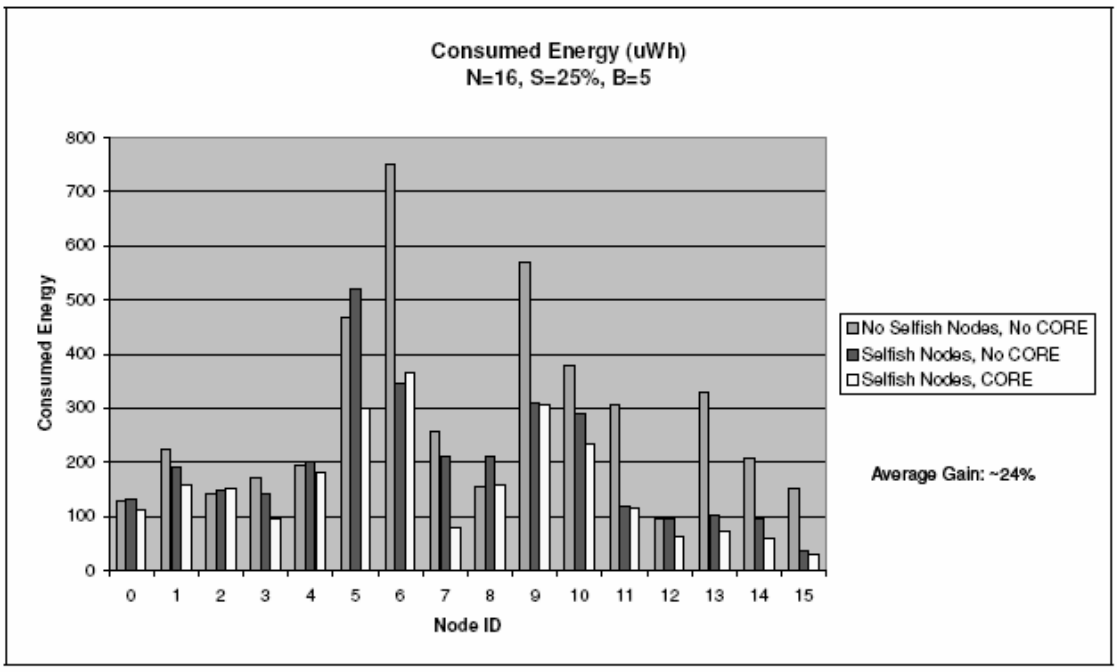
In this section we discuss the results of our simulation study: the plots presented hereafter are organized in two sections, one that shows the nodes' energetic consumption, and one that shows detection and punishment capabilities of a CORE enabled network.
Throughout our simulation study, we were able to assess not only that CORE introduce a low power consumption overhead (also with respect to other cooperation enforcement mechanisms available in the literature), but also that it entails a significant power saving for legitimate nodes that use CORE. Thus, CORE stimulates cooperation of selfish nodes to the network operation and introduces an incentive for legitimate nodes to use CORE as a cooperation enforcement mechanism since they save energy.

5.2.1. Energetic consumption

The following plots (Figures 5.3, 5.4, 5.5) present the average energetic consumption (per node) provided by the GloMoSim simulation statistics for different scenarios. We considered both static and dynamic networks and we varied the path diversity parameter (as described in the setup section) and the percentage of selfish nodes. Note that in the following plots, when the percentage of selfish nodes is 6%, only one node misbehave: precisely, in our plots the node with *ID*=6 is misbehaving. On the other hand, when the percentage of selfish nodes is 25%, there are 4 selfish nodes in the network: the following plots depict a network in which nodes with *ID*=2, 6, 9, 11 are misbehaving.



(a)

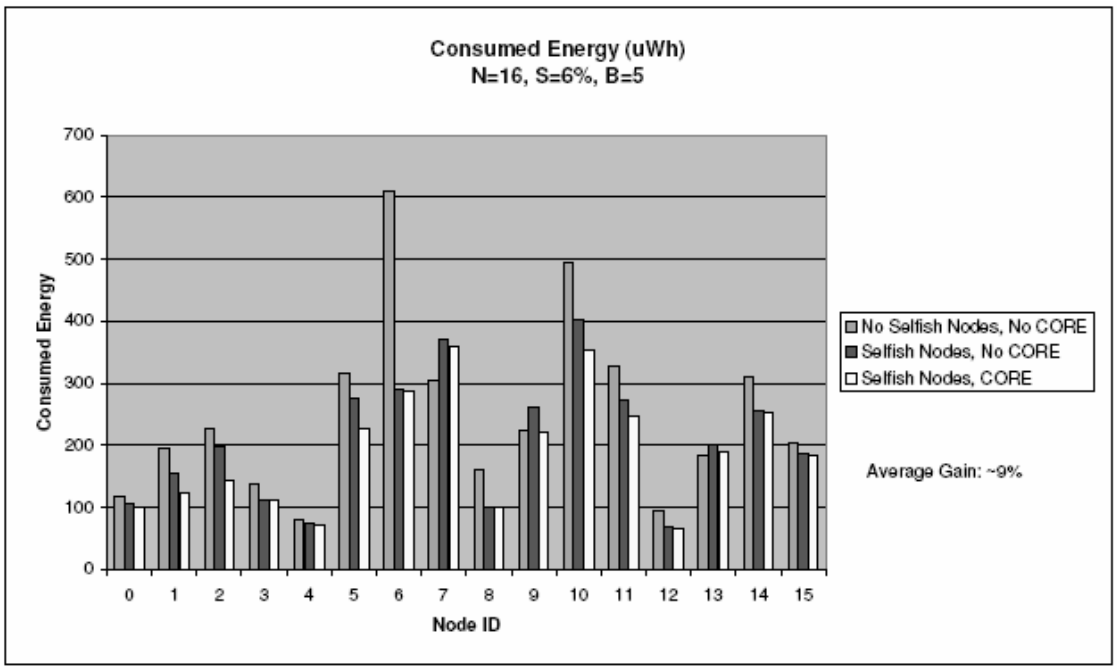


(b)

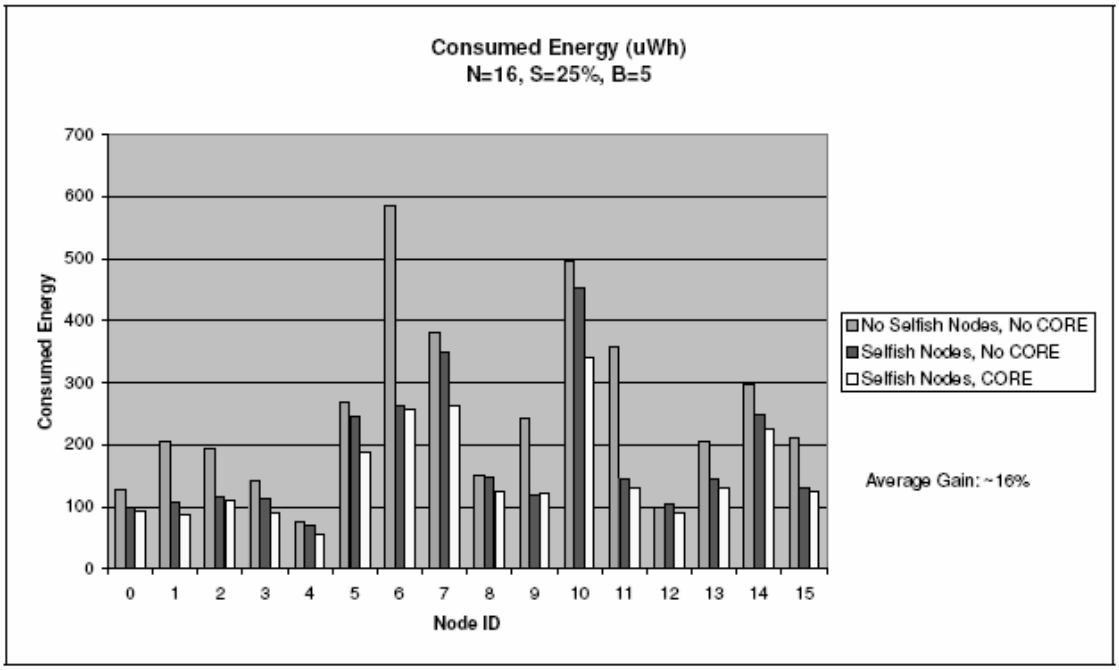
Figure 5.3. Static network, S\% of selfish nodes: routes with more than 1 hops and high path diversity.

The first observation that we can make by an overall analysis of the plots derives from a comparison between the power consumption of an ad hoc network without selfish nodes

and a defenseless network with a defined percentage of selfish nodes. If it is not surprising to see that selfish nodes save energy by dropping data packets, it is interesting to observe that also legitimate nodes' power consumption is impacted by the presence of selfish nodes: in average, also legitimate nodes might gain in energetic savings when selfish nodes are present in the network.
Indeed, consider for example the following *n+2*-hop path from a legitimate source *S* to the corresponding destination *D*: *<S, N1, N2, N3, N4,..., Nn, D>*
If we consider the extreme scenario in which *N1* is dropping data packets because of a *selfish* behavior, also all the other down-link nodes {*N2...N_n*} towards the destination will save energy because they will not receive any data packets to forward.
On the other hand, in the opposite scenario in which the last hop before the destination (namely, *Nn* in our path example) is *selfish*, all the up-link nodes towards the source will waste energy by forwarding data packets that will never reach the destination.
The use of a multi-path routing scheme with a high *path-diversity* would mitigate the waste in terms of energy of legitimate nodes that detected the presence of a selfish node in the route; at the same time, a multi-path routing scheme allows nodes to "probe" multiple routes and eventually increase detection capabilities.



(a)



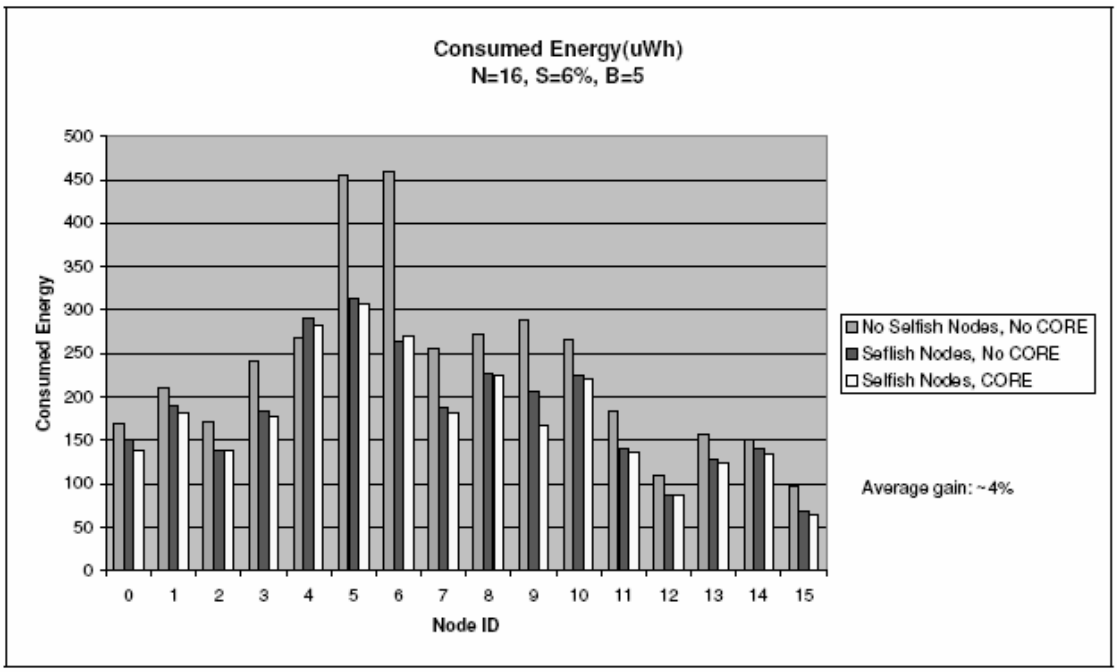
(b)

Figure 5.4. Static network, S\% of selfish nodes: routes with more than 1 hop and low path diversity.

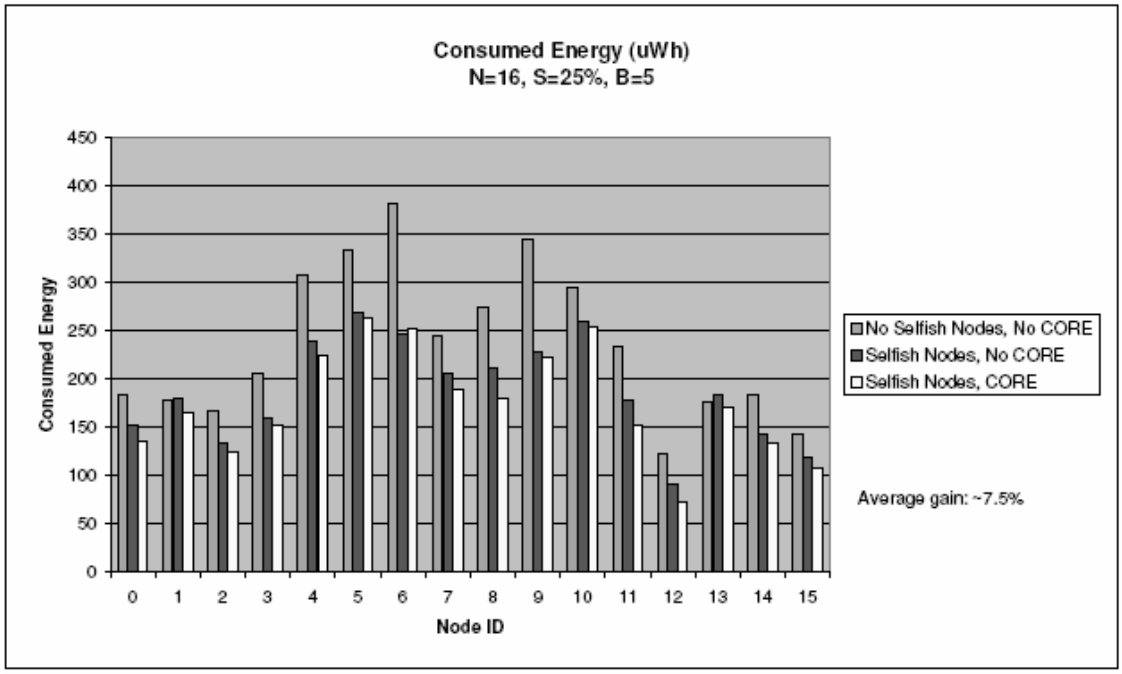
On the other side, when comparing energetic consumption of nodes in a defenseless network as opposed to nodes in a CORE-enabled network, it is possible to see that nodes

in a CORE-enabled network save energy by punishing selfish sources that originate data traffic. Consider the following path, in which the source node S has been detected as selfish by (all) her neighbors: *<S, N1, N2, N3, N4,..., Nn, D>*

If the network has no countermeasures to cope with node selfishness, every node on the path towards the destination wastes energy by forwarding "illegitimate" traffic generated by a selfish source that does not cooperate to the network operation.
On the contrary, in a CORE-enabled network if node *N1* detected a selfish behavior of the source, she will then punish the selfish source by denying data forwarding. Moreover, all nodes down-link towards the destination will benefit from the punishment of the selfish source performed by *N1* and will save energy.



(a)



(b)

Figure 5.5. Dynamic network, S\% of selfish nodes [pause time = 0].

The plots presented in the previous figures also show the *average gain* in terms of energetic consumption of a CORE-enabled network with respect to a defenseless network, without taking into account the gain of selfish nodes: as it is possible to see in

the figures, average gains vary depending on mobility settings as well as percentage of selfish nodes in the network and path diversity parameter. In Figure 5.5 we set the *pause time* parameter to 0 seconds in order to tackle with an extreme mobility scenario: nodes constantly move with a speed uniformly distributed between 1m/s and 20m/s.
When comparing plots with the same amount of selfish nodes in the network but different mobility settings, it is possible to see that average gains in static networks (~17%) are higher than in the dynamic case (~4-5%): indeed, node mobility not only introduces a high amount of traffic overhead due to frequent link breaks but, as it will be shown in the punishment efficiency plots, it has an impact on the detection and punishment mechanisms.
Furthermore, it is possible to deduce from the plots that the average energetic gain is higher when a larger portion of nodes of the network is set to be selfish: this result follows the discussion on the position (and number) of selfish nodes in a path from a source to a destination and the number of selfish sources in the network, as described in the beginning of this section.
As it will be clearer in the section dedicated to punishment efficiency plots, also the *path diversity* parameter has an influence on the average energetic gain: detection and punishment capabilities are impacted by the presence of multiple path towards and from a selfish node. A reduced path diversity, which we simulated through a small number of CBR connections in the network, degrades the detection and punishment capabilities of CORE.

5.2.2. Punishment efficiency

In this section we present the plots for the punishment efficiency (p_E) metric versus the pause time parameter: when we consider a static network, the pause time has no influence on the results, whereas for the dynamic case the impact of node mobility is more important. For every pause time value (i.e. pause time = {0, 10, 100, 300, 600, 900} seconds) we run 20 experiments varying the selfish node percentage and the selfish node position (i.e. the selfish node ID) in the network. In the static case, we also varied the path diversity parameter.
Figure 5.6 presents a summary of the punishment efficiency for all simulation scenarios we used, while Figures 5.8, 5.9 are enriched with measurement errors for every specific scenario we considered.

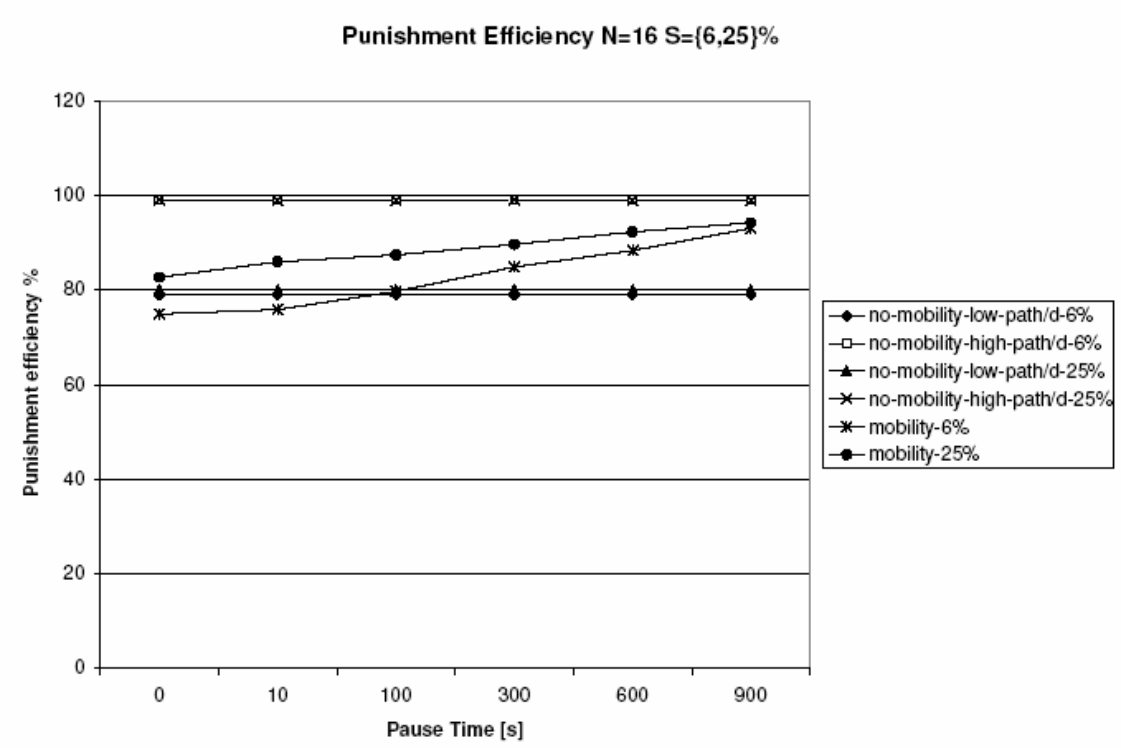


Figure 5.6. Summary of p_E for different simulation scenarios.

In Figure 5.6 it is possible to observe that, in average, punishment efficiency is higher for a static network as compared to a dynamic network. Furthermore, our plots show that the path diversity parameter has a significant influence on the punishment efficiency (and detection capabilities) of a CORE-enabled network. Lastly, as for the energy consumption plots, a higher percentage of selfish nodes in the network produce higher punishment efficiency.
To explain the behavior of the punishment efficiency plots, we refer now to the following specific scenarios.

Static network

When the nodes of the network do not move, the punishment efficiency depends on the path diversity parameter. Let's give a practical example to explain how path diversity impacts both detection and consequently punishment capabilities of CORE.
The following figure depicts the gird network used in our simulation, where node *N6* is selfish.

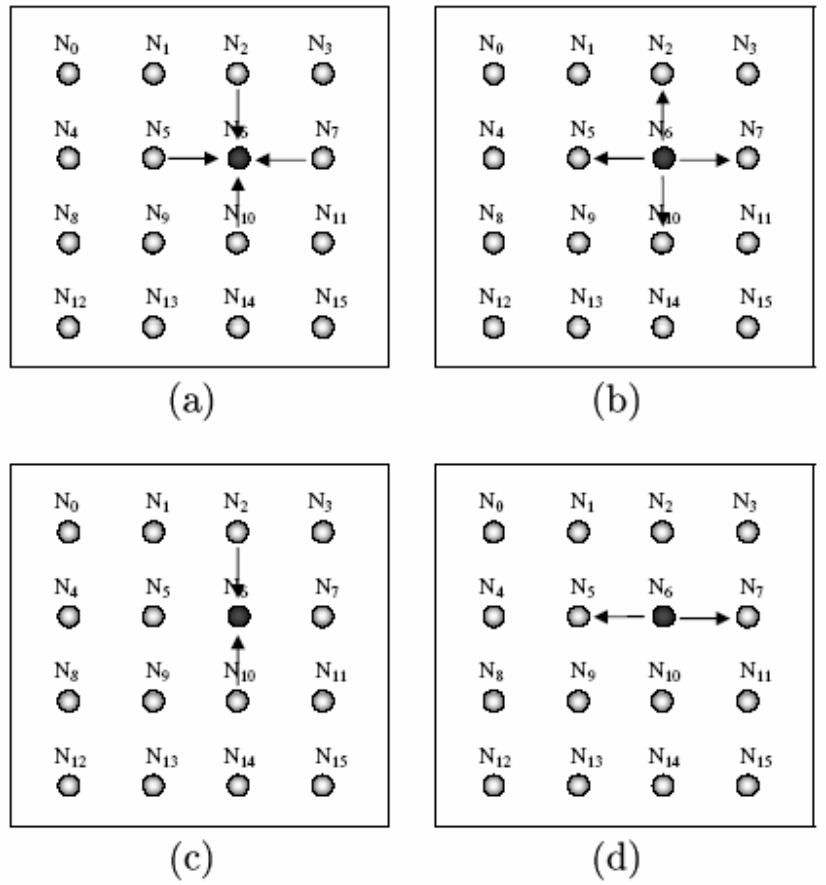


Figure 5.7. Static grid network with 1 selfish node: path diversity example.

Detection capabilities as a function of path diversity are depicted in Figures 5.7 (a) and (c), while punishment capabilities are depicted in Figures 5.7 (b) and (d). When path diversity is high, all neighboring nodes of node *N6* have some routes that use the selfish node as a relay (either because they are sources or because they are relaying packets on behalf of distant nodes): in this case the detection mechanism assures that all neighboring nodes detect and classify node *N6* as selfish. In the same way, when path diversity is high, if node *N6* has routes that use at least one neighboring node, the punishment efficiency reaches almost 100%. Figure 5.8 shows an average value of 99% since our path diversity has been manually imposed through an appropriate traffic pattern setting, which however does not guarantee the same results that we would have obtained by using a true multi-path routing protocol.
On the other side, when path diversity is low (Figures 5.7 (c) and (d))only nodes *N2* and *N10* uses routes that go through the selfish node *N6*: only two nodes over four neighbors would detect the selfish behavior. Now, in the non negligible case in which node *N6* uses only routes that go through node *N3* and *N7*, the punishment efficiency would be 0%.

This example shows the limitations of CORE: in the absence of a reputation distribution mechanism, legitimate nodes that never directly experienced transmission failures that

can be attributed to a selfish behavior would not punish the selfish node. However, this limitation can be overcome in different ways: we already mentioned that a multi-path routing protocol that exploits path diversity would inherently solve the problem. Furthermore, in [2] we proposed a CORE enhancement that relies on *peer nodes* which are able to observe selfish behavior without being involved in a forwarding transaction.
Figure 5.8 shows the average punishment efficiency that we experienced in our simulations: in average, the punishment efficiency is below 100%; however we believe it sufficient to represent a strong incentive for selfish nodes to participate to the network operation.

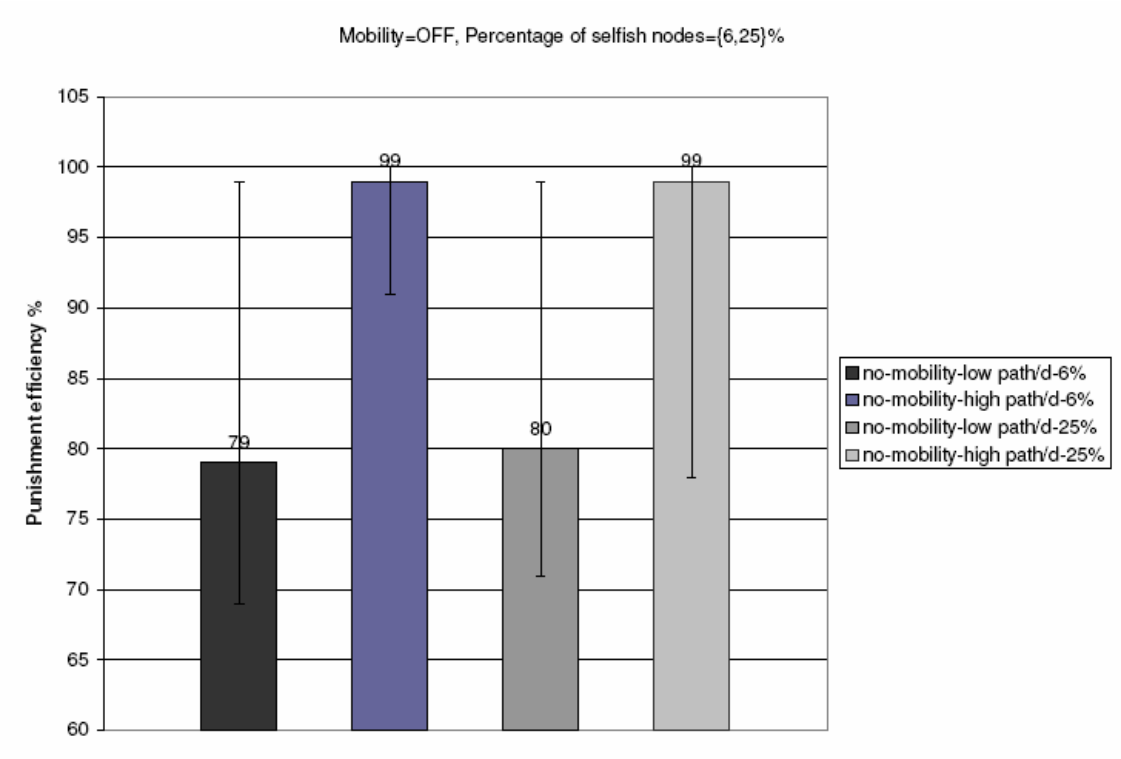
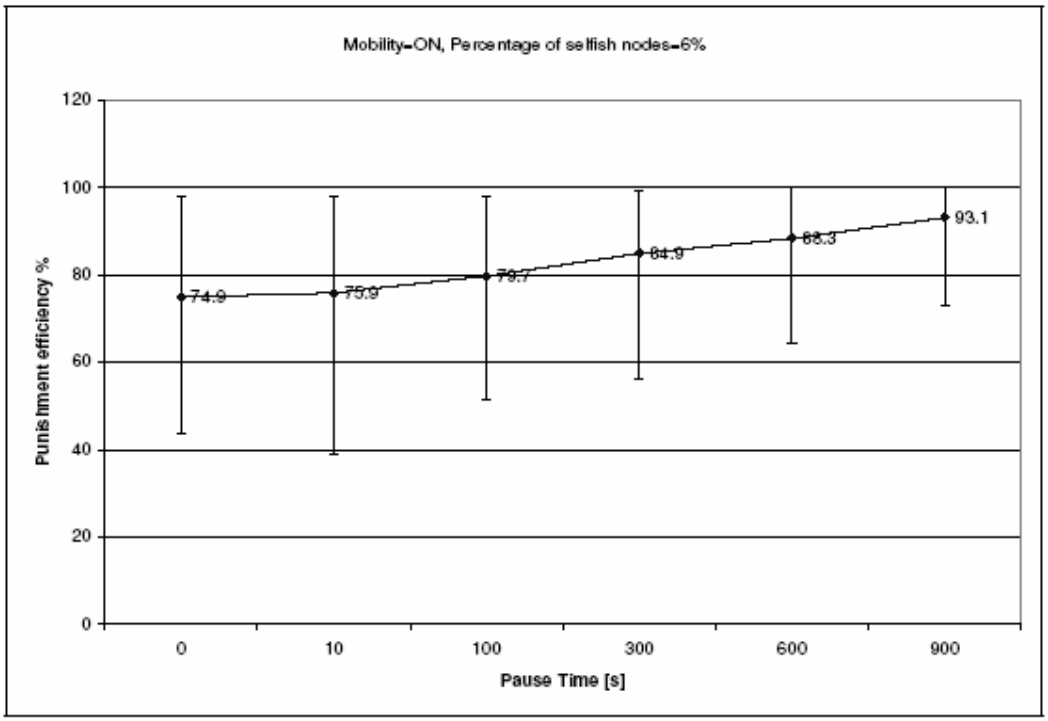


Figure 5.8. p_E for a static network, with measurement errors.

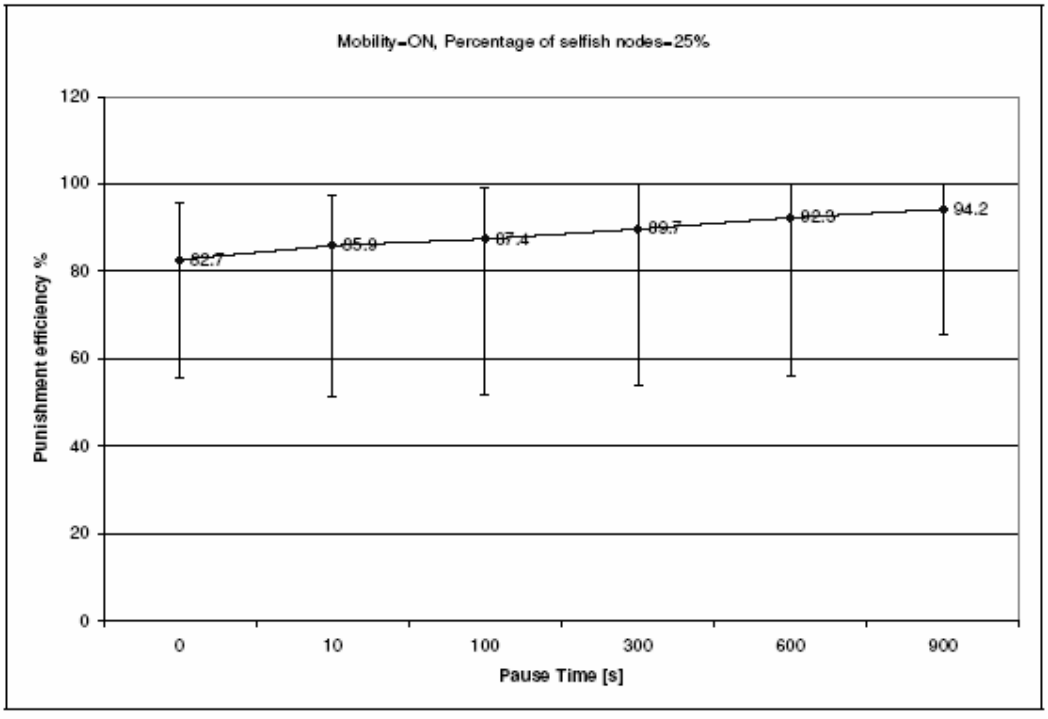
Dynamic network

By taking into account node mobility, parameters such as route hop count and path diversity are more difficult to control with respect to the static case. In the plots presented in the following figures, we vary the percentage of selfish nodes in the network and we analyze the variation of the punishment efficiency metric with respect to the pause time parameter of the Random Waypoint mobility model. The nodes' initial position is defined using the same grid network showed for the static case, which is then deformed due to node movements.
As it is possible to observe in figures 5.9 a) and b), punishment efficiency grows in the pause time, and reaches high efficiency values (i.e. ~100%) starting from a pause time of

900s. On the other side, when node mobility is very high, i.e. when nodes continuously move to random destinations, punishment efficiency is comparable to the one evaluated for the static network with low path diversity.
Since the nodes of the network rely on the DSR routing protocol which uses only one path that is reconstructed every time a link break is detected, path diversity is hard to estimate, even b imposing the adequate traffic patterns. We believe that path diversity registers negative variations due to node mobility: for example, neighbors that detected a selfish behavior could move away from the selfish node before she sends packets that would be dropped by the punishment mechanism. Other nodes that never used the selfish node as a relay (i.e., they could not detect it as being selfish), could fall within the wireless radio range of the selfish node and thus be used as relays for forwarding selfish traffic.
Even if the punishment efficiency offered by CORE in a dynamic network can be estimated as an effective incentive for node cooperation, we envision in our future work to use a variation of the DSR protocol that exploits multiple paths and that do not use gratuitous route reply or route error messages, which we deem a source of a low path diversity.



(a)



(b)

Figure 5.9. p_E for a dynamic network, with measurement errors.

False positives

The last plot we present shows punishment errors due to the shortcomings of the watchdog technique.

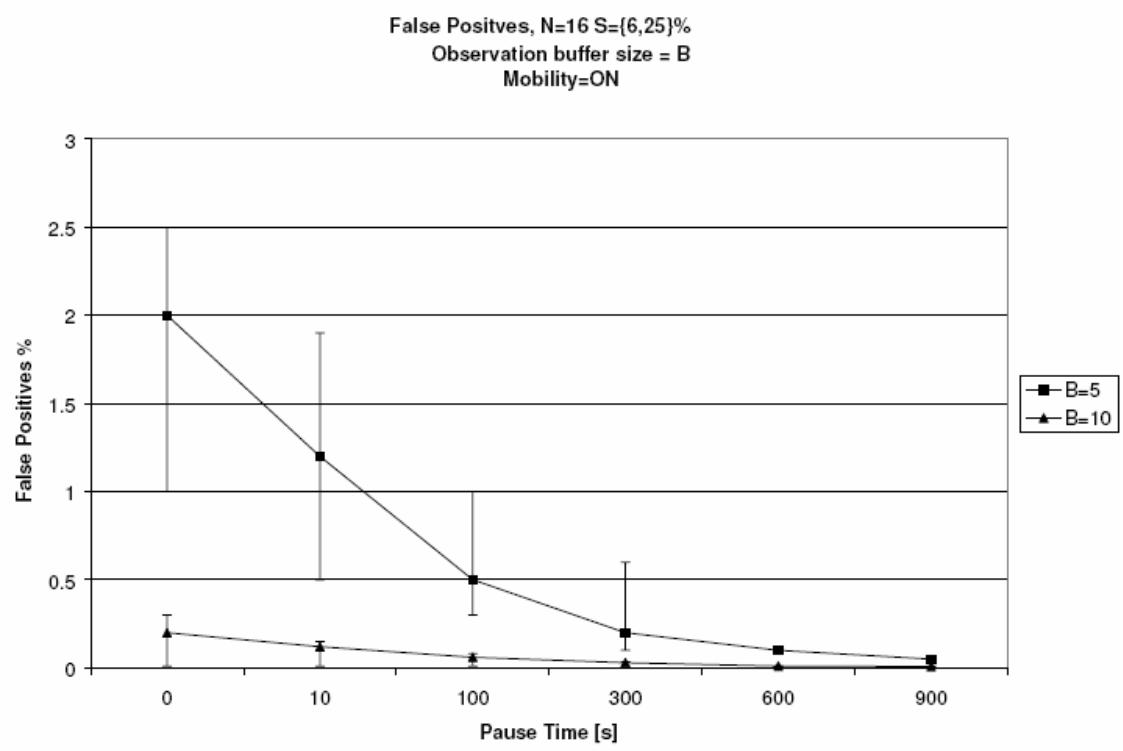


Figure 5.10. False positives with 25\% of selfish nodes in a dynamic network.

In the plots, we varied the *B* parameter, which has always been set to *B=5* in all other simulation runs. We believe that the filtering function and the filter parameters used to evaluate reputation metrics for neighboring nodes is the main responsible for errors in the classification of selfish nodes. Assuming that the monitoring mechanism is imperfect, we want to study which is the impact of the parameter *B* on the false positive metric.
As it is possible to see in Figure 5.10, the false positive metric is inversely related to the pause time parameter, which we varied as in the punishment efficiency evaluation.
When the observation buffer size is *B=5*, a non negligible percentage of false positives appear only in the case of a high mobility scenario: however, globally, for every legitimate node in the network, only 2 packets per 100 packets sent by legitimate nodes might be dropped erroneously. Varying the pause time parameter, or by doubling the buffer size *B=10*, is sufficient to render the percentage of false positives almost negligible.

5.3. Discussion

The results section provides a detailed analysis of CORE performance under different networking scenarios: as we observed, the punishment efficiency is for most of the scenarios adequate to promote node cooperation since performance of selfish nodes is drastically reduced in a CORE-enabled network.
Furthermore we demonstrated throughout our simulation study that CORE provides not only **incentives for selfish nodes to cooperate**, but also that CORE is **attractive for legitimate nodes** in that they might save up to 24% of energy which, as a consequence, extends the network lifetime in the case that all nodes use CORE.
In some cases, we noticed that the distribution of reputation information could increase detection capabilities when path diversity is low or when we consider high node mobility scenarios. However we suggest that using a multiple-path routing protocol that exploits high path diversity would inherently solve the problems pointed out in the results section.

Furthermore, by using the extended version of CORE which implements the peer validation technique described in section [2], punishment efficiency can be improved also in the case of a single-path routing protocol. Our simulation study has also been used as a basis for the design of a test-bed implementation of CORE and might be used for the fine-tuning of CORE parameters.

We believe, however, that the selfishness model implemented in our simulation study (and in other simulation study available in the literature) is not sufficient to grasp the salient aspects of node cooperation when "rational" end-users selfishly operate the nodes of the network. Indeed, a static selfishness model in which a misbehaving node never changes her behavior is not realistic: if changes in the selfish behavior are not allowed, then it is impossible to show in our simulations the capabilities of CORE to stimulate cooperation. In our work presented in [2], we build an analytical model of an ad hoc network in which rational selfish agents operate the network maximizing their profits in terms of energy consumption while at the same time knowing that other agents in the network could do the same or could use a cooperation enforcement mechanism like CORE to enforce node cooperation.

5.4. References

[1] Glomosim, available at http://pcl.cs.ucla.edu/projects/glomosim/

[2] P. Michiardi, Cooperation enforcement and network security mechanisms for mobile ad hoc networks, PhD Thesis Dissertation, available at PASTEL, Paris Institute of Technology, http://pastel.paristech.org/archive/00001114/

[3] P. Michiardi and R. Molva, Report on a Working Session on Security in Wireless Ad Hoc Networks, ACM Sigmobile Mobile Computing and Communication Review, 2002 volume 6, number 4

[4] P. Michiardi and R. Molva, Analysis of Coalition Formation and Cooperation Strategies in Mobile Ad hoc Networks, Elsevier - Ad hoc Networks Journal (Special Issue, 2004

[5] E. Altman and A. Kherani and P. Michiardi and R. Molva, Non cooperative forwarding in Ad hoc Networks, INRIA Sophia Antipolis Research Report RR-5116, February 2004

[6] E. Altman and A. Kherani and P. Michiardi and R. Molva, Non cooperative forwarding in Ad hoc Networks, in Proceedings of IFIP Networking Conference, Waterloo, Canada, May 2005

[7] E. Altman and A. Kherani and P. Michiardi and R. Molva, Some game-theoretic problems in wireless ad hoc networks, in Proceedings of the EURO-NGI Workshop, Dagstuhl, Germany, October 2004

6. MIDDLEWARE

Mobile ad hoc networks (MANETs) represent complex distributed systems comprised of wireless mobile nodes that can freely and dynamically self-organize into arbitrary and temporary, "ad-hoc" network topologies. This spontaneous form of networking allows people and devices to seamlessly exchange information in areas with no pre-existing communication infrastructure, e.g., disaster recovery environments. While the early MANET applications and deployments have been military oriented, civilian applications have also grown substantially since then. Especially in the past few years, with the rapid advances in mobile ad hoc networking research, mobile ad hoc networks have attracted considerable attention and interests from commercial business industry, as well as the standards community. The introduction of new technologies, such as the Bluetooth and IEEE 802.11, greatly facilitated the deployment of ad hoc technology outside of the military domain, generating a renewed and growing interest in the research and development of MANET.

While ad hoc networking applications appeared mainly in specialized fields such as emergency services, disaster recovery and environment monitoring, MANET flexibility makes this technology attractive for several applicative scenarios like, for example, in personal area and home networking, law enforcement operation, search-and-rescue operations, commercial and educational applications, and sensor networks. Currently developed mobile ad hoc systems adopt the approach of not having a middleware, but rather rely on each application to handle all the services it needs. This constitutes a major complexity/inefficiency in the development of MANET applications. Indeed, most of the MANET research concentrated on the enabling technologies, and on networking protocols (mainly routing), see [CCL03], while research on middleware platforms for mobile ad hoc networks is still in its infancy. Recently, in research circles, some middleware proposals for mobile ad hoc environments appeared in [BCM05, MPR01, MCZE02].

Their emphasis is on supporting transient data sharing [MPR01] between nodes in communication range, data replication for disconnected operations [BCM05, MCZE02], or both [H03]. To achieve this, classical middleware technologies have been adopted. These include tuple spaces, mobile agents, and reactive programming through the usage of events' publishing/subscribing [ADGS02, PC02]. While these technologies provide service abstractions that highly simplify application development, their efficiency in ad hoc environments is still an open issue.

Ad hoc networking shares many concepts, such as distribution and cooperation, with the peer-to-peer (p2p) computing model [SGF02]. A defining characteristic of p2p systems is their ability to provide efficient, reliable, and resilient message routing between their constituent nodes by forming virtual ad hoc topologies on top of a real network infrastructure. The difference with traditional distributed computing systems is the lack of a central authority that controls the various components; instead, nodes form a dynamic and self-organizing system. The applications best suited for p2p implementation are those

where centralization is not possible, relations are transient, and resources are highly distributed [PC02]. In particular, the range of applications covered by the p2p model includes file sharing, distributed search and indexing, resource storage, collaborative work, etc. The key aspect of p2p systems is the ability to provide inexpensive, but at the same time scalable, fault tolerant and robust platforms. For example, file sharing systems, like Gnutella [KM02], are distributed system where the contribution of many participants with small amounts of disk space results in a very large database distributed among all participant nodes.

The distributed nature of ad hoc networks fits well the p2p model of computation. Systems based on the p2p model are those where centralization is not possible, relations are transient, and resources are highly distributed. These are also main requirements and characteristics of MANET environments. This duality makes exploiting the p2p paradigm for designing middleware platform for MANETs a very promising direction.

In this chapter we investigate the efficiency of p2p middleware platforms when implemented in mobile ad hoc networks. Specifically, we focus on two well known platforms like Gnutella and Pastry. Through simulations, we show the limitations and the inefficiencies that these p2p systems exhibit in MANET environments. Finally, the chapter ends by showing, through experimentation, the potentialities of a cross-layer interaction between a proactive routing protocol and a p2p platform that offers the same functionalities and semantic of Pastry [DR01]. We give perspectives on how costs and complexity of building and maintaining a Pastry-like overlay network can be reduced through cross-layer interactions.

6.1. Performance of Peer-to-peer platforms in ad hoc environments

A key challenge to the usability of a data-sharing peer-to-peer system is implementing efficient techniques for search and retrieval of shared data. The best search techniques for a system depend on the needs of the distributed application. For example, applications like group multicasting, web caches or archival systems focus on availability, and need guarantees on content location (if it exists). However, these requirements are usually met at the expense of flexibility, for example by having search indexes organized by data identifiers, which allow quick lookup procedures by limiting the subject space, or imposing strict rules on their format, and by exactly controlling how the search index should be organized in the distributed system. In contrast, other kinds of applications, like for example file sharing or publish/subscribe systems, require the ability to issue rich queries such as regular expressions, meant for a wide range of users from autonomous organizations. Moreover, this second class of applications requires a greater respect to the autonomy of individual peers, without requiring them to host parts of the distributed search index. These requirements clearly relax assumptions and expectations on the performance of the p2p system. The aforementioned differentiation on the requirements of distributed data sharing applications, led to two p2p computational models: structured and unstructured platforms.

Structured platforms are such that peers organize themselves in a distributed search index (also called a structured overlay network), that usually contains information on the exact location of each shared data. In these systems, each peer maintains only a partial knowledge of the index, establishing "key" network relationships with other peers, so to make possible an almost complete coverage of the search structure. The main idea is to map both peers and data identifiers on the same logical space, and assuming that a peer with a logical identifier P gets relevant information about data logically close to P. This approach allows for subject-based lookup procedures, where a peer with the identifier D of the wanted data item (e.g., a file name or a multicast group identifier), initiates a distributed search algorithm that hop after hop in the structured overlay ends up on the peer logically closest to D. There are various proposals in the area of structured data sharing. For example, Pastry [DR01] and Chord [SMKK01] organize the overlay as a distributed ring of identifiers, while CAN [RFHKS01] use the concept of a distributed quad-tree on an n-dimensional space. All these platforms achieve optimal lookup performances, guaranteeing retrieval of shared content information in a logarithmic number of hops in the overlay, and requiring each node to establish a little number of relationships in the overlay.

Unstructured platforms are such that peers do not self-organize in a distributed search index, and are not required to maintain relevant information about shared content owned by other entities (e.g., a distributed search index). Peers establish network relationships in a pseudo-random fashion starting from a given entry point (i.e., a boot peer), and look for shared data initiating flood search procedures. This approach does not match availability guarantees like in the case of structured platforms, but allows for content-based lookup procedures based on regular expressions, to retrieve shared data. Content-based lookups are directly applied on the published content, and assume that a large number of peers get hit by search requests, for example through query propagation schemes based on flooding. Platforms like Gnutella [KM02] or KaZaa [KaZaa] witness the flexibility offered by the unstructured approach in supporting very large-scale file sharing applications on the Internet. Moreover, the characteristic of being open platforms, with discussion and development forums, brought existing systems to an established maturity, where available protocol specifications make it easy to adopt and deploy them with new implementations, introducing innovative optimizations directly in real test-beds.

In the context p2p computing for mobile ad hoc networks, it is advisable to consider both unstructured and structured approaches, as they could better support one or the other kind of application for distributed data sharing. Furthermore, an initial evaluation of the capacity and the performance of existing data sharing platforms in ad hoc environments would provide an important starting point for future discussions and new proposals. To this end, in the following we show algorithmic details regarding Gnutella and Pastry, respectively two representatives of the unstructured and structured classes. Through simulative and experimental results, we provide clear pictures about their capacity and performance when employed in ad hoc scenarios.

6.1.1. The Gnutella protocol

In this section, we describe the Gnutella protocol for overlay maintenance and data lookup. For more details, please refer to the latest specification [KM02]. Some of the information in this section is not part of the original protocol (e.g., the behavior of a peer accordingly to its connectivity in the overlay), but represents details added for clarity. Note that the Gnutella specification makes distinction between ordinary and super peers. Super-peers are those making up the overlay and providing the search infrastructure, and are usually represented by nodes with permanent Internet connection. In contrast, ordinary (or leaf) peers have intermittent connectivity, so they don't take part to the overlay formation, but simply attach themselves to an arbitrary number of super-peers, proxying queries through them. In the context of this chapter, we are interested in the general properties of the overlay formation protocol, so we don't consider ordinary peers in following discussion.

State maintenance. Gnutella operations rely on the existence of an unstructured overlay network. Peers open and maintain application layer connections among them, forming logical links in the overlay. Messages dedicated to peer discovery, link control and data lookup, are then exclusively sent along the overlay. As each peer is allowed to open only a limited amount of connections, establishing direct relationships with a few other peers, message forwarding is a necessary cooperative task, in order to achieve a broad coverage of the overlay. The messages lifespan is controlled by assigning bounded Time To Live (i.e., at application layer), which decrements at each logical hop.

To establish a connection, as shown in Figure 6.1, a peer P1 initiates a handshaking procedure with a peer P2, sending a request message (C-request). Peer P2 could then reply using either a connection accept (C-accept) to signal its willingness to link with P1, or a connection reject (C-reject) to signal for example that it has no more available connection slots. The handshaking ends up successfully for P1 when it receives C-accept message, and for P2 when it receives a confirm message (C-confirm) from P1. These events trigger an update of the connection tables, and both peers will then behave accordingly to their internal status. This mechanism assumes that each peer is given a boot-server to establish its entry point (or first connection) in the overlay. In real Gnutella networks, this information comes directly from the user, or through a lookup against a Gnutella Web Cache (See http://www.gnucleus.com/gwebcache/ as an example), where peers that are already part of the overlay publish their addresses.

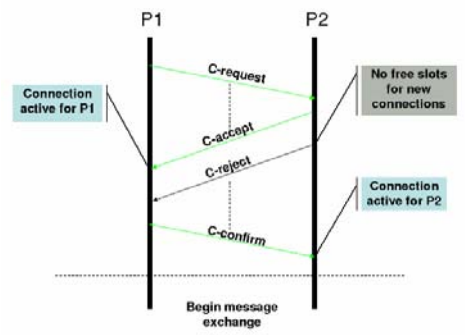


Figure 6.1. Gnutella handshaking procedure.

Gnutella peers usually store information on the state of active connections in a dedicated table, which we refer to as connection table. The size of the connection table is expected to range inside a lower and an upper bound (LB and UB). These bounds are not directly reported in the Gnutella specification [KM02], but provide a systematic way of modeling the behavior of Gnutella peers [CCR04]. In particular, they guarantee that each peer opens a minimal amount of links (at least LB), without overdoing the overall connectivity (exceeding UB), and consequently abusing of network resources. These two limits influence peers behavior, identifying three operational states:

While the table size is smaller than LB, the peer remains in a connecting state, where it i) performs peer discovery, ii) initiates connections towards discovered peers, and iii) accepts incoming connection requests from foreign peers.

As the table size reaches LB, the peer enters a connected state, where it stops doing peer discovery and does not initiate connection requests anymore. In this state it still keeps accepting incoming connection requests from remote peers, as long as its table has free slots available. Clearly, if the table size falls back down under LB, the peer transits back in state connecting.

Finally, when the table size reaches the UB, the peer enters a full state, where it also stops accepting incoming connection requests. Again, as the table size falls down under UB, the peer returns in state connected.

In each of the above states, the system performs data lookup on demand (i.e., driven by the user), and periodically probes its active connections, using one-hop probe Ping messages. A connection becomes not active if the peers are not able to probe-ping each other for more than a specified amount of time. Finally, a peer can intentionally drop active connections by issuing Bye messages.

```
AddPong(pongmsg) ⇒ add Pong to cache
PurgePongCache() ⇒ delete stale cache entries
...
HandlePing(pingmsg, connection)
  if (pingmsg.isProbePing()) then
    pong = newPongMsg(local credentials)
    pong.ttl = 1
    SendPong(connection, pong)
  else if (pingmsg.isDiscoveryPing()) then
    PurgePongCache()
    if (PongCache.size() ≥PT) then
      pongs = PongCache.SelectPongs(PT)
      for all pong in pongs do
        pong.ttl = pingmsg.hops + 1
        SendPong(connection, pong)
      end for
    else
      if (PeerTable.state ≠ FULL) then
        pong = newPongMsg(local credentials)
        pong.ttl = pingmsg.hops + 1
        SendPong(connection, pong)
      end if
      if (pingmsg.ttl > 1) then
        pingmsg.ttl − −
        pingmsg.hops + +
        for all c in PeerTable such that c ≠ connection
        do
          ForwardPing(c, pingmsg)
        end for
      end if
    end if
  end if
...
HandlePong(pongmsg)
  if (pongmsg.ttl + pongmsg.hops ≤ TTL_MAX) then
    AddPong(pongmsg)
    if (pongmsg.ttl + pongmsg.hops > 1) then
      connection = GetMsgOriginator(pongmsg.msgID)
      ForwardPong(connection, pongmsg)
    end if
  end if
```

Symbols and constants	
PT	Pong Threshold, is the minimum number of valid entries in the Pong cache that a peer should have in order to directly answer a query. The Gnutella specification suggests 10 as fair value.
FULL	This indicates the state full of the peer behavior.
TTLmax	This is the maximum TTL value that can be assigned to a message. The Gnutella specification suggests 7 as the correct value.
Objects	
Name	**Field or Method**
PingMsg or PongMsg	ttl = message Time to Live
PingMsg or PongMsg	hops = number of hops in the overlay performed by the message
PingMsg or PongMsg	msgID = unique message identifier
PingMsg	IsProbePing() = checks whether this Ping is a Probe Ping (i.e., ttl = 1 and hops = 0)
PingMsg	IsDiscoveryPing() = checks wheter this Ping is a valid discovery Ping (ttl + hops <= TTLMAX)
PeerTable	This object represents the connection table
PeerTable	state = current peer behavior

Algorithm 1. Pseudo-code for Gnutella peer discovery and pong caching.

Peer discovery, pong caching and queries. We now briefly describe how peers discover each other, pointing at Algorithm 1 for a detailed pseudo-code description. After having established their first connection, peers discover other agents by sending over it multi-hop discovery Ping messages with TTL equal to 7 [KM02]. At each hop in the overlay, discovery Pings get their TTL field decremented before being forwarded over each active connection listed by the current peer (except the one where the Ping came from). This bounds the horizon of the application layer broadcast to 7 hops, ``the edge'' where discovery Pings are discarded due to TTL expiration. Peers receiving a valid discovery Ping, reply back with a Pong message containing credentials for future connections (i.e., network address and port number). Note that this last step is executed only if the peer is not in state full. In fact, in this state the peer could not even accept incoming connection requests. Pong replies are given enough TTL so that they are able to reach the Ping originator, which can then use the embedded credentials to open new connections. Note that each Pong reply is back propagated along the overlay path of the related Ping. In fact, the originators of discovery Pings associate unique identifiers to the

messages, making possible for intermediate peers to remember where Pong replies should be forwarded.

The standard discovery procedure can be enhanced with Pong caching. On receiving Pong messages, a peer stores the embedded credentials in a local cache. Incoming Ping messages could then be directly answered if the local cache contains enough items (i.e., up to a predefined threshold), without further forwarding the discovery Ping. In this case, a certain number of items are selected from the Pong cache and returned all together in a series of replies to the originator. Otherwise, if the cache does not contain enough items, the peer performs the standard Ping forwarding procedure. This caching scheme significantly reduces the discovery overhead.

Queries are handled similarly to discovery Pings. On receiving a Query message, a peer looks up the locally shared content using the constraints contained in the Query. If one or more matches are found, the peer replies back with a Query Hit, providing pointers to local results. In any case, the peer decrements the TTL field, forwarding the Query message to its neighbors. Subsequent data downloads are carried out outside the overlay through direct file transfers.

Performance of Gnutella in mobile ad hoc environments. To better understand the capacity and limitations of Gnutella when employed in mobile ad hoc environments, we performed a set of simulations to put the platform through typical ad hoc scenarios, using the Network Simulator (ns2 version 2.27). In this chapter, we only report some results related to i) scenarios with mobile nodes moving with different patterns, and ii) scenarios where we recreated partitioning of the physical network, referring to [CGT05] for a complete analysis and a detailed discussion. In these scenarios we analyzed the overhead of the protocol as the amount of network traffic generated in a time unit, and its capacity of building the overlay, measured as the average number of per peer connections (i.e., average peer degree).

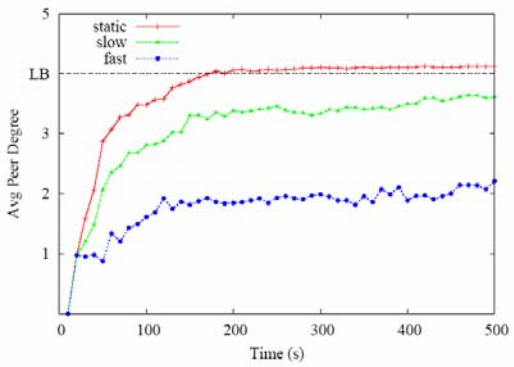


Figure 6.2. Average Gnutella peer degree under increasing nodes mobility, with the OLSR routing protocol.

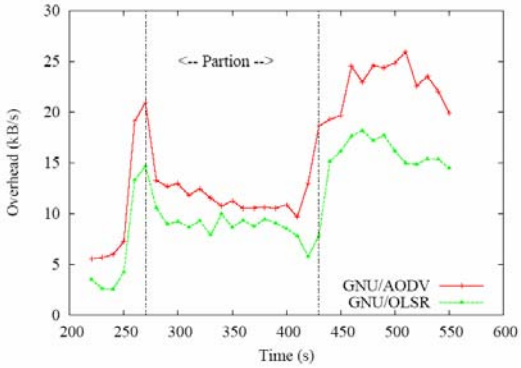


Figure 6.3. Effects of network partitioning on the overhead generated by Gnutella peers, with both OLSR and AODV.

The plots reported in Figure 6.2 show the average degree achieved by Gnutella peers under three different mobility patterns. This study highlighted that nodes mobility severely impacts on the overlay formation capacity of Gnutella. Only in static scenarios the peers were able to reach (on average) the minimum amount of connectivity (LB), fixed to 4 connections per-peer [CCR04]. When we introduced mobility patterns based on the random waypoint mobility model, the average peer degree fell down to 3.7 connections per peer in a slow configuration where nodes moved up to 5m/s, and down to 2 connections per peer in fast configuration where nodes moves up to 15m/s. These conditions resulted in high rates of overlay partitioning and consequent low rates of information discovery.

The study related to Figure 6.3 draws the attention on the reaction of Gnutella peers under network partitioning situations. These experiments were carried out for 600 seconds simulation time, leaving an overlay of 30 peers stabilize its activity in the first 200 seconds, introducing a network partitioning around time 270, and finally resuming the original network around time 430. This was achieved by placing the 30 nodes in mutual visibility on a static grid, and letting those in the center move in opposite directions so as to break the network in two halves. The plots report the reactions (in term of amount generated network traffic [kB/s]) of Gnutella peers in correspondence of the network partitioning and rejoining events (i.e., see vertical lines). We could observe bursts of networking activity in correspondence of the beginning and termination of the network partitioning: up to 300% of traffic increase in the former, and up to 200% in the latter case. The straightforward explanation for this behavior is that peers transit back in state connecting while the network breaks up in two halves, and perform even broader discovery procedure when the connectivity is restored at the physical layer. The plots also show that Gnutella behavior is independent from which routing algorithm (i.e., OLSR or AODV) is in use at network layer. However, the implementation of OLSR used in these simulations provided routes of better quality: the overhead generated by Gnutella on top of AODV results bigger of a factor ranging between 10% and 20%.

From these simulations, we could verify that although Gnutella meets important requirements for the management of data-sharing overlay network, it was not designed for ad hoc networks, and suffers from node mobility, causing peers not to achieve minimum connectivity requirements. Moreover, when put through network partitioning, the protocol generates traffic bursts in correspondence of topological reconfigurations. This is clearly not desirable in mobile ad hoc situations where the network partitions frequently, or where groups of nodes enter and leave the network.

A cross-layer optimization of Gnutella. Exploiting the cross-layer interface presented in Deliverable D13, we realized an event-based interaction between Gnutella peers and OLSR agents at the network layer. The main re-design addressed the Gnutella peer discovery procedure, replacing PING flooding with peer advertisements - also referred to as Optional Information (OI) - spread around in conjunction with routing information. We introduced two classes of cross-layer events:

1. *SpreadOI* events, to which the routing agent subscribes, receiving notifications from the Gnutella platform. These events are used to ask the OLSR agent to advertise local peer credentials around, together with the next Hello or Topology Control message, respectively if the node is a Multi Point Relay or a leaf node.
2. *RecvOI* events, to which the local Gnutella peer subscribes, in order to receive notifications from the underlying OLSR agent. These events are used to notify the local peer about the credential advertisement of a remote peer, received together with a routing control message.

This cross-layer framework allows each peer to periodically advertise its credentials, additionally enabling reaction to events like the reception of new advertisements signaling the discovery of a new peer, and the failure of peers due expiration of old advertisements.

On receiving cross-layer events, peers fill up a local table of advertisement generated by foreign agents. This advertisement table took the place of the Pong cache of the legacy implementation. Moreover, as advertisements travel the network along with routing control packets, we were able to get each time an accurate estimation of the physical distance (in number of hops) of the peer originating the advertisement. This topological information enriched the advertisement table, and allowed us to play a smarter overlay formation protocol, introducing a link selection policy based on the physical distance of discovered peers. In other words peers were able to prioritize the establishment of closer connections over further ones, with the goal of building an overlay network topologically closer to the physical network. Note that this deterministic selection of overlay links is meaningful in ad hoc environments, where one could eventually assume a small number of opportunistic (and hence heterogeneous) participants to the Gnutella network. Additionally, the network participants get reshuffled by mobility and by the arrival of new nodes. The same rational would not apply on the Internet, where overlay formation with random walks has been proved more effective than deterministic approaches. Apart from the new peer discovery approach, we left unaltered the rest of the Gnutella protocol (i.e., connection handshaking, link probing and queries), as well as the states identified by the peer degree and the LB and UB bounds.

We modified our Gnutella peers to react on three kinds of situation, in correspondence of the updating of the advertisement table. On receiving an advertisement from a new peer, the agent always attempts a connection request if it is in state connecting. If the current state is connected, the agent checks the physical distance of the new peer. In case this is closer than any of the already connected peers, the agent attempts a connection request to it. The same procedure triggers in state full, but in this case the peer additionally drops the furthest connection, once the new one becomes active. Similar steps are performed for an advertisement refresh, because with nodes mobility, a peer can receive subsequent advertisements from the same foreign agent at different distances. The third situation is the expiration of an advertisement due to consecutive miss of refreshes. This event is detected internally by the advertisement table, which then directly notifies the peer, causing a connection drop in case the Gnutella protocol didn't detect it using probe Pings.

The cross layer approach eliminates the need of bootstrap servers, reducing and stabilizing the overhead of the protocol, and finally improving the connectivity and the quality of the generated overlays. The next section proves the validity of these claims,

evaluating the cross-layer version of Gnutella (XL-Gnutella). XL-Gnutella peers issue credential advertisements every 30 seconds.

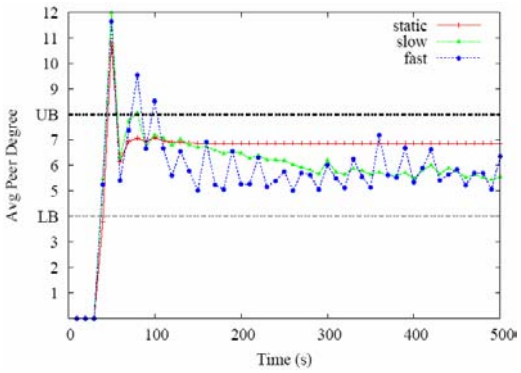


Figure 6.4. Average XL-Gnutella peer degree under increasing nodes mobility.

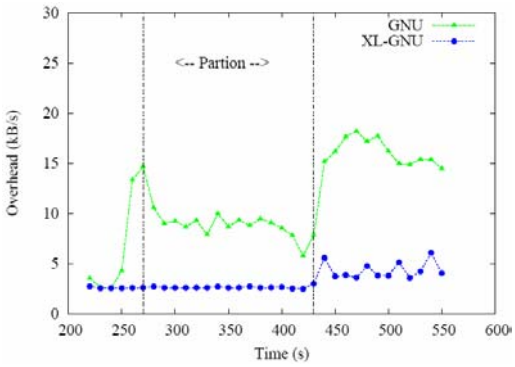


Figure 6.5 Comparison of the overhead generated by Gnutella and XL-Gnutella under network partitioning.

As shown in Figure 6.4, the behavior of the average peer degree which falls down from 7 to 5-6 connections per peer as the mobility increases, but remains in the [LB - UB] range. However, fast node mobility induces frenetic oscillations of the average degree. By stressing XL-Gnutella with the network partitioning scenarios, we obtained again interesting results: cross-layer peers loose connectivity when the network splits in two halves, but they timely recover the original level when links are re-established at the network layer. Additionally, this happens with no traffic bursts in correspondence of topological reconfigurations, which is not the case for the legacy protocol (as shown in Figure 6.5).

In the last set of experiments, we wanted to prove the effectiveness of our topology-aware link selection policy. To this end, we studied the *path stretch* generated by XL-Gnutella and Gnutella overlays. The path stretch is defined as the ratio of the end-to-end delay (measured in number of hops in the physical network) along the path connecting two peers in the overlay, to that along the direct unicast path in the physical network. The path stretch measures how far (from a topological point of view) the overlay is from the physical network, and characterizes the overhead induced by the former on the latter. By definition, the direct unicast between two nodes in the physical network has a path stretch of unity. The closer the path stretch of a p2p platform to unity, the better. In order to perform this analysis, we turned on periodic query issues with TTL equal to 7, to get multi-hop overlay messages also for XL-Gnutella.

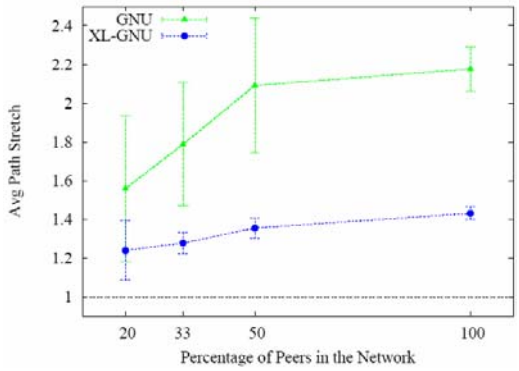


Figure 6.6. Comparison of the average path stretch produced by Gnutella and XL-Gnutella under increasing peer densities.

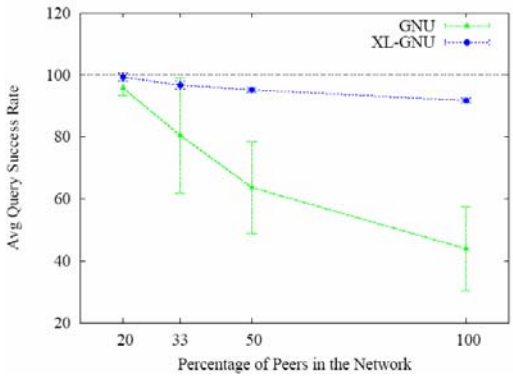


Figure 6.7. Comparison of the average query success rate produced by Gnutella and XL-Gnutella under increasing peer densities.

Peer density	Gnutella	XL-Gnutella
20%	8.5%	0%
33.3%	25%	0%
50%	29.4%	0%
100%	62%	0.1%

Table 1. Comparison of Gnutella and XL-Gnutella average overlay partitioning rate under increasing peer densities.

We prepared scenarios with 40 static nodes uniformly distributed on a rectangular area, configuring an increasing percentage of them as peers (respectively 20%, 33%, 50% and 100%). Figure 6.6 shows the obtained average path stretch with 95% confidence interval. Not only XL-Gnutella produces overlays significantly closer to the underlying networks when compared to the legacy Gnutella (e.g., respectively 1.35 against 2.1 with a 50% of peers), but the cross-layer protocol exhibits a more stable behavior with smaller variances. Moreover, by looking at the average overlay partitioning rates (see Table 1), it was important to notice that with a one-to-one nodes/peers correspondence, 62% of Gnutella peers weren't able to reach each other in the overlay, compared with the 0.1% for XL-Gnutella on the same density. Finally, the numbers shown for overlay partitioning were directly verifiable in a simple experiment on the query success rate. We used the same simulation scenarios as for the path stretch, but additionally distributed shared content on the peers, before making them issue queries on it. This allowed peers to reply with hit messages when reached by query constraints matching their local shared content. We loaded a different file name on each peer, forcing it to issue one query for each file shared by the other peers. The results are reported in Figure 6.7, where the Gnutella query success rate clearly degrades accordingly to the overlay partitioning rate: from nearly 95% with 20% peer density (and 8.5% of partitioning), down to 43% with 100% peer density (and 62% partitioning). In the same Figure, the 95% confidence interval highlights once again the stability exhibited by XL-Gnutella in satisfying queries on existing content.

6.1.2. The Pastry protocol

The overlay network defined by Pastry [DR01] is represented by a large circular space of 2^{128}-1 logical identifiers, also called ring overlay. Each Pastry node chooses a 128-bit identifier (nodeId), which represents a logical position in the ring. The nodeId is calculated at join time, when the node hashes one of its physical identifier (e.g., IP address, hostname, public key etc.) through a strong hash function H. The function H uniformly distributes inputs in the ring space, and guarantees little chances of mapping two different physical identifiers on the same nodeId, and great chances of scattering nodes with closer nodeIds far apart in the ring.

Subject-based message routing. The fundamental service offered by a Pastry ring, is that peers exchange messages through a subject-based mechanism. The idea is to associate a logical subject (or a key) to an application layer message, and route it hop by hop in the ring, until it arrives at a peer with a nodeId that results the closest to the message's subject. This final peer is considered the root for the message and is responsible to handle the message content at the application layer. The association between messages and subjects happens through the same hash function H used for mapping nodes to logical addresses, which guarantees the same aforementioned distribution properties. To give an example, consider a file sharing application where each file is represented by its name. The typical interaction with Pastry would be to create two types of messages, one to publish in the ring the sharing of a file associated to a given name, and another to lookup the node (or the nodes) sharing a file with a given name. In this scenario publish and lookup messages get routed through the same logical identifiers, and the corresponding root peers associate them at application layer.

The subject-based routing policy used by Pastry is based on a numerical proximity metric between message subjects and nodeIds. From the algorithmic standpoint, consider logical identifiers to be represented as a sequence of digits with base 2^b, where the parameter b is defined a priori. At each step of a routing procedure, a Pastry peer P forwards the message with subject K to a peer Q whose nodeId shares with K a prefix that is at least one digit (b bits) longer than the prefix shared with P. If no such node is known, P tries to forward the message to a peer L that has the same common prefix with K, but is numerically closer to K with respect to P (this can be easily identified by looking at the digit after the common prefix). The expected maximum number of hops in the overlay between a source and a destination peers is equal to $\lceil \log_{2^b} N \rceil$ in an overlay of N nodes [DR01].

State representation. To support this routing procedure, each peer maintains information related to other peers in the overlay, using the following data structures:

- *Routing table*. This structure is organized into $\log_{2^b} N$ rows with 2^{b-1} entries each. Each entry at row n of the routing table refers to a node whose nodeId shares with the local nodeId the first n digits, but whose $n + 1^{th}$ digit differs (it has the same value of the column index). If there are no nodeIds with this characteristic, the entry

is left empty. In practice, a destination node is chosen, between those known by the local node, based on the proximity of its logical identifier to the value of the key. This choice provides good locality properties, but only in the logical space. In fact nodes that are logically neighbors, have a high probability to be physically distant. In addition, the choice of the parameter b determines a trade-off between the size of this data structure and the maximum number of hops in subject-based routing procedures, that is expected to be equal to $\lceil \log_2{}^b N \rceil$, and simulations results confirmed this [DR01].

- *Neighborhood set*. This structure represents the set of nodes that are physically close to the local node. The neighborhood set is not normally used in routing messages, but could be useful for maintaining physical locality properties among nodes.
- *Leaf set*. This structure represents the set of nodes with the closest logical identifiers to the current node. The leaf set is centered on the local node P, with half of the identifiers larger than P, and the other half smaller than P. The leaf set represents the perfect knowledge that each peer has of its logical contour.

In routing a given message, the node first checks to see if the related subject falls within the range of nodeIds covered by its leaf set. If so, the message is directly forwarded to the destination node, namely the leaf set entry whose nodeId is logically closest to the message subject. If the subject is not covered by the leaf set, then the routing table is used, and the message is forwarded to a node that shares a common prefix at least one digit longer than the local nodeId. Sometimes, it is possible that the appropriate entry in the routing table is empty, or that the associated node is currently disconnected from the network, but the overlay is still not updated; in this case the message is forwarded to a node (if any exists) that shares the same prefix as the local node, but is numerically closer to subject. Each Pastry data structure entry maintains a correspondence between the logical identifier of each node and its credentials (IP address and port number), in order to allow the establishment of direct peer connections driven by application needs.

State management. The main procedures used by Pastry to establish and maintain the overlay network (i.e., the previously presented data structures), consists of join and disjoin operations. First of all, when a new node, say X, decides to join the overlay, it needs to initialize its internal data structures, and then informs other nodes of its presence. It is assumed that the new node knows at least one of its physical neighbors, say A, which already takes part to the overlay. Typically, this bootstrap node can be located automatically, for instance by sending "expanding ring" queries, or be obtained by the system administrator through outside channels. Node X then asks A to route a special "join" message with the key equal to X. Like any message, Pastry routes the join message to the existing node Z whose id is numerically closest to X, passing through some intermediate nodes. In response to the "join" request, nodes A, Z, and all the intermediate peers, send the content of their tables to X. At this point, the new node X processes the received information, and initializes its own structures in the following way:

The neighborhood set is initialized with the contents of that of node A, since it is a physical neighbor of X.

The leaf set is initialized with that of node Z, which has the closest existing nodeId to X.

The i^{th} row of the routing table is initialized with the corresponding row of the routing table of the i^{th} node (B_i) encountered in the routing path from A to Z (as it shares a prefix of length i with X).
At the end, X informs all the newly known nodes about its arrival, transmitting a copy of its resulting state. This procedure ensures that X initializes its state with appropriate values, and that the state in all other involved nodes is updated.

The management of departure nodes is another important feature of Pastry. In [DR01] it is assumed that Pastry nodes may fail or depart without warning. In particular, a node is considered failed when its logical neighbors can no longer communicate with it. To this aim, nodes in the leaf set are periodically probed with UDP ping messages. Leaf entries that do not reply to probe pings are considered failed, and get replaced by entries of the leaf set relative to the live node with the largest index on the side of the failed node. In this way each node can easily repair its leaf set, and the delay with which it becomes aware of logical neighbors failure depends on the probing frequency. A similar probing mechanism is used to maintain a consistent neighbor set. Instead, a node realizes that an entry in its routing table is failed, only when it attempts to connect to it to forward an application message. This event does not normally delay message routing, as another destination node could be selected. Anyway, the failed routing table entry has to be replaced. To this end, the peer contacts the entries belonging to the same row of the failed one, asking for a nodeId that can replace it. If none of them has a pointer to a live node with the appropriate prefix, the local node has to contact nodes belonging to the successive row of the routing table. In this way, many remote connections could be required to manage single entries of the routing table.

The maintenance procedures explained above, highlight the complexity related to the management of a structured overlay network. The many remote connections needed to check the validity of table entries, considerably increase the overhead introduced on the underlying network. In addition, peers that are considered failed have no way to get back in the ring apart from performing once again a join procedure. For example, if a node temporary looses its network connection, it has to reboot the system and join the overlay again. This limitation represents a major problem in mobile ad hoc networks, where frequent topology reconfigurations could cause situations of intermittent connectivity, which clash with the low tolerance offered by Pastry maintenance procedures.

To better understand the overhead introduced by ring-maintenance operations in Pastry, we analyzed its behavior in a small real test-bed. Specifically, we evaluated an open source implementation of Pastry called FreePastry [FreeP]. During our study, we focused on both reactive and proactive solutions for the routing protocol at the network layer, using AODV in the former case and OLSR in the latter. A complete analysis of the results obtained from our real testbed has been presented in [BCDP05].
For our study, we set up a real ad hoc network of 8 nodes inside the CNR campus in Pisa, where the structural characteristics of the building and the nearby presence of access points and measurement instrumentations, limit the transmission capabilities of nodes. These force the establishment of a multi-hop ad hoc network. The network topology

configuration used for the experiments related to FreePastry is shown in Figure 6.8, where all nodes ran one of the routing protocols while only 6 ran a distributed application on top of FreePastry. In particular, nodes B and G just worked as routers, allowing the packet forwarding from the source to the destination through the optimal path.

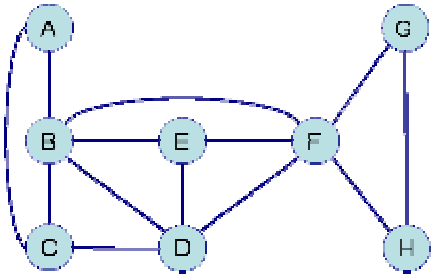


Figure 6.8 Experimental network topology

From experimental results we noticed that, due to the limited number of peers participating to the FreePastry ring, the overlay data structures maintain logical identifiers of almost all peers. For this reason, there were rare generations of multi-hop subject-based routing procedures. However, operations needed to create and maintain the overlay, introduced high overheads on the underlying ad hoc networks, causing errors depending on the used routing protocol. In fact, FreePastry implements maintenance operations using both UDP and TCP connections to remote peers, introducing further overhead on the network as overlay relationships do not take into account neighborhood information.

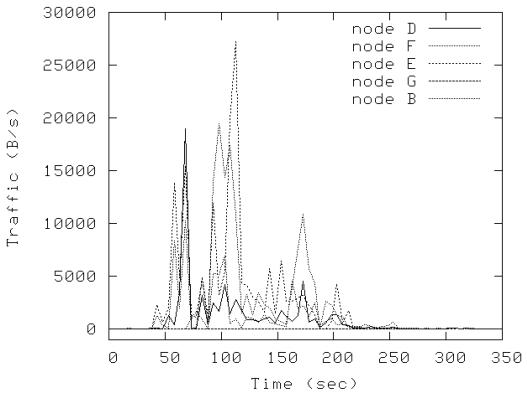


Figure 6.9. Pastry on OLSR: traffic related to main nodes

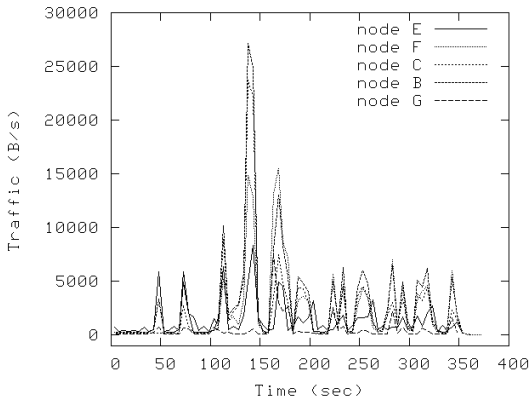


Figure 6.10. Pastry on AODV: traffic related to main nodes

In order to obtain a performance evaluation of Pastry, we numerically calculated the traffic generated to maintain the overlay, in addition to the one produced by the routing protocol at the network layer. Specifically, Figure 6.9 and 10 show the amount of traffic observed by some nodes of the network, respectively running OLSR and AODV. In both cases the amount of traffic generated by node B and G, which just worked as routers, was

negligible. In fact, also the proactive protocol does not introduce a high overhead on the network, encapsulating different routing messages (Hello, and Topology Control) into single routing packets. By configuring an incremental ring formation, UDP and TCP connections used by FreePastry peers to collect initial overlay information, generates evident traffic peaks. Even if the total amount of traffic is not so high related to the WiFi bandwidth, those peaks can negatively influence additional transmission of application data. Furthermore, we could observe that the successful creation of the overlay depends on the reliability of the path discovery process implemented by the routing protocol. In the case of AODV, a high number of TCP retransmissions and connection failures occurred, mainly due to the delay introduced to discover a route toward a destination, and to the use of unidirectional links as valid routes.

To summarize, experimentations carried out with FreePastry, showed the generation of heavy overheads related to overlay management procedures, due to a high number of remote connections opened between peers. The lack of attention toward the usage of network resources greatly reduces the overall system performances in ad hoc networks. In order to improve the system, Pastry nodes should become aware of the underlying network topology, maintaining a tight correspondence between physical and logical address space. The work in [CDH] presents a reorganization of Pastry's overlay exploiting network proximity information, in order to improve application performances and network usage. This solution is based on additional location discovery protocol that estimates physical distances between nodes, showing the potential improvement introduced by proximity information at the expense of running another protocol in conjunction to the p2p platform. The last section of this chapter shows an alternative solution based on a full cross-layer protocol stack architecture, which allowed us to greatly optimize ring management procedures by exploiting interactions between the p2p platform and routing protocols at the network layer.

Performance of Pastry in mobile ad hoc environments. As in the case of Gnutella, we performed ns2 simulations to understand the capacity of the platform in typical ad hoc scenarios. By putting Pastry through the same mobility scenarios used for Gnutella, we studied the capacity of building the overlay, measured as the average number of entries in Pastry routing tables, and the rate of unsuccessful subject-based message routing procedure. For the latter we re-created a simple subject distribution model, in which each Pastry peer was configured to be root for one subject, and performed a route procedure toward the subject maintained by other peers. We considered successful a route procedure for a subject k, if it terminated at the peer responsible for k.

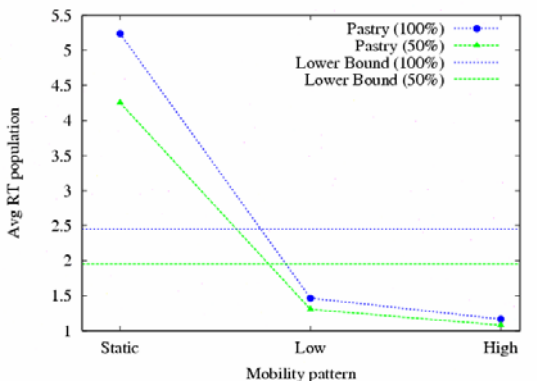


Figure 6.11. Average number of Pastry routing table entries collected by peers under pattern of increasing nodes mobility.

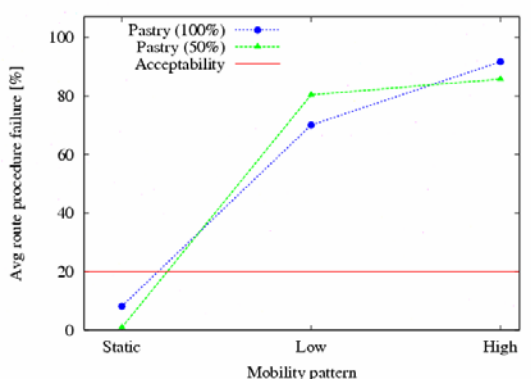


Figure 6.12. Effects of patterns of increasing nodes mobility on the rate of unsuccessful subject-based routing procedures.

In our simulation models, we configured a network of 30 mobile nodes, and then varied the density of Pastry peers to be exactly 100% and 50% of the network size (respectively 30 and 15 Pastry peers). The plots reported in Figure 6.11 show the average number of routing table entries collected by Pastry peers under patterns of increasing nodes mobility. The Pastry protocol [DR01] defines a logarithmic lower bound (i.e., $\log_2^b(n)$ where n is the size of the overlay) on the number of entries that each peer should collect in internal tables, in order to guarantee high rates of successful subject-based routing procedures (the original Pastry paper [DR01] reports a maximum of about 5% failed routing procedure as acceptable values). The horizontal lines in Figure 6.11 report the lower bounds relative to the two overlay sizes (respectively $\log_4(30)$ and $\log_4(15)$). As already observed in the case of Gnutella, nodes mobility has a big negative impact on the protocol capacity of building the overlay. The scenarios with mobile nodes (in both slow and fast configurations) determined a sharp reduction in the amount of ring overlay knowledge that each Pastry peer was able to collect, respectively 1.5 for the slow scenario and 1.2 for the fast scenario, in the case of a one to one nodes-peers correspondence.

The above results were confirmed by a study on the failure rate of subject-based routing procedures. As applications based on structured platforms focus on availability, subject-based routing procedures should maintain very low failure rates, for example below 20% in the case of mobile ad hoc environments. Medium or higher failure rates determine the non-usability of platform. The results reported in Figure 6.12 show that the performance of a straightforward implementation of the Pastry protocol quickly degrades under patterns of increasing nodes mobility: from around 5-10% in the case of static network, to 70-80% and more in the case of slow and fast mobility patterns.

6.1.3. CrossROAD: a Cross-layer Ring Overlay for AD hoc Networks

From the results illustrated in the previous sections, we can conclude that the two analyzed p2p platforms have significant performance degradation when operating over an ad hoc network. Specifically, the simulation results indicate that, only in static configurations, the protocols we analyzed are able to construct overlays that meet necessary quality criteria, and guarantee good performance. Furthermore, experiments with real testbeds indicate that, also in static configurations, while the FreePastry implementation correctly operates on top of our multi hop ad hoc network, severe problems have been identified from the performance standpoint. These are mainly due to several factors that affect a MANET behavior in a real environment. IEEE 802.11 operates in the ISM spectrum and hence experienced a lot of noise from external sources. The quality of the wireless links is therefore highly variable and this affects the higher layer protocols (e.g., routing, forwarding and transport) behavior. At the end, this affects the overlay construction and management. For example, as FreePastry maintenance operations are also based on TCP services, the platform performance was negatively affected by poor TCP performances. In addition, as Pastry operates its own subject-based routing ring independently from the underlying ad hoc network, it introduces a significant and bursty overhead on the ad hoc network.

Both simulation results and experiences with a real implementation provided us with some indications for solving the performance problems in MANET environments. Specifically, results related to FreePastry indicate that significant performance benefits can be expected by exploiting routing information (extended with services information) at the middleware layer. This allows realizing ring overlay maintenance operations avoiding the big overheads connected with implementing it via middleware operations. Similar indications have been obtained for Gnutella [CGT05].

To verify the effectiveness cross-layering also from an experimental standpoint, we have designed and implemented CrossROAD (Cross-layer Ring Overlay for AD hoc networks) [D05, D13], a ring overlay platform with Pastry-like semantic, optimized through cross-layering. In order to compare and contrast the performances of CrossROAD with those of Pastry, we used a simple Distributed Messaging (DM) application traffic generator. Nodes running DM set up and maintain an overlay network related to this service. Once a node has created/joined the overlay, the application provides the possibility to create/delete one or more mailboxes and send/receive short messages. As mailboxes get distributed on the ring overlay, their physical location is randomly selected applying the hash function to the associated identifier, which is also used to address the related messages. To perform an experimental comparison between CrossROAD and FreePastry, we considered the same 8-node network shown in Figure 6.8. In the first set of experiments, we measured the overhead introduced by the two platforms. Figure 6.13 shows the overhead produced by CrossROAD to maintain the overlay, while the results related to FreePastry are those reported in Figures 6.9 and 6.10.

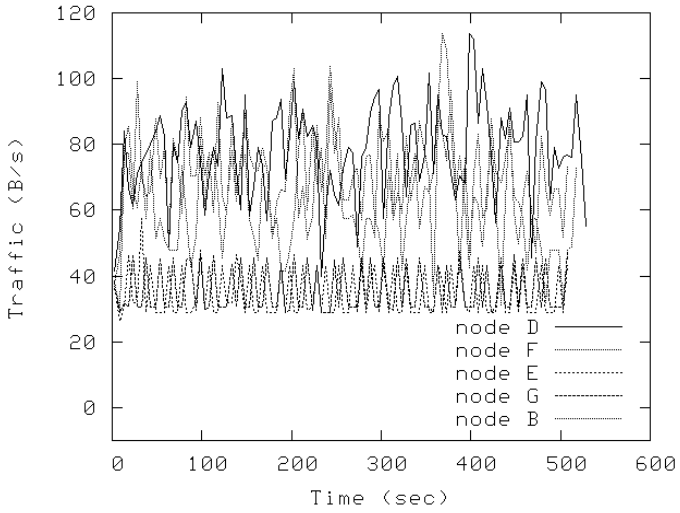


Figure 6.13. CrossROAD overhead related to main nodes.

The figure points out that the overhead traffic is for all nodes less than 100B/sec, which is much lower than that observed with FreePastry. In addition, traffic peaks introduced by FreePastry, corresponding to TCP and UDP connections used to initialize and maintain the overlay data structures, completely disappear in CrossROAD. Another important feature of CrossROAD is represented by the timeliness with which every node becomes aware of the other participants (see [BCDG05]). This is an important property to guarantee a correct behavior of the overlay when network partitioning and rejoining occur.

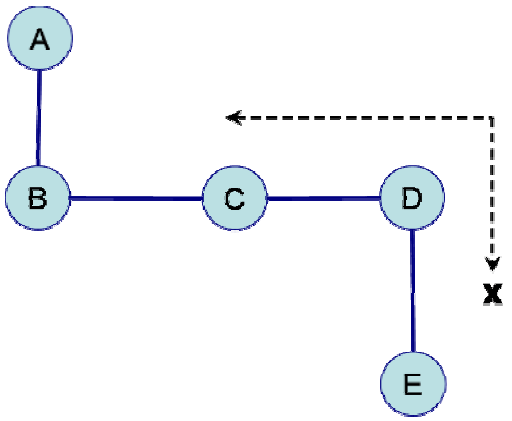


Figure 6.14. Network partitioning scenario.

To highlight this, we analyzed a possible network partitioning and the consequent reaction of CrossROAD in the overlay management. To do this, a new network topology, shown in Figure 6.14, has been set up. The network consists of 5 nodes, and only nodes in adjacent positions are in the transmission range of each other. All nodes are also CrossROAD peers. When all nodes are correctly connected to the overlay, node C starts

periodically sending an application message with a specified key value (period equal to 1sec). Initially, the key value results to be logically closest to the node identifier of B, hence node C sends those messages directly to B. Then after about 30sec, node C starts moving towards position X with a speed of about 1m/sec, generating a network partitioning: nodes A and B create an independent ad hoc network as well as nodes C, D, and E. Since the direct link B-C is lost, the cross-layer interaction between CrossROAD and the routing protocol allows node C to become aware of the network partitioning and the consequent removal of nodes A and B from the overlay. Hence, the successive messages sent by node C on the overlay with the same key value, are directly sent to the new best destination: node D. After 2 minutes, node C starts coming back to the initial position re-establishing a single ad hoc network. At this point, the following messages are sent again to node B. As shown in Figure 6.15, CrossROAD, correctly manages data distribution in case of overlay and network partitioning. Specifically, the figure shows that i) during the first phase (a single ad hoc network), node C data are stored on node B; ii) when the partition occurs, after a transient, node C data are delivered to node D; and iii) when the network is again connected, C data are again stored on node B.

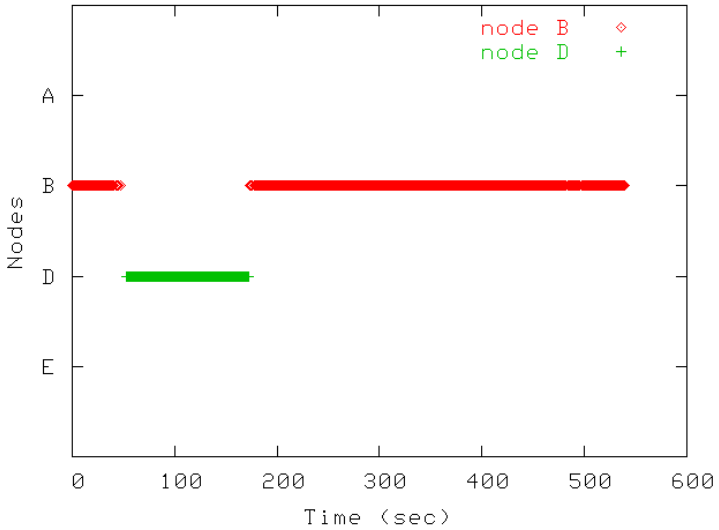


Figure 6.15. CrossROAD data distribution during network partitioning.

6.2. Summary and Conclusions

Ad hoc networks are distributed systems composed of self-organized wireless nodes. As these systems cannot benefit from any centralized infrastructure, networking functionalities, like packet forwarding, routing and network management, as well as application services, should be distributed among user devices. The distributed nature of ad hoc networking finds in the peer-to-peer (p2p) interaction its natural model of computation.

Recently, several self-organizing overlay platforms have been proposed for building decentralized and distributed applications for the Internet. The variety of applications and services realizable on top of these overlays suites also ad hoc scenarios. Thus, having them working in ad hoc environments would be an advantage for the MANET technology. However, it is not clear how these overlays should be ported, and how they will perform on ad hoc networks.
In this chapter we focused both on structured and unstructured p2p platforms, investigating their effectiveness when operating on top of a mobile ad hoc network. Specifically, we investigated via simulation the performance of Pastry and Gnutella, as representatives of the structured and the unstructured classes. Our results indicated that only in static scenarios these platforms are able to construct effective overlays. Furthermore, measurements on the performance of FreePastry (an open source implementation of Pastry) on a real testbed highlighted that also in static scenarios overlay management procedures may introduce a significant overhead on ad hoc networks.
To summarize, our results indicated that in MANET environments, implementing an overly that is completely independent from the network physical topology results in poor system performance. To avoid this, we used the MobileMAN cross-layer architecture to re-design key components of two well-known data-sharing platforms, letting them usable in mobile ad hoc scenarios.

6.3. References

[ADGS02]	E. Anceaume, A. K. Datta, M. Gradinariu, and G. Simon "Publish/subscribe scheme for mobile networks", Proc. ACM Workshop On Principles Of Mobile Computing 2002, pp. 74 – 81.
[BCDP05]	E. Borgia, M. Conti, F. Delmastro, and L. Pelusi, "Lessons from an Ad hoc Network Test-bed: Middleware and Routing issues", Ad Hoc & Sensor Wireless Networks An International Journal, Vol. 1 N. 1-2, 2005.
[BCDG05]	E. Borgia, M. Conti, F. Delmastro, and E. Gregori, "Experimental comparison of Routing and Middleware solutions for Mobile Ad Hoc Networks: Legacy vs. Cross-Layer approach", in Proc. of the Workshop on Experimental Approaches to Wireless Network Design and Analysis (in conjunction with SIGCOMM 2005), Philadelphia, PA, USA, Aug 2005.
[BCM05]	P. Bellavista, A. Corradi, and E. Magistretti "Lightweight Replication Middleware for Data and Service Components in Dense MANETs", Proc. 6th IEEE Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM 2005), Taormina, June 13-16, 2005.
[CCL03]	I. Chlamtac, M. Conti, and J. Liu, "Mobile Ad hoc Networking: Imperatives and Challenges", Ad Hoc Networks Journal, Vol.1 N.1 January-February-March, 2003.
[CCR04]	M. Castro, M. Costa, and A. Rowstron, "Peer-to-peer overlays: structured, unstructured, or both?", Technical report, 2004. Microsoft Research, Cambridge, Technical Report MSR-TR-2004-73.
[CDH]	M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, "Exploiting Network

	proximity in peer-to-peer overlay networks", Technical Report available at http://freepastry.rice.edu/PAST.
[CGT05]	M. Conti, E. Gregori, and G. Turi, "A Cross-Layer Optimization of Gnutella for Mobile Ad hoc Networks", Proceedings of the 6th ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2005), Urbana-Champaign, IL, USA, May 2005.
[D05]	F. Delmastro, "From Pastry to CrossROAD: Cross-layer ring Overlay for AD hoc networks", in Proc.of Workshop of Mobile Peer-to-Peer 2005 in conjunction with IEEE PerCom 2005, Kauai Island, Hawaii, March 2005.
[D13]	MobileMAN Deliverable 13, http://cnd.iit.cnr.it/mobileMAN.
[DR01]	P. Druschel and A. Rowston, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", in IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, November 2001.
[FreeP]	FreePastry, www.cs.rice.edu/CS/Systems/Pastry/FreePastry.
[H03]	Klaus Hermann, "MESHMdl - A Middleware for Self-Organization in Ad hoc Networks", Proc. IEEE Workshop on Mobile and Distributed Computing (MDC 2003) in conjunction with ICDCS 2003, 19 May 2003.
[KaZaa]	http://www.kazaa.com.
[KM02]	T. Klinberg and R. Manfredi, "Gnutella Protocol Specification v0.6". http://rfc-nutella.sourceforge.net/src/rfc-0 6-draft.html, June 2002.
[MCZE02]	C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich, "XMIDDLE: A Data-Sharing Middleware for Mobile Computing," Wireless Personal Communications, vol. 21, pp. 77–103, 2002.
[MPR01]	A. L. Murphy, G. P. Picco, and G.-C. Roman, "Lime: A middleware for physical and logical mobility", in Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21), Phoenix, AZ, USA, pp. 524–233, April 16-19, 2001.
[PC02]	I. Pratt and G. Crowcroft, "Peer-to-Peer systems: Architectures and Performance", Networking 2002 tutorial session, Pisa, Italy, May 2002.
[RFHKS01]	S. Ratsanami, P.Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network", in Proc. SIGCOMM, San Diego, CA, Aug. 2001, pp. 161-172.
[SGF02]	R. Schollmeier, I. Gruber, and M. Finkenzeller, "Routing in Mobile Ad Hoc and Peer-to-Peer Networks. A Comparison.", in Proc. of Networking 2002 Workshops, Pisa, Italy, May 2002.
[SMKK01]	I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications", in Proc. SIGCOMM, San Diego, CA, Aug. 2001, pp. 149-160.

7. APPLICATIONS

7.1. UDDI4m Experimental Evaluation

In this section we present an evaluation of the UDDI4m implementation. This evaluation focuses on two features:

1. the load balancing in data sharing among the nodes of the network;
2. the amount of resource used when the application runs.

The first evaluation analyzes the load balancing of data distributed on the peer nodes that form the UDDI4m overlay network. The second evaluation is aimed to understanding if our application can run on resources constrained devices, or in other words, which are the requirements for running our application. Indeed, the devices that can form an ad hoc network are heterogeneous: laptops, PDAs or mobile phones etc., and the amount of their resources, i.e. memory and CPU, are highly variable.
The evaluation presented hereafter was performed in an ad hoc environment to have realistic results.
In this section we first provide a brief presentation of the UDDI4m implementation model, and then we describe the experimental environment set-up. A discussion on resource constraints to run our application concludes the section.

7.1.1. Implementation model

The UDDI4m mechanism is implemented by tuning the UDDI standard [UD02] for ad hoc environments. To know more details regarding the UDDI4m implementation, please refer to Deliverables D13 and [UM05].
In the Figure 7.1 is shown the interaction among the layer middleware and new service layer. The UDDI4m software architecture is fully modular. The modules that are been implemented in the UDDI4m mechanism are:

- The *UDDI4m client* generates requests to the *UDDI4m server* module, using the ID-Number, connects to the (local or remote) *UDDI4m server* to publish or recovery contents.
- The *UDDI4m server* side is divided in two part: one block (*UDDI4m Manager*) that implements and provides the API to publish the information on databases (*publishing API*) and to retrieve the information from databases (*inquiry API*), these API are used from *UDDI4m_Service* module that is the core of the UDDI4m service because implements the methods of the service: *publishService*, *findService*, *updateService* and *deleteService*; a block (*DB Management*) that implements the data structure.
- *UBR4m Table* module manages the database, translating the client requests in queries to the database. This module is optional, so it cannot be installed on all

nodes of the network, e.g. devices with resource constraints, like IPAQs, cannot manage the *UBR4m Table*, therefore if these devices want to publish their services they must publish them on other nodes.

- The *UDDI4m Message* module implements the messages exchanged among nodes in the network by middleware layer.

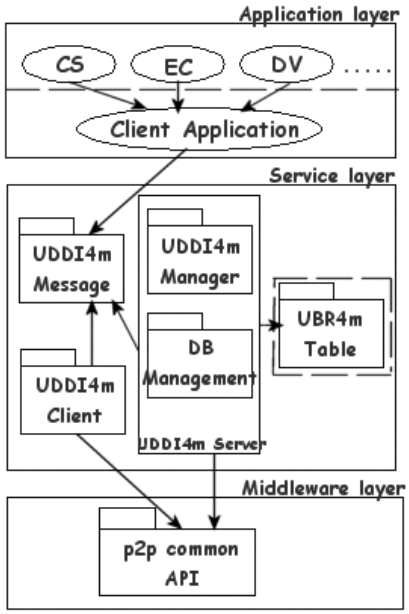


Figure 7.1. UDDI4m software architecture

The functionalities of the UDDI4m mechanism that have been implemented during the experiments are: the publication and recovery of a service.
When we run the web application, we open a web server (Mozilla) to connect to the local home page where it is possible choosing between service publication, and discovery (see Figure 7.2 for the code flows). The messages, generated during these phases are sent to the middleware and the data are stored in the registry of nodes that are reachable in the ad hoc network.

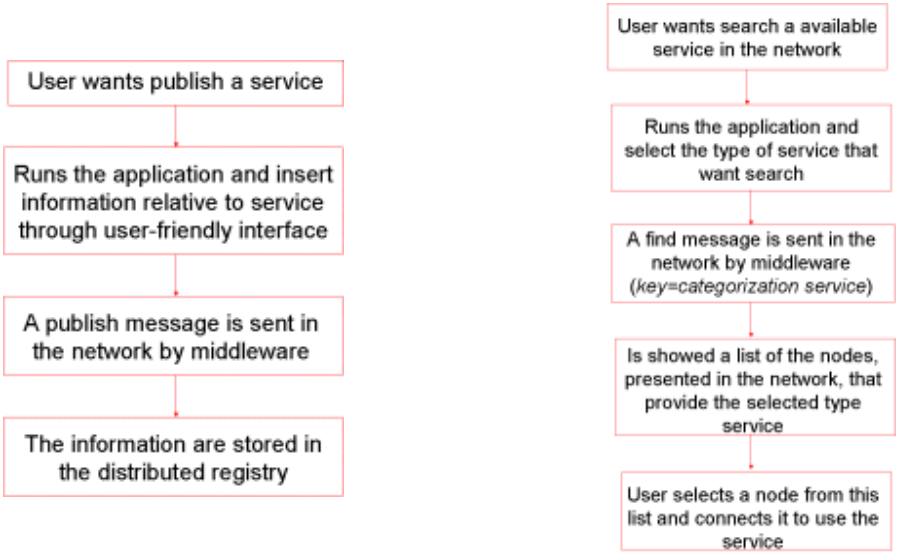


Figure 7.2. code flows during a publication and find of a service, respectively

7.1.2. Environment set-up

We considered different scenarios for the two evaluations.

Data Load Balancing
To evaluate how data are stored in the UBR4m registry of the nodes in the network, we are considered a scenario with 6 peer nodes running the application, and 5 service categories. We assumed that the services, available in the ad hoc network, can be classified in the following service categories: Content Sharing (CS); chat (chat), Video Conference (VC), Device (DV), E-Commerce (EC).

We used 8 laptops (IBM R50 centrino 1.6GHz) with integrated card wireless PRO 2200 b/g, 2 laptops were routers while the others 6 were running the application; all laptops communicated by a single hop. Each laptop published 2 different service categorizations; in Table 7.2 we report the services published by each laptop.

Node number	*First publication*	*Second Publication*
1	CS	VC
2	CS	chat
3	VC	DV
4	CS	EC
5	DV	EC
6	CS	DV

Table 7.2 service categories per node

Table 7.3 shows the number of publication for each service category.

Service Categorization	*Publication number*
CS	4
chat	1
VC	2
DV	3
EC	2

Table 7.3 publication number for each service categorization

Wasted Resource

To evaluate the resource wasted by a device when running the application, we have configured 2 laptops (hp Compaq nx9010 with wireless card Compaq 802.11b), and we have evaluated the resource wasted during a service publication estimating the amount memory and CPU used from the processes that run when the publication is activated. The processes that we considered are:

1. Java virtual machine (*java*) because the language used in the UDDI4m and middleware implementation is java language, so the application, when runs, can invocate java class;
2. Graphic Interface (*X*) because we run the operative system with graphic interface to show the home page of web application;
3. database (*mysql*) because a service publication involved insert of data in the database;
4. server web (*mozilla*) because must run to run our web application

The wasted resources are calculated from information recovered with *top command line*. While publishing a service, the top command store in an out file the information relative to:

- SIZE: the size of the task's code plus data stack space;
- RSS: total amount of physical memory used by the task;
- %CPU: the task's share of the CPU time expressed as a percentage of total CPU time processor;
- %MEM: the task's share of the physical memory.

We calculated the resources wasted in the worst case: the case of a publication that involves all the tables of the UBR4m registry.

7.1.3. Results and Discussions

Data Load Balancing

By considering the scenario described above -- where we detailed the publication numbers for each service category: 4 services of Content Sharing (CS), 3 of Device (DV), 2 of Video Conference (VC) and E-Commerce (EC) and 1 of chat (chat) -- the data spreading among the peer nodes is shown in Table 7.4.

Node Number	*Number of service categorization*
1	1
2	1
3	0
4	3
5	0
6	0

Table 7.4 load balance data in the peer nodes

The node 4 is more overloaded than other nodes: its registry has stored the information related to 3 service categorization; other 2 nodes have 1 service categorization only, while 3 nodes have nothing to store. This indicates the need to implement dynamic (i.e. adaptive to the status of the system) criteria to decide where to publish in order to obtain a more equilibrated distribution of the data on the network nodes.

Wasted Resource

We have considered the case of a publication of the service using our application and we have investigated the wasted resource to understand if this application can be installed on resources constrained devices.

The processes we considered are explained in the previous section and the resources wasted by each process are shown in Table 7.5, where we show the worst case of each process.

	SIZE(KB)	RSS(MB)	%CPU	%MEM
java	34172	23	4,9	3,5
X	78996	12	4,1	1,3
mysql	11280	11	0,3	1,1
mozilla	23568	23	3,7	2,4

Table 7.5 result obtained from top command line

Specifically, the total SIZE is 148016 KB and the total RSS is 69MB, this requirement can be satisfied from many devices (e.g., PDAs) that you can find in the market.

The laptop involved in this evaluation had a 750 MHz CPU, and the percentage %CPU indicates the wasted CPU time by each process. The process that used more CPU time is the java virtual machine but it is a low percentage that can be satisfied from devices with limited resources.

The most interesting information is the %MEM; from this index, considering that the laptop used during the experiments has 1GB RAM, we can calculate the wasted memory by the application. More precisely, from the *top command* information, the used memory results 950MB, so calculating the percentage of each process we obtained the results shown in the Table 7.6.

Java	*X*	*mysql*	*mozilla*
33,25MB	12,35 MB	10,45 MB	22,80 MB

Table 7.6 950MB (used memory) * %MEM of each process

The total wasted memory is 78.85 MB in the worst case. The most demanding process is the java virtual machine. If we wish to reduce the memory requirements we can avoid considering the database (as explained in deliverable D13 we have this option in the implementation).

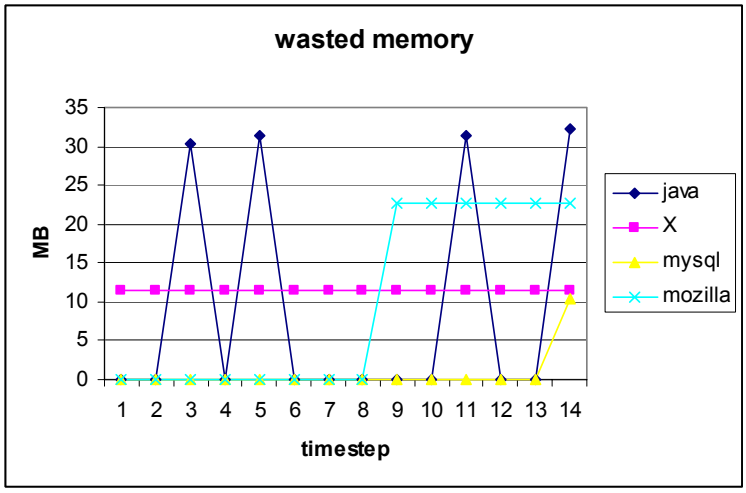


Figure 7.3 wasted memory trend

The above result represents the worst case, but while the application runs there are time instants in which some processes are down. In Figure 7.3, we show the wasted memory trend of each process during all application flow, the values of wasted memory are sampled at interval time of 5 seconds. We can observe that the Graphic Interface (X) is always up and uses 12.35 MB of memory; the mozilla process is up when we launch the web browser and uses 22.80 MB of memory. These 2 processes, on a PDA device or mobile telephone, can be lighter. The mysql process is optional, a node may have not the UBR4m registry. Finally, the java process is the most critical process because uses a lot of memory when the application invocate a java class. In our case, there are 4 calls to java classes and in these time instants the wasted memory is around 32 MB.

7.1.4. References

[UD02]	UDDI Version 2.03 Data Structures Reference, UDDI Committee Specification, 19 July 2002.
[UM05]	P. Cremonese, V. Vanni, “UDDI4m: UDDI in Mobile Ad Hoc Network”, WONS 2005, St Moritz,

7.2. The Whiteboard Application

Even though research on MANETs has been very active in the last decade, real applications addressed to people outside the research community still have to be developed. The typical simulation-based approach for the performance evaluation of MANETs is one of the main reasons of this. Often, simulation results turn out to be quite unreliable if compared to real-world measurements [ABCGP05, GLNT05], and real world experiments are highly required for MANET applications to become reality, despite their high costs (in terms of time to set up) and intrinsic limitations (number of nodes).

By leveraging the self-organising nature of MANETs, group-communication applications can be an outstanding opportunity from this standpoint. In this work, we focus on a significant example of this class of applications, and we evaluate complete networking solutions that could be used to develop it. Specifically, we consider the Whiteboard application (WB), which implements a distributed whiteboard among MANET users (see Figure 7.4). Each MANET user runs a WB instance on her device, selects a topic she wants to join, and starts drawing on the canvas. Drawings are distributed to all nodes subscribed to that topic, and rendered on each canvas. We believe that these simple, "Plug&Play" applications will be of great value for MANET users.

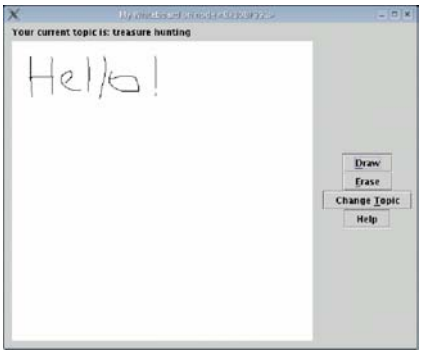


Figure 7.4: The Whiteboard interface.

Developing this kind of applications in MANETs is a challenging task. In this work we present the networking solutions we have studied and tested to this end. We present alternative networking frameworks for supporting WB-like applications. Then, we compare a standard P2P system (Pastry [RD01]) with CrossROAD [D05], the P2P system optimised for MANETs that we have designed within these frameworks. Advantages of the CrossROAD approach are presented by means of experimental results.

The main contribution of this work is evaluating through real experiments complete networking solutions for developing distributed applications such as WB in real-world MANETs. We evaluate our prototype at two different levels, i.e., we quantify i) the QoS perceived by WB users, and ii) the quality of the multicast tree generated by Scribe. First of all, we show how a proactive routing protocol performs better than a reactive one with

regard to this kind of applications. Then, we highlight that a solution based on Pastry and Scribe is not suitable for MANET environments. WB users perceive unacceptable high data loss and delays. Furthermore, both the Pastry overlay network and the Scribe multicast tree get frequently partitioned. This results in some WB users to be completely isolated from the rest of the network. Finally, we show that some of these problems can be avoided by using CrossROAD. Specifically, the structure of the Scribe tree is quite more stable when CrossROAD is adopted, and partitions problems experienced with Pastry completely disappear. Thus, CrossROAD turns out to be a very promising P2P system for MANET environments.

7.2.1. WB integration in MANETs

Group-communication applications such as WB are distributed, self-organising, decentralised in nature. Designing them on top of P2P systems guarantees a great flexibility and optimised performances exploiting P2P policies to distribute and recover information. Figure 7.5 depicts the abstractions we have used to support WB. The network level provides basic connectivity among nodes through IP-like routing and transport protocols. On top of them, a structured overlay network, comprising nodes that participate in the WB application, is built. The overlay abstraction is the fundamental substrate for any P2P application, providing functionalities such as logical node addressing (instead of topological, IP-like addressing) and subject-based routing. Finally, an additional multicast level is used to efficiently distribute contents generated by application users to all nodes in the overlay. These abstractions make quite straightforward develop group communication applications. They hide the complexity of low-level communications, group management, and data distribution, and provide a robust, flexible, self-organising networking environment.

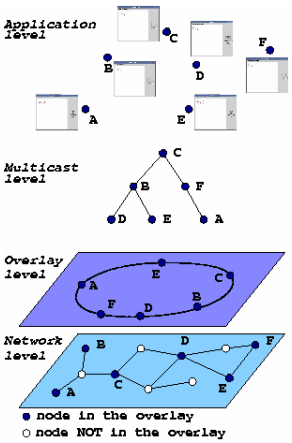


Figure 7.5: Abstraction supporting WB.

Figure 7.6 shows the complete networking solutions we have used to support WB in real-world MANETs. We have defined a first architecture (referred to as legacy), that uses state-of-the-art components to implement the abstractions in Figure 7.5. Specifically, it uses either AODV [AODV] or OLSR [OLSR] at the network level, Pastry [RD01] at the middleware level, and Scribe [CDKR03] at the multicast level. While AODV and OLSR

represent standard models for ad hoc reactive and proactive routing protocols, Pastry and Scribe have been designed for wired networks. The evaluation of the "legacy solution" indicates weaknesses of these components, and ways to improve them. In order to optimize the entire system performances, a cross-layer architecture, as depicted on the right-hand side of Figure 7.6, has been proposed in [CMTG04]. Specifically, the NeSt module allows cross-layer interactions between protocols at different layers. To this aim, NeSt provides well-defined interfaces and data abstractions to protocols [CMTG04], joining the advantages of cross-layering and the scalability of traditional layered approach. CrossROAD represents an optimised solution at the middleware layer that exploits cross-layer interactions with a proactive routing protocol (OLSR in this case) in order to optimise the creation and management of the overlay network. In this work we do not discuss any other MANET-optimised solutions that could be integrated into the cross-layer architecture. However, other such components both at the routing level (Hazy Sighted Link State [SSR01]), and at the multicast level (X-layer Scribe) are being studied and currently under development.

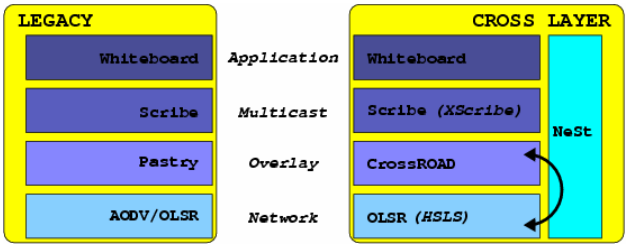


Figure 7.6: Network solutions: legacy (left) and cross-layer (right).

7.2.2. WB and its middleware support

For the reader convenience, before discussing the results of our experiments we now briefly recall the main characteristics of Pastry, CrossROAD, and Scribe.

Pastry and CrossROAD

Pastry is a P2P system based on a DHT to build a structured overlay network (ring) at the middleware level. A logical identifier (node id) is assigned to each node hashing one of its physical identifiers (e.g., IP address, hostname). Messages are sent on the overlay by specifying a destination key k belonging to the logical identifiers' space. Pastry routes these messages to the node whose id is numerically closest to k value. To route messages, Pastry nodes maintain a limited subset of other nodes' logical ids in their internal data structures (middleware routing tables). Periodic data exchange between nodes of the overlay is needed to update the state of the overlay. Finally, in order to initially join the overlay network, each Pastry node executes a bootstrap procedure, during which it initialises its middleware routing table by collecting portions of other nodes' routing tables. Specifically, each nodes has to connect to an already existing Pastry node (i.e., it needs to know its IP address) in order to correctly start the bootstrap procedure.

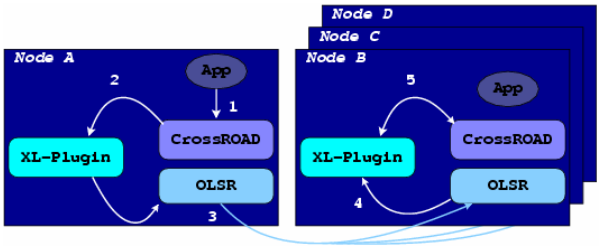


Figure 7.7: Cross-layer interactions between CrossROAD and OLSR.

The bootstrap phase and the periodic data exchange between nodes constitute the main network overhead of Pastry. CrossROAD [D05] provides the same Pastry functionalities through the P2P commonAPI [DZDK+03], but it drastically reduces the overlay management traffic by exploiting cross-layer interactions with a proactive routing protocol. Specifically, CrossROAD implements a Service Discovery protocol that exploits the broadcast flooding of routing packets to distribute services information. An example of cross-layer interaction between CrossROAD and OLSR is shown in Figure 7.7. Each application running on CrossROAD has to register itself by specifying a service id (step 1). The list of service ids registered at the local node (Node A in the figure) is maintained by the Cross-Layer Plugin (XL-Plugin), which can be seen as a portion of the NeSt module (step 2). The XL-Plugin embeds the list of local service ids into periodic Link-State Update packets generated by OLSR (step 3). On the other nodes of the network (nodes B, C, D in the figure), upon receiving LSU packets containing such list, the routing level notifies XL-Plugin to store the list in its internal data structures. This way, each CrossROAD node has a complete knowledge of all the other nodes providing the same service in the MANET, and it is able to autonomously build the overlay network without generating any further management traffic (step 5). Furthermore, in case of topology changes, the status of the overlay network converged as quickly as the routing protocol does.

Scribe

Scribe exploits Pastry-like routing to build multicast groups (see Figure 7.8). From the standpoint of the application running on Scribe, the group is identified by a topic. Scribe uses the hash function provided by Pastry (or CrossROAD) to generate the topic id (tid) in the logical space of node ids. In order to join the Scribe tree, nodes send a join message on the overlay with key equal to tid (e.g., node E in the figure). This message reaches the next hop (B in the figure) towards the destination on the overlay network. The node originating the join message (E) is enrolled as a child of B. If not already in the tree, B itself joins the tree by generating a join message anew. Eventually, such a message reaches the node whose id is the closest one to tid (C in the figure) and is not propagated further. This node is defined as the root of the Scribe tree. Join messages for further nodes (D in the figure) which occur to share some part of the overlay path towards the root node have to travel just up to the closest branching point (B in the figure).

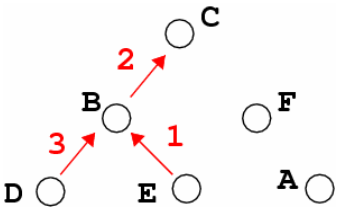


Figure 7.8: Scribe building the tree.

Application messages are sent on the overlay with key equal to tid (e.g., D sends a message in Figure 7.9). Hence, they reach the Scribe root, which is in charge of delivering them over the tree. To this end, it forwards the messages to its children, which further forward them to their children, and so on.

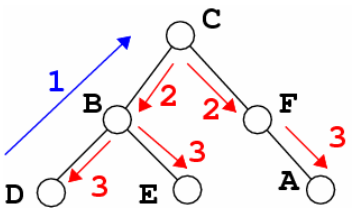


Figure 7.9: Scribe delivering messages.

Finally, the Scribe maintenance procedure is as follows. Each parent periodically sends a HeartBeat message to each child (application-level messages are used as implicit HeartBeats). If a child does not receive any message from the parent for a given time interval (20 sec. in the default case), it assumes that the parent has given up, and re-executes the join procedure. This simple procedure allows node to discover parent failures, and re-join the tree, if the case.

7.2.3. Experimental Environment

The experiments reported in this work are based on a static MANET. This allows us to highlight limitations that originate from Pastry and Scribe design, rather than to mobility. Extending the results in the case of mobility is subject of future work.

The experiment test-bed is as depicted in Figure 7.10. We set up an indoor MANET consisting of 8 nodes. To have a homogeneous test-bed, all nodes are IBM ThinkPad R50 laptops. We use the built-in Intel PRO-Wireless 2200 802.11 card, with ipw2200 driver (on Linux 2.6 kernel). The data rate is set to 11 Mbps. In addition the transmission power of each card has been adjusted to reproduce the topology shown in the figure and obtain a multi-hop ad hoc network. During the experiments, nodes marked A through to F participate in the overlay network, and run the WB application (they will be throughout referred to as “WB nodes”). Nodes marked with “R” are used just as routers. It is worth pointing out that this setup lies within the “802.11 ad hoc horizon” envisioned in [GLNT05], i.e. 10-20 nodes, and 2-3 hops. Therefore, it is a valid example of possible real-world MANETs.

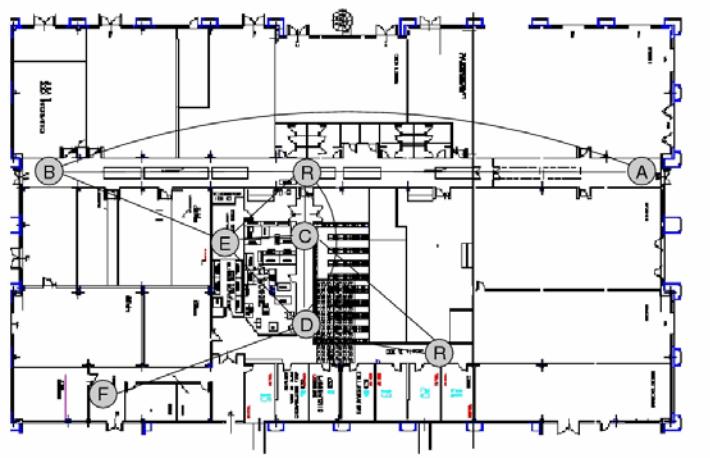


Figure 7.10: Map of the experiment setup.

In order to have a controllable and reproducible setup, a human user at a WB node is represented by a software agent running on the node. During an experiment, each software agent interleaves active and idle phases. During an active phase, it draws a burst of strokes on the canvas, which are sent to all the other WB nodes through Scribe (please note that in our experiment each stroke generates a new message to be distributed on the Scribe tree). During an idle phase, it just receives possible strokes from other WB nodes. After completing a given number of such cycles (a cycle is defined as a burst of strokes followed by an idle time), each agent sends a Close message on the Scribe, waits for getting Close messages of all the other nodes, and shuts down. Burst sizes and idle phase lengths are sampled from exponentially distributed random variables. The average length of idle phases is 10 s, and is fixed through all the experiments. On the other hand, the average burst size is defined on a per-experiment basis. As a reference point, we define a traffic load of 100% as the traffic generated by a user drawing, on average, one stroke per second. Finally, the number of cycles defining the experiment duration is fixed through all the experiments. Even at the lowest traffic load taken into consideration, each agent draws – on average – at least 50 strokes during an experiment. For the performance figures defined in this work (see below) this represents a good trade-off between the experiment duration and the result accuracy.

Some final remarks should be pointed out about the experiment start-up phase. Nodes are synchronised at the beginning of each experiment. Then, in the Pastry case, the Pastry bootstrap sequence occurs as follows (the same schedule is also used to start CrossROAD, even though a CrossROAD node does not need to bootstrap from another node): node C starts first, and generates the ring. Nodes E and D start 5 seconds after C, and bootstrap from C. Node B starts 5 seconds after E and bootstraps from E. Node A starts 5 seconds after B and bootstraps from B. Finally, node F starts 5 seconds after D and bootstraps from D. After this point in time, the Scribe tree is created and, finally, WB instances start sending application messages (hereafter, WB messages). This way, the Scribe tree is built when the overlay network is already stable, and WB starts sending when the Scribe tree is completely built.

Performance Indices

Since Pastry and Scribe have been conceived for fixed networks, we investigate if they are able to provide an adequate Quality of Service to users in a MANET environment. To quantify the "WB user satisfaction" we use two performance indices:

- **Packet Loss**: at each node *i*, we measure the number of WB messages received and sent (R_i and S_i, respectively) during an experiment; the packet loss experienced by node *i* is defined as $pl_i = \frac{R_i}{\sum_i S_i}$.
- **Delay**: the time instant when each packet is sent and received is stored at the sending and receiving node, respectively. This way, we are able to evaluate the delay experienced by each node in receiving each packet. If d_{ij} is the delay experienced by node *i* in receiving packet *j*, and N_i the total number of packets received by *i* during an experiment, the average delay experienced by node *i* is defined as $D_i = \frac{\sum_j d_{ij}}{N_i}$.

Furthermore, we define two more indices, to quantify the quality of the multicast tree created by Scribe.

- **Node Stress**: for each node, it is defined as the average number of children of that node. If t_{ij} is the time interval (within an experiment) during which node *i* has n_j children, the average node stress of node *i* is $NS_i = \frac{\sum_j n_j t_{ij}}{\sum_j t_{ij}}$.
- **Re-subscriptions**: for each node, we count the number of times (during an experiment) this node sends new subscriptions requests, because it can't communicate with the previous parent anymore.

7.2.4. Performance with Pastry

The results we report in this section are obtained by using Pastry as DHT and either OLSR or AODV as routing protocol. Experiments are run by increasing the traffic load starting from 20% up to 80%.

Before presenting the results in detail, let us define what hereafter will be referred to as "crash of the Scribe Root Node". In our configuration Pastry assigns node ids by hashing the IP address and the port used by Scribe on the node. Hence, each node always gets the same node id. Furthermore, the topic used by the WB users is always the same. Under the hypothesis that Pastry generates a single ring encompassing all WB nodes, the Root of the Scribe tree (i.e., the node whose id is closest to the WB topic id) is the same through all the experiments, and is node C in Figure 7.10. This node will be throughout referred

to as the Main Scribe Root Node (MSRN). Due to the Scribe algorithm, each WB message to be distributed on the tree is firstly sent to MSRN, and then forwarded over the tree. Often, this is an excessive load for MSRN, which, after some point in time, becomes unable to deliver all the received messages. Instead, messages are dropped at the MSRN sending queue. We refer to this event as a crash of MSRN. Of course, since the application-level traffic is randomly generated, the MSRN crash is not a deterministic event.

User Satisfaction

Figures 5.2.8 and 5.2.9 show the packet loss and the delay indices experienced by the WB nodes considering experiments where the MSRN does not crash. Specifically, we consider AODV experiments with 10% and 20% traffic load, and OLSR experiments with 20%, 50% and 80% traffic load, respectively. There is no point in running AODV experiments with higher traffic load, since performances with AODV are quite bad, even with such a light traffic load. In the figure legend we also report the rings that Pastry builds during the bootstrap phase (please note that, theoretically, just one ring should be built, encompassing all WB nodes). Finally, an "x" label for a particular node and a particular experiment denotes that for that experiment we are not able to derive the index related to the node (for example, because some component of the stack crashed during the experiment).

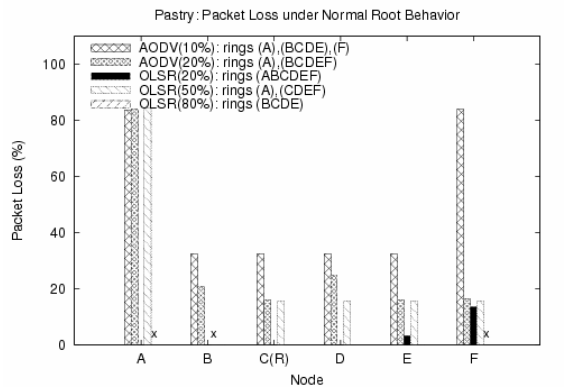


Figure 7.11: Packet Loss w/o MSRN crash

Figure 7.11 allows us to highlight an important Pastry weakness. If a WB node is unable to successfully bootstrap, it starts a new ring, and remains isolated for the rest of the experiment. In MANET environments, links are typically unstable, and the event of a WB node failing to contact the bootstrap node is quite likely. Clearly, once a node is isolated, it is unable to receive (send) WB messages from (to) other nodes for the rest of the experiment, and this results in packet losses at all nodes. In the "AODV 10%" experiment, nodes A and F are isolated, and create their own rings. This results in packet loss of about 80% at those nodes (i.e., they just get their own WB messages, which is about one sixth of the overall WB traffic), and about 33% at nodes B, C, D and E. Similar remarks apply to the "OLSR 50%" experiment. It is more interesting to focus on the

"AODV 20%" experiment. In this case, node A is isolated, while nodes B, C, D, E and F belong to the same ring. As before, A's packet loss is about 80%. The packet loss at the other nodes due to the isolation of node A is about 18% (one sixth of the overall traffic). It is interesting to notice that nodes B and D experience a higher packet loss, meaning that they are unable to get WB messages generated within the "main" Pastry ring (i.e., nodes B, C, D, E, F). Finally, in the case "OLSR 20%", Pastry is able to correctly generate a single ring, and the packet loss is quite low. In the case "OLSR 80%", nodes A and F crash. However, the packet loss experienced by the other nodes is negligible.

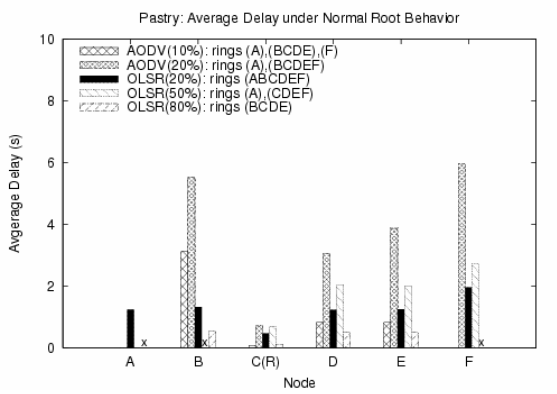


Figure 7.12: Delay w/o MSRN crash

Similar observations can be drawn by focusing on the delay index (Figure 7.12). First of all, it should be pointed out that the delay related to nodes that are the sole member of their own ring (e.g., node A in the "AODV 10%" case) is obviously negligible. Even though – in general – the delay in this set of experiments is low, it can be noted that better performances are achieved by using OLSR instead of AODV. Finally, it should be noted that MSRN (node C) always experiences a lower delay with respect to the other nodes in the same ring.

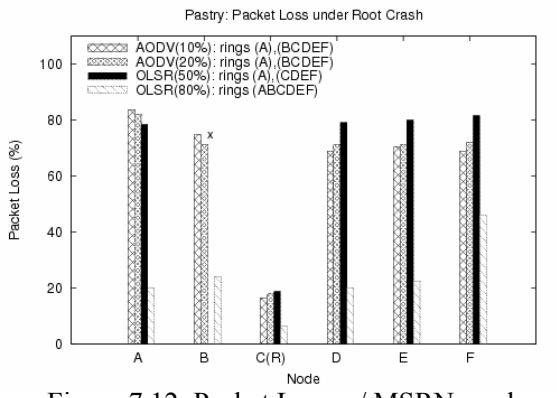


Figure 7.12: Packet Loss w/ MSRN crash

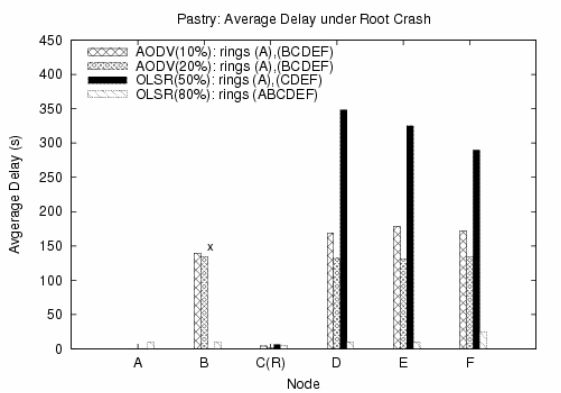


Figure 7.14: Delay w/ MSRN crash

Figures 5.2.10 and 5.2.11 show the packet loss and the delay indices in cases of MSRN crash. The packet loss experienced by nodes in the same ring becomes higher than in cases where MSRN does not crash. In the first three experiments, node A isolation causes a packet loss of about 18% on the other nodes. Hence, the remaining 60% packet loss is ascribed to the MSRN crash. Quite surprisingly, OLSR with 80% traffic load shows better performance than OLSR with 50% traffic load. It is also interesting to note that the packet loss at MSRN is always lower than at other nodes in the same ring. This highlights that MSRN is able to get, but unable to deliver over the Scribe tree WB messages generated by other nodes. Similar observations can be drawn by looking at Figure 7.14, as well. The delay experienced by nodes B, D, E and F can be as high as a few minutes, either by using AODV or OLSR. Finally, the delay experienced by MSRN is very low in comparison to the delay experienced by the other nodes.

To summarise, the above analysis allows us to draw the following observations. The Pastry bootstrap algorithm is too weak to work well in MANETs, and produces unrecoverable partitions of the overlay network. This behaviour is generally exacerbated by AODV (in comparison to OLSR). Furthermore, MSRN is clearly a bottleneck for Scribe. MSRN may be unable to deliver WB messages also with moderate traffic loads, resulting in extremely high packet loss and delay. Moreover, the performance of the system in terms of packet loss and delay is unpredictable. With the same protocols and traffic load (e.g., OLSR and 50% traffic load), MSRN may crash or may not, resulting in completely different performance figures. In cases where MSRN crashes, packet loss and delay are clearly too high for WB to be actually used by real users. However, even when MSRN does not crash, the high probability of WB users to be isolated from the overlay network makes Pastry based solutions too unreliable. These results suggest that Pastry and Scribe need to be highly improved to actually support group communication applications such as WB in MANET environments.

Multicast Tree Quality

In this section we analyse the node stress and re-subscription indices, with respect to the same experiments used in the previous section.

Figures 5.2.12 and 5.2.13 plot the average node stress with and without MSRN crashes, respectively. In both cases, the node stress is significantly higher at MSRN than at any other node. This means that the Scribe tree is a one-level tree, and MSRN is the parent of all the other nodes. This behaviour is expected, and can be explained by recalling the way Scribe works. In our moderate-scale MANET, all nodes are in the Pastry routing table of each other. Hence, Scribe join messages reach MSRN as the first hop, and MSRN becomes the parent of all other nodes (in the same ring). Together with the way application-level messages are delivered, this phenomenon explains why MSRN is a bottleneck, since it has to send a distinct message to each child when delivering WB messages over the tree. This is a major limitation of the Scribe algorithm, and optimisations of the P2P system are clearly not sufficient to cope with it.

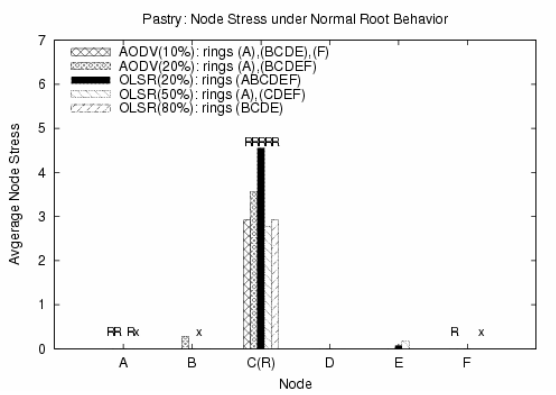


Figure 7.15: Node Stress w/o MSRN crash

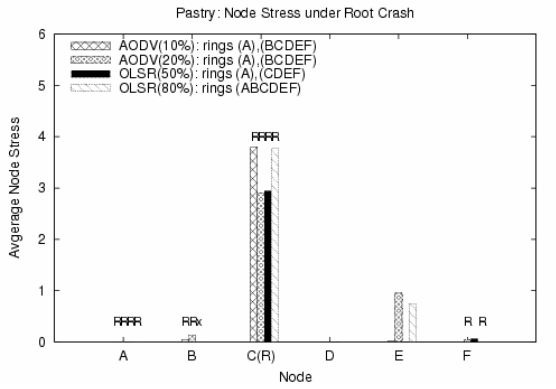


Figure 7.16: Node Stress w/ MSRN crash

In Figures 7.15 and 7.16 we have added "R" labels to indicate nodes that occur to become Scribe Root during the corresponding experiment. When MSRN does not crash (Figure 7.15) other nodes become Scribe root only as a side effect of a failed Pastry bootstrap. On an isolated WB node, Scribe builds a tree which consists only of the node itself that is thus the root. However, Scribe partitions may also occur due to congestion at the Pastry level in cases where MSRN crashes. By looking at Figure 7.16, it can be noticed that

nodes other than MSRN may become root also if they belonged (after the Pastry bootstrap phase) to the same overlay network of MSRN. This phenomenon occurs, for example, at node A in the OLSR 80% case, and at node B and F (whenever they become root). It should be noted that a node with id n1 (other than MSRN) becomes root when i) it looses its previous parent, and ii) the Pastry routing table does not contain another node id n2 such that n2 is closer to the WB topic id than n1. Figure 7.16 shows that the congestion at the Pastry level is so high that the Pastry routing table of some nodes becomes incomplete (i.e., MSRN disappears from other nodes' routing table). Thus, the Scribe tree gets partitioned in several isolated sub-trees. Clearly, this contributes to the high packet loss measured in these experiments. Another effect of Pastry congestion during MSRN crashes is a possible reshaping of the Scribe tree. Figure 7.16 shows that the average Node Stress of E is close to 1 in the "AODV 20%" and "OLSR 80%" cases. This means that MSRN disappears from the Pastry routing table of some node, which – instead of becoming a new root – finds node E to be the closest one to the WB topic id. This phenomenon could be considered a benefit, since it reduces the MSRN node stress. However, it derives from an incorrect view of the network at the Pastry level, originated from congestion.

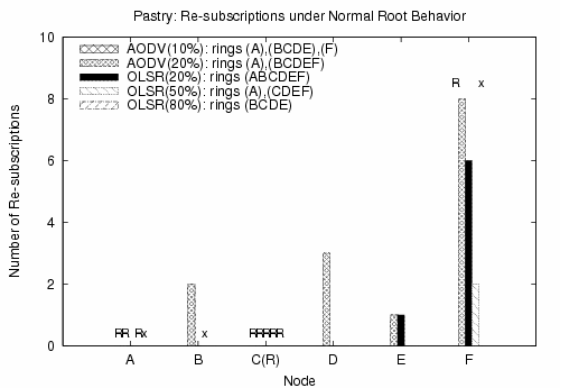


Figure 7.17: Re-subscriptions w/o MSRN crash

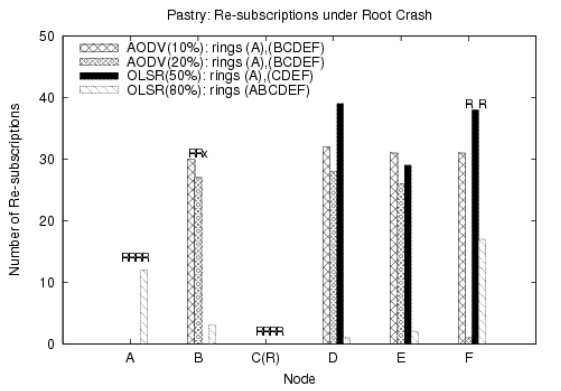


Figure 7.18: Re-subscriptions w/ MSRN crash

Figures 7.17 and 7.18 show the re-subscription index for the same set of experiments. Figure 7.17 shows that, when MSRN does not crash, the Scribe tree is quite stable. Most of the re-subscriptions occur at node F, which is the "less connected" node in the network (see Figure 7.10). In these experiments, the performance in the AODV cases is worse than in OLSR cases. Furthermore, upon MSRN crashes (Figure 7.18), the number of re-subscriptions increases drastically, even in case of "well-connected nodes" (i.e., node B, D and E). MSRN crashes make other nodes unable to get messages from their parent (i.e., MSRN itself), increasing the number of re-subscriptions. It is interesting to point out that this is a typical positive-feedback control loop: the more MSRN is congested, the more re-subscriptions are sent, and more congestion is generated.

To summarise, the multicast tree generated by Scribe on top of Pastry is quite unstable, especially in cases of MSRN crashes. The tree may get partitioned in disjoint sub-trees, and many re-subscriptions are generated by nodes. Furthermore, Scribe is not able to generate a well-balanced multicast tree, since MSRN is the parent of all other nodes. Directions to optimise Scribe are discussed in Section 5.2.6.

7.2.5. Improvements with CrossROAD

In this section we show that using a P2P system optimised for MANETs is highly beneficial to the stability of the Scribe tree. In this set of experiments, we use CrossROAD instead of Pastry, and set the traffic load to 20%, 50% and 100%, respectively. We concentrate on the performance figures related to the quality of the multicast tree, i.e., the average node stress (Figure 7.19) and the number of re-subscriptions (Figure 7.20). Complete evaluations of the User Satisfaction parameters, as well as, further optimisations of the Scribe algorithm are for further studies.

The first main improvement achieved by using CrossROAD is that neither the overlay network, nor the Scribe tree, gets partitioned. CrossROAD is able to build a single overlay network in all the experiments. Furthermore, even at very high traffic loads (e.g., 100%), MSRN is the only root of the Scribe tree. Therefore, CrossROAD is able to overcome all the partition problems experienced when Pastry is used.

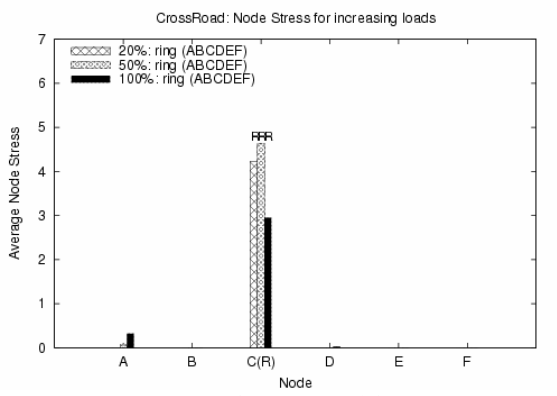


Figure 7.19: Node Stress with CrossROAD

Figure 7.19 clearly shows that the node stress still remains quite unbalanced among the nodes. MSRN is typically the parent of all other nodes, and this contributes to make it a bottleneck of the system, as highlighted above. This behaviour is expected, since it derives from the Scribe algorithm, and cannot be modified by changing P2P system.

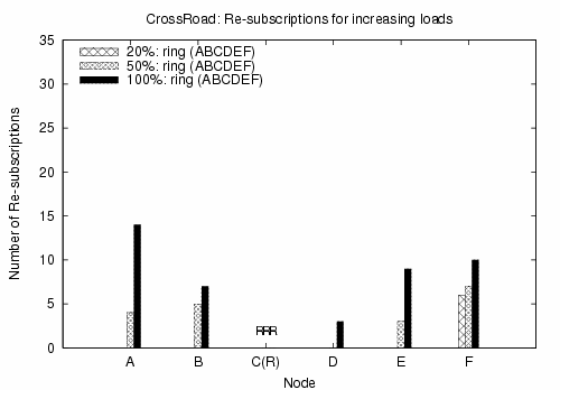


Figure 7.20: Re-subscriptions with CrossROAD

Finally, Figure 7.20 shows that the Scribe tree is more stable (i.e., requires less re-subscriptions) using CrossROAD instead of Pastry. To be fair, we have to compare Figure 7.20 with both Figures 7.17 and 7.18. It is clear that CrossROAD outperforms Pastry when used on top of AODV. The "20%" case of CrossROAD should be compared with the "OLSR 20%" case of Figure 7.17, since in both experiments the overlay network is made up of all nodes. The number of re-subscriptions measured at node F is the same in both cases, while it is higher at node E when Pastry is used. The CrossROAD "50%" case shows a higher number of re-subscriptions with respect to the "OLSR 50%" case in Figure 7.17. However, it should be noted that in the latter case the overlay network encompasses less nodes, and hence the congestion is lower. It should also be noted that, with the same nodes in the overlay network, with the same protocol stack and traffic load, Pastry experiments may suffer MSRN crashes (Figure 7.18). In this case, the number of re-subscriptions is much higher than in the CrossROAD case. Finally, results in the CrossROAD "100%" case should be compared with the "OLSR 80%" case of Figure 7.18, since the overlay network is the same in both experiments. CrossROAD achieves comparable performance, and at some nodes it outperforms Pastry, even if the application traffic is significantly higher.

Overlay Management Overhead

In the previous section we have shown that adopting CrossROAD significantly improves the performance of Scribe. In this section we highlight that one of the main reasons for this improvement is the big reduction of the network overhead. This is a key advantage in MANET environments.

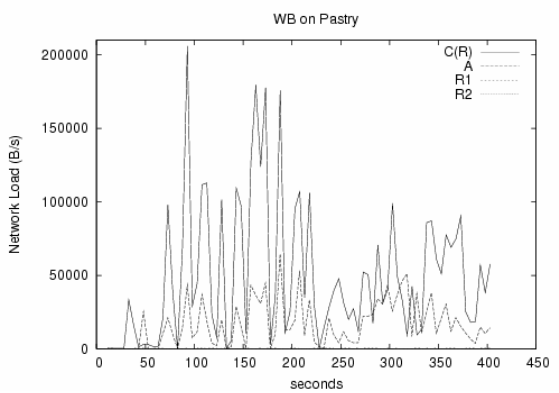


Figure 7.21: Network Load with Pastry

Figure 7.21 shows the network load experienced by nodes A, C and by the two nodes which just act as routers, during the Pastry "OLSR 80%" experiment in which MSRN crashes (we do not take into account AODV experiments, since OLSR has clearly shown to outperform AODV). Each point in the plot is computed as the aggregate throughput (in the sending and receiving directions) over the previous 5-seconds time frame. We take into consideration the traffic related to the whole network stack, from the routing up to the application layer. Specifically, nodes A and C are representative for WB nodes, pointing out the difference with nodes that just work as routers. The discrepancy between the curves related to node A and C confirms that the MSRN node has to handle a far greater amount of traffic with respect to the other WB nodes, due to the Scribe mechanisms. Furthermore, it should be noted that the curves related to the two routers can hardly been distinguished in Figure 7.21, since they are about 400Bps. This means that the lion's share of the load on WB nodes is related to Pastry, Scribe and the WB application.

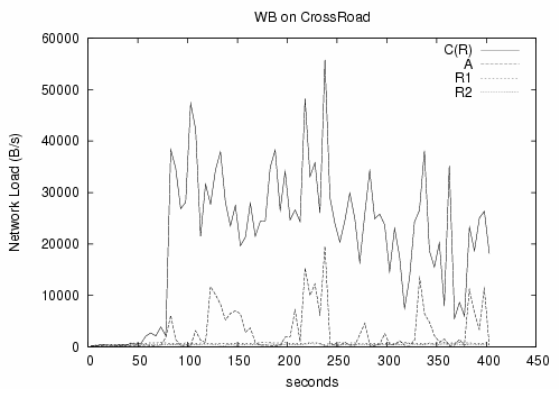


Figure 7.22: Network Load with CrossROAD

Figure 7.22 plots the same curves, but related to the "100%" CrossROAD experiment. Also in this case, MSRN (node C) is more loaded than the other WB nodes. However, by comparing Figures 7.22 and 7.23 we can highlight that the Pastry network load is far higher than the CrossROAD network load. By considering the average value over all

nodes in the MANET, the Pastry load is about 3 times greater than the CrossROAD load. More specifically, the average load of C and A is 48.5 KB/s and 16.5 KB/s in the Pastry case, while drops to 21.1 KB/s and 2.96 KB/s in the CrossROAD case. The reduction of the network load achieved by CrossROAD is thus 56% at node C and 82% at node A. Since the other stack components are exactly the same, CrossROAD is responsible for this reduction (the actual reduction is even higher, since the application-level traffic is 100% in the CrossROAD case). Furthermore, it should be noted that, during several time intervals, the load of node A is just slightly higher than that of "routing" nodes. This suggests that the additional load of CrossROAD management with respect to the routing protocol is very limited.

7.2.6. Conclusions and Future Works

Results presented in this work allow us to draw the following conclusions. Pastry and Scribe seem not to be good candidates to support group communication applications in MANET environments. Pastry is particularly weak during the bootstrap phase, causing the overlay network to be partitioned into several sub-networks, and some nodes to be unable to join application services. Further partitions may occur in the Scribe tree due to congestion at the Pastry level. Finally, the delivery algorithm implemented by Scribe generates a severe bottleneck in the tree, which is highly prone to get overloaded. All these limitations result in unacceptable levels of packet loss and delay for applications. Many of these problems can be avoided by adopting a cross-layer optimised P2P system such as CrossROAD. Thanks to the interactions with a proactive routing protocol CrossROAD is able to avoid all the partition problems experienced with Pastry, and to drastically reduce the network overhead. Clearly, CrossROAD cannot solve the problem of bottlenecks in the Scribe trees. Therefore, optimized versions of Scribe are required for group communication applications such as WB to be really developed in MANETs. The direction we are exploring is building a single distribution tree, optimised through cross-layer interactions with a proactive routing protocol. Building a single-tree, instead of a new tree for each source node, allows for a more scalable solution. In addition, cross-layering allow us to retain the subject-based features of Scribe (e.g., locating a tree by means of its topic), while exploiting also topological information to build the tree. Furthermore, the tree can be built in a completely distributed way, by exploiting greedy policies such as those used in YAM [CC97] and ALMA [GKF04]. Finally, the data-distribution phase can be optimised so as to avoid each message to be sent to the MSRN and then delivered to other nodes. For example, while travelling towards MSRN, messages can be duplicated and delivered at each branching point in the tree. These policies are expected to drastically mitigate the bottleneck problems experienced by Scribe. Furthermore, they raise very interesting arguments about causal ordering of messages that we are planning to address, as well.

7.2.7. References

[ABCGP05] G. Anastasi, E. Borgia, M. Conti, E. Gregori and A. Passarella, "Understanding the Real Behavior of Mote and 802.11 Ad hoc Networks: an Experimental Approach", Pervasive and Mobile Computing, Vol. 1, Issue 2, pp. 237-256, July 2005.

[AODV] AODV, Dept. of Information technology at Uppsala University (Sweden), http://user.it.uu.se/ henrikl/aodv/.

[CC97] K. Carlber and J. Crowcroft, "Building Shared Trees Using a One-to-Many Joining Mechanism", ACM Computer Communication Review, pp. 5-11, Jan. 1997.

[CDKR02] M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "SCRIBE: A large-scale and decentralised application-level multicast infrastructure", IEEE Journal on Selected Areas in Communication (JSAC), Vol. 20, No, 8, October 2002.

[CJKR+03] M. Castro, M. B. Jones, A-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang and A. Wolman, "An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer overlays", Infocom 2003, San Francisco, CA, April, 2003.

[DZDK+03] F. Dabek and B. Zhao and P. Druschel and J. Kubiatowicz and I. Stoica, "Towards a common API for Structured Peer-to-Peer Overlays", Proc. of the the 2nd International Workshop on Peer-to-peer Systems (IPTPS'03), Berkeley, CA, Feb. 2003.

[CMTG04] M. Conti, G. Maselli, G. Turi, and S. Giordano, "Cross layering in mobile ad hoc network design", IEEE Computer, Feb. 2004.

[D05] F. Delmastro, "From Pastry to CrossROAD: Cross-layer Ring Overlay for Ad hoc networks", in Proc. of Workshop of Mobile Peer-to-Peer 2005, in conjuction with the PerCom 2005 conference, Kauai Island, Hawaii, Mar. 2005.

[D04] M. Dischinger, "A flexible and scalable peer-to-peer multicast application using Bamboo", Report of the University of Cambridge Computer Laboratory, 2004, available at http://www.cl.cam.ac.uk/Research/SRG/netos/futuregrid/dischinger-report.pdf.

[GKF04] M. Ge, S.V. Krishnamurthy, and M. Faloutsos, "Overlay Multicasting for Ad Hoc Networks", Proc. of the Third Annual Mediterranean Ad Hoc Networking Workshop (MedHocNet 2004), June 2004.

[GLNT05] P. Gunningberg and H. Lundgren and E. Nordström and C. Tschudin, "Lessons from Experimental MANET Research", Ad Hoc Networks Journal, (Special Issue on "Ad Hoc Networking for Pervasive Systems"), Vol. 3, Number 2, March 2005.

[OLSR] OLSR, Andreas Tonnesen, Institute for informatics at the University of Oslo (Norway), http://www.olsr.org.

[RD01] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", Middleware 2001, Germany, November 2001.

[SSR01] C.A. Santivanez, I. Stavrakakis, and R. Ramanathan, ”Making link-state routing scale for ad hoc networks”. In Proceedings of the 2nd ACM Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC’01), 2001.

7.3. VoIP

Voice over IP (VoIP) provides voice telephony service utilizing the exiting wide spread Internet infrastructure. VoIP has been around for several years but it is gaining momentum especially because of its low cost deployment requirements, and its low cost service for users.

7.3.1. VoIP introduction

VoIP is a speech transmission technology utilizing the packet switched networks. It is built on top of IP protocol. In order to provide users with similar experience to the analog telephony service, VoIP must be able to deliver non-interrupted continuous stream of voice with as minimum delay as possible. This basic requirement will govern the selection of transport protocol and trade offs between real time audio delivery and high quality audio.

TCP/IP provides mechanisms to offer guaranteed packet delivery. However, this does not suite the requirements and needs of VoIP. TCP/IP makes sure a packet is delivered and the packets arrive in order, using retransmission. However, the TCP mechanisms result in the overall increase of the end-to-end delay. VoIP requires sending packets as fast as possible and delivering them to receiver after as short delay as possible. VoIP includes mechanisms at the reception to alleviate some of the problems inherent in the IP protocol, where Jitter buffering is the most important. Even though IP protocol offers only a best-effort service, packets can be delivered out of order, corrupted, duplicated, with different delay for each packet or the packets can even bet lost. Therefore, VoIP uses the IP stack to send separate datagrams (i.e. UDP transport) quickly without any congestion control or retransmission mechanism. Afterwards, the appropriate techniques are applied to buffer and re-order the packets in order to provide a voice service resilient to packet loss and variable packet arrival delay.

7.3.2. VoIP techniques

VoIP takes care of transmitting the voice after recording and breaking it down into small packets at the transmitter. The receiver will receive those packets and has to reconstruct and play them back. The transmitter starts by sampling the analog audio signal, digitizing it to audio bytes at a sampling frequency. Then it compresses sampled audio and generates audio packets that will be transmitted using a transport protocol such as Real Time Protocol (RTP [1]) that provides packet sequence order and additional information for reconstructing the audio stream in the receiver. The receiver takes care of unpacking the RTP messages, interpreting the sequence numbering and implementing the buffering of audio packets to ensure continuous playback. After buffering enough packets, the receiver decompresses the audio packets and reconstructs the audio samples to play them back.

The typical value (i.e. PCM format) for audio streams is 8000 Hz sampling frequency, 8 bit per sample, resulting in a 64 kbps audio stream.

7.3.3. VoIP testbed

The VoIP testbed we have developed uses components from other research institutions [2]. In order to increase the QoS in Ad hoc networks, enhancements in the VoIP application have been implemented. The experiment was carried out in the laboratories of the Consiglio Nazionale delle Ricerche (IIT-CNR), Pisa, Italy. The test logs and additional information can be found in the project site at networking laboratory [3]. The experiment measures the overall performance of audio sessions using VoIP in Ad hoc wireless LAN environment.

The VoIP client is running in both Laptops and iPAQs and additionally they require the following software: a GSM library [4] and an RTP library [5]. The tests are performed with two different routing protocols; OLSR and ADOV. We analyze the routing protocol effect in the overall performance.

7.3.4. Testbed objectives

The objective of the testbed is to measure the overall performance of VoIP sessions in ad hoc wireless LAN networks. VoIP applications have to deal with new requirements because of the mobility and nature of Ad hoc networks. The Ad hoc routing protocols do not affect the VoIP sessions after the route is established. However, the routing protocols have to ensure reliable routes and they have to react quickly to route changes to provide a smooth VoIP service.
The testbed analyses the VoIP service from an end-to-end point of view and test different situations to see the effect of certain parameters in the overall performance.

The main parameters considered in the testbed to analyze the performance of VoIP service in ad hoc networks, in terms of QoS, signaling overhead, etc, are the following.

- Routing protocol; OLSR or AODV.
- Receiver jitter buffer length.
- RTP payload length.

The two routing protocol selected are a proactive routing protocol (i.e. OLSR), and a reactive routing protocol (i.e. ADOV). These two routing protocols have been chosen to analyze the signaling overhead, the route stability and the reaction to link breaks provided by each protocol.

The Jitter buffer length has two main consequences from audio session quality point of view. Increasing the jitter buffer length will reduce the perceived pauses in audio playback. This results in smooth playback but will increase the over all delay. ITU-T recommends a maximum delay of 400 ms, and 250 ms for a quality audio session. A high delay is translated in a bad experience for the user. On the other hand, if we reduce the jitter buffer, the overall delay is reduced. However the overall result is a low quality session with lots of pauses in the playback. These two extremes result in an annoying and difficult to understand conversation. Therefore the jitter buffer is a balance between these two extremes.

The other parameter consists of the RTP payload length, which consists of the amount of audio data inserted on each RTP message. If we increase the RTP payload (i.e. we increase the amount of audio data in each RTP packet), then we enhance the audio playback at the receiver since each packet holds enough audio data to play until the next packet arrives. Nevertheless, increasing the payload means longer time to record resulting in higher over all delay. In addition, if one packet is lost, a larger mount of audio data is lost resulting in longer pauses in the playback.

7.3.5. Testbed Metrics

We have collected an exhaustive amount of data from the tests in order to analyze the performance.

Table 7.7. Transmitter Metrics

Statistics	
RTP Traffic %	The percentage of RTP bytes sent versus the total amount of bytes captured at the transmitter.
GSM data %	The percentage of GSM bytes sent versus the total bytes captured. The GSM packet length is 33 bytes.
Signaling to Capacity Overhead %	The percentage of bytes related to the routing protocols (i.e. OLSR or AODV) sent and received versus the total number of bytes captured.
Signaling Overhead %	The percentage of bytes related to the routing protocols sent and received versus the total number of GSM bytes successfully sent.
Succ GSM packets	Total number of GSM packets successfully sent.
Succ GSM bytes	Total number of GSM bytes successfully sent.
Succ Audio time	Total audio data sent. This is equal to the successful GSM packets sent multiplied by 20ms of audio samples sent per each GSM packet.
Succ Audio %	The percentage of total amount of audio data sent during the test.

Table 1.8 Receiver Metrics.

Statistics	
RTP Traffic %	The percentage of RTP bytes received versus the total amount of bytes captured at the receiver.
GSM data %	The percentage of GSM bytes received versus the total amount of bytes captured at the receiver.
Lost Packets	Number of RTP packets lost calculated from gaps in the sequence number.
Succ packets	Number of RTP packets received correctly without duplicates or errors.
Duplicates	Number of duplicate RTP packets received.
Duplicate %	The percentage of duplicate RTP packets received versus to the

	total amount of RTP packets received.
Signaling to capacity Overh. %	The percentage of bytes related to the routing protocol (OLSR/AODV) sent and received versus the total amount of bytes captured.
Signaling overhead %	The percentage of bytes related to the routing protocol sent and received versus the total amount of GSM bytes successfully received.
Succ GSM packets	Total number of GSM packets successfully received.
Succ GSM bytes	Total number of received GSM bytes.
Succ Audio time	The total audio received. This is equal to the successful GSM packets received multiplied by 20ms of audio samples sent on each GSM packet.
Succ Audio %	The percentage of total amount of audio data received during the test.

Table 7.9 End to End statistics.

Statistics	RTP
Sender	
Highest Seq Num	Highest sequence number generated by the sender.
Representing (sec)	Time equivalent to the highest sequence number.
Receiver	
Highest RX Seq Num	Highest Sequence number received.
Lost RX	The amount of RTP packets lost counted from gaps the in sequence number of the RTP packets received.
Duplicate RX	The amount of duplicate RTP packet received.
Succ RX	The amount of RTP packets successfully received.
Received (sec)	The amount of audio successfully received.
Link	
Lost in Link	The amount of packets lost in the link.
QoS	
Overall	The percentage of RTP packets successfully received versus the total amount of RTP packets generated.
Loss in Link	The percentage of RTP packets lost in the link versus the total amount of RTP packets transmitted.
Time lost	The amount of audio samples (i.e. in ms) lost.
Time lost %	The percentage of audio lost.
Jitter	The Jitter measured at the receiver during the test.
Jitter, during cont period	The Jitter measured in the receiver during a continuous reception time.

7.3.6. Testbed layout

The test was carried out at the IIT-CNR laboratory and it included four nodes (i.e. 2 iPAQs and 2 laptops). The gray areas in Figure 7.23 represent rooms, and the white areas are the corridors where we carried out the tests. The numbers beside each node represent the last part of IP address in 10.0.0.X. For example, the number 6 means IP address 10.0.06. The diagram also shows where we used iPaqs and laptops. We carried out several tests with this Topology only.

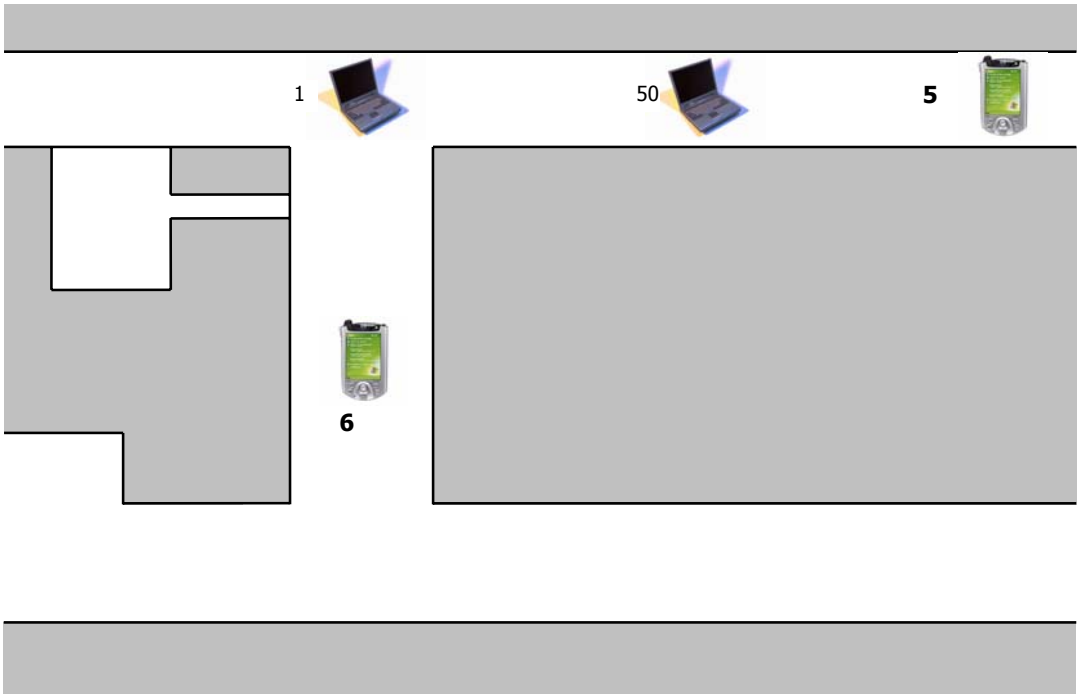


Figure 7.23. Testbed layout

The testing equipment used are:

1. Two HP 3850 iPAQs, running Familiar linux distribution. They are equipped with PCMCIA wlan cards.
    a. 206 MHz Intel StrongARM processor
    b. 64 MB memory
    c. http://h18000.www1.hp.com/products/quickspecs/10977_na/10977_na.HTML
    d. Familiar project: http://familiar.handhelds.org/
    e. VoIP client version 1, JRTP Lib version 2.9, GSM codec version 06.10.
    f. AODV version
    g. OLSR version
2. Two IBM laptops
    a. AODV version
    b. OLSR version

The iPAQs has version 1 of RTP client. This client utilizes the following components:

- GSM library version 06.10 (http://kbs.cs.tu-berlin.de/~jutta/toast.html)
- JRTPLib version 2.9 (http://research.edm.luc.ac.be/jori/jrtplib/jrtplib_old.html)

Test conditions

The testing environment was extremely noisy since the laboratory had several wireless networks. There are electromagnetic and a cellular-network labs in the vicinity of the testing area. The wireless link between two iPAQs having a line of sight path was quite bad even at short distances. Trying to ping between those two iPAQs will result on 10%-15% packet loss, even at short distances. We faced a great deal of troubles just trying to setup the topology to make sure that each node sees the next one only, and the link between them was good enough.

7.3.7. Test Cases

The tests consist of setting up a VoIP session between nodes 5 and 6 while changing the session parameters (i.e. routing protocol, GSM buffer, RTP payload) to analyze the performance. The network topology as shown in 7.23 consists of two iPAQs separated by 2 laptops. The distance between two nodes was between 5 to 10 meters and each node cannot see more than the next one.

Test Number	5.1	6.1	7.1	8.1
Protocol	OLSR	OLSR	AODV	AODV
GSM Buffer	60 msec	100 msec	60 msec	100 msec
GSM/RTP	3	3	3	3

Test Case 5.1

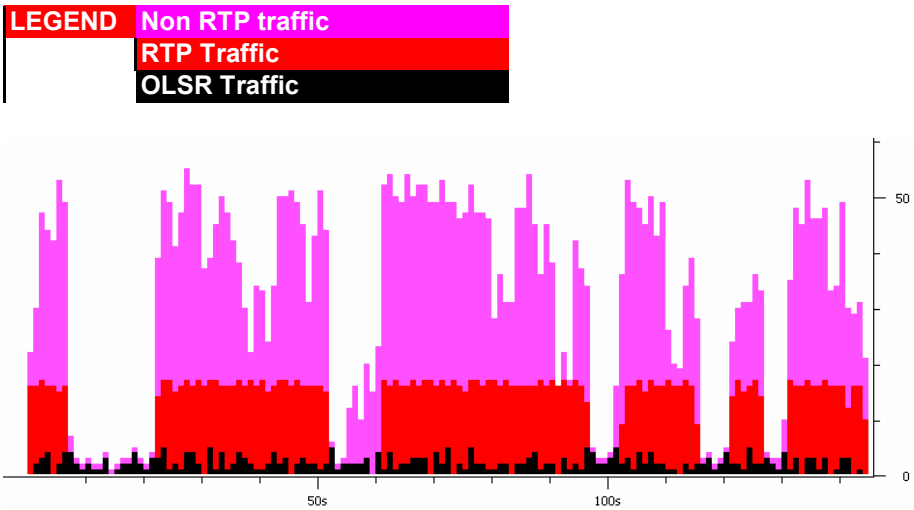


Figure 7.24. Traffic measurements test 5.1.

Statistics

From Node 5 to Node 6

Statistics	RTP
Sender	
Highest Seq Num	2379
Representing (sec)	142.74
Signaling Over head	12.64%
Receiver	
Highest RX Seq Num	2379
Lost RX	949
Duplicate RX	0
Succ RX	1430
Representing (sec)	85.8
Signaling Over head	6.10%
Link	
Lost in Link	949
QoS	
Overall	60.11%
Loss in Link	39.89%
Time lost	56.94
Time lost %	39.89%
Jitter	0.52708
Jitter, during cont period	0.52708

From Node 6 to Node 5

Statistics	RTP
Sender	
Highest Seq Num	3108
Representing (sec)	186.48
Signaling Over head	14.82%
Receiver	
Highest RX Seq Num	2838
Lost RX	1269
Duplicate RX	76
Succ RX	1493
Received (sec)	89.58
Signaling Over head	18.73%
Link	
Lost in Link	1615
QoS	
Overall	48.04%
Loss in Link	51.96%
Time lost	96.9
Time lost %	51.96%
Jitter	0.617955
Jitter, during cont period	0.182292

Observations

There were a lot of link breaks and the percentage of packets lost in the link is 40%. The end-to-end delay was low but the traffic showed an asymmetric behavior (i.e. traffic going from node 6 to 5 where totally lost).

Test Case 6.1

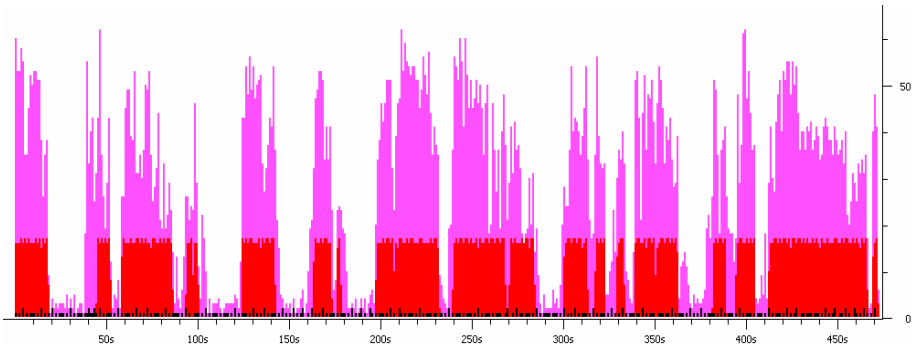


Figure 7.25. Traffic measurements test 6.1

Statistics

From Node 5 to Node 6

Statistics	RTP
Sender	
Highest Seq Num	7758
Representing (sec)	465.48
Signaling Over head	14.64%
Receiver	
Highest RX Seq Num	7758
Lost RX	3602
Duplicate RX	0
Succ RX	4156
Received (sec)	249.36
Signaling Over head	20.65%
Link	
Lost in Link	3602
QoS	
Overall	53.57%
Loss in Link	46.43%
Time lost	216.12
Time lost %	46.43%
Jitter	0.869446
Jitter, during cont period	0.045297

From Node 6 to Node 5

Statistics	RTP
Sender	
Highest Seq Num	7062
Representing (sec)	423.72
Signaling Over head	15.96%
Receiver	
Highest RX Seq Num	7062
Lost RX	3777
Duplicate RX	132
Succ RX	3153
Received (sec)	189.18
Signaling Over head	18.88%
Link	
Lost in Link	3909
QoS	
Overall	44.65%
Loss in Link	55.35%
Time lost	234.54
Time lost %	55.35%
Jitter	0.758336
Jitter, during cont period	0.083759

Observations

Link breakage is more frequent and shorter in this experiment.

Test Case 7.1

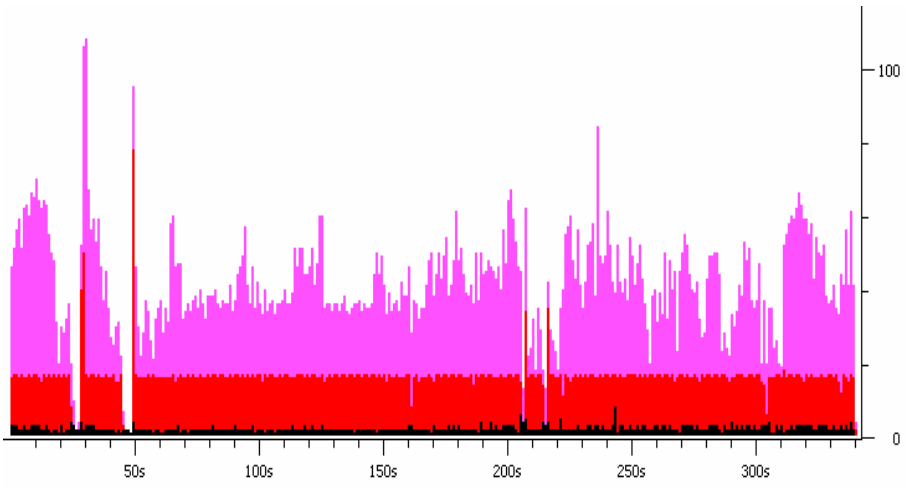


Figure 7.26 Traffic measurements test 7.1

Statistics

From Node 5 to Node 6

Statistics	RTP
Sender	
Highest Seq Num	5574
Representing (sec)	334.44
Signaling Over head	8.95%
Receiver	
Highest RX Seq Num	5574
Lost RX	282
Duplicate RX	59
Succ RX	5233
Received (sec)	313.98
Signaling Over head	8.69%

From Node 6 to Node 5

Statistics	RTP
Sender	
Highest Seq Num	5467
Representing (sec)	328.02
Signaling Over head	8.62%
Receiver	
Highest RX Seq Num	5467
Lost RX	3621
Duplicate RX	19
Succ RX	1827
Received (sec)	109.62
Signaling Over head	28.30%

Link	
Lost in Link	341
QoS	
Overall	93.88%
Loss in Link	6.12%
Time lost	20.46
Time lost %	6.12%
Jitter	0.107875
Jitter, during cont period	0.073521

Link	
Lost in Link	3640
QoS	
Overall	33.42%
Loss in Link	66.58%
Time lost	218.4
Time lost %	66.58%
Jitter	0.700415
Jitter, during cont period	0.151487

Observations

This test case is almost the best one we had. The lost in link packets percentage is very low (6%). The delay was very acceptable, the quality was clear.

Test Case 8.1

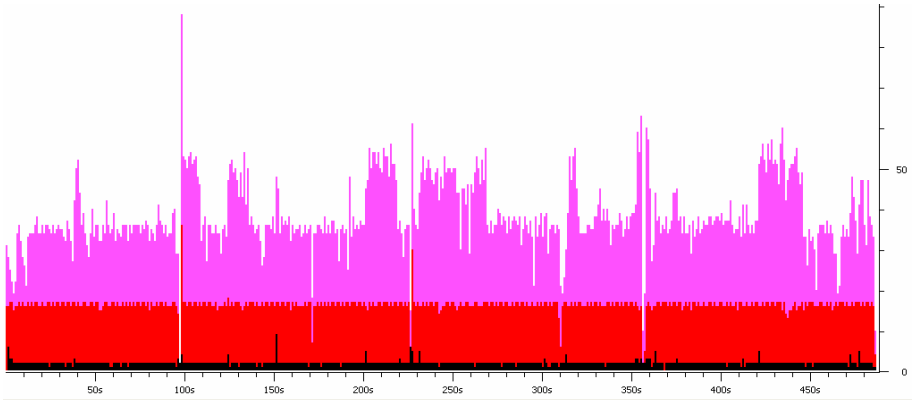


Figure 7.27 Traffic measurements test 8.1

Statistics

From Node 5 to Node 6	
Statistics	RTP
Sender	
Highest Seq	7974

From Node 6 to Node 5	
Statistics	RTP
Sender	
Highest Seq	8547

Num	
Representing (sec)	478.44
Signaling Over head	7.91%
Receiver	
Highest RX Seq Num	7974
Lost RX	1399
Duplicate RX	433
Succ RX	6142
Received (sec)	368.52
Signaling Over head	10.46%
Link	
Lost in Link	1832
QoS	
Overall	77.03%
Loss in Link	22.97%
Time lost	109.92
Time lost %	22.97%
Jitter	0.579396
Jitter, during cont period	0.055566

Num	
Representing (sec)	512.82
Signaling Over head	9.00%
Receiver	
Highest RX Seq Num	8543
Lost RX	5755
Duplicate RX	37
Succ RX	2751
Received (sec)	165.06
Signaling Over head	25.67%
Link	
Lost in Link	5796
QoS	
Overall	32.19%
Loss in Link	67.81%
Time lost	347.76
Time lost %	67.81%
Jitter	1.8566112
Jitter, during cont period	0.1595774

The tests were repeated changing the following variables to determine the optimum audio parameters.

1. Size of audio device buffers.
2. Number of audio device buffers.
3. RTP payload length
4. Code timing.

Test Number	Audio buffers	Buffer size (byte)
1	2	1024
2	4	512
3	8	512
4	4	1024

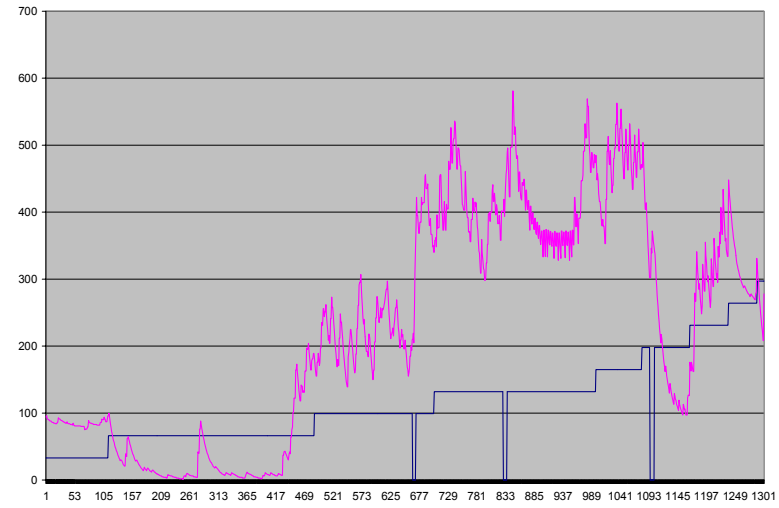
Test Case 1

Test with 2 iPAQs with no middle nodes and with the following RTP parameters.

- RTP initial jitter buffer length: 60 msec.
- Maximum payload length: 10 GSM Packets

- Using audio device for recording and playback (not files)
- 2 Audio buffers, each 1024 bytes long

The payload length was dynamically changed during the call.

GSM packets (RTP Payload length)	Observation
1	Very bad, too much popping sound, a lot of discontinuity, voice can not be understood
2	Acceptable, low delay, infrequent popping sound, voice clear and of good quality
3	Acceptable, low delay, no popping sound at all, voice understandable, clear and of high quality
4	Acceptable, acceptable delay, clear and understandable voice of high quality
5-7	Acceptable, somewhat acceptable delay, clear and understandable voice of high quality
8-10	Very good quality, significant delay, if a packet is lost, there is a long pause in voice, therefore, the need to ask to repeat what was said is noticed.
The best jitter values are when the payload length is 2. It then grows too much as the payload length increases.	 Jitter plot, also shown is the payload length

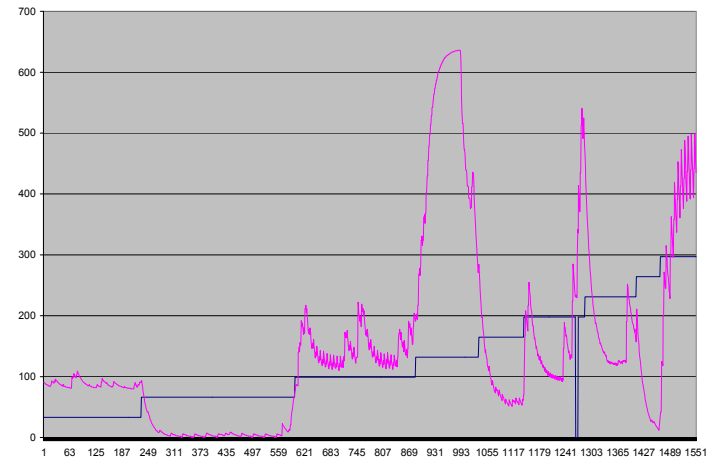
Test Case 2

Test with 2 iPAQs with no middle nodes and with the following RTP parameters.

- RTP initial jitter buffer length: 60 msec.
- Maximum payload length: 10 GSM Packets
- Using audio device for recording and playback (not files)

- 4 Audio buffers, each 512 bytes long

The payload length was dynamically changed during the call. The delay with these smaller buffers was lower than the case with larger buffers.

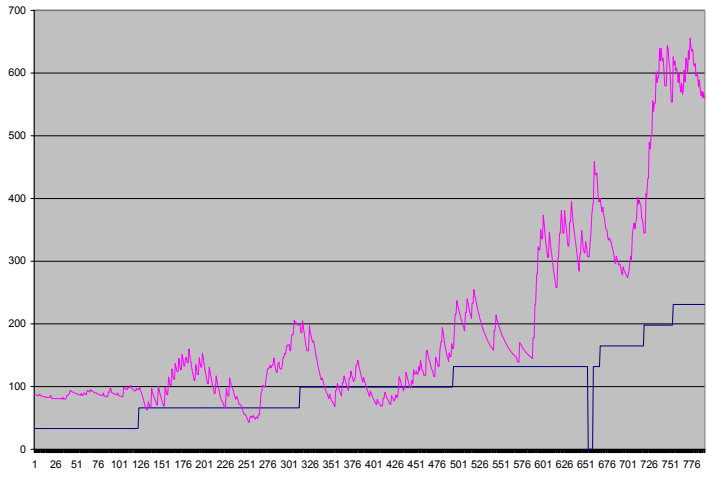
GSM packets (RTP Payload length)	Observation
1	Very bad, too much popping sound, a lot of discontinuity, voice can not be understood
2	Very good, low delay, infrequent popping sound, voice clear and of good quality
3	Excellent, low delay, no popping sound at all, voice understandable, clear and of high quality
4	Excellent, acceptable delay, clear and understandable voice of high quality
5-7	Excellent, somewhat acceptable delay, clear and understandable voice of high quality
8-9	Very good quality, significant delay, if a packet is lost, there is a long pause in voice, therefore, the need to ask to repeat what was said is noticed.
The jitter is the best for payload length 2. 3 is also acceptable	

Test Case 3

Test with 2 iPAQs with no middle nodes and with the following RTP parameters.

- RTP initial jitter buffer length: 60 msec.
- Maximum payload length: 10 GSM Packets
- Using audio device for recording and playback (not files)
- 8 Audio buffers, each 512 bytes long

The payload length was dynamically changed during the call. The delay with the smaller buffers was lower than the case with the larger buffer.

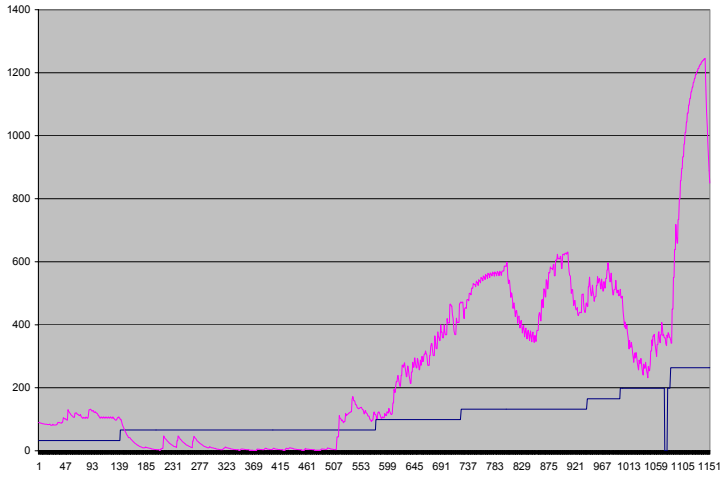
GSM packets (RTP Payload length)	Observation
1	Very bad, too much popping sound, a lot of discontinuity, voice can not be understood
2, 3	Very good, low delay, infrequent popping sound, voice clear and of good quality
4-9	Bad quality, strange high volume popping sound, with a constant poping frequency that decreases as the payload increases. The delay increases from 4 to 10.
By far the worst setting. The jitter values are never as good as any other experiment.	

Test Case 4

Test with 2 iPAQs with no middle nodes and with the following RTP parameters.

- RTP initial jitter buffer length: 60 msec.
- Maximum payload length: 10 GSM Packets
- Using audio device for recording and playback (not files)
- 4 Audio buffers, each 1024 bytes long

The payload length was dynamically changed during the call. The delay with these smaller buffers was lower than the case when the buffers are larger.

GSM packets (RTP Payload length)	Observation
1	Very bad, too much popping sound, a lot of discontinuity, voice can not be understood
2, 3	Very good, noticeable delay, voice clear and of good quality
4-9	Good audio quality, but high delay.
Note that when payload length goes over 4, the jitter becomes significant.	

7.3.8. Results and Analysis

From the tests we deduce that choosing a payload length of only 1 GSM packet per RTP packet is not a good choice at all. The results in all the cases were a high popping sound, unclear voice and a large number of short pauses.

The best audio parameters settings were using 4x512 bytes buffers, and the payload length can be ranging from 2 to 5, 2 sometimes having popping sound, 3 being the best, and 5 with somewhat more delay but excellent quality. The selection was made to start any audio session with 3 GSM packets per payload.

7.3.9. Future work

There can be many areas of improvements for the VoIP client, among which:

- Testing against different types of codecs is essential. AMR is a good candidate for this.
- Deeper traces of audio device performance, in order to get a better control on the delay. Maybe a different kind of sound system should be used.
- Support of conference calls.
- Support for video.

7.3.10. References

1. H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 1889 Jan. 1996.
2. AODV-UU, Ad Hoc On Demand Distance Vector implementation created at Uppsala Universtity (http://user.it.uu.se/~henrikl/aodv/).
3. Helsinki University of Technology, Networking Laboratory (http://www.netlab.hut.fi/~mayyash/pisa/index.htm).
4. GSM library v06.10 (http://kbs.cs.tu-berlin.de/~jutta/toast.html)
5. RTP library v2.9 (http://research.edm.luc.ac.be/jori/jrtplib/)
6. Programmers Guide to OSS http://www.opensound.com/pguide/index.html
7. Ethereal user guide: http://www.ethereal.com/docs/user-guide/

7.3.11. Ethereal Analysis

Each test case and each section contains analysis, statistics and traffic graphs obtained using the Ethereal tool. It is worth mentioning that this tool was used to sniff everything being heard on the channel, whether it was sent or received to the current node, or some traffic between two other nodes that just happened to be heard on the current node.
The filters that were used in this tool to get the required plots and statistics are:

1. Time Limit

frame.time >= "Oct 1, 2004 20:24:26.502118" && frame.time <= "Oct 1, 2004 20:33:23.024789"

The time stamps are examples only. It is used to limit packet filtering between the starting and ending time stamps. The reason for limiting time periods is to analyze the relevant part of the experiment. During the execution of the experiment, there was a starting period of time when the testers were just getting ready and an ending period of time when the testers were finishing up. The logs reflects this as sniffed packets but with no RTP traffic. The time stamps were decided by looking at the logs as a whole then selecting the best period of time when actually talking was happening.

2. RTP With Time limit

ip.ttl==64 && udp.port == 55000 && ip.src == 10.0.0.6 && !icmp.type && frame.time >= "Oct 1, 2004 19:44:31.599465" && frame.time <= "Oct 1, 2004 19:53:28.418081"

This filter is used to list all sent RTP packets, in this example from node 6. It filters out RTCP, ICMP and those packets that are sniffed as being forwarded by the next node. When a packet is sent, the IP TTL field is set to 64, when it is received by the next node, TTL field is decremented to 63 and then transmitted to the following node, however, this

transmission is heard by the first node and logged in the TCP dump as a second copy of the same RTP packet except that IP TTL field is 63. This is important so as not to have double the number of RTP packets sent in the analysis.

3. Received RTP packets with IP TTL field 63

ip.ttl== 63 && udp.port == 55000 && ip.src == 10.0.0.6 && !icmp.type && frame.time >= "Oct 1, 2004 20:24:26.502118" && frame.time <= "Oct 1, 2004 20:33:23.024789"

In this example, the filter lists packets sent from node 6 to node 5 with IP TTL field 63, meaning those packets making the trip in two hope only. This happened sometimes because, for example, it was still possible for laptop 50 and node 6 to see each other directly. The filter also removes ICMP packets.

4. Received RTP packets with IP TTL field 62

ip.ttl==62 && udp.port == 55000 && ip.src == 10.0.0.6 && !icmp.type && frame.time >= "Oct 1, 2004 20:24:26.502118" && frame.time <= "Oct 1, 2004 20:33:23.024789"

Same as filter 3, except it filters for those packets making the trip in 3 hops.

5. Received RTP packets with IP TTL field 63 OR 62

(ip.ttl== 63 || ip.ttl==62) && udp.port == 55000 && ip.src == 10.0.0.6 && !icmp.type && frame.time >= "Oct 1, 2004 20:24:26.502118" && frame.time <= "Oct 1, 2004 20:33:23.024789"

A combination of filters 3 and 4. It gives a total idea about the received RTP traffic whether it made it in 2 or 3 hops.

6. AODV Traffic

aodv && frame.time >= "Oct 1, 2004 20:24:26.502118" && frame.time <= "Oct 1, 2004 20:33:23.024789"

This filter will list the AODV routing traffic between the two time stamps. It lists both received and sent AODV packets.

7. OLSR Traffic

olsr && frame.time >= "Oct 1, 2004 18:27:52.979226" && frame.time <= "Oct 1, 2004 18:30:18.468502"

This filter will list the OLSR routing traffic between the two time stamps. It lists both received and sent OLSR packets.

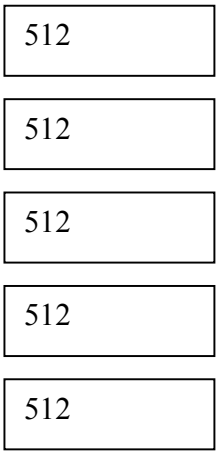
Audio device capabilities

The audio system model used is OSS (Open Sound System). This system provides audio buffers for playback and recording. They are filled sequentially upon request. The code can specify how many buffers the system should use and how large each buffer is.

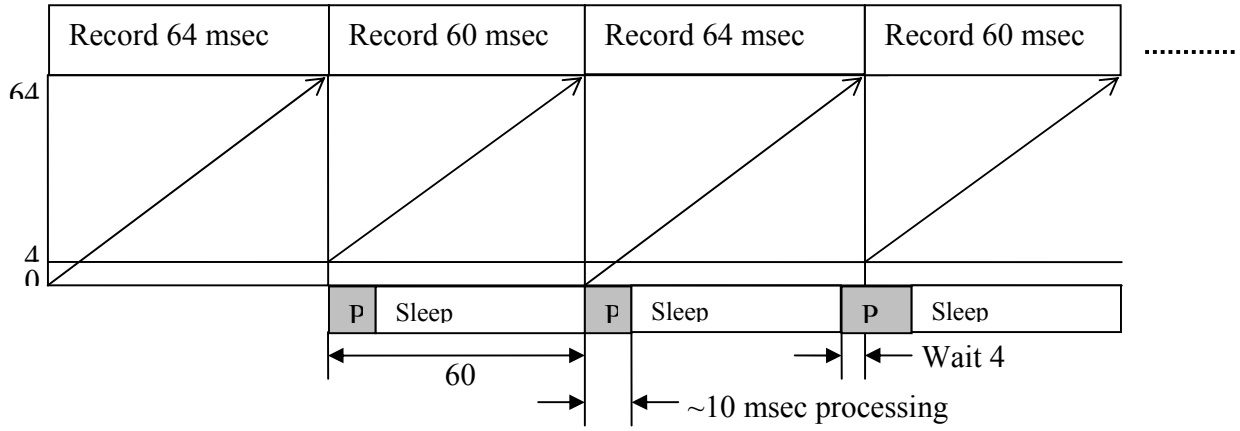
It is important to keep the audio buffer segments small specifically in realtime applications [OSS manual page 97]. Following the recommendation of OSS manual, very short segments will not be used [OSS manual page 98]. Every buffer under run will result in a popping sound [OSS manual page 98]

Since GSM encoding requires 13 bits per sample, 8000 sample per second (8KHz), the audio device is configured to record 16 bits sample at 8KHz mono stream. The GSM library used accepts 16 bits per sample audio steam and handles that correctly to encode and decode GSM packets. In other words, the incompatibility between 16bit audio on iPAQ and the 13bit GSM codec requirements is wrapped and solved inside the GSM

codec library. Since each GSM packet represents 20 msec of audio, this gives 160 samples at 8KHz 160 (sample) = 20/1000 (sec) * 8000 (sample/sec)
Each sample is 16 bit, hence the block of bytes to be requested from the audio device each time a GSM packet is to be sent is 2 * 160 = 320 bytes. Most of our trials should that the best situation is 3 GSM packets, i.e. 960 Bytes.
The best buffer setup, as observed from the trials, is 4x512 bytes audio buffers. The observation is that the more fragmented the audio buffer, the better achieved performance, but some limits apply.
First, the larger one segment means it will introduce buffers of the audio device looks like:



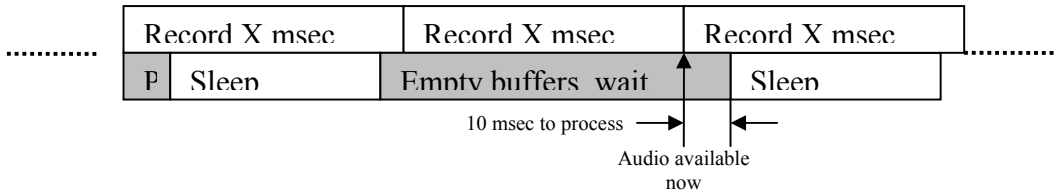
The software starts by requesting 960 bytes from the audio device, and waits until they are available. Then it starts encoding and sending them, meanwhile, it keeps the audio device busy recording the next audio data. On average, the measured time the code takes to encode and send 60 msec of audio is 10 msec.
The audio device will have to record full segments before delivering back the audio data [OSS manual page 95]. It will record 2 segments full of 512 bytes. 512 bytes equals 256 sampls, each 16 bit. That is equivalent to 256/8000 = 32 msec. For the audio device to record 2 of those segments, it requires 64 msec. It is there where the misalignment between the request audio time by the code and the audio time the audio device must spend in delivering it. The code is asking for 60 msec of audio, but the audio device can deliver that in 64 msec, of course, after recording 64 msec worth of audio, not just 60 msec. It is not possible to interrupt the audio device while recording once the 60 msec are available, and it is not possible to retrieve them once they are available, the audio device must finish the current segment. The cycle should look like this:



P: Processing: Reads 60msec worth of bytes, leaves the rest. Then it encodes the GSM packets, packs them in RTP packets, and then sends them. Typically, 10 msec is needed.

Note how the 4 msec wait is to be added to every cycle every 2 cycles. First the audio device records 64 msec, then the code takes out 60 msec and leaves 4, the second cycle, the audio device has to record 60 msec only, since there are already 4 left from last cycle. The code in this case will wake up in exactly the same time when the audio device finishes recording, and will be able to take 60 msec immediately. It leaves the buffer segments totally empty. Now the cycle repeats, the audio device will spend 64 msec recording, after 60 msec, the code wakes up to request 60 msec, but it has to wait 4 msec until recording is complete.

Ideally, the code should expect enough full segments to send when it wakes up. A delay is introduced if that does not happen. If the code wakes up and the audio device for some reason is still recording, or if the code wakes up and there are not enough full fragments, then a call from the code to the audio device to get the recorded audio will be blocked until the audio device can fill enough segments and deliver the requested audio. This is added delay to the sending time, and in the worst case, specially of the audio device is configured in a different way that will not allow the aforementioned cycle, it will be the time to record 2 segments, which is the reason why sometime transmission deadlines are missed. A solution to this problem is the jitter buffering at the receiver side. The following diagram illustrates:



P: Processing: Reads 60msec worth of bytes, leaves the rest. Then it encodes the GSM packets, packs them in RTP packets, and then sends them. Typically, this takes 10 msec

8. MOBILEMAN MEDIUM SCALE TEST BED

An extensive experimentation of MobileMAN solutions was carried out in Pisa at CNR (Italian National Research Council) last summer in five different days: 29 May, 3 June, 11 June, 17 June and 23 July 2005. The participants of MobileMAN Experimentation were selected from students of Computer Engineering of University of Pisa. Since we have received many requests to participate in this experimentation, a selection between candidates based on technical skills was necessary. In particular, we were interested in selecting students with a basic knowledge of Linux operating system and the ability to configure wireless cards in ad hoc mode. Moreover, we chose students that were willing to use their own laptops during the experimentation in order to obtain a heterogeneous configuration of the network. Specifically, the following 20 students were selected among Bachelor's degree candidates (first group) and Master's degree candidates (second group):

- Gaetano Anastasi, Giovanni Bianchi, Roberto Corradi, Marco D'Alò, Danilo Levantesi, Fabrizio Lovino, Damiano Macchi, Matteo Mattei, Luca Melette, Luca Niccolini, Mario Olivari, Stefano Pallicca
- Annalisa Bizzarrini, Chiara Boldrini, Edoardo Canepa, Mario Di Francesco, Salvatore Gerace, Ilaria Giannetti, Iacopo Iacopini, Giacomo Santerini

These two groups of people, coordinated by CNR research assistants, Eleonora Borgia and Franca Delmastro, set up in CNR campus a MobileMAN network involving up to 23 nodes.

The experimentation focused on the analysis of different layers of the protocol stack to compare a legacy architecture performance with those of a cross-layer architecture. In particular we focused on:

- a comparative performance analysis of two different routing protocols (OLSR and AODV) both in static and mobile scenarios;
- a comparative analysis of two different middleware platforms (Pastry for the legacy architecture and CrossROAD for the cross-layer architecture). In case of Pastry an open source implementation called FreePastry has been used on top of the two routing protocols. On the contrary, CrossROAD, which requires a proactive routing protocol, ran exclusively on top of OLSR. In addition its implementation has been enhanced with a cross-layer service discovery protocol. Both systems were analyzed in static and mobile scenarios.

A great number of experiments had been conducted during this experimentation. A detailed description of all of them is available in Appendix A. In following sections the most meaningful experiments are described and main results on system performance are detailed.

8.1. Experimental Environment

All the experiments took place at the ground floor in CNR campus in Pisa. Since more than 20 nodes were involved in this experimentation, a wide area has been used for testing a medium scale network. Hence, in addition to the CED Area used in previous experimentations, the Conference Area located in the adjacent building was also used (see Figure 8.1). The structural characteristics of these buildings strictly determine the transmission capabilities for nodes of a wireless network located within. Rooms are generally delimited by masonry padding walls situated between reinforced concrete pillars; in addition, in the CED area some locations are separated by either "sandwich panels" of plastic materials which don't reach the height of the ceiling, or metal panels till the ceiling. Wireless links are also influenced by the presence nearby of Access Points and measurement instrumentations which introduce quite a lot of noise.
Moreover, about 30-40 people work in this floor every day and get around from office to office or towards service areas with coffee machines, toilets, etc. This makes the transmission coverage characteristics of the floor and the stability of the links modify continuously and in an unpredictable manner. For this reason all the experiments were executed during Saturday or non-working days to reduce human interferences maintaining a realistic environment to test an ad hoc network.

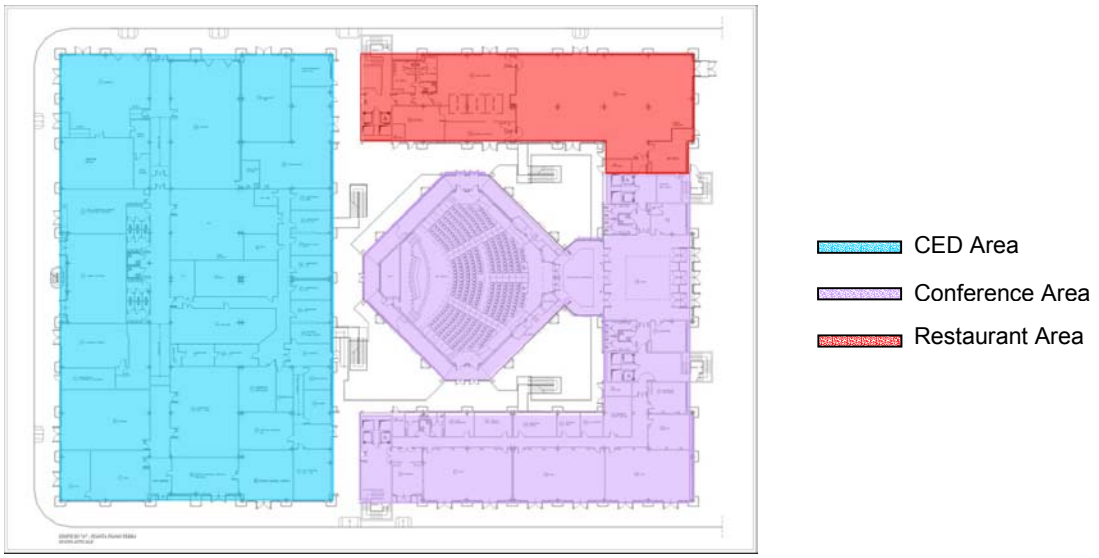


Figure 8.1: Experiment Area

Devices

Devices used for these experiments were laptops running Linux with different hardware capabilities. They were equipped with wireless cards compliant to IEEE 802.11b standard working at the constant data rate of 11Mbps. Most laptops were equipped with an integrated wireless card, while for the others PCMCIA cards were used. The variety of devices caused appearing/disappearing of some links in different experiments, depending

on the power of wireless cards. In the following list, devices are referred to as numbers corresponding to the last byte (digit) of their IP address assigned during the experimentation.

Laptops

.1:
Model: IBM ThinkPad R40 Series - Centrino® Mobile Technology
Processor: Intel® Pentium® M - 1300MHz
Wireless LAN PC Card: D-Link DWL 650

.20:
Model: IBM ThinkPad R40 Series - Centrino® Mobile Technology
Processor: Intel® Pentium® M - 1300MHz
Wireless LAN PC Card: Intel PRO/Wireless 2100 Integrated wireless card

.40:
Model: IBM ThinkPad R40 Series
Processor: Mobile Intel® Pentium® 4 - 2 GHz
Wireless LAN PC Card: D-Link DWL 650

.52, .53,.54, .55, .56, .57, .58, .59, 111:
Model: IBM ThinkPad R50 Series
Processor: Mobile Intel®– 2.000GHz
Wireless LAN PC Card: Intel PRO/Wireless 2200 Integrated wireless card

.100:
Model: Asus LK8470
Processor: Intel® Pentium® 3 – 1.13 GHz
Wireless LAN PC Card: D-Link DWL 650

.104:
Model: ECS G553 - Centrino ® Mobile Technology
Processor: Intel® Pentium® M – 1.6 GHz
Wireless LAN PC Card: Intel PRO/Wireless 2200 Integrated wireless card

.105:
Model: ASUS L8400L
Processor: Intel® Pentium® 3 – 800 MHz
Wireless LAN PC Card: Sitecom

.106:
Model: CDC Premium HI BRITE
Processor: Intel® Pentium® M – 1.6 GHz
Wireless LAN PC Card: Intel PRO/Wireless 2200BG Integrated wireless card

.107:
Model: HP Pavilion ze4932ea
Processor: Intel® Pentium® M - 1.4 GHz
Wireless LAN PC Card: Broadcom 802.11b/g 54

.108:
Model: ASUS Z8000
Processor: Mobile Intel® Pentium® 4 – 3 GHz
Wireless LAN PC Card: Broadcom Integrated wireless card

.109:
Model: ASUS z8100 A4D
Processor: Amd 3000+
Wireless LAN PC Card: Sitecom WL100
.110:
Model: Acer Travelmate 292 LMi
Processor: Mobile Intel® Pentium® 4 - 1.5 GHz
Wireless LAN PC Card: Intel 802.11b/g Integrated wireless card
.112:
Model: ASUS a28
Processor: Mobile Intel® Pentium® 4 - 3.4 GHz
Wireless LAN PC Card: Broadcom Integrated wireless card
.113:
Model: HP pavilion ze4500
Processor: AMD Athlon XP 2500+
Wireless LAN PC Card: Sitecom WL-112
.115:
Model: Acer Travelmate 8104 WLMi
Processor: Intel® Pentium® M – 2 GHz 2
Wireless LAN PC Card: Intel PRO/Wireless 2915ABG Integrated wireless card

Software

As in the MobileMAN experimentation on small-scale ad hoc networks, the main goal of this work was to test different implementations of protocols at different layers. In particular routing, middleware and application solutions were considered using more recent software versions. We divided all experiments in two main groups based on the layer of interest.

The first set of experiments focused on evaluating a reactive and a proactive routing protocol. We used UU-AODV v.0.8. [1], developed by Uppsala University (Sweden), as reactive protocol. On the other hand we used UNIK-OLSR v.0.4.8 [2], developed by University of Oslo (Norway) as proactive protocol.

The second set of experiments focused on evaluating performance of two middleware platforms on this real ad hoc network of 23 nodes. First of all we analyzed FreePastry, starting from previous experimental results presented in [3]. Then we evaluated CrossROAD [4] performance, exploiting a first prototype of cross-layer architecture based on UNIK-OLSR that has been already presented in [4]. The software implementation of Pastry used in these experiments was FreePastry-1.3 [5], an open-source implementation developed by the RICE University, while CrossROAD has been entirely developed by the IIT Institute of CNR in Pisa. All laptops have been equipped with the j2sdk-1.4.02 [6] Java Virtual Machine. In addition, each type of experiment related to Pastry was repeated running UNIK-OLSR and UU-AODV to completely

evaluate the system performance and the impact of these two different routing protocols on a real ad hoc network of such dimensions.

A simple application of Distributed Messaging (DM) was installed on top of both systems, aimed at creating the overlay network and distributing data. This application implements the P2P common API originally proposed in [7] and also developed by FreePastry. Specifically, the application developed for FreePastry 1.3 implements main features of the original definition, while in case of CrossROAD a cross-layer enhancement of the same commonAPI (XL-CommonAPI) has been defined in [8] and completely developed. The main feature of these interfaces consists of defining a common interface between applications and middleware platforms based on structured overlay networks, to guarantee the portability of several services on different overlays. In particular, in case of XL-CommonAPI, cross-layer information mainly comes from the routing protocol, and they are exported to applications to optimize their behaviour on top of ad hoc networks.

DM specifies the IP address of the local node as its logical identifier. Its numerical representation strictly depends on the hash function chosen by the underlying system (e.g. FreePastry and CrossROAD use the SHA-1 hash function, but FreePastry also adds a random quantity to the final value). In case of Pastry, the application recovers the IP address of the bootstrap node from a configuration file, since the local node must know one of its neighbours to join the overlay. If no bootstrap node is specified, the local node creates a new overlay. The bootstrap procedure is needed by Pastry to initialize overlay data structures used to communicate with other nodes of the system. Then, periodical monitoring procedures are used to update the same data structures. On the other hand, CrossROAD does not need any bootstrap procedure. The cross-layer service discovery protocol spreads and collects services information exploiting the proactive flooding of the routing protocol. Hence, CrossROAD autonomously builds its overlay collecting IP addresses of nodes providing the same service, and applying the hash function to their values. No bootstrap nodes are required, and the overlay data structures are initialized and updated with the same frequency of the routing protocol updates which contain the services information.

In both cases, once the local node has created/joined the overlay, the application allows users to create/delete a mailbox, and to send/receive messages to/from a specific mailbox. The subject-based routing of these systems requires that a key is assigned to each message. The key represents a logical identifier of the overlay so that the related message is sent to the node logically closest to the value of the key. For this reason, in order to create a mailbox and store it on a node of the system, users have to specify an identifier that represents the key of the “CREATE” message. Then they can send and receive messages from the same mailbox specifying its identifier.

To make experiments as much automatic as possible, a set of scripts has been developed and the application has been enhanced with some features that allow the periodical generation of messages with random keys, guaranteeing the data distribution on all nodes of the network.

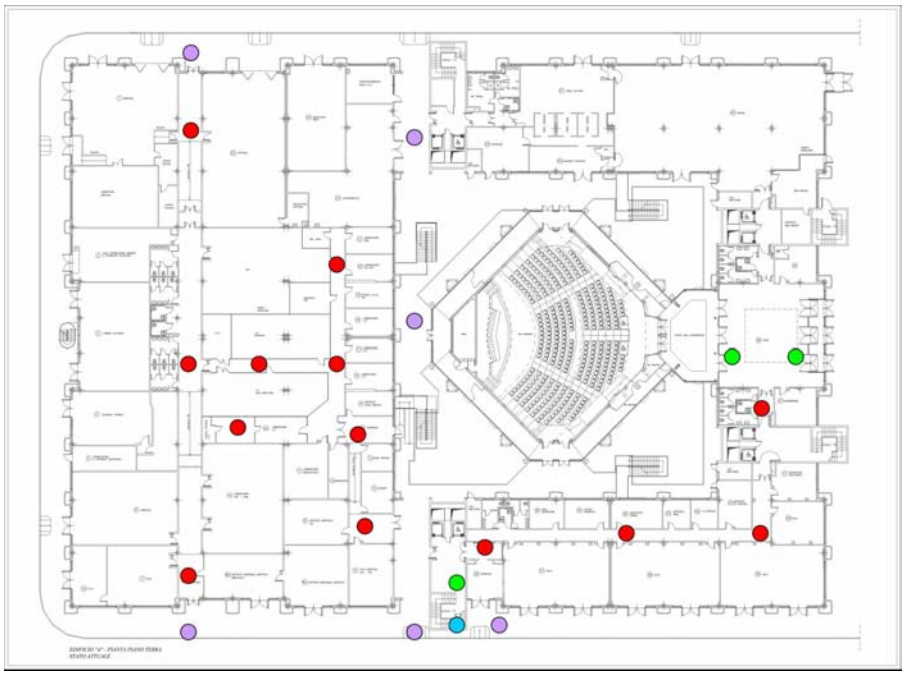


Figure 8.1. Physical position of nodes.

8.2. The network topology

The first step to set up the ad hoc network and start investigating software features was configuring the network topology. We had 23 nodes to be distributed inside the CNR campus to carry out a multi-hop ad hoc network as much large, as possible. For this reason we used a heterogeneous environment consisting of indoor and outdoor spaces since not all buildings are strictly connected between them. We started from the same configuration used in the experimental session of July 2004 with 12 nodes, explained in [3]. Since we used a greater number of laptops with different capabilities (also for the transmission range of wireless cards), a new measurement of the link connectivity had to be done. In this case the interested area was extended from the CED area to the neighborhood of the conference area as shown in Figure 8.1. Most part of nodes (17) was located inside buildings. In particular we placed 13 at the ground floor (red circles), three at the first floor (green circles), and one on the stairs (the blue circle). The last six nodes were located outside the buildings (violet circles) along the street or the corridor between the involved buildings. In order to verify the coverage area of every device, each node started running UNIK-OLSR for five minutes storing the kernel routing table in a log file every second. Then, we analyzed the set of 1-hop neighbors of each node to define the final network topology. Considering a large multi-hop ad hoc network we could test and evaluate features and performance of a complete MANET architecture. For this reason, since many devices had a wireless card with a high transmission power, we had to reduce

it on single nodes (if allowed by the driver of the wireless card) to remove some redundant links. We repeated this procedure many times to check that the obtained configuration was stable. Figure 8.2 shows the final network topology, where straight lines point out the presence of stable links (two nodes directly see each other), dashed lines show the presence of weaker links (the communication between two nodes is affected by a considerable packet loss). We thus obtained a multi-hop MANET of 23 nodes with the maximum extension of eight hops. To simplify the explanation of single experiments, we referred to the network topology through the graph illustrated in Figure 8.3.

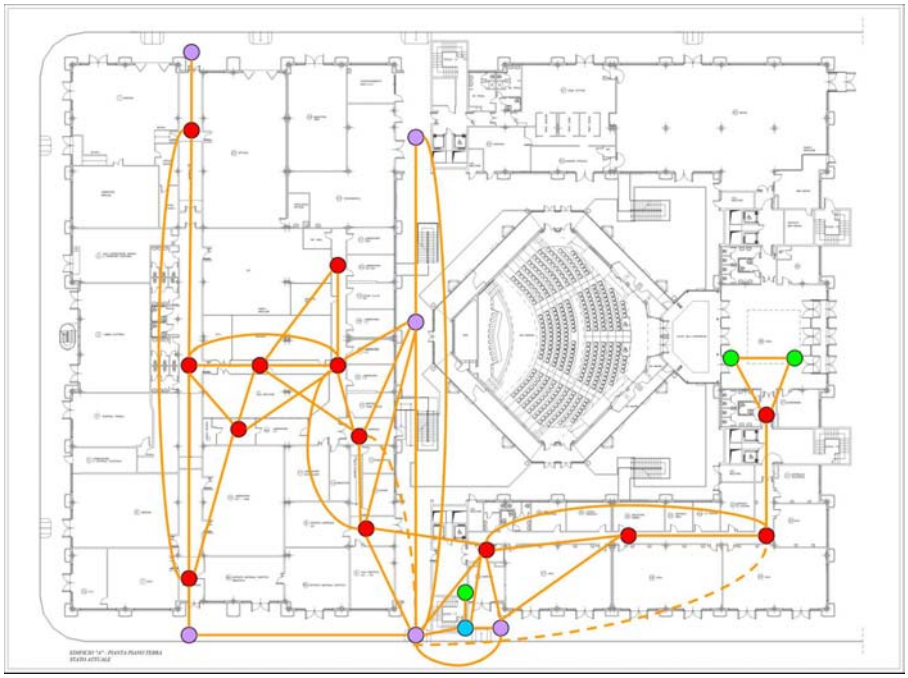


Figure 8.2. Network topology

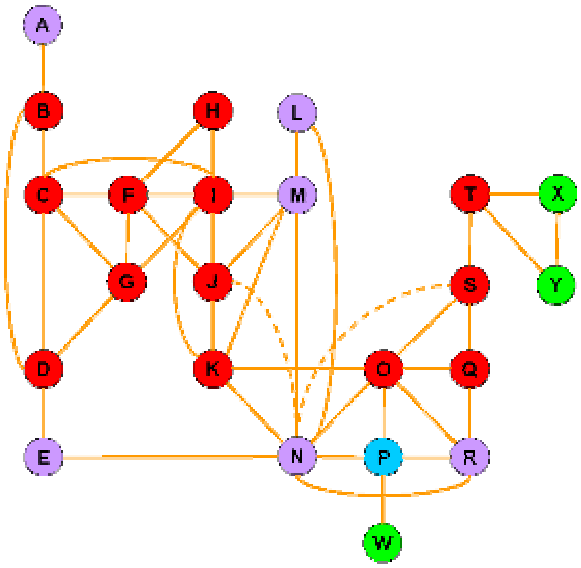


Figure 8.3. Topology graph

8.3. Routing Algorithms Experiments

The second step was investigating the performance of OLSR and ADODV routing protocols for MANET in *static* and *mobile* scenarios. We analysed several parameters to make a comparison between them. In particular we focused on:

- the Overhead introduced in the network due to routing messages
- the Packet Loss suffered at the application level
- the Delay introduced in data transfer

In mobile networks case, we concentrated the analysis on packet loss and average delay to analyze the network reconfiguration due to topology changes. In addition to routing protocols we introduced some application traffic using the Ping utility. This guarantees that AODV also runs in a complete manner, otherwise its routing information is reduced only to the exchange of Hello packets and no route is calculated.

We performed the following type of experiments (more details are in Appendix A):

a. STATIC SCENARIO:

Experiment 1: all nodes started running the OLSR protocol together. After 30sec the external nodes A and Y started pinging all the other nodes in the network, for 1 minute each, with a random sequence. The two sequences used for the ping operation were different and precisely:

Pinging sequence for node A: R, S, C, T, N, Q, Y, E, F, M, X, H, K, B, L, O, I, J, G, P, W, D.

Pinging sequence for node Y: E, F, M, X, H, K, B, L, O, I, J, G, P, W, D, A, R, S, C, T, N, Q.

Experiment 2: all nodes started running the OLSR protocol together. After 30sec all the nodes started pinging for 1 minute all the other nodes in the network with a random sequence. Table A.1 in Appendix A shows the sequences used by each node during the ping operation. Nodes ran the routing protocol for other 30 sec before stopping. The whole experiment took 23 minutes.

Experiment 3: in this case we used ADOV as routing protocol; the methodology is that of the previous experiment, and the sequences used in the ping operation are the same of Table A.1 to have a direct comparison between the two routing protocols.

b. MOBILE SCENARIO:

Experiment 1: this experiment is a "2-Central nodes Swap" in which the central nodes J and N changed their position during a continuous ping operation from two external nodes. More precisely, all nodes started running the OLSR protocol together (t=0). After 30sec the external node Y started pinging node A continuously for 210 sec. At t=90 nodes, J and N (two central nodes) started moving and swapped their positions after 30sec (t=120), then they remained in this new configuration until the end of the experiment. At t= 240 the experiment ended, so the ping operation and the routing protocol stopped running.

Experiment 2: this experiment can be referred as "4-Central nodes Swap". The four central nodes J, M, O and N changed their positions in clockwise manner during a continuous ping operation from two external nodes. More precisely, all nodes started running the OLSR protocol together (t=0). After 30sec (needed for network stabilization), the external node Y started pinging node A continuously for 300 sec. At t=90 node N started moving and reached the position of node J in 30 sec (event 1), then it remained in this new location until the end of the experiment. At t=120 node J started moving toward node M and reached the new position after 30 sec (event 2). Event 3 started at t=150 when node M moved and it finished when M reached the position of node O at t=180. In the same instant node O moved to the initial location of node N (event 4) stopping after 30 sec. At t=330 the experiment ended, so the ping operation and the routing protocol stopped running. Figure 8.4 shows the 4 events of the experiments.

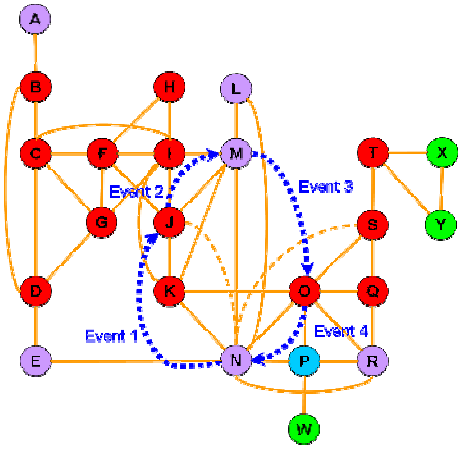


Figure 8.4. 4-Central nodes Swap scenario

Experiment 3: this experiment can be referred as Roaming node: all the nodes were static except the "roaming node" that moved crossing the entire network. The reference scenario is shown in Figure 8.5. The experiment lasted 410 sec. After an initial phase for OLSR (protocol) stabilization (30 sec), node Y started pinging node A for 380 sec. At t_1=90 from the initial position near node A, the pinger Y started moving inside the building along the corridor with a speed of about 1 m/sec. It reached position of node C after 30 sec (t_2=120), position of node I after 15 sec (t_3=135), then node K after 15 sec (t_4=150), node O after 15 sec (t_5=165), position of node S after 30 sec (t_6=195) and finally it reached the opposite side of the network near node X after other 30 sec (t_7=225). Once it has reached the last position, it immediately moved in the opposite direction following the reverse path and taking the same lags as in the forward path (from t_7 to t_f). Finally it reached the starting position near node A after 2 min and 15 sec (t_f=350).

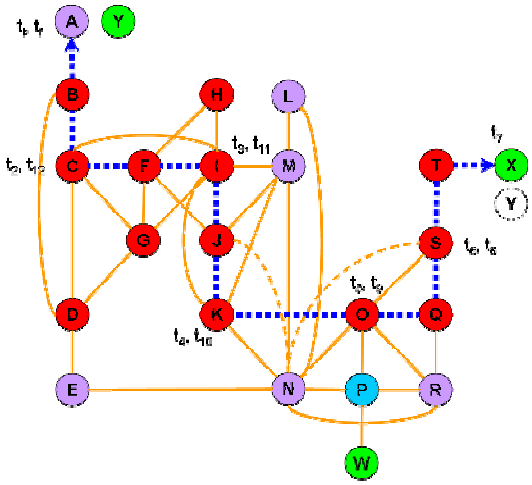


Figure 8.5. Roaming node scenario

Experiment 4: in this case we repeated experiment 1 with the reactive routing protocol running on each node.

Experiment 5: we repeated experiment 2 running AODV on each nodes.

Experiment 6: experiment 3 is repeated with AODV.

8.3.1. Static Scenario

In this section we compare OLSR and AODV referring to results obtained in experiment 2 and 3 (the reference scenario is shown in Figure 8.2). Specifically, we evaluate the protocols' performance by using the three indices introduced in the previous section.

Throughput analysis

Figure 8.6 presents the total overhead introduced by OLSR and AODV as a function of the experiment 2 time instants. The curves are generated taking into account the amount of control traffic generated by each node and forwarded by it, averaged between all nodes taking part to the experiment.

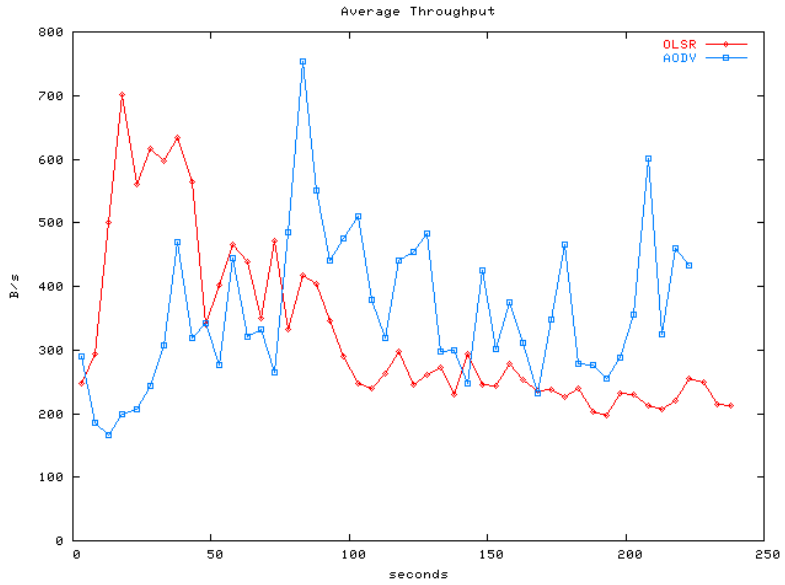


Figure 8.6. Overhead introduced by OLSR and AODV routing protocols.

As it clearly appears from the picture, OLSR and AODV have different behaviour. The proactive protocol produces a load of about 600 Bps in the starting phase (first 40 sec), then its load decreases to 400 Bps for the next 50 sec, finally a new steady state is achieved till the end around 250 Bps. On the contrary, AODV reaches a steady phase with a load of 400 Bps between 40 and 80 sec, then its load doubles with a peak of about

750 Bps around 90 sec, finally it stabilizes again varying from 300 and 500 Bps till the end of the test-run. OLSR introduces a higher overhead during the starting phase, then after a second phase in which its throughput coincides with AODV throughput, OLSR performs better for the rest of the experiment. From the quantitative standpoint, OLSR overhead falls in a range of [200-700] Bps, while using the reactive routing protocol it is around [180-750] Bps. These results confirm that also in medium scale network the overhead introduced by routing protocols, either using proactive and reactive approaches, doesn't affect negatively the system performance, indeed it reduces the available 802.11 bandwidth only of a small quantity.

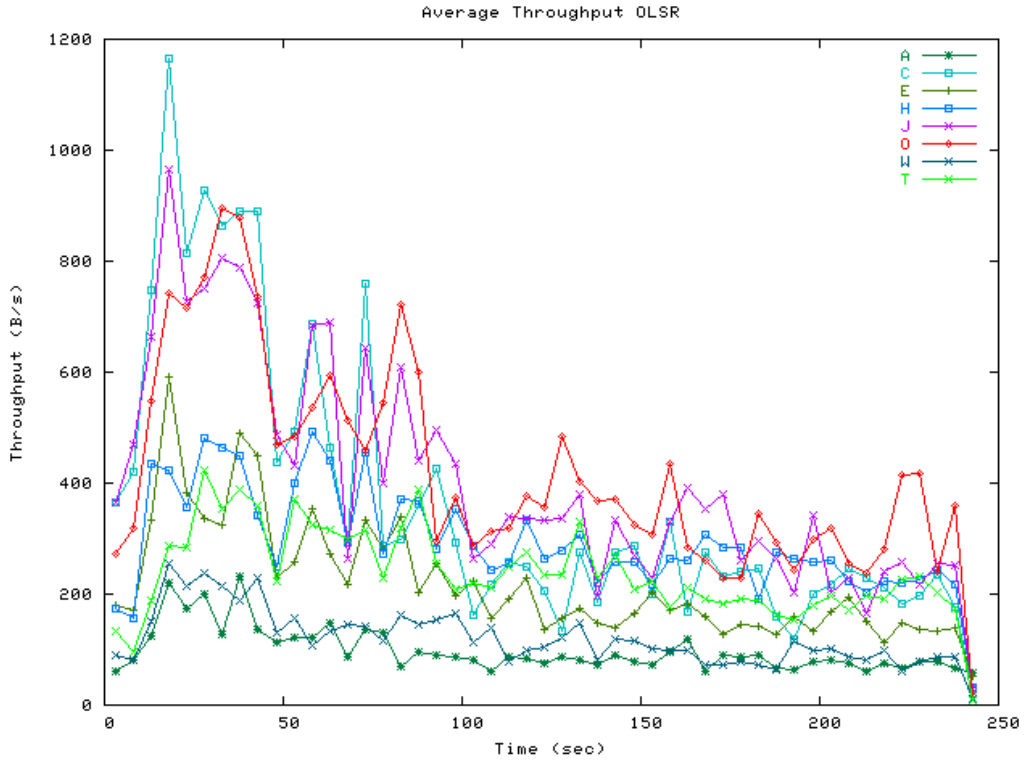


Figure 8.7. Overhead introduced by OLSR for different nodes.

Figure 8.7 and Figure 8.8 show the average overhead introduced by OLSR and AODV, respectively, for different nodes depending on their position in the network. Since there are 23 nodes in the network, in order to obtain a sharper graph only some of them are plotted. Referring to OLSR results (Figure 8.7), note that node C, J, and O observe the highest load since they are better connected with the rest of the network with 5 or more neighbors each, instead nodes E, H, and T have an intermediate load since they have less neighbors. At last nodes A and W obtain the lowest traffic around 100 Bps because they are located in marginal position since they are leaves for the network (see Figure 8.3). For AODV (see Figure 8.8) node F and O have the highest throughput, instead an intermediate load is performed by nodes L and B. In this case the lowest load is experienced by node E that, even though it isn't a leaf, has a marginal location with only

two neighbors. An explanation of this behavior is the reactive nature of AODV that makes the throughput also dependent on the application traffic

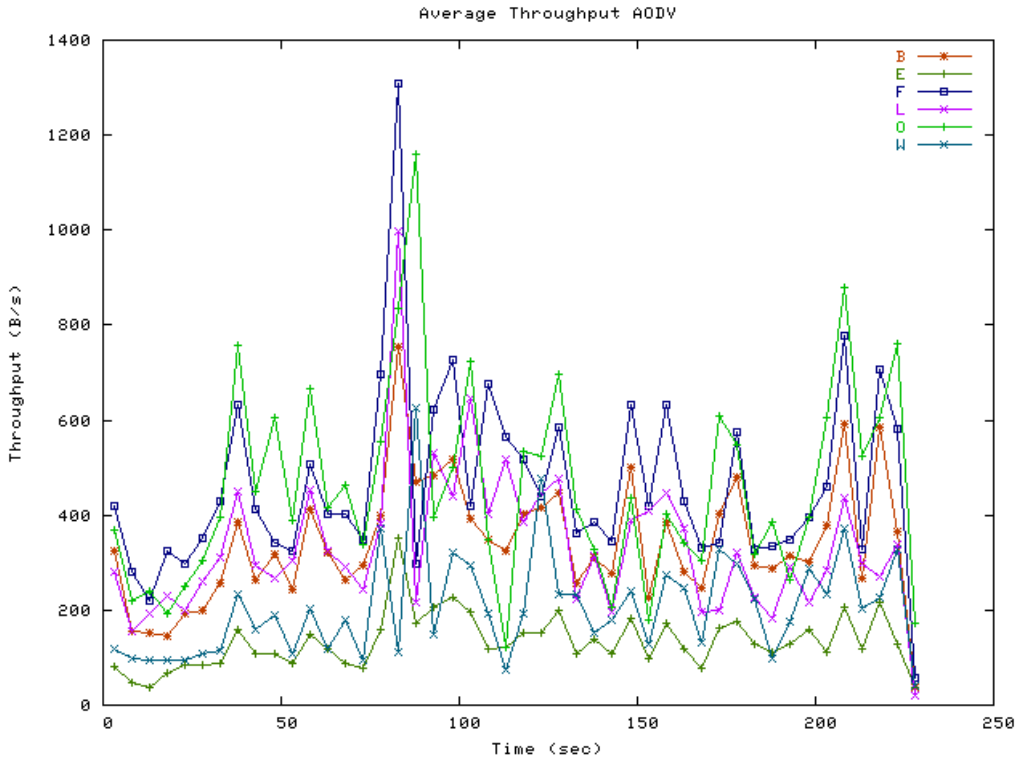


Figure 8.8. Overhead introduced by AODV for different nodes.

Packet Loss analysis

In the following paragraphs we analyze network's performance taking experiment 2 and 3 as reference scenarios. Table 8.1 shows the overall Packet Loss suffered by the routing protocols for different number of hops. Each percentage is obtained averaging all Ping operations between two nodes distant *x* hops. Looking at the obtained results we can notice that OLSR performs better than AODV delivering packets with less packet loss. In particular OLSR has a good behavior at 1-hop distance delivering a huge number of data, and then it suffers a packet loss of [15%-45%] with nodes distant more than 2 hops. Finally it performs an error of more than 50% when ICMP packets pass through 6 or 7 nodes. On the contrary, problems with the reactive protocol are more evident. AODV doesn't work properly also in the nearby performing a packet loss of 20%, its performance decreases experiencing a 50% loss rate when communicating with nodes at a 2-hop distance. Finally its behavior drastically degenerates when the distance becomes more than 3 hops, finishing successfully only 10% of Ping operations at 7-hop distance.

	1	2	3	4	5	6	7
OLSR	5%	15%	28%	35%	45%	52%	67%
AODV	20%	51%	51%	61%	67%	86%	89%

Table 8.1. Overall Packet Loss suffered by OLSR and AODV for different number of hops.

Another observation can be derived taking into account results from the indoor string topology as explained in Section 3.1. To summarize, in that scenario the OLSR performance is acceptable in all Ping operations towards each node in the string, instead AODV loses 50% of ICMP packets when communicating with nodes at 3-hop distance (see Table 3.1.). On the contrary, in this medium-scale environment, also with few hops we observe a high percentage of undelivered packets. Possible explanations are the different network size (small vs. medium) and the complexity of the experiment (1 Ping operation vs. 23 simultaneously Ping operations). In particular with concurrent connections in the network each node can act as destination for a Ping operation and also as router for another one. Thus the probability of collision at MAC layer is increased considerably.

Delay analysis
To evaluate the delay introduced by the routing protocols we measured the end-to-end latency for completing a Ping operation between couples of nodes. In particular we consider two different delays in the network:

- Average delay to deliver the 1° successful packet
- Average delay to deliver all successful packets

Figures 8.9 and 8.10 present graphs of delay (expressed in milliseconds) suffered by the routing protocols for different number of hops. Each value is calculated averaging time required to complete Ping operations between two nodes distant *x* hops.

Figure 8.9 shows the average delay needed to complete successfully the first ping operation. As it clearly appears, OLSR curve is (as expected) lower than AODV curve due to the different nature of the routing protocols. In particular, OLSR curve increases almost linearly up to 6 hops, and then it doubles at 7 hops. The reason is the instability of the network that implies a network reconfiguration and hence an increase of time. On the contrary, AODV curve is a step function. It needs about 2-sec delays to discover routes to neighbours, then it introduces delays of about 10 sec in the range [2-5] hops and finally it requires 15-17 sec to discover valid paths towards nodes distant more than 6 hops. These high delays are due to the several attempts performed in the route discovery process. In

fact each node makes about 5-6 attempts in order to discover a valid route to its destination.

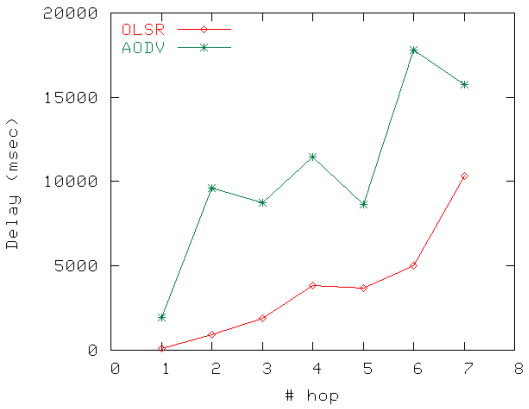


Figure 8.9. Average Delay on 1° packet suffered by OLSR and AODV for different number of hops.

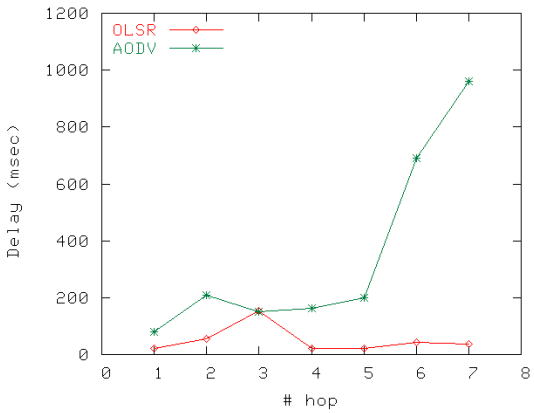


Figure 8.10. Average Delay on delivered packets suffered by OLSR and AODV for different number of hops.

Looking at Figure 8.10, note that OLSR requires delays in the range of (20msec, 60msec) independently from the number of crossed hops, while AODV introduces higher delays. More precisely, AODV Ping connections perform the following delays: about 200 msec when they are shorter than 6 hops, about 700 msec towards nodes distant 6 hops and about 1 sec toward nodes at 7-hop distant. The log files give an indication of these high values. AODV doesn't maintain the first discovered path to the same destination for the entire duration of the connection, but it requires 1 or 2 attempts in order to re-establish a valid route to the destination.

8.3.2. Mobile Scenario

In this section we compare routing protocols considering mobile scenarios with nodes that change positions during the entire experiment. In the first scenario referred as Roaming node, a node moves along the network; in the second scenario two central nodes exchange their position and it is referred as 2-Central node swap; in the third experiment 4 central nodes change their position (4-Central node swap). In all the performed experiments each mobile node moves in the network with a speed of about 1m/s, i.e., low mobility scenarios. We analyze their results with particular attention to the packet loss and the introduced delays for network reconfiguration.

Packet Loss analysis

Considering the Roaming node experiment (experiment 3 and 6), OLSR experiences a 25% packet loss, while AODV correctly delivers only 50% of packets. Examining the log files, we observe that for OLSR the packet loss mainly occurs in the way back between node X and Q (see Figure 8.5). In particular during this gap the Y routing table becomes empty and node Y recovers the route to the destination only near node Q, so all packets are completely lost. AODV instead loses all ICMP packets when node Y goes beyond node D, so when the connection becomes longer than 4 hops none of ICMP packets

reaches the destination. Increasing the complexity of the proposed scenarios the performance of the routing protocols becomes unacceptable: only a little percentage of Ping operations is completed successfully. Two are the main causes. Firstly, the Ping operation between node Y and A is a 7-hop connection. In the previous section we have shown how the network's performance decreases in static networks. Secondly, adding mobility, the complexity of the network increases. To better understand, let's consider the similar scenario analyzed in the small scale network (see Section 3.1). Also in that case more than 40% of packets were lost with both routing protocols. Hence, these two factors cause the network breakdown.

Delay analysis
To evaluate the delay introduced in the network in the three scenarios we investigated the time needed to update the routing tables with a valid route to the destination after topology changes, independently from the correct delivery of packets. All the values are measured from the standpoint of node Y. Starting from the Roaming node scenario, OLSR performs a delay in a range of [4-9] sec to update routing table each time the connection length increases of 1 hop. AODV suffers delays between [4-10] secs to discover routes to node A from 1 to 4 hops long. The route discovery process takes more than 10 sec when Y goes beyond node O (in this case routes are 5 or 6 hops long), but since valid paths are maintained in the routing table only for few seconds no ICMP packets are delivered successfully. Both protocols do not introduce any additional delay in the reverse path. In the 2-Central node Swap the tested routing protocols lose a valid path to the destination for 30 sec needed for central nodes' exchange. OLSR is able to reconfigure an 8-hop route properly only after 5 sec from the end of the exchange, instead AODV requires other 90 sec to discover a valid route to node A. In the 4-Central nodes scenario OLSR and AODV suffer high delays. In particular, OLSR loses the routing table's entry to node A from 1 to 4 sec after the start of each event and it needs delays between 5 sec to 50 sec to reconfigure properly the routing table. AODV becomes aware of the new event after 3-8 sec from the beginning and needs from 5 to 10 sec to re-establish a valid route with also a peak of 60 sec when all network changes are completed. Note that, in case of AODV, most of the discovered routes are stored in routing tables only for few seconds, thus the discovery process is repeated frequently. This is due to the nature of AODV that discovers also unstable paths. As consequence Ping packets aren't received correctly by the destination decreasing the overall system performance. On the contrary, even though OLSR generates higher delays to network reconfigurations due to a slow propagation of topology changes, its new paths are maintained in the routing tables till the beginning of the new event. In fact the proactive protocol looks for more stable routes and this allows the source to send and receive application data successfully.

8.3.3. Conclusions

We investigated the performance of OLSR and AODV setting up a network of 7-8 hops size with up to 23 nodes. We performed an extensive set of experiments comparing them in static and mobile scenarios. The throughput analysis confirms that also in medium

scale networks the use of a proactive protocol doesn't reduce the system performance since it introduces an overhead of the same order of AODV. In addition with an OLSR a higher amount of data is delivered successfully even with long connections. Referring to delay introduced in the network, OLSR response times are much better than AODV. Finally, considering the mobile scenario, even though OLSR is slower than AODV to propagate the network changes, it performs better than the reactive protocols discovering more stable routes and hence delivering more application data.

8.4. Middleware Experiments

In the middleware experiments we compared CrossROAD and *FreePastry* in *static* and *mobile* scenarios. In static scenarios we mainly analysed the overhead introduced by the overlay management on the network in terms of throughput and delay. On the other hand, in case of mobile scenarios, we focused on CrossROAD performance to distribute data on the overlay nodes and the responsiveness of the cross-layer architecture to topology changes. The network topology used in these experiments is shown in Figure 8.11.

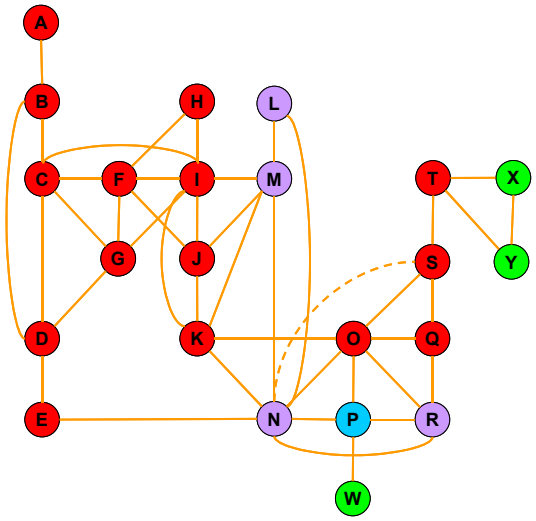


Figure 8.11. Network topology

Since Pastry requires the knowledge of a bootstrap node to join the overlay, each node running FreePastry must define a priori the IP address of that node to directly send its join request. Obviously, the bootstrap node has to be active before other nodes sending their join request. A time interval is thus associated with each join procedure to be compliant with the bootstrap sequence and avoid connection failures during this phase. To maintain a correspondence between the two different sets of experiments, the same bootstrap sequence is also used for CrossROAD experiments. A description of the overlay set up, and the joining procedure for each set of experiments, is detailed in Appendix A. In addition for all these experiments all nodes were synchronized and started running the routing protocol for 30 seconds to have the network topology stabilized. Then they ran the DM application with different start-up delays, and they are all active after 60 seconds from the starter of the overlay. In particular, to numerically evaluate system performance grouped by throughput, delays, and data distribution, we defined different types of experiment:

- Experiment 1: All nodes started running the routing protocol, followed by DM application on top of CrossROAD or FreePastry. In this experiment no application messages were sent in order to evaluate only the overhead introduced by the overlay management on the routing protocol. DM ran for 4 minutes, and then each node explicitly closed it. The routing protocol stopped after 30 seconds after DM ending.

- Experiment 2: All nodes started running the routing protocol, followed by DM application. In this experiment one application message ("Create Mailbox" message) of 100 Bytes with a random key was generated and sent on the network by each node every 100 msec for 120 seconds. Before sending those messages, nodes waited for all the others joining the overlay. Therefore, the first message was sent (on each node) after 60 seconds from the first node that had started the overlay. The "Create Mailbox" message requires to the destination node to store a mailbox with the specified identifier as the key of the message. It does not require any reply.

- Experiment 3: All nodes started running the routing protocol, followed by DM application. In this experiment one application message ("Create Mailbox" message) of 100 Bytes with random key was generated and sent on the network by each node (except for nodes **A** and **Y**). Messages were sent with a period of 100 msec for a last of 120 seconds. At the same time, **A** and **Y** generated a "Get" message with a random key using the same frequency of other nodes. As in the previous case, before sending all messages, nodes waited for all the others joining the overlay. The "Get" message notifies to the destination node the request of the list of messages stored in the mailbox with logical identifier equals to the key of the message (the mailbox had to be previously created by a "Create" message). The node selected as best destination for this kind of messages has to directly reply to the sender with the list of messages. Using a timestamp inside the application message, a round-trip delay can be measured.

- Experiment 4: All nodes started running the routing protocol, followed by DM application. In this experiment 10 nodes (**A, Y, T, R, M, H, I, C, E, G**) generated a "Get" message every 100 msec for 120 seconds, while the others (**N, D, F, J, O, P, S, B, K, W, Q, X**) only maintained the overlay, receiving messages and replying directly to their sender.

- Experiment 5: It is the same procedure of experiment 2 changing the frequency of messages: all nodes generated a "Create Mailbox" message every 500 msec for 120 seconds. This experiment was conducted to analyze the effect of a traffic reduction on the system performance.

- Experiment 6: data distribution in case of delayed joining of the overlay with CrossROAD. In this experiment node **L** was not available, and all nodes, except for **N** and **M**, started running UNIK-OLSR and XL-plugin creating two different ad hoc networks (see Figure 8.12). They only ran the routing protocol for 30 seconds to have the two network topologies stabilized. Then they ran DM application with different delays following the same sequence specified for the first set of experiments on CrossROAD. **N** and **M**, which are central nodes, started the routing protocol and the overlay with a delay of 2 minutes joining the two networks in an only one. On the other hand, when all nodes of the first group correctly participated in the overlay, **A** and **Y** started sending a "Get" message every 200 msec for 6 minutes.

A detailed description of each experiment can be found in Appendix A. In the following sections the analysis of main performance results obtained by this experimentation is presented. In particular, the overhead to maintain these two different overlays in terms of throughput is presented in Section 0. Then, measured delays to distribute and recover data are analyzed in Section 8.4.2 and, finally, the responsiveness of the cross-layer interactions between CrossROAD and OLSR to distribute data in a partitioned overlay is described in Section 0.

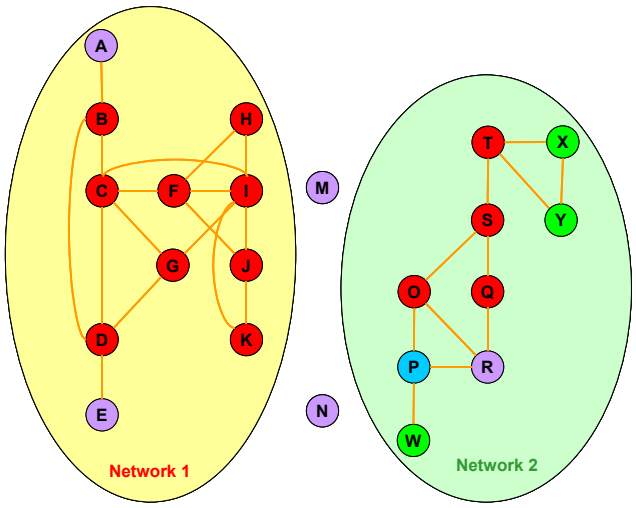


Figure 8.12. Network topology of the delayed joining of the overlay.

8.4.1. Throughput analysis

To analyse the overhead introduced on the ad hoc network by the different overlays (Pastry or CrossROAD), we considered the experiment number 2, in which every node generated an application message every 100 msec for 120 seconds after the initial phase of 30 seconds to stabilize the network topology using the routing protocol only. We defined the average throughput as the aggregation of the overlay management throughput, the application data traffic, and the routing traffic sampled every second and mediated on the number of network nodes. Considering the routing traffic together with the overlay and application traffic is important in case of CrossROAD, because, on the opposite of Pastry, CrossROAD does not introduce additional overhead to maintain the overlay data structures at the middleware layer, but it exploits the routing protocol to distribute services information and locally compute contents of the overlay data structures. For this reason the overhead introduced to manage the overlay is moved at the routing layer, thus slightly increasing the routing traffic (see Figure 8.14b). The average

throughput is shown in Figure 8.13 considering three different cases: CrossROAD, FreePastry running on top of OLSR, and FreePastry on top of AODV. As shown by the figure, the overhead introduced by Pastry is much higher than that of CrossROAD, either in case of OLSR or AODV routing protocols. This is mainly due to periodical TCP and UDP connections needed by FreePastry to monitor the status of other nodes of the overlay and consequently update overlay data structures. On the other hand, in case of CrossROAD, each node becomes aware of changes in the overlay directly from the cross-layer interactions with the proactive routing protocol. The cross-layer service discovery protocol does not significantly overload the routing protocol since service information piggybacked in routing packets consists of few bytes: the service identifier (int value of 32 bit) and the port on which the service is provided (int value of 16 bit). This information is spread on the network with the same frequency of Hello packets (every 2 seconds).

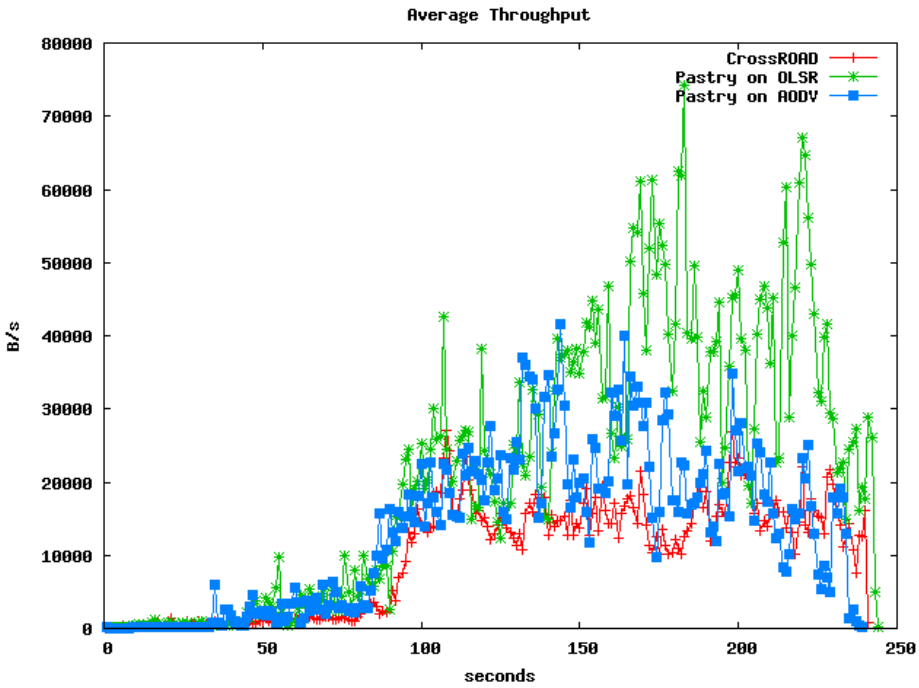


Figure 8.13. Average aggregate throughput.

Note that the average throughput is close to zero for the first 30 seconds of the experiment compared to high values in the second part of the experiment. In fact nodes spent 30 seconds running only the routing protocol to stabilize the network topology, and then they spent 60 seconds for the bootstrap procedure to wait for all nodes joining the overlay. In case of Pastry, when the second node tried to connect to its bootstrap, a set of data exchange is flooded on the network and increases the average throughput. Instead, in

case of CrossROAD, the throughput assumes low values until nodes started sending application messages. Actually, the routing throughput is not zero. In fact, as is shown by Figures 8.14 a) and b), in the first phase AODV assumes the lowest values since it only sends Hello packets to discover one hop neighbours. On the other hand, OLSR enhanced with XL-plugin coincides with the original protocol, since in that phase no services information are flooded on the network. Instead, from 30 to 90 seconds the throughput increases of about 60%, since in that period nodes started running CrossROAD at different instants and they received service information from the others. After the bootstrap procedure, the overhead approaches to the same values of OLSR, considering the periodical sending of service information on the network with the same frequency of Hello packets.

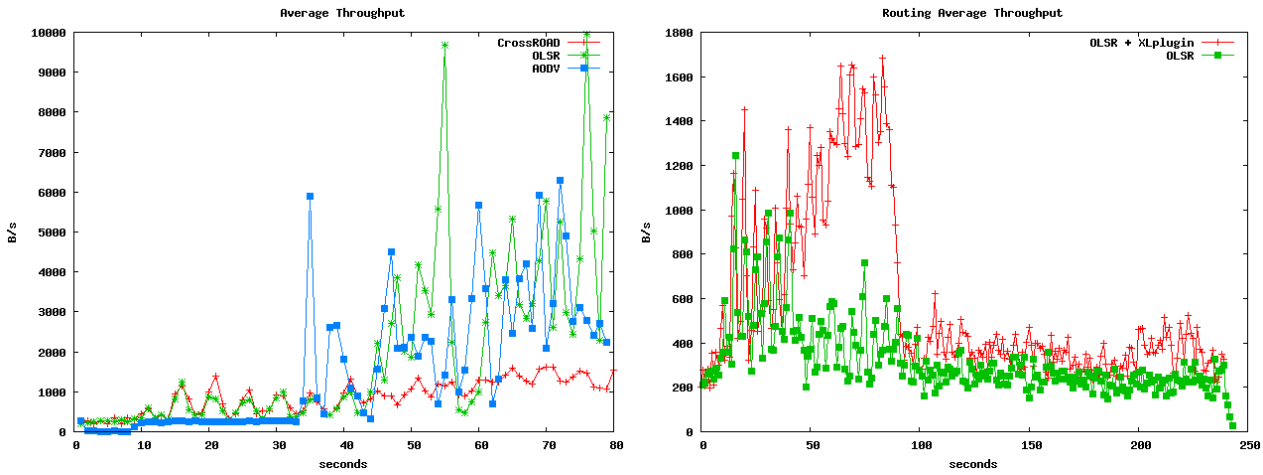


Figure 8.14

a) Average aggregate throughput, the first phase. b) Routing throughput

Since the additional overhead introduced by the cross-layer interaction on OLSR is negligible, the average throughput of CrossROAD corresponds to the data traffic introduced by DM application, while in case of Pastry the overlay management is much higher than the application traffic (see Figure 8.15).

In addition, comparing the average throughput of FreePastry on AODV and OLSR on each single node, we noticed that some groups of nodes measured highly different throughput in these experiments. This is mainly due to the bootstrap procedure needed to join the overlay in case of Pastry. Figures 8.16 (a, b, and c) show the state of the overlay after the joining phase. In case of CrossROAD every node of the network participated in the same overlay, since the cross-layer service discovery protocol notifies other nodes of connection/disconnection events. On the other hand, in case of Pastry running on top of

OLSR, four overlays had been carried out and five in case of Pastry on AODV. These phenomena depend on connection failures occurred during the join procedure, and they influence the entire last of the experiment. In fact, when a node fails the connection to its bootstrap node, it creates a new overlay. The failure can be due to the absence of a route to the destination, or to the instability of the selected link. If some other nodes consequently connect to the failed node, they join the new overlay and a future rejoining with the original overlay is not possible. In Pastry, nodes are not aware of the network topology and each of them is responsible only for maintaining information related to its overlay. Hence, the amount of data to be exchanged in a small overlay is lower than that exchanged in a wider one (see Table 8.2).

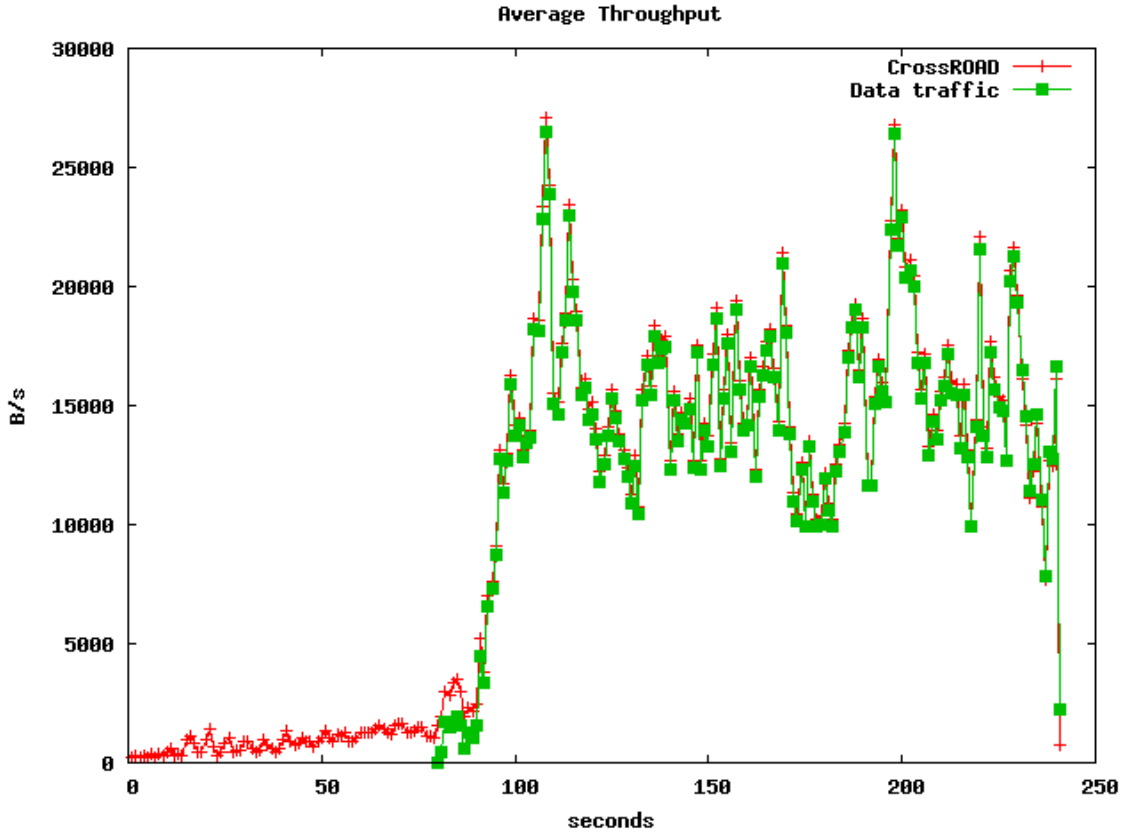


Figure 8.15. Average aggregate throughput and data traffic.

In Table 8.2 some entries are empty because during the execution of the experiment not every node correctly ran the `tcpdump` utility to store sent and received packets. In case of CrossROAD, the distribution of the average throughput of single nodes mainly depends on the location of nodes in the network topology, since most connected nodes

and central ones not only see packets generated and received by themselves, but also packets that they forward to other nodes. On the other hand, in case of Pastry, nodes that had built an independent overlay, without other participants, measured a very low throughput (e.g. node **A** in OLSR measured 446 Bps, while node **G** in AODV measured 550.56 Bps). In these cases the measured throughput refers to the routing overhead and to periodical tentative connections to the original bootstrap node to recover information on other nodes taking part in the same overlay. In these cases application messages are only sent to the local node without involving the network socket, since every random key is nearest to the local node identifier. In addition, nodes that originated an overlay of two or three participants measured about the same amount of throughput either in case of AODV and OLSR. For example, nodes **K** and **J** in the OLSR experiment measured about the same throughput of nodes **A** and **B** in the AODV experiment. On the other hand, highest values of throughput were measured by nodes participating in the original overlay. Since the number of nodes in the original overlay in case of OLSR is greater than that in case of AODV, the average throughput of Pastry on OLSR is higher than that of Pastry on AODV. In fact, the average throughput shown in Figure 8.13 represents the average on all nodes of the network, independently from the number of overlays.

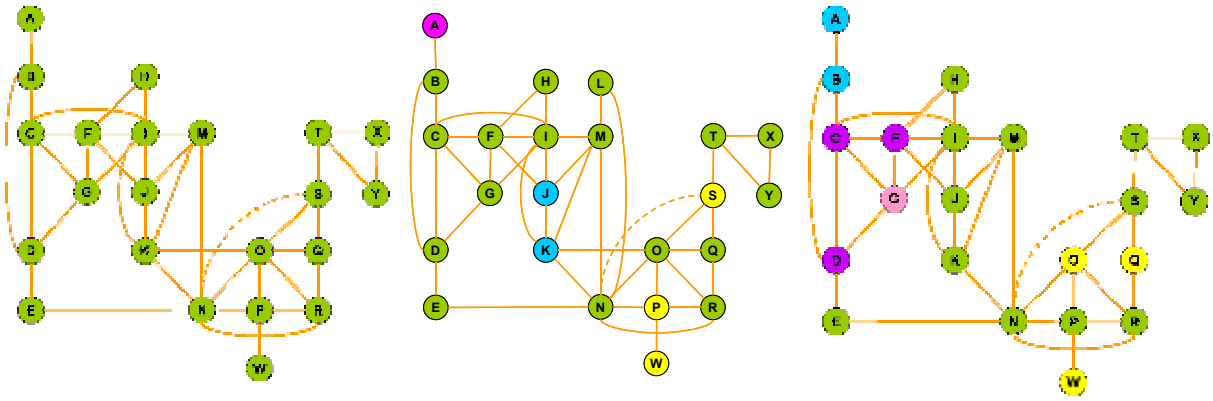


Figure 8.16.

a) CrossROAD overlay b) Pastry on OLSR overlays c) Pastry on AODV overlays

Nodes	CrossROAD Bps	Pastry on OLSR Bps	Pastry on AODV Bps
A	7864.83	446.02	9818.63
B	14340.44	21105.81	10121.92
C	6987.27	50445	15074.81
D	9825.43	29758.44	7810.33
E	6425.7	10801.22	2434.22
F	13929.36	67684.04	8875.53
G	17517.3	50711.24	550.56
H	7485.49	3626.05	5308.8
K	3625.77	9750.221	102910.1
J	14441.47	9983.44	-
I	11457.44	3662.84	10988.85
M	2499.88	6712.37	8308.95
N	13353.95	74777.96	10070
O	13171.84	10574.02	5368.29
P	10455.38	3007.5	-
Q	-	10171.61	10960.5
R	10684.71	57114.13	5334.54
S	-	5985.35	-
T	13830.02	12480.14	14180.41
W	3667.63	3463.5	5418.92
X	13217.73	-	-
Y	12741.6	10300.04	11917.92

Table 8.2. Average throughput per node.

.

8.4.2. Delays Analysis

To analyze the distribution of delays measured by nodes to send a specific message and receive the related reply, we considered the experiment number 4. In this experiment 10 nodes (**A, Y, T, R, M, H, I, C, E, G**) generated a "Get" message every 100 msec for 120 seconds, while the others (**N, D, F, J, O, P, S, B, K, W, Q, X**) only maintained the overlay, receiving messages and replying directly to the sender. Even though only a part of nodes is involved in the message transmission, this experiment is the most suited for analysing delays. In fact in each "Get" message a timestamp was added to the original content by the sender. When it thus receives the related reply maintaining the original timestamp, it directly computes the delay as the difference between the local time and the original timestamp.

Percentiles	CrossROAD	Pastry on OLSR (main overlay of 15 nodes)	Pastry on AODV (main overlay of 12 nodes)
0,6	599 msec	11.171 sec	9.138 sec
0,7	2.306 sec	20.032 sec	16.055 sec
0,8	4.692 sec	34.846 sec	28.823 sec
0,9	10.648 sec	46.340 sec	75.475 sec
0,95	23.025 sec	61.858 sec	88.701 sec
0,99	60.468 sec	111.560 sec	105.649 sec

Table 8.32. Percentiles of the delay distribution.

The delay distribution and related percentiles, shown in Figure 8.17 and Table 8.3, respectively, highlight that delays reach the order of 100 seconds in case of Pastry and 60 seconds in case of CrossROAD, but the most part of them is concentrated on the following time intervals: (0, 100msec) and (100msec, 500msec). In order to have a consistent view of the distribution, only packets generated and received by nodes of the main overlay were considered. In fact, in case of the smaller overlays, delays are reduced to few milliseconds because few packets have to be managed by the involved nodes. High delays measured in the main overlay can be due to processing data packets and concurrently managing overlay data structures with other TCP and UDP connections (in case of Pastry). In addition, since the network topology is also characterized by redundancy, and in some cases unstable links, the distribution of data packets through TCP connections can be delayed by many retransmissions, increasing the related timeout.

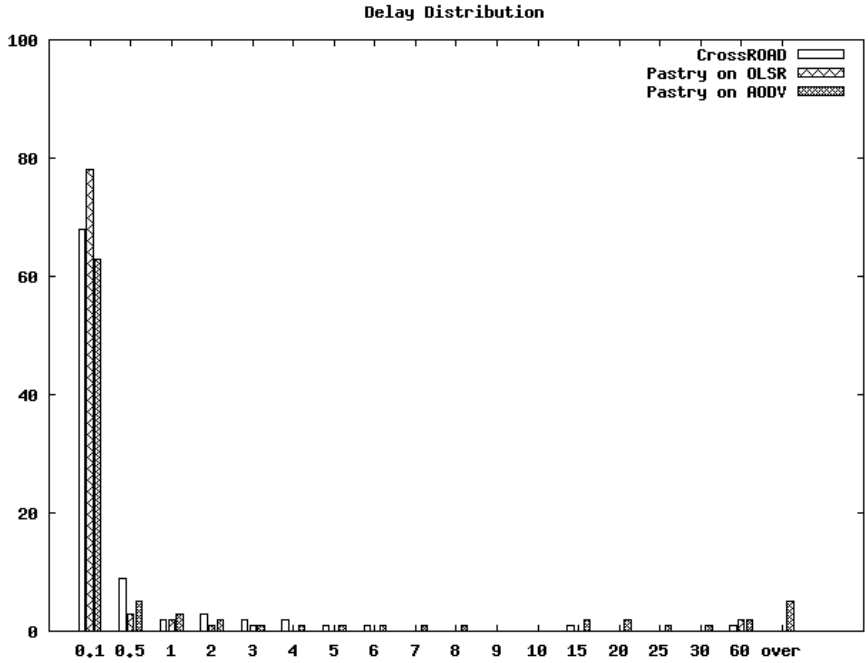


Figure 8.17. Delays distribution (seconds)

8.4.3. Data Distribution in case of delayed joining of the overlay

To analyse the responsiveness of the cross-layer interactions of CrossROAD with the routing protocol, we carried out a set of experiments in which central nodes started the routing protocol and the overlay with a delay of 2 minutes after the others. In this way, even though nodes were not moving, the topology changes, and CrossROAD became aware of these changes; this has an important consequence for the application. We analysed results from experiment number 6, where, referring to the topology graph shown in Figure 8.12, all nodes, except for **N** and **M**, started running UNIK-OLSR and XL-plugin creating two different ad hoc networks. **N** and **M**, which are central nodes, started the routing protocol and the overlay with a delay of 2 minutes joining the two networks in one. Note that the first group of nodes not only created two different ad hoc networks but, running CrossROAD and DM application, they also generated two different overlays. In addition, when all nodes of the first group correctly participated in the overlay, **A** and **Y** started sending a "Get" message every 200 msec for 6 minutes. Since the key of each message was randomly selected, messages were originally sent to random destination inside the network area of the sender (i.e. node **A** sent messages to nodes of Network 1, while node **Y** sent messages to nodes of Network 2). Then, when nodes **N** and **M** joined the routing protocol, the cross-layer service discovery protocol flooded the service information of all participating nodes on the entire network. From that moment senders became aware of the topology change through the cross-layer interaction with the routing

protocol, and their random messages were sent also to nodes located in the other network area. In addition, when **N** and **M** also joined the overlay, they became also possible receivers of those messages. As is shown by Figure 8.18, messages are initially distributed on nodes of the same area of the sender and, after the first 100 packets, they are also sent on the other area.

8.4.4. Conclusions

By setting up a large-scale ad hoc network, we really examined system features and performance in real conditions, using both static and mobile scenarios. Even though this kind of experimentation is difficult to be carried out, due to the high number of people and devices involved, it allowed us to analyze advantages and drawbacks of a complete MANET architecture. Analyzing performance of a simple distributed application on top of two different p2p systems, we pointed out advantages of using a cross-layer approach to exploit network topology information at the middleware layer, and drawbacks of using a legacy p2p system on ad hoc networks. The main purpose of a structured overlay network consists of defining a good policy to distribute workload on all nodes of the network.

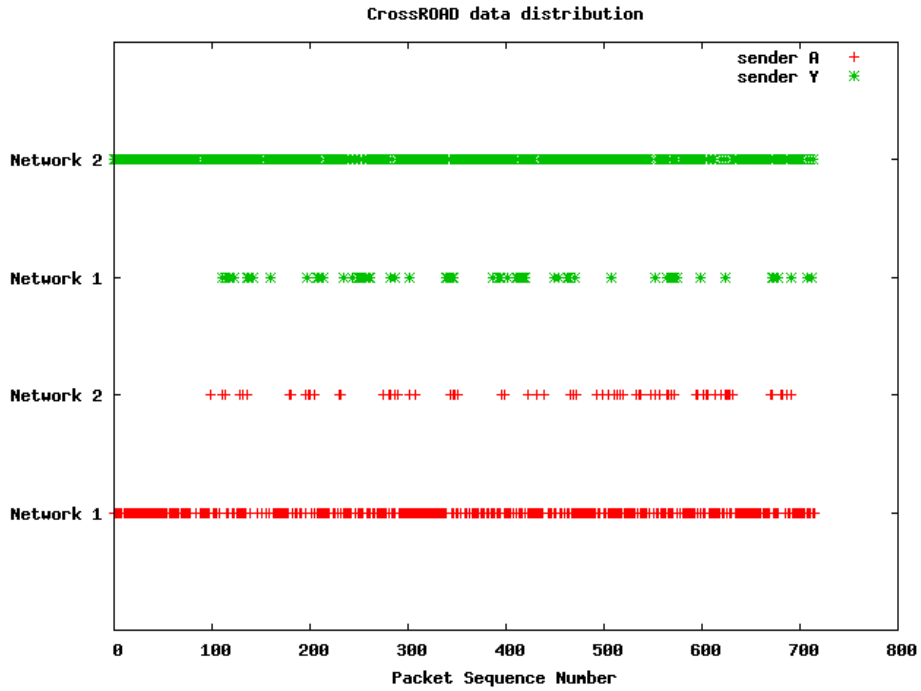


Figure 8.18. CrossROAD data distribution in case of partitioned overlay and consequently joining.

In a legacy solution, for fixed Internet, there are no connectivity problems and the joining procedure that requires a bootstrap node is not a problem. On the other hand, on ad hoc networks characterized by unstable links and mobile nodes, the high possibility of connection failures during this phase negatively influences the system performance. In addition, using a great number of UDP and TCP connections to update overlay data structures increases the network overhead and delays of application messages. From these results we pointed out that a cross-layer solution developed to optimize the structured overlay on ad hoc networks increases the system performance making nodes independent of each other in managing the overlay, and able to notify them their connection or disconnection events through a proactive routing protocol. The overhead needed to maintain the overlay is thus transferred at the routing layer, but it is quite negligible. However, high application delays have to be investigated to study further optimizations in order to improve the system performance.

8.5. References

[1]	http://www.olsr.org
[2]	http://user.it.uu.se/~henrikl/aodv.
[3]	D8: MobileMAN First phase, http://cnd.iit.cnr.it/mobileMAN/pub-deliv.html
[4]	D13: MobileMAN domain modelling, http://cnd.iit.cnr.it/mobileMAN/pub-deliv.html
[5]	FreePastry, www.cs.rice.edu/CS/Systems/Pastry/FreePastry.
[6]	Java API specification, http://java.sun.com.
[7]	P. Druschel J.Kubiatowicz I. Stoica F. Dabek B. Zhao, "Towards a common API or structured peer-to-peer overlays", Proc. of IPTPS'03 Workshop, Berkeley, CA, Feb. 2003.
[8]	M. Conti, F. Delmastro, and E. Gregori, "Cross-Layer extension of the P2P CommonAPI for structured overlay networks". Technical Report available at http://cnd.iit.cnr.it/people/fdelmastro/

8.6. Appendix A: Journal of the Experiments

28th May: *Setting up the topology*

During the first day we investigated the network topology. Our goal was to create an ad hoc network of 23 nodes as much large, as possible in a heterogeneous environment. For this reason we used a heterogeneous environment consisting of indoor and outdoor spaces since not all buildings are strictly connected between them. We started from the same configuration used in the experimental session of July 2004 with 12 nodes, explained in [3]. Since we used a greater number of laptops with different capabilities (also for the transmission range of wireless cards), a new measurement of the link connectivity had to be done. In this case the interested area was extended from the CED area to the neighbourhood of the conference area as shown in Figure A.1.

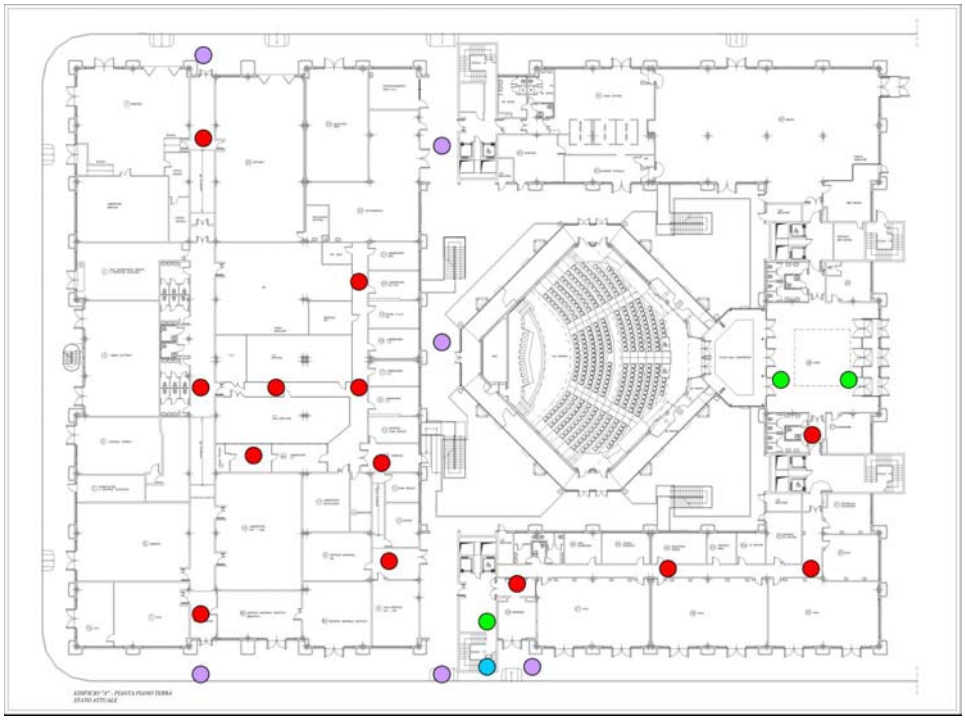


Figure A.1. Physical position of nodes.

Most part of nodes (17) was located inside buildings. In particular, 13 at the ground floor (red circles), three at the first floor (green circles), and one on the stairs (the blue circle). The last six nodes were located outside the buildings (violet circles) along the street or

the corridor between the involved buildings. In order to verify the coverage area of every device, each node started running UNIK-OLSR for five minutes storing the kernel routing table in a log file every second. Then, we analyzed the set of 1-hop neighbors of each node to define the final network topology. Considering a large multi-hop ad hoc network we could test and evaluate features and performance of a complete MANET architecture. For this reason, since many devices had a wireless card with a high transmission power, we had to reduce it on single nodes (if allowed by the driver of the wireless card) to remove some redundant links. Most of the transmission power of cards had been set to 12 dBm. We repeated this procedure many times to check the obtained configuration was stable. Figure A.2 shows the final network topology, where straight lines point out the presence of stable links (two nodes directly see each other), dashed lines show the presence of weaker links (the communication between two nodes is affected by a considerable packet loss). We thus obtained a multi-hop MANET of 23 nodes with the maximum extension of eight hops.

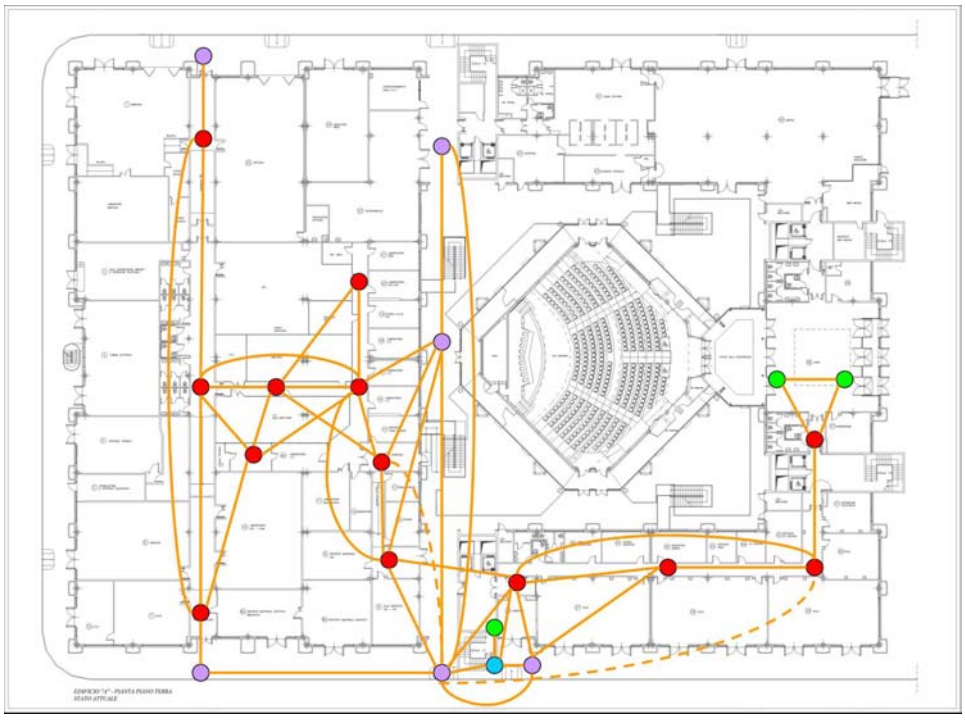


Figure A.2. Network topology.

In the following paragraphs we refer to the network topology as the graph shown in Figure A.3.

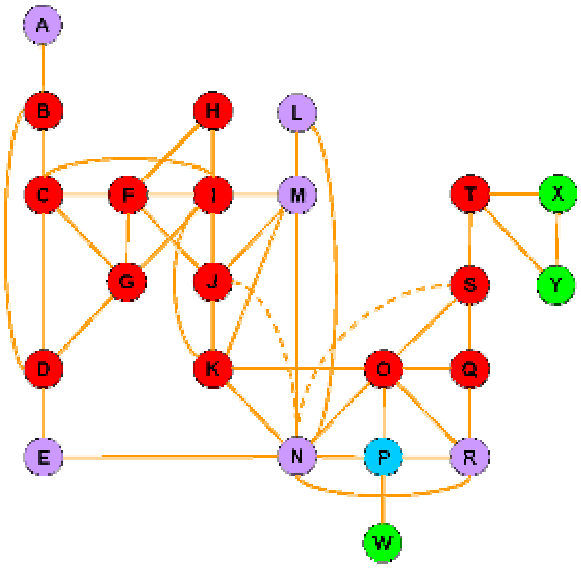


Figure A.3. Network topology graph.

3rd June: Experiments with Routing Protocols on static network

In this set of experiments we investigated the behavior of the selected routing protocols considering only static scenario. In addition to UNIK-OLSR and UU-AODV we introduced some application traffic using the Ping utility. This guarantees that AODV also run in a complete manner, otherwise its routing information is reduced only to the exchange of Hello packets and any route is calculated.

Experiment 1: all nodes started running the OLSR protocol together. After 30sec the external nodes A and Y started pinging all the other nodes in the network, one minute each, with a random sequence. The two sequences used for the ping operation were different and precisely:

Pinging sequence for node A: R, S, C, T, N, Q, Y, E, F, M, X, H, K, B, L, O, I, J, G, P, W, D.

Pinging sequence for node Y: E, F, M, X, H, K, B, L, O, I, J, G, P, W, D, A, R, S, C, T, N, Q.

At the end of the ping operation each node ran OLSR for other 30sec and then stopped. The experiment lasted 23 minutes.

Experiment 2: all nodes started running the OLSR protocol together. After 30sec all the nodes started pining for 1 minute all the other nodes in the network with a random sequence. Table A.1 shows the sequences used by each node during the ping operation. Nodes ran

the routing protocol for other 30 sec before stopping. The whole experiment took 23 minutes.

Experiment 3: in this case we used ADOV as routing protocol; the methodology and the last are equal to the previous experiment, and the sequences used in the ping operation are the same of Table A.1 in order to have a direct comparison between the two routing protocols.

PINGER	PINGING SEQUENCE
A	R, S, C, T, N, Q, Y, E, F, M, X, H, K, B, L, O, I, J, G, P, W, D.
B	L, O, I, J, G, P, W, D, A, R, S, C, T, N, Q, Y, E, F, M, X, H, K.
C	T, N, Q, Y, E, F, M, X, H, K, B, L, O, I, J, G, P, W, D, A, R, S.
D	A, R, S, C, T, N, Q, Y, E, F, M, X, H, K, B, L, O, I, J, G, P, W.
E	F, M, X, H, K, B, L, O, I, J, G, P, W, D, A, R, S, C, T, N, Q, Y.
F	M, X, H, K, B, L, O, I, J, G, P, W, D, A, R, S, C, T, N, Q, Y, E.
G	P, W, D, A, R, S, C, T, N, Q, Y, E, F, M, X, H, K, B, L, O, I, J.
H	K, B, L, O, I, J, G, P, W, D, A, R, S, C, T, N, Q, Y, E, F, M.
I	J, G, P, W, D, A, R, S, C, T, N, Q, Y, E, F, M, X, H, K, B, L, O.
J	G, P, W, D, A, R, S, C, T, N, Q, Y, E, F, M, X, H, K, B, L, O, I.
K	B, L, O, I, J, G, P, W, D, A, R, S, C, T, N, Q, Y, E, F, M, X, H.
L	O, I, J, G, P, W, D, A, R, S, C, T, N, Q, Y, E, F, M, X, H, K, B.
M	X, H, K, B, L, O, I, J, G, P, W, D, A, R, S, C, T, N, Q, Y, E, F.
N	Q, Y, E, F, M, X, H, K, B, L, O, I, J, G, P, W, D, A, R, S, C, T.
O	I, J, G, P, W, D, A, R, S, C, T, N, Q, Y, E, F, M, X, H, K, B, L.
P	W, D, A, R, S, C, T, N, Q, Y, E, F, M, X, H, K, B, L, O, I, J, G.
Q	Y, E, F, M, X, H, K, B, L, O, I, J, G, P, W, D, A, R, S, C, T, N.
R	S, C, T, N, Q, Y, E, F, M, X, H, K, B, L, O, I, J, G, P, W, D, A.
S	C, T, N, Q, Y, E, F, M, X, H, K, B, L, O, I, J, G, P, W, D, A, R.
T	N, Q, Y, E, F, M, X, H, K, B, L, O, I, J, G, P, W, D, A, R, S, C.
X	H, K, B, L, O, I, J, G, P, W, D, A, R, S, C, T, N, Q, Y, E, F, M.
Y	E, F, M, X, H, K, B, L, O, I, J, G, P, W, D, A, R, S, C, T, N, Q.
W	D, A, R, S, C, T, N, Q, Y, E, F, M, X, H, K, B, L, O, I, J, G, P.

Table A.1. Sequence for the ping operation used by each node.

June 11th, 2005: Experimenting CrossROAD in static scenarios

Every node of the network ran CrossROAD and DM application on top of UNIK-OLSR. Since CrossROAD exploits a cross-layer interaction with OLSR, the prototype of the cross-layer architecture has to run on all nodes. As presented in [D13], it has been developed as a dynamic library for latest versions of UNIK-OLSR (v.0.4.8 and v.0.4.9) and it has been called *XL-plugin*.

XL-plugin is dynamically loaded by the OLSR daemon at the start-up but it is involved in the entire system only when a new instance of CrossROAD is created. Each instance of CrossROAD is strictly related to a particular service. It thus provides the specific service identifier to XL-plugin, which forwards the same information to the routing protocol as additional information to be piggybacked in routing packets. Therefore, the proactive flooding of the routing protocol guarantees a complete knowledge not only of the network topology but also of the list of services currently provided.

For all the experiments in static scenarios, all nodes started running UNIK-OLSR and XL-plugin. They only ran the routing protocol for 30 seconds to have the network topology stabilized. Then they ran DM application with different delays trying building a single overlay network. Specifically, referring to Figure A.3:

- **N** started DM as the first node of the overlay;
- **E, K, M, L, R, O** started DM after 10 seconds from **N**, as its 1-hop neighbours;
- **D, J, Q, S, P** started DM after 20 seconds from **N**, as its 2-hops neighbours;
- **C, B, G, F, I, T, Y, W** started DM after 30 seconds from **N**, as its 3-hops neighbours;
- **A, H, X** started DM after 40 seconds from **N**, as its 4-hops neighbours.

This order has been defined to reproduce the starting sequence of Pastry, where each node has to connect to one of its physical neighbour to join the overlay. Even though CrossROAD does not need to group nodes dependently on their physical position, this

allows us to better evaluate CrossROAD and XL-plugin in publishing services identifiers and building the overlay network. Single sessions of experiments are detailed below:

- Experiment 1: All nodes started running OLSR and XL-plugin, followed by DM application as specified in the aforementioned sequence. In this experiment no application messages were sent in order to evaluate the overhead introduced by the overlay management on the routing protocol. DM ran for 4 minutes, and then each node explicitly closed it producing a DisconnectMessage from the XL-plugin to notify the disconnection from the overlay. The routing protocol stopped after 30 seconds after DM ending.

- Experiment 2: All nodes started running OLSR and XL-plugin, followed by DM application as specified in the aforementioned sequence. In this experiment one application message ("Create Mailbox" message) of 100 Bytes with a random key was generated and sent on the network by each node every 100 msec for a last of 120 seconds. Before sending those messages, nodes waited all the others to join the overlay. Therefore, the first message was sent on each node after 60 seconds from N had started the overlay. The "Create Mailbox" message notifies to the destination node the storage of a mailbox with the specified identifier as the key of the message on itself. It does not require any reply.

- Experiment 3: All nodes started running OLSR and XL-plugin, followed by DM application as specified in the aforementioned sequence. In this experiment one application message ("Create Mailbox" message) of 100 Bytes with random key was generated and sent on the network by each node, except nodes **A** and **Y**, every 100 msec for a last of 120 seconds. At the same time, **A** and **Y** generated a "Get" message with a random key using the same frequency of other nodes. Before sending all messages, nodes waited all the others to join the overlay. Therefore, the first message was sent on each node after 60 seconds from N had started the overlay. The "Get" message notifies to the destination node the request of the list of messages stored in the mailbox with logical identifier equals to the key of the message (the mailbox had to be previously created by a "Create" message). The node selected as best destination for this kind of messages has to directly reply to the sender with the list of messages. Using a timestamp inside the application message, a round-trip delay can be measured.

- Experiment 4: All nodes started running OLSR and XL-plugin, followed by DM application as specified in the aforementioned sequence. In this experiment 10 nodes (**A, Y, T, R, M, H, I, C, E, G**) generated a "Get" message every 100 msec for 120 seconds, while the others (**N, D, F, J, O, P, S, B, K, W, Q, X**) only maintained the overlay receiving messages and replying directly to their sender.

- Experiment 5: It is the same procedure of experiment 2 changing the frequency of messages sending: all nodes generated a "Create Mailbox" message every 500 msec for 120 seconds. This experiment was conducted to analyze the effect of a traffic reduction on the system performance.

June 11th, 2005: Experimenting FreePastry on top of OLSR in static scenarios

In this set of experiments every node of the network ran FreePastry and DM application on top of UNIK-OLSR. As previously said, each node can join the overlay by directly connecting to another participating node or it can create its own overlay. We defined the starting sequence of these experiments setting the bootstrap node for each node of the network as one of its one hop neighbours. In addition, different nodes that had the same bootstrap started the join procedure with a delay of 1 second from each other, to reduce the probability of overlay partitioning due to failed join procedures. Specifically, we assumed that:

- **N** started DM as the first node of the overlay;
- **E**, **K**, **M**, and **R** joined the overlay using **N** as bootstrap node; the first one started DM with a delay of 10 seconds after **N**.
- **I** and **L** joined the overlay using **M** as bootstrap node; the first one started DM with a delay of 10 seconds after M.
- **O** and **P** joined the overlay using **R** as bootstrap node; the first one started DM with a delay of 10 seconds after **R**.
- **D** joined the overlay using **E** as bootstrap node; it started DM with a delay of 10 seconds after **E**.
- **B, C**, and **G** joined the overlay using **D** as bootstrap node; the first one started DM with a delay of 10 seconds after **D**.
- **A** joined the overlay using **B** as bootstrap node; it started DM with a delay of 10 seconds after **B**.
- **F** joined the overlay using **C** as bootstrap node; it started DM with a delay of 10 seconds after **C**.
- **J** joined the overlay using **K** as bootstrap node; it started DM with a delay of 10 seconds after **K**.
- **H** joined the overlay using **I** as bootstrap node; it started DM with a delay of 10 seconds after **I**.
- **Q** and **S** joined the overlay using **O** as bootstrap node; the first one started DM with a delay of 10 seconds after **O**.

- **W** joined the overlay using **P** as bootstrap node; it started DM with a delay of 10 seconds after **P**.

- **T** and **Y** joined the overlay using **S** as bootstrap node; the first one started DM with a delay of 10 seconds after **S**.

- **X** joined the overlay using **Y** as bootstrap node; it started DM with a delay of 10 seconds after **Y**.

All nodes only ran the routing protocol for 30 seconds to have the network topology stabilized. Then they ran DM application with specified delays trying building a single overlay network. Single sessions of experiments are detailed below:

- Experiment 1: All nodes started running OLSR followed by DM application as specified in the aforementioned sequence. In this experiment no application messages were sent in order to evaluate only the overhead introduced by the overlay management on the routing protocol. DM ran for 4 minutes, and then each node explicitly closed it. The routing protocol stopped after 30 seconds after DM ending.

- Experiment 2: All nodes started running OLSR followed by DM application as specified in the aforementioned sequence. In this experiment one application message ("Create Mailbox" message) of 100 Bytes with a random key was generated and sent on the network by each node every 100 msec for a last of 120 seconds. Before sending those messages, nodes waited all the others to join the overlay. Therefore, the first message was sent on each node after 60 seconds from N had started the overlay. The "Create Mailbox" message notifies to the destination node to store a mailbox with the specified identifier as the key of the message. It does not require any reply.

- Experiment 3: All nodes started running OLSR followed by DM application as specified in the aforementioned sequence. In this experiment one application message ("Create Mailbox" message) of 100 Bytes with random key was generated and sent on the network by each node (except for nodes **A** and **Y**). Messages were sent with a period of 100 msec for a last of 120 seconds. At the same time, **A** and **Y** generated a "Get" message with a random key using the same frequency of other nodes. As in the previous case, before sending all messages, nodes waited all the others to join the overlay. The "Get" message notifies to the destination node the request of the list of messages stored in the mailbox with logical identifier equals to the key of the message (the mailbox had to be previously created by a "Create" message). The node selected as best destination for this kind of messages has to directly reply to the sender with the list of messages. Using a timestamp inside the application message, a round-trip delay can be measured.

- Experiment 4: All nodes started running OLSR followed by DM application as specified in the aforementioned sequence. In this experiment 10 nodes (**A, Y, T, R, M, H, I, C, E, G**) generated a "Get" message every 100 msec for 120 seconds, while the others (**N, D, F, J, O, P, S, B, K, W, Q, X**) only maintained the overlay receiving messages and replying directly to their sender.

June 17th, 2005: Experimenting FreePastry on top of AODV in static scenarios

In this session all the experiments of FreePastry on top of OLSR were repeated running FreePastry on top of UU-AODV. The bootstrap sequence and the typology of experiments were maintained unchanged.

June 17th, 2005: Experimenting CrossROAD in case of delayed joint of the overlay

In this set of experiments (type experiment 6) node **L** was not available, and all nodes, except for **N** and **M**, started running UNIK-OLSR and XL-plugin creating two different ad hoc networks (see Figure A.4). They only ran the routing protocol for 30 seconds to have the two network topologies stabilized. Then they ran DM application with different delays following the same sequence specified for the first set of experiments on CrossROAD. **N** and **M**, which are central nodes, started the routing protocol and the overlay with a delay of 2 minutes joining the two networks in an only one. On the other hand, when all nodes of the first group correctly participated in the overlay, **A** and **Y** started sending a "Get" message every 200 msec for 6 minutes.

In this scenario when **N** and **M** joined the experiment, other nodes, that had been already running the overlay, became aware of the new participants thanks to the cross-layer interactions with the routing protocol. In fact nodes **A** and **Y** in the first phase of the experiment (when **N** and **M** were not active) sent messages only to nodes of their network, while after the joining of central nodes, they distributed messages on all nodes of the network.

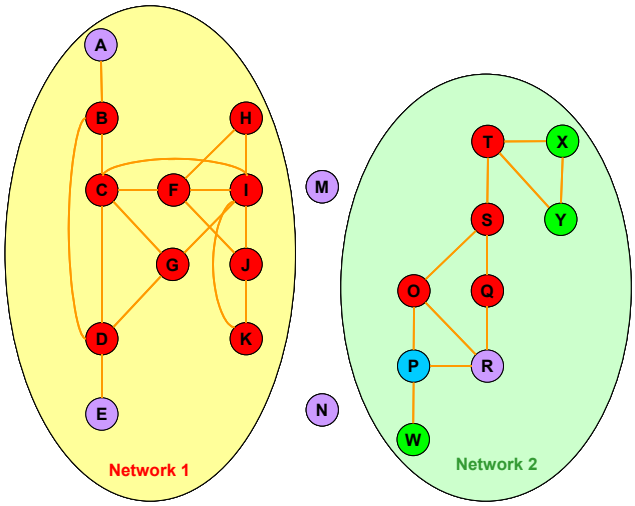


Figure A.4. Initial network topology for the delayed joint overlay experiment both in case pf CrossROAD and FreePastry

June 17th, 2005: Experimenting FreePastry on OLSR in case of delayed joining of the overlay

The initial network topology of this set of experiments is the same of the Figure A.4. Since central nodes started the experiment with a delay of 2 minutes after other nodes, the original sequence of bootstrap nodes used for previous experiments with FreePastry has to be updated. Specifically:

- **R** and **E** are the new starters of the overlay;
- **I** and **J** joined the overlay using F as bootstrap node;
- **K** joined the overlay using **J** as bootstrap node.
- **N**, that was the original starter of previous experiments, joined the overlay with a delay of 2 minutes using **E** as bootstrap node;
- **L** and **M**, the other central nodes, joined the overlay using **N** as bootstrap node. Their delays is 2 minutes and 10 seconds, to wait **N** to be active in the overlay.

Starting delays of the overlay of these nodes were updated depending on the instant when their bootstrap nodes had joined the overlay.

One of the main drawbacks of FreePastry on ad hoc network consists of the high probability of bootstrap failure. This is mainly due to the presence of unstable links in the network topology and consequently failures in remote connections to join and collect overlay data structures information. For this reason generally more than one ring overlay was constituted in most of the experiments, and no rejoining operation is provided by Pastry. Specifically in this set of experiments, having more than one ring, nodes **A** and **Y** couldn't distribute messages on all nodes of the network, and results are not meaningful as in case of CrossROAD.

17th June: Experiments with Routing Protocols on mobile network

In this set of experiments we added mobility to some nodes in the network in order to investigate how the routing protocols react to changes of topology. Hence we decided to reduce the number of active ping operation to only one connection between the two most distant nodes. This type of experiments can be referred as 2-Central node Swap.

Experiment 1: all nodes started running the OLSR protocol together (t=0). After 30sec node Y started pinging node A continuously for 210 sec. At t=90 nodes M and N (two central nodes) started moving and swapped their positions after 30sec (t=120), then they remained in this new configuration until the end of the experiment. At t= 240 the experiment ended, so ping operation and routing protocol stopped running.

Experiment 2: we repeated the same experiment with AODV too, so the involved nodes and the last are the same of experiment 1.

23rd July: Experiments with Routing Protocols on mobile network

In this set of experiments we continued investigating the behaviour and the performance of UNIK-OLSR and UU-AODV in presence of mobile nodes. As in the previous day of experimentation, we reduced the number of active ping operation to only one connection between the two most distant nodes in order to simplify the complexity of the network.

Furthermore, we had to decrease the number of nodes in the network to 22 for the whole day due to technical problems with some laptops.

Experiment 1: this experiment is a 2-Central nodes Swap one in which the central nodes J and N changed their position during a continuous ping operation from two external nodes. More precisely, all nodes started running the OLSR protocol together (t=0). After 30sec the external node Y started pinging node A continuously for 210 sec. At t=90 nodes J and N (two central nodes) started moving and swapped their positions after 30sec (t=120), then they remained in this new configuration until the end of the experiment. At t= 240 the experiment ended, so the ping operation and the routing protocol stopped running.

Experiment 2: this experiment can be referred as 4-Central nodes Swap: the four central nodes J, M, O and N changed their positions in clockwise manner during a continuous ping operation from two external nodes. More precisely, all nodes started running the OLSR protocol together (t=0). After 30sec needed for network stabilization, the external node Y started pinging node A continuously for 300 sec. At t=90 node N started moving and reached the position of node J in 30 sec (event 1), then it remained in this new location until the end of the experiment. At t=120 node J started moving toward node M and once reached the new position after 30 sec (event 2). Event 3 started at t=150 when node M moved and it finished when M reached the position of node O at t=180. In the same instant node O moved to the initial location of node N (event 4) stopping after 30 sec. At t=330 the experiment ended, so the ping operation and the routing protocol stopped running. Figure A.5 shows the 4 events of the experiments.

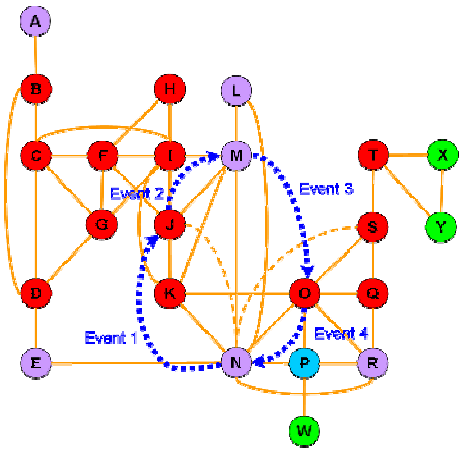


Figure A.5 4-Central nodes Swap scenario

Experiment 3: this experiment can be referred as Roaming node: all the nodes were static except the "roaming node" that moved crossing the entire network. The reference scenario is shown in Figure A.6. The experiment lasted 410 sec. After an initial phase for OLSR (protocol) stabilization (30 sec), node Y started pinging node A for 380 sec. At t_1=90 from the initial position near node A, the pinger Y started moving inside the building along the corridor with a speed of about 1 m/sec. It reached position of node C after 30 sec (t_2=120), position of node I after 15 sec (t_3=135), then node K after 15 sec

(t_4=150), node O after 15 sec (t_5=165), position of node S after 30 sec (t_6=195) and finally it reached the opposite side of the network near node X after other 30 sec (t_7=225). Once it has reached the last position, it immediately moved in the opposite direction following the reverse path and taking the same lags as in the forward path (from t_7 to t_f). Finally it reached the starting position near node A after 2 min and 15 sec (t_f=350).

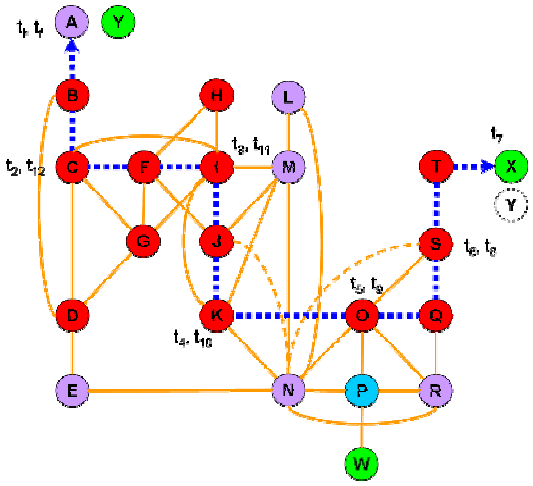


Figure A.6. Roaming node scenario

Experiment 4: in this case we repeated experiment 1 with the reactive routing protocol running on each node.

Experiment 5: we repeated experiment 2 running AODV on each nodes.

Experiment 6: experiment 3 is repeated with AODV.

July 23th, 2005: Experimenting CrossROAD in a mobility scenario

In this set of experiment we used 22 nodes with only 2 central nodes as shown in Figure A.7a. The mobility scenario chosen for CrossROAD experiments consists of a temporary partitioning of the network in two separate ad hoc networks and the subsequent re-conjunction. To this aim central nodes started moving along blue dashed lines (see Figure A.7b) causing the loss of all central links and the consequent partitioning of the network and the overlay. After having the new topology stabilized for 1 minute, they came back in the original positions, rejoining the original network topology. In Figure A7.b temporary links are drawn as green dashed lines.

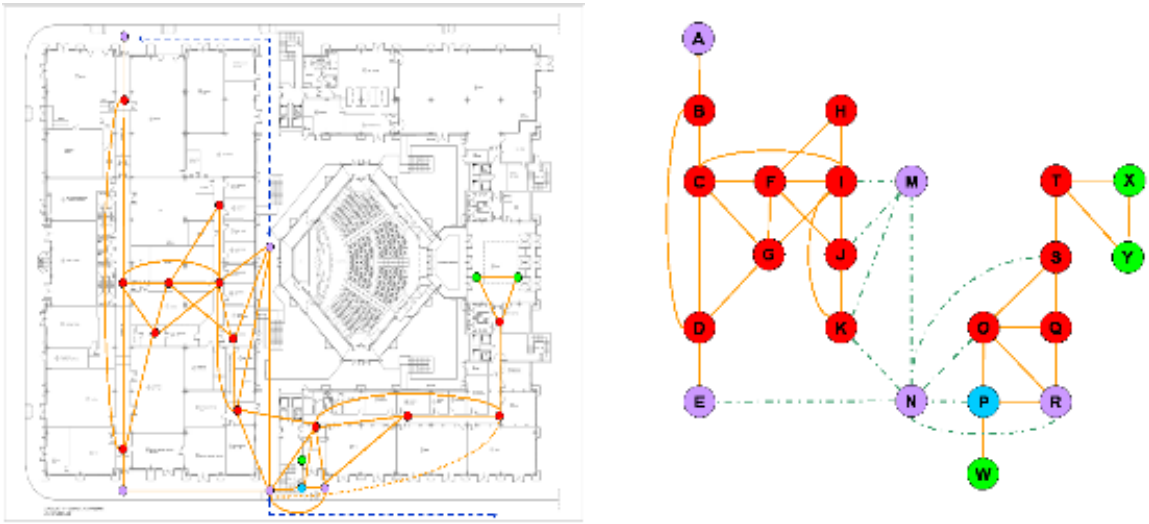


Figure A.7.

a) Physical position of nodes in the mobility scenario.

b) Topology graph in the mobility scenario.

All nodes started running OLSR, XL-plugin, CrossROAD and DM application with the same sequence of other experiments. Node **A** sent a "Get" message with key **K*** that was logically closest to node **Y** every 100 msec for the entire last of the experiment. In addition, nodes **M** and **N** started moving after 30 seconds from the beginning of the application. They continuously moved in two different directions for 120 seconds, and then they stayed in their new position for 60 seconds to stabilize the topology. After that they spent 120 seconds to come back in the original positions, and after other 60 seconds the experiment ended. When **M** and **N** reached their new position, the network was partitioned in two different networks. **A** became aware of the topology change through CrossROAD and it sent the message to the new best destination for K*, that was identified by node **B**. Actually, in this particular experiment, node **A** didn't have a stable route towards node **Y**, and for this reason it couldn't send the message to the real best destination. It sent the selected message to node **B** for the entire last of the experiment. For this reason we couldn't obtain new results.