
Department of Information Engineering, University of Pisa, Italy

*Ph.D. Thesis - XVIII Cycle*

# Design, Implementation and Experimental evaluation of CrossROAD: a novel P2P platform for MANETs

Ph.D. Candidate

*Franca Delmastro*

Advisors

*Prof. Giuseppe Anastasi*

*Ing. Enrico Gregori*

February 2006

# Contents


1. Introduction 3

2. Thesis Contribution 5

3. Thesis Layout 7

4. MANET: Multi-hop Ad hoc NETworks 9
   - 4.1. Introduction . . . 9
   - 4.2. Multi hop Ad hoc Networks . . . 10
   - 4.3. Architecture and Protocols . . . 11
     - 4.3.1. Enabling Technologies . . . 11
     - 4.3.2. Networking Protocols . . . 12
     - 4.3.3. Middleware and Applications . . . 12

5. Routing Protocols for Ad Hoc Networks 15
   - 5.1. AODV: Ad hoc On demand Distance Vector routing . . . 16
   - 5.2. OLSR: Optimized Link-state routing protocol . . . 18

6. P2P platforms in wired networks 21
   - 6.1. Structured overlay networks . . . 22
     - 6.1.1. Content-Addressable Network . . . 23
     - 6.1.2. Chord . . . 24
     - 6.1.3. Pastry . . . 25

7. Small-scale Ad Hoc network test-bed: Routing and Middleware performance 29
   - 7.1. Test-bed reference architecture . . . 30
   - 7.2. Experiments Environment . . . 32
   - 7.3. Experiments warm-up: a qualitative analysis . . . 34
     - 7.3.1. Unik-OLSR Testing . . . 34
     - 7.3.2. UU-AODV Testing . . . 35
     - 7.3.3. FreePastry Testing . . . 36
   - 7.4. Quantitative analysis . . . 37
     - 7.4.1. AODV and OLSR Performance . . . 37
       - 7.4.1.1. Experiment 1 . . . 38
       - 7.4.1.2. Experiment 2 . . . 39

7.4.2. Performance of FreePastry on Ad Hoc Networks . . . . . 40
7.5. Lessons learned and further work . . . . . 46

8. The Cross-Layer Architecture 49
8.1. The Network Status . . . . . 50
8.2. Examples of Cross-Layer interactions . . . . . 52

9. From Pastry to CrossROAD:
Cross-layer Ring Overlay for AD hoc networks 55
9.1. Node Identifiers and Service Discovery protocol . . . . . 55
9.2. Managing overlay data structures . . . . . 57
9.3. CrossROAD Subject-based routing . . . . . 58

10. Software Architecture 61
10.1. XL-plugin: the NeSt implementation . . . . . 61
10.2. The P2P CommonAPI and its cross-layer enhancements . . . . . 66
10.3. CrossROAD Software Architecture . . . . . 72

11. CrossROAD on a small-scale ad hoc network testbed 77
11.1. Testbed architecture and experiments environment . . . . . 77
11.2. Experiments and performance evaluation . . . . . 78
11.2.1. Routing performance . . . . . 78
11.2.2. Overlay network performance . . . . . 81
11.3. Conclusions . . . . . 84

12. CrossROAD on a medium-large scale ad hoc network testbed 87
12.1. The network topology . . . . . 88
12.2. Middleware Experiments . . . . . 89
12.3. Network load analysis . . . . . 91
12.4. Delays Analysis . . . . . 97
12.5. Data Distribution in case of delayed joining of the overlay . . . . . 98
12.6. Conclusions . . . . . 99

13. Distributed Applications on CrossROAD 101
13.1. A Group-Communication application: the Whiteboard . . . . . 101
13.1.1. Scribe Overview . . . . . 102
13.1.2. Experimental Environment . . . . . 103
13.1.3. Performance with Pastry . . . . . 105
13.1.4. Multicast Tree Quality . . . . . 108
13.1.5. CrossROAD Improvements . . . . . 109
13.1.6. Overlay management overhead . . . . . 111
13.2. Content Sharing: The UDDI approach . . . . . 113

14. Conclusions 117

A. XL-CommonAPI: the complete specification 121

# List of Figures


4.1. MANET Layered Architecture . . . . . 11

5.1. AODV Route Discovery . . . . . 17
5.2. AODV unidirectional links . . . . . 17
5.3. MPRs selection . . . . . 18

6.1. Pastry Overlay Data Structures . . . . . 26
6.2. Example of Pastry routing . . . . . 27

7.1. Reference Architecture . . . . . 30
7.2. Experiments Area . . . . . 32
7.3. Experiments Scenario . . . . . 33
7.4. Routing Network Topology . . . . . 38
7.5. OLSR overhead . . . . . 39
7.6. AODV overhead . . . . . 40
7.7. OLSR overhead in case of disconnection event . . . . . 41
7.8. AODV overhead in case of disconnection event . . . . . 41
7.9. Network Topology for FreePastry Experiments . . . . . 42
7.10. Traffic Load on each node running Pastry on top of OLSR . . . . . 43
7.11. Traffic Load on each node running Pastry on top of AODV . . . . . 43
7.12. Node F traffic on an OLSR network . . . . . 44
7.13. Node F traffic on an AODV network . . . . . 45
7.14. Node A traffic profile (AODV case) . . . . . 45

8.1. Cross-Layer Architecture . . . . . 50

9.1. Cross-layer interactions for the service discovery . . . . . 56
9.2. Sending of an application message . . . . . 57
9.3. Overlays subject-basd routing . . . . . 59

10.1. Overview of the software architecture . . . . . 62
10.2. UNIK-olsr software architecture . . . . . 62
10.3. CrossROAD and routing protocol interaction through the plug-in definition. . . . . 64
10.4. Nodes communication example. . . . . 64
10.5. XL-plugin as cross-layer interaction between CrossROAD and OLSR. . . . . 65

10.6. XL-plugin internal data structures. . . . . 65
10.7. Sequence Diagram . . . . . 66
10.8. XL-Common API class diagram. . . . . 68
10.9. CrossROAD package diagram . . . . . 72
10.10 CrossROAD activity diagram . . . . . 76

11.1. String topology . . . . . 79
11.2. Experimental network topology . . . . . 81
11.3. CrossROAD: Throughput related to main nodes. . . . . 82
11.4. Pastry: Throughput related to main nodes. . . . . 83
11.5. CrossROAD mobility experiment . . . . . 84
11.6. CrossROAD reaction to network partitioning . . . . . 84

12.1. Testbed area . . . . . 87
12.2. Physical position of nodes . . . . . 88
12.3. Network topology . . . . . 89
12.4. Topology graph . . . . . 89
12.5. Network topology for delayed joining experiment . . . . . 92
12.6. Average aggregate network load . . . . . 93
12.7. Average aggregate network load, the first phase . . . . . 94
12.8. Average network load . . . . . 94
12.9. Routing traffic load . . . . . 95
12.10 CrossROAD overlay . . . . . 96
12.11 Pastry on OLSR overlays . . . . . 97
12.12 Pastry on AODV overlays . . . . . 97
12.13 Delay Distribution . . . . . 98
12.14 CrossROAD data Distribution . . . . . 99

13.1. Two nodes running WB . . . . . 102
13.2. Scribe building the tree (a), and disseminating messages (b). . . . . 103
13.3. Network Topology . . . . . 104
13.4. Network Load with Pastry . . . . . 112
13.5. Network Load with CrossROAD . . . . . 112
13.6. UDDI4m nodes on CrossROAD overlay . . . . . 114

14.1. Integration of mobile and fixed p2p systems. . . . . 118

A.1. Packages Diagram: XL-CommonAPI and its interactions with p2p systems and applications. . . . . 121
A.2. Class Diagram: Id interface . . . . . 122
A.3. Class Diagram: IdFactory interface . . . . . 122
A.4. Class Diagram: NodeHandle interface . . . . . 123
A.5. Class Diagram: Message interface . . . . . 123
A.6. Class Diagram: RouteMessage interface . . . . . 123
A.7. Class Diagram: Endpoint interface . . . . . 124

A.8. Class Diagram: OverlayRoutingTable interface . . . . . 126
A.9. Class Diagram: NetworkRoutingTable interface . . . . . 127
A.10.Class Diagram: Node interface . . . . . 127
A.11.Class Diagram: Application interface . . . . . 127
A.12.Class Diagram: InitCommonAPI class . . . . . 128

# List of Tables

11.1. Overall Packet Delivery Ratio. . . . . 81
11.2. CrossROAD delays for the overlay construction. . . . . 82

12.1. CrossROAD delays for the overlay construction. . . . . 95
12.2. Delays distribution . . . . . 96

# Acknoledgments

During these years I had the pleasure to work with several people who helped me with their experience and gave me their invaluable support addressing my research.

First of all, I would like to express my gratitude to my advisors, Prof. Giuseppe Anastasi of the Information Engineering Department (University of Pisa), Dr. Enrico Gregori and Dr. Marco Conti of the Institute of Informatics and Telematics of CNR in Pisa. Their guidelines, suggestions, and encouragements supported me in all this time.

Then, I would like to thank all my colleagues and friends with which I shared this important experience, especially Eleonora, Andrea, Luciana, Raffaele, Patrizia, Giovanni e Gaia. Working with them I learned that the collaboration is the best help you can give.

Finally, I would like to give my special thanks to my husband Giacomo and all my family for their loving support.

# 1. Introduction

Mobile ad hoc networks (MANETs) represent complex distributed systems composed of wireless mobile nodes, which can freely and dynamically self-organize themselves 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 Bluetooth and IEEE 802.11, greatly facilitated the deployment of ad hoc technology outside the military domain, generating a renewed and growing interest in 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: 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 they need. This constitutes a major complexity/inefficiency in the development of MANET applications. Indeed, most of the MANET research is 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 [MC02] [Her03] [MPR01] [MCZE02b] [BCM05].

Their emphasis is on supporting transient data sharing [MPR01] between nodes in communication range, data replication for disconnected operations [BCM05] [MCZE02a], or both [Her03]. 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 [MCZE02a] [MPR01] [ADGS02]. In addition, in order to develop distributed applications aimed at sharing resources between nodes, peer-to-peer (p2p) systems represent a reliable model also for ad hoc networks. In fact, ad hoc networking shares many concepts, such as distribution and cooperation, with the p2p computing model [SGF02], representing a natural paradigm for these networks. However, while all these technologies provide service abstractions that highly simplify application development in wired networks, their efficiency in ad hoc environments is still an open issue.

Focusing on p2p systems, one of their main defining characteristic 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. Best suited applications 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.

In this thesis we investigate the efficiency of p2p middleware platforms when implemented in mobile ad hoc networks and their possible optimizations. This work started from the study of existent p2p systems developed for Internet with particular attention to *structured overlay networks* [RFH+01] [SMK+01] [RD01]. The analysis led to the selection of a particular solution (Pastry [RD01]) for its scalability with high numbers of nodes, and to an experimental evaluation of its performance on small-medium scale MANETs [BCDP05] [CDT05] [d16]. Experimental results highlithed limitations and drawbacks of this system on ad hoc networks, encouraging the definition of an optimized solution based on a *cross-layer* architecture [CMTG04]. Specifically, a new p2p system for MANETs has been defined [Del05] and an exhaustive performance analysis on small and medium scale ad hoc networks [BCDG05] [d16] highliths its advantages. In addition, several applications already developed for Internet have been ported on this new system [DP05] [d13] further optimizing their performance.

# 2. Thesis Contribution

The main contribution of this thesis is to provide a new optimized p2p system for ad hoc networks based on a cross-layer architecture in order to exploit many distributed applications developed for the Internet on top of legacy p2p systems even in MANET environments. Using known services on MANETs can represent a good incentive for users to adopt this technology in the daily use supporting mobility and cooperation between users and devices. However, simulations and experimental results [CDT05] show that existent solutions developed for wired networks cannot be adopted in MANETs, since they introduce high network overhead not supporting nodes mobility and intermittent connectivity. For this reason a new solution called *CrossROAD* has been designed [Del05].

This system inherits the main charateristics of structured overlay networks, with particular reference to the Pastry model [RD01]. At the same time it drastically reduces the network overhead needed by the original system to maintain the overlay data structures and it correctly manages network partitioning and nodes mobility exploiting a cross-layer interaction with a proactive routing protocol.

The main reason of low performance of legacy p2p systems on ad hoc networks is represented by the complete lack of correspondence between physical and logical address spaces. In fact, structured p2p systems define a logical space where nodes and data are mapped through a Distributed Hash Table (DHT). A logical identifier is associated to each node independently from their physical position, and data are distributed among nodes following a specific policy called *subject-based routing*: a key value is associated to each message to be sent on the network and the destination node is selected as that whose logical identifier is the closest one to the key value. Thus, nodes that are physical neighbors are probably distant on the logical space and data are delivered to their final destination following a proximity logic between nodes known by the sender. Since these systems have been thought for Internet, where thousands of nodes can be involved in the same service, every node does not know all the other participants. For this reason each message could pass through several nodes physically distant before reaching the final destination, even if it is a physical neighbor of the sender.

A cross-layer architecture as defined in [CMTG04] allows the interaction between protocols belonging to different layers of the legacy protocol stack. Each protocol can share its information with the others supporting their optimization. In particular, using a proactive routing protocol at the network level and exporting topology information to other protocols of the stack, allows each node to have a complete knowledge of the network topology and to become aware of nodes mobility and connection/disconnection events. In case of p2p systems, this guarantees the possibility of maintaining a correspondence between logical and physical spaces and it is also exploited

to spread upper-layer information on the network through the proactive flooding of the routing protocol.

CrossROAD exploits these two main features of the cross-layer interaction with a proactive routing protocol. Therefore it drastically reduces the network overhead and guarantees a high responsiveness of the system to nodes mobility and network partitioning. In addition, since CrossROAD implements a common interface for structured p2p systems already defined in [DZD+03], it supports all distributed applications developed on top of these systems and it can furtherly optimize them through cross-layer extensions applied to the same interface [CDG].

During this thesis CrossROAD has been designed, developed and tested together with a prototype of cross-layer architecture aimed at only middleware and routing interactions. An exhaustive experimental analysis of the entire system performance has been conducted on small and medium scale ad hoc networks, highlithing advantages of this new solution compared with Pastry system. In addition several applications have been developed on top of it and further optimizations for ad hoc networks are currently under development.

# 3. Thesis Layout

The thesis is organized as follows. First of all a description of multi hop ad hoc networks is given in Chapter 4 with particular attention to architecture and protocols. Specifically, in Chapter5 we focus on routing protocols for ad hoc networks, detailing main features of a reactive and a proactive solution. Then an overview of existent p2p systems on wired networks is given in Chapter 6 with particular attention to structured overlay networks and the Pastry model. A preliminary experimental analysis of routing protocols and Pastry on a small-scale testbed is detailed in Chapter 7. Starting from this results, we describe main fetaures of a cross-layer architecture in Chapter 8 and the design of CrossROAD and the software architecture of the entire system are described in Chapter 9 and 10 respectively. Then an extensive experimental evaluation of CrossROAD on small and medium-large scale testbeds is given in Chapter 11 and 12. Finally, description and performance evaluation of some distributed applications developed on top of CrossROAD are detailed in Chapter 13 followed by conclusions and future work.

# 4. MANET: Multi-hop Ad hoc NETworks

## 4.1. Introduction

The hardware and software progress of the last ten years provided the basic elements (wearable computers, several wireless-network technologies, devices for sensing and remote control, etc.) for developing pervasive computing and communication systems [CD04]. We can envisage a physical world with pervasive, sensor-rich, network-interconnected devices embedded in the environment [ECPS02]. In these systems the environment is saturated with computing and communication capabilities, that interact among them, and with the users. In this environment, virtually everything (from key chains to computers, and PDAs) is connected to the network, and can originate and respond to appropriate communications. The nature of ubiquitous devices makes wireless networks the easiest solution for their interconnection. Furthermore, in a pervasive computing environment, the infrastructure-based wireless communication model is often not adequate: it takes time to set up the infrastructure network, while the costs associated with installing infrastructure can be quite high. These costs and delays may not be acceptable for dynamic environments where people and/or vehicles need to be temporarily interconnected in areas without a pre-existing communication infrastructure (e.g., inter-vehicular and disaster networks), or where the infrastructure cost is not justified (e.g., in-building networks, specific residential communities networks, etc.). In these cases, ad hoc networks provide a more efficient solution [CD04].

In the ad hoc networking paradigm, a group of mobile devices self-organizes to create a network by exploiting their wireless network interfaces, without the need of any pre-deployed infrastructure. The simplest ad hoc network is a peer-to-peer network formed by a set of stations within the range of each other that dynamically configure themselves to set up a temporary single-hop ad hoc network. Bluetooth piconet is probably the most widespread example of single-hop ad hoc network [Bid01]. 802.11 WLANs can also be implemented according to this paradigm, thus enabling laptops communications without the need of an access point. Several emerging wireless-network standards support the ad hoc networking paradigm. IEEE 802.15.4 for short-range low data rate ($<$ 250 kbps) networks (also known as Zigbee), Bluetooth (IEEE 802.15.1) for personal area networks, and the 802.11 standards family for high-speed medium-range ad hoc networks [CD04].

Single-hop ad hoc networks interconnect devices that are within the same transmission range. This limitation can be overcome by exploiting the multi-hop ad hoc paradigm. In a multi-hop network, often referred to as MANET, the network nodes (e.g., the users mobile devices) must cooperatively provide the features usually provided by the network infrastructure (e.g. routers, switches, servers). Nearby nodes can communicate directly by exploiting a single-hop wireless technology

(e.g., Zigbee, Bluetooth, 802.11, etc.) while devices that are not directly connected communicate by forwarding their traffic via a sequence of intermediate devices ([CCL03], [BCGS04]).

Multi-hop ad hoc networking is not a new concept having been around for over twenty years, mainly as tactical networks. Recently, the emerging of wireless networking technologies for consumer electronics are pushing multi-hop networks outside the military domain. Pure multi-hop mobile ad hoc networks are attractive for specialized scenarios like disaster recovery, vehicle-to-vehicle communications, and home networking but have a very limited mass-market deployment. To turn mobile ad hoc networks into a commodity, we should move to a more pragmatic scenario in which multi-hop ad hoc networks are used as a flexible and *low cost* extension of Internet. Indeed, a new class of networks is emerging from this view: the mesh networks [BCG05]. Unlike MANETs, where every routing node is mobile, routing nodes in mesh networks are stationary and form the networks backbone, which has connections to Internet. Client nodes connect to the mesh nodes and use the backbone to access the Internet. Mesh networks scenarios are moving multi-hop wireless networking from emergency-disaster-relief and battlefield scenarios to the main networking market.

## 4.2. Multi hop Ad hoc Networks

Mobile ad hoc networks are formed dynamically by a set of mobile nodes that are connected via wireless links without using any existing network infrastructure, such as a base station, for their operation (hence they are also referred to as infrastructure-less). The nodes are free to move randomly and organize themselves arbitrarily. Thus, the networks wireless topology may change rapidly and unpredictably. Such networks may operate in a standalone fashion, or may be connected to the Internet. The ad hoc networks flexibility and convenience do come at a price. Indeed, in addition to the traditional problems of wireless networking, the multi-hop nature, and the lack of fixed infrastructure add a number of characteristics, complexities, and design constraints that are specific to these networks [CCL03]:

- *Autonomous and infrastructure-less.* MANET does not depend on any established infrastructure or centralized administration. Each node operates in distributed peer-to-peer mode, acts as an independent router and generates independent data. This makes fault detection and management very difficult.
- *Multi-hop routing.* Every node acts as a router and forwards each others packets to enable information sharing between mobile hosts.
- *Dynamically Changing Network Topologies.* Nodes mobility causes changes in the network topology frequently and unpredictably. This may produce route changes, network partitions, and possibly packet losses.
- *Variation in Link and Node Capabilities.* Nodes might have different software/hardware resources and configurations, thus generating a heterogeneous network which is complex to manage. For example nodes may be equipped with heterogeneous wireless cards which produces asymmetric links.

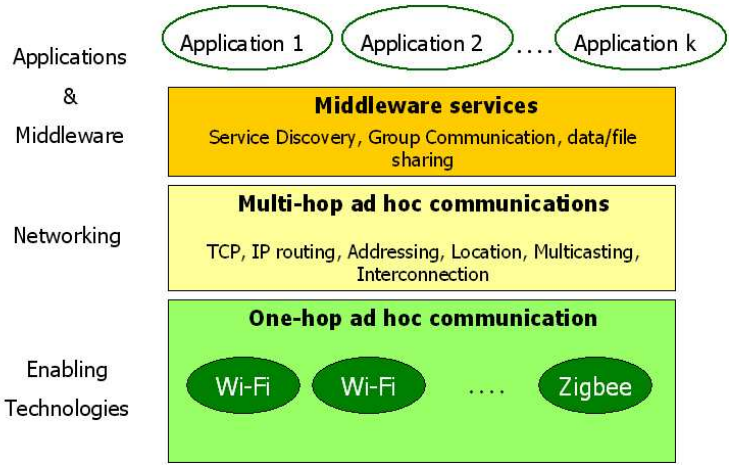



Figure 4.1.: MANET Layered Architecture

To cope with these issues, several functions and protocols need to be implemented in MANET protocol stack. This produced extensive research activities both in academy and in industry. The IETF MANET working group has the main role in standardizing protocols for mobile multi-hop ad hoc network [man]. IETF MANET WG proposes a view of mobile ad hoc networks as an evolution of the Internet. This mainly implies an IP-centric view of the network, and the use of a layered architecture. This paradigm has greatly simplified network design and make easier the interconnection to the Internet. However, as explained in Section 4.3, this approach limits the development of efficient solutions which are very important in a resource-constrained environment. For this reason, new MANET organizations based on the cross-layering approach are emerging.

## 4.3. Architecture and Protocols

In this section, we briefly review the main protocols (see Fig.4.1) required to support distributed applications in a mobile ad hoc environment. In the presentation we will follow a layered approach starting from the enabling technologies up to the applications.

### 4.3.1. Enabling Technologies

Enabling Technologies are a fundamental building block of ad hoc networks by providing wireless interfaces able to guarantee direct single-hop communications between users devices. Currently, connectivity is provided by wireless LANs (e.g., 802.11 WLAN standard), or somewhat smaller and less expensive Bluetooth devices. However, to provide the network connectivity to sensor nodes, technologies cheaper, simpler, lighter, and with lower power than these are necessary. These solutions should be able to support low bit rates (e.g., less than 100 Kbps), short ranges (few meters), low power requirements, and above all, they must be extremely inexpensive. The IEEE 802.15.4 specification, also known as Zigbee, is one of the most promising solutions for short-range, low data rate (< 250 kbps), personal area networking. Specifically, 802.15.4 is designed

to address wireless networking requirements for industrial control, home automation and control, inventory management, as well as wireless sensor networks [CGH+02]. A detailed discussion of enabling technologies for Ad hoc Wireless Networks can be found in [CD04].

### 4.3.2. Networking Protocols

The aim of the networking protocols is to use the one-hop transmission services provided by the enabling technologies to construct end-to-end (reliable) delivery services, from a sender to one (or more) receiver(s). This is the set of problems on which researchers mainly focused on [CCL03]. The primary objective of networking protocols is to deliver a message from a sender to a receiver outside its transmission range by exploiting the services offered by intermediate nodes. The simplest solution is based on flooding the messages through the network. Of course, flooding does not scale, and hence this approach is only suitable for limited size networks. Controlling the flooding area and/or the number of messages to flood can help to refine the technique. For example, flooding can be used only when the first message have to be sent from the sender to the receiver to discover the receiver location, then successive packets can be delivered along the path discovered when delivering the first message. Identifying the path between the sender and the receiver and then delivering the packets along this path is exactly the objective of routing and forwarding algorithms, respectively. Numerous routing protocols and algorithms have been proposed, and their performance under various network environments and traffic conditions have been studied and compared. Several surveys and comparative analysis of MANET routing protocols have been published, see e.g. [BR04].

The classification of routing protocols for MANETs is detailed in Chapter 5 with particular attention to two specific solutions.

### 4.3.3. Middleware and Applications

Mobile ad hoc systems currently developed have the applications directly running on top of the transport layer by exploiting via the legacy socket interface the services offered by the TCP and UDP protocols. No middleware services are generally implemented, but each application has to handle all the services it needs. This constitutes a major complexity/inefficiency in the applications development, which can be fixed with the development of middleware platforms tuned on the unique MANET features. Currently, research on middleware for mobile ad hoc networks is still in its infancy and only recently middleware proposals for mobile ad hoc environments appeared, e.g., see [BCM05], and references herein. As ad hoc networks shares many concepts with the peer-to-peer (p2p) computing system, exploiting the p2p paradigm for designing middleware platforms for MANET is emerging as a very promising direction [CDT05]. In ([BCDG05], [CGT05]) the authors investigate the efficiency of p2p middleware platforms when implemented on mobile ad hoc networks. In spite of the massive research efforts, multi-hop ad hoc networks have still limited usage scenarios. More precisely, currently multi-hop ad hoc networks are generally used either for implementing tactical networks (e.g., military communications, automated battlefields) or for

supporting communications in specialized civilian scenarios (disaster recovery, planetary exploration, vehicular networks, etc). On the other hand, general-purpose civilian applications able to push a mass-market deployment of these networks have not yet been identified. A first step in this direction is reported in [d10] where new applications (city cab, mobile games, and shopping mall) have been identified that can leverage the ad hoc technology to provide valuable services to the user.

In this thesis we investigate a complete MANET architecture involving all these protocols, from the network layer up to the application layer, in order to understand how this new technology can become part of the daily life of million of users and what kind of enhancements are necessary to improve it.

# 5. Routing Protocols for Ad Hoc Networks

The highly dynamic nature of a mobile ad hoc network results in frequent and unpredictable changes of network topology, adding difficulty and complexity to routing among mobile nodes. Challenges and complexities, coupled with the critical importance of routing protocols in establishing communications among mobile nodes, make routing area the most active research area within the MANET domain. They also represents the basis on which all upper layer protocols are developed. for this reason a deep analysis of existent routing protocols and their performance is necessary before studying the rest of the architecture.

Numerous routing protocols and algorithms have been proposed, and their performance has been studied and compared under various network environments and traffic conditions. Several surveys and comparative analysis of MANET routing protocols have been published [BRT99], [BR04]. [Per00] provides a comprehensive overview of routing solutions for ad hoc networks, while an updated and in depth analysis of routing protocols for mobile ad hoc network is presented in [BR04]. A preliminary classification of the routing protocols can be done via the type of cast property, i.e. whether they use a *Unicast*, *Geocast*, *Multicast*, or *Broadcast* forwarding.

*Broadcast* is the basic mode of operation over a wireless channel. Each message transmitted on a wireless channel is generally received by all neighbors at one-hop from the sender. The simplest implementation of the broadcast operation to all network nodes is by classical flooding, but this may cause the broadcast storm problem due to redundant re-broadcast [NTCS99]. Schemes have been proposed to alleviate this problem by reducing redundant broadcasting. [SW03] surveys existing methods for flooding a wireless network intelligently.

*Unicast* forwarding means a one-to-one communication, i.e., one source transmits data packets to a single destination. This is the largest class of routing protocols found in ad hoc networks.

*Multicast* routing protocols come into play when a node needs to send the same message, or stream of data, to multiple destinations.

*Geocast* forwarding is a special case of multicast that is used to deliver data packets to a group of nodes situated inside a specified geographical area. Nodes may join or leave a multicast group as desired, on the other hand, nodes can only join or leave a geocast group only by entering or leaving the corresponding geographical region.

This work only focuses on unicast routing protocols whose primary goal is the correct and efficient route establishment and maintenance between couples of nodes, so that messages may be delivered reliably and in a timely manner.

They are typically divided into two main categories: *proactive* routing protocols and *reactive on-demand* routing protocols [BRT99].

Proactive routing protocols derive from legacy Internet distance-vector and link-state protocols. They attempt to maintain consistent and updated routing information for every pair of network nodes by propagating, proactively, route updates at fixed time intervals. Since the routing information is usually maintained in tables, these protocols are sometimes referred to as Table-Driven protocols.

Reactive on demand routing protocols, on the other hand, establish the route to a destination only when it is requested through the route discovery process. Once a route has been established, it is maintained until either the destination becomes inaccessible, or until the route is no longer used, or expired [BRT99] [BR04].

Actually, another category of routing protocols exists in literature: *hybrid* routing protocols. They integrate characteristics of these two families and exhibit proactive or reactive behaviours depending on the situation. ZRP (*Zone Routing Protocol*) [Haa97] belongs to this category, taking advantage of proactive discovery within a node's local neighborhood, and using a reactive protocol for communicating with these neighbors.

Most work on routing protocols is being performed in the framework of the IETF MANET working group [man], where four routing protocols are currently under active development. These include two reactive routing protocols, AODV [PR99] and DSR [JM96], and two proactive routing protocols, OLSR [rfc] and TBRPF [tbr]. There has been good progress in studying protocols behavior (almost exclusively by simulation), as can be seen in the large conference literature. However, today AODV and OLSR are the most mature from the implementation standpoint; for the others either updated implementations are not available (DSR) or there are no freely available implementations (TBRPF). For this reason, reliable implementations of AODV [aod] and OLSR [Ton] have been integrated in this test-bed, and their main characteristics are explained in following sections.

## 5.1. AODV: Ad hoc On demand Distance Vector routing

As previously mentioned, reactive routing protocols depart from the legacy Internet approach adding a particular optimization: to discover a route only when it is needed. AODV minimizes the number of route broadcasts by creating routes on-demand via a route discovery procedure that works as follows. Whenever a traffic source needs a route to a destination, it initiates a route discovery by floodig a route request (RREQ) for the destination in the network, and then it waits for a route reply (RREP). When an intermediate node receives the first copy of a RREQ packet, it sets up a reverse path to the source using the previous hop of the RREQ as the next hop of the reverse path. Specifically, if it has a valid route available for the destination, it unicasts a RREP back to the source via the reverse path, otherwise, it re-broadcasts the RREQ packet. Duplicate copies of the RREQ are immediately discarded upon reception at every node. Once the RREQ reaches the destination, it unicasts the RREP back to the source and if it is correctly received by the source, it establishes a forward path to the destination at each hop. A simple example is shown in Fig.5.1.

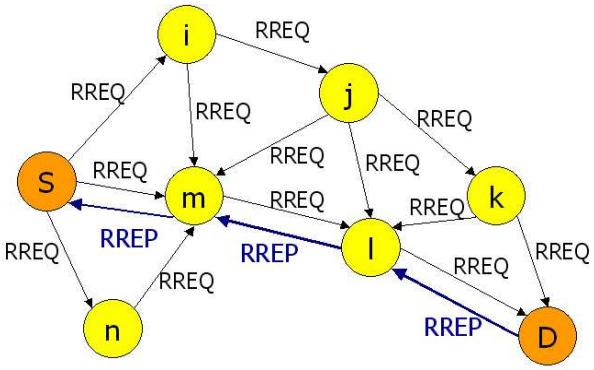



Figure 5.1.: AODV Route Discovery

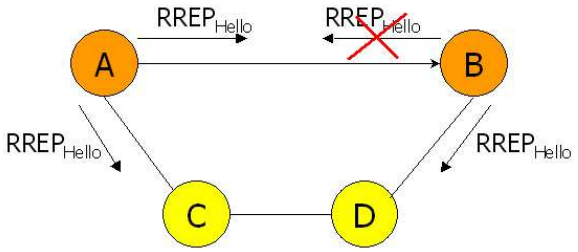



Figure 5.2.: AODV unidirectional links

To guarantee the use of only bidirectional links in route discovery procedures and the consequent definition of a path towards a destination, AODV defines some techniques [MD02] and a selected one, called *BlackListing*, has been adopted in the last specification of the protocol. Specifically, when a node detects a RREP transmission failure (e.g. in case it does not receive a RREP or it receives one with a different reverse path in respect to the RREQ path), it inserts the next hop of the failed RREP into a "blacklist" set. Thus, the blacklist set of a node indicates the set of nodes from which it has unidirectional links. Then, when a node receives a RREQ from one of the nodes of its blacklist set, it discards it to avoid the creation of a reverse path with unidirectional links.

In addition, to have at least a partial view of the network topology even in absence of application traffic, AODV allows nodes to learn about their neighbors through Hello messages. Periodically each node checks if it has sent a broadcast packet (e.g. a RREQ) in the last time interval. If it has not, it broadcasts a RREP message with TTL equals to one. Thus, even if there is no request to establish a specific route, nodes become aware of their neighbors.

Even though AODV prevents unidirectional links in a route discovery through BlackListing, it does not consider the possibility of having a unidirectional link caused by the failure of a Hello message. In Fig.5.2 a simple example is shown. In particular, considering the unidirectional link between nodes A and B. When these nodes send their broadcast Hello messages, node B correctly receives the Hello message of node A, but node A cannot receive Hello message of node B. Thus, node B, having received the RREP from A, it inserts a valid entry in its routing table as the 1-hop route to A, not realizing that it is a unidirectional link. Therefore, when node B has to forward an application message to A, it directly sends it through the 1-hop path completely failing. At this point it has to

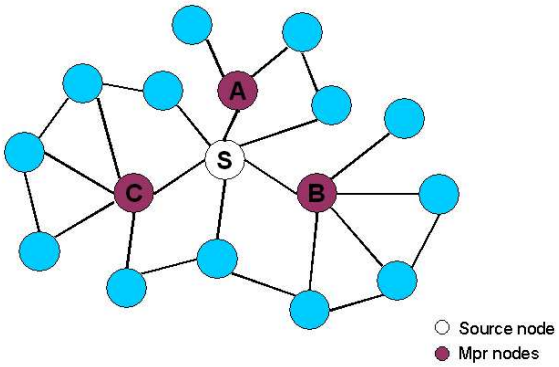



Figure 5.3.: MPRs selection

execute a route discovery procedure to establish an alternative valid route to the destination. In addition, since Hello messages are periodically sent by every node, valid routes obtained by the reactive procedure are substituted by these unstable routes causing several connection failures. For this reason, we affirm that using AODV also unidirectional links are considered as valid routes in the forwarding protocol.

## 5.2. OLSR: Optimized Link-state routing protocol

OLSR [rfc] is an optimization of the classical link-state algorithm developed for mobile ad hoc networks. It exchanges topology information with other nodes of the network regularly exploiting the key concept of *Multipoint Relays* (MPRs). MPRs are selected nodes which forward broadcast messages during the flooding process. No other nodes are responsible for forwarding control traffic on the network.

A node selects its set of MPRs among its 1-hop neighbors with bidirectional links (validated by Hello messages exchange), such that this set covers all its 2-hop neighbors. Consider a node S, the MPR set of S, denoted as MPR(S), is a subset of 1-hop neighbors of S that satisfies the following condition: every 2-hop neighbor of S has a symmetric link with at least one member of MPR(S). On the other hand, each node has to maintain information about nodes that selected it as MPR, since it must forward messages related only to those nodes. This list is called *MPR Selector Set*. A simple example is shown in Fig.5.3 where A, B, and C represent the MPR set of S.

Compared to a classical broadcasting mechanism, where all nodes foward a message after having received its first copy, this technique drastically reduces the message overhead, since link state information is generated only by MPRs. In addition, the protocol can be further optimized if each MPR chooses to report only links between itself and its MPR selectors. Thus, also partial link state information is distributed on the network.

All information about the neighborhood of each node and the selection of its MPRs result from Hello messages exchange. In fact, a Hello message contains the list of 1-hop neighbors from which the sender has received a Hello specifying the link type (unidirectional or bidirectional in case the received message already contains the sender in the neighbor list). Through this exchange each

node knows all its neighbors in the 2-hop range. At this point it can select its MPRs and they will be announced in the subsequent Hello messages that are generally sent every 2 seconds.

In addition, in order to build the rest of the network topology, each MPR periodically broadcasts specific control messages called *Topology Control (TC)* messages. They contain the list of neighbors who have selected the sender node as the MPR. The interval between the transmission of two TC messages depends on whether the MPR Selector set is changed or not, since the last TC message has been transmitted. If no changes occurs, the next TC message is sent after its normal interval, instead it may be sent after a specified minimum interval, starting from the transmission of the last TC. Based on information contained in TC messages, the network routing table is computed.

# 6. P2P platforms in wired networks

Since the distributed nature of ad hoc networks fits well the p2p model of computation and they also share a lot of features, we start studying p2p system developed for wired networks, to understand if their main characteristics can also be exploited in ad hoc networks.

Specifically, a key challenge to the usability of a data-sharing p2p system is implementing efficient techniques for search and retrieval of shared data. The best search techniques for a system depend on the need of the distributed application. For example, applications like group multicasting, web caches or archival systems focus on availability, and they 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: *unstructured* and *structured* platforms.

*Unstructured* platforms are such that 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. They are not required to maintain relevant information about shared content owned by other entities (e.g., a distributed search index). This approach does not match availability guarantees, but it introduces *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], KaZaa [kaz] and the recently appeared BitTorrent [Coh03], 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.

*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. The key 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 is the basis of *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 by 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. All these platforms achieve optimal lookup performance.

In the following section main characteristics of structured p2p systems are described, and three particular solutions are proposed.

## 6.1. Structured overlay networks

Peer-to-peer systems based on the creation of an overlay network essentially represent ways of building distributed hash table over an Internet-like network technology, in order to distribute information among nodes taking part in the same service. The fundamental problem of peer-to-peer systems can be translated in a simple question: once I have some data *D* and some attributes *A* describing it, how do I map them consistently over a zone (physical area, logical area or set of nodes) of the distributed community in a scalable and resilient way? This problem finds several solutions in peer-to-peer applications following different policies. For example, let's say *D* is the information describing a service functionality and parameters needed to bind it, while *A* contains some service keywords. On the one hand the service provider has to have a way to publish *D*, while on the other a client peer needs a way to discover where to find a service matching the keywords specified in *A*. Policies based on flooding service descriptions or service queries on all nodes are not scalable, while the usage of brokers (third-party nodes holding service descriptions and answering service queries) is not resilient. Further, in case of ad hoc networks, nodes do not know a priori the other nodes in the network and the link connectivity is highly dynamic, thus it is not possible to elect specific nodes to assume the role of centralized servers, since they would represent possible points of failure or attack. In addition, techniques based on flooding service descriptions or service queries are not scalable with the number of nodes and they also generate a big overhead on the network.

In literature there are several p2p systems based on a structured overlay network, mainly exploiting a Distributed Hash Table (DHT): Pastry [RD01], CAN [RFH+01], Chord [SMK+01] are some of them. All these systems are characterized by some common features:

- ✧ They define a logical address space where nodes and data are mapped through the use of a distributed hash function. The structure and dimensions of this logical space is different depending on the specific system: CAN uses a *d*-dimensional Cartesian coordinate space on a *d*-torus; Pastry and Chord use a circular address space (called *ring*).
- ✧ All data to be stored in the overlay has to be represented as a pair (key,value), where the hash function is applied to the value of the key and it is deterministically mapped on a node of the overlay selected through different metrics. This particular policy is called *subject- based*

*routing* since data is characterized by a unique identifier (key) representing the subject used to choose the best destination where to be stored and from where it can be recovered.

- ✧ All nodes taking part in the overlay locally store the information related to a limited set of other participants. For this reason, it is possible that the best destination for a specific data is not known by the sender of the message, and it could be necessary to forward the message through intermediate nodes establishing a multi-hop routing at the middleware level.
- ✧ In order to join the overlay and initialize its internal data structures, each node has to know at least one node already connected to the overlay. For this reason these systems cannot be considered completely self-organizing since they require a priori the knowledge of a bootstrap node.
- ✧ Each overlay is associated to a single service or to a set of services provided by all participant nodes.

Naturally all these systems are also characterized by several differences, with particular reference to the policy defined to distribute and recover data, the structure of the information stored on the local node to manage the overlay, and the complexity introduced by their lookup procedures. In following sections a brief description of these single systems is given with particular attention to the Pastry model that has been chosen as a reference for our solution on ad hoc networks.

### 6.1.1. Content-Addressable Network

A Content-Addressable Network (CAN)[RFH+01] is a virtual d-dimensional Cartesian coordinate space on a d-torus. At any point in time, the entire coordinate space is dynamically partitioned among all the nodes in the system such as every node *owns* its individual, distinct zone within the overall space. This virtual coordinate space is used to store (key,value) pairs. specifically a key K is deterministically mapped onto a point P in the coordinate space using a uniform hash function. The corresponding (K,V) pair (where V is the associated value) is then stored at the node that owns the zone within which the point P lies. To retrieve an entry corresponding to key K, any node can apply the same deterministic hash function to map K onto point P and then retrieve the corresponding value from the point P. If the point P is not owned by the requesting node or its immediate neighbours, the request must be routed through the CAN infrastructure until it reaches the node in whose zone P lies. Efficient routing is therefore a critical aspect of CAN.

A node learns and maintains the IP addresses of those nodes that hold coordinate zones adjoining its own zone. This set of immediate neighbours in the coordinate space serves as a coordinate routing table that enables routing between arbitrary points in this space. Intuitively, routing in a CAN works by following the straight line path through the Cartesian space from source to destination coordinates. Each CAN node maintains a coordinate routing table holding the IP addresses and the virtual coordinate zones of its immediate neighbours in the virtual space. The immediate neighbours are nodes located in adjacent positions in the virtual space. In a d-dimensional space, they are those nodes with d-1 coordinate ranges in common with the reference node, and border on only one dimension. Each CAN message will be routed according to its destination coordinates,

calculated by the hash function and included within the message. At each routing step, the node forwards the message to the immediate neighbor with coordinates closer to the one of the message destination. In this mechanism, resilience is guaranteed by more than one possible path reaching the destination, but the complete path can result much longer than the physical path to connect the same nodes. This is mainly due to the lack of correspondence between the logical address space and the physical network topology.

This algorithm provides that each node maintains *O(d)* state and the lookup cost is $O(dn^{1/d})$ for *d* dimensions and *n* nodes, that represents the number of hops at the application level to distribute data among nodes.

In order to create and maintain the overlay, CAN is characterized also by a set of procedures to manage its internal data structures consequently to selected events such as the boostrap of the local node, the joining of another node, data distribution through the routing policy, and the departure of some other nodes. All these procedures are extensively explained in [RFH+01].

### 6.1.2. Chord

Chord [SMK+01] provides a distributed lookup mechanism built on top of a consistent hash function. The consistent hash function assigns an *m*-bit identifier to each node and key using a SHA-1 algorithm as a base hash function. A node's identifier is chosen by hashing the node's IP address, while a key identifier is produced by hashing the key value. The identifier length *m* must be large enough to make the probability of two nodes or keys hashing to the same identifier negligible (generally it is set to 128). Identifiers are ordered on an identifier circle modulo $2^m$. Key k is assigned to the first node whose identifier is equal to or follows the identifier of k in the logical space. This node is called the successor node of key k, denoted by *successor(k)*. If identifiers are represented as a circle of numbers from $0$ to $2^m - 1$, then *successor(k)* is the first node clockwise from k. The circle is also called a Chord ring. Consistent hashing is designed to let nodes enter and leave the network with minimal disruption. To maintain the consistent hashing mapping when a node *n* joins the network, certain keys previously assigned to *n*'s successor now become assigned to *n*. When node *n* leaves the network, all of its assigned keys are reassigned to *n*'s successor. No other changes in assignment of keys to nodes need occur. Over the described distribution scheme, a simple but inefficient lookup procedure consists in having each node simply forwarding lookup queries to the immediate successor in the ring, until they end up in the node maintaining the correspondent (key, value) pair. The retrieval of the correct location using this approach, may however require the linear traversal of the entire ring (linear cost). By maintaining more per node routing information (not only the link to the successor in the Chord ring), the system implements a more efficient (logaritmic cost) lookup procedure. Let *m* be the number of bits in the key/node identifiers. Each node *n* maintains a routing table with up to *m* entries, called the *finger table*. The $i^{th}$ entry in the table at node *n* contains the identity of the first node *s* that succeeds *n* by at least $2^i - 1$ on the identifier circle, i.e., *s* = *successor(n*+$2^{i-1}$*)*, where $1 <= i <= m$ (and all arithmetic is modulo $2^m$). We call node *s* the $i^{th}$ finger of node *n*. A finger table entry includes both the Chord identifier and the IP address (and port number) of the relevant node. Note that the first finger of *n* is its immediate successor on the circle. The lookup operation, extended to use

finger tables, works as follows: at each node *n*, if the key *k* in the lookup query falls between *n* and its successor, the procedure ends, returning the successor identifier. Otherwise, *n* searches its finger table for the node whose ID most immediately precedes *k*, that will know more about the identifier circle in the region of *k* than *n* does. By repeating this process , *n* learns about nodes with IDs closer and closer to *k*.

This scheme has two important characteristics. First, each node stores information about only a small number of other nodes, and knows more about nodes closely following it on the identifier circle than about nodes farther away. Second, a node's finger table generally does not contain enough information to directly determine the successor of an arbitrary key k. Specifically, in a system with *N* nodes, each node maintains information only about *O(*$logN$*)* other nodes and resolves all lookup procedures via *O(*$logN$*)* messages to other nodes. In addition Chord maintains information about joining and leaving events through no more than *O(*$log^2N$*)* messages for each event.

### 6.1.3. Pastry

Pastry [RD01] shares with Chord the ring abstraction, but implements a radically different routing strategy. The overlay network defined by Pastry is represented by a circular address space of 128 bits. To each node in the Pastry network is assigned a 128-bit node identifier (nodeId) that is used to indicate a node's position in the logical address space, which ranges from 0 to $2^{128} - 1$. The nodeId is assigned randomly when a node joins the system simply hashing one of its physical identifier (IP address, hostname, public key or others). Using a distributed hash function it is assumed that nodeIds are generated such that the resulting set of nodeIds is uniformly distributed in the 128-bit space. As a result of this random assignment of nodeIds, with high probability, nodes with adjacent nodeIds are physically scattered. The same hash function is also used to map data on the overlay. It is applied on the key value specified for each data to be sent on the overlay, and it represents the subject of its routing at the middleware level.

In particular, the routing policy defined by Pastry is based on a numerical proximity metric between the hash value of the message key and the nodeIds of all nodes of the overlay. From the algorithmic standpoint, keys and nodeIds are represented as a sequence of digits with base $2^b$, where the parameter *b* is defined a priori. In this way, Pastry routes messages to the node whose nodeId is numerically closest to the given key. To this aim, in each routing step, a node normally forwards the message to a node whose nodeId shares with the key a prefix that is at least one digit (*b* bits) longer than the prefix that the key shares with the local nodeId. If no such node is known, the message is forwarded to a node whose nodeId shares a prefix with the key as long as the current node, but the following digit is numerically closer to the key than the local nodeId.

To support this routing procedure, each node maintains the information related to other nodes of the overlay in the following data structures:

- *Routing table*. A node's routing table 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

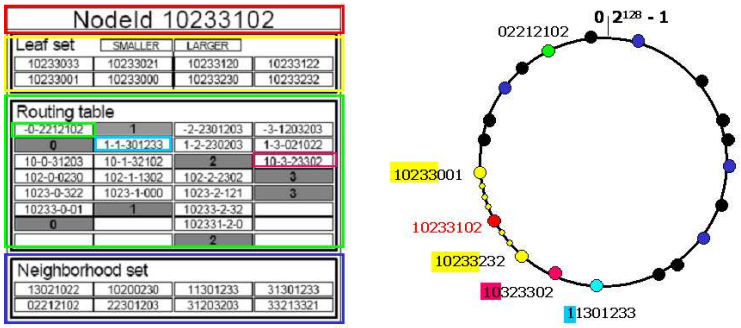



Figure 6.1.: Pastry Overlay Data Structures

index). If there are no nodeIds with this characteristic, the entry is left empty. To each entry is also associated the IP address of the selected node with the appropriate prefix to directly connect to it. In practice, a destination node is chosen based on the proximity to the value of the key and to the local node. This choice provides good locality properties, but only in the logical space. In fact nodes that are logically neighbors, are probably physically distant. It has been demonstrated in [RD01] that the maximum number of routing hops between a source and a destination is equal to $log_{2^b} N$ , in this way the systems scales with the number of participating nodes, and the choice of the parameter $b$ involves a trade-off between the size of this data structure and the maximum number of hops.

✧ *Neighborhood Set*. The neighborhood set (*M*) contains the nodeIds and IP addresses of the nodes that are physically closest to the local node. The neighborhood set is not normally used in routing messages; it is useful in maintaining physical locality properties of nodes.

✧ *Leaf set*. The leaf set *L* is the set of nodes with identifiers numerically closest to the current node. A half of the identifiers in *L* are larger than the one of the present node, while the other half are smaller (it is an interval centered around the present identifier). The leaf set represents the set of nodes that are logically closest to the local node, and for this reason it is used as the first step in the routing policy.

In routing a given message, the node first checks whether the key falls within the range of nodeIds covered by its Leaf Set. In this case, the message is forwarded directly to the destination node, namely the node in the leaf set whose nodeId is logically closest to the message key. If the key 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 with the key by at least one more digit that the present identifier.

Sometimes, it is possible that the appropriate entry in the routing table is empty or the associated node is not reachable. In this case the message is forwarded to a node whose id shares a prefix with the key at least as long as that shared by the local node, and it is numerically closer to the key than the local node's id.

Fig.6.1 shows an example of Pastry data structures for a node with logical identifier 10233102 and the logical positions of known identifiers on the circular address space. Considering the same local node, in Fig.6.2 an example of the subject-based routing for a specific key is given. Specifically,

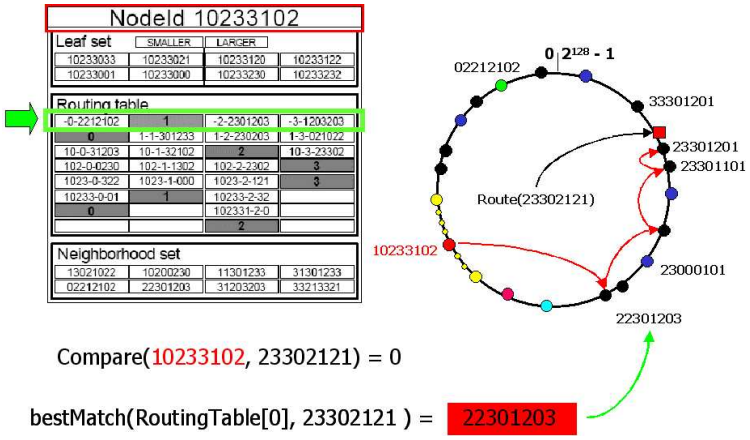



Figure 6.2.: Example of Pastry routing

as previously said, if the local node wants to send a message, it has to compare its nodeId with the key identifier. In this case the key value (22301203) has no common digit with the local id, so Pastry checks the contents of the row 0 of the routing table and selects the nodeId closest to the key value. At this point the message is sent and the receiver will forward the message in the same way until it arrives at the node closest to the key. Due to the limited dimensions of the Pastry routing table, each node maintains only a limited part of nodes of the overlay, providing a multi-hop routing to reach the best destination of each message. However, in wired networks where thousands of nodes can be involved in the same service, maintaining a complete view of the overlay makes the system impractical to scale to large number of nodes, and with this policy it scales better than the other systems. For this reason we chose Pastry as a reference model also for ad hoc networks evaluating its performance in such environments.

To better understand Pastry behavior also on ad hoc networks, it is necessary to explain other main procedures used to establish and maintain the overlay network. These procedures consist of join and disjoin operations.

First of all, when a new node enters the system, it needs to initialize its internal data structures, and then it has to inform other nodes of its presence. As the other p2p systems, Pastry assumes that each node must know at least another node involved in the system, if possible one of its physical neighbors. Such a node can be located automatically, for instance, using "expanding ring" IP multicast, or it can be obtained by the system administrator through outside channels. Let us assume the new node identifier is $X$ and the known node is $A$. Node $X$ asks $A$ to route a special "*join*" message with the key equal to $X$ on the overlay. 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 receiving the "join" request, nodes $A$, $Z$, and all nodes encountered on the path connecting them, send the content of their tables to $X$. At this point, the new node $X$ processes this information and then initializes its own tables 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 $Z$ (new logical neighbor of $X$).

- ✧ The $i^{th}$ row of the routing table is initialized with the $i^{th}$ row of the routing table of the $i^{th}$ node ($B_i$) encountered in the routing path from *A* to *Z* (it shares a prefix of length i with *X*).

Finally, *X* informs any nodes that need to be aware of 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 affected nodes is updated.

Another important feature of Pastry is the management of departure nodes. In [RD01] a node is considered failed when 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 generally delay message routing, since 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.

During this work the Pastry model has been chosen as a reference also for ad hoc networks because it scales with the number of nodes better than the other systems and a free implementation [fre] is available, together with a rich set of applications developed on top of it. The main purpose of this work is to present a performance evaluation of this system on a real ad hoc networks of small/medium scale, in order to understand advantages and limits on MANET environments and to propose an innovative solution optimized for mobile ad hoc networks. The first phase of our experimental evaluation is described in the next chapter.

# 7. Small-scale Ad Hoc network test-bed: Routing and Middleware performance

Although MANET research has been ongoing for some time, there are relatively few experiences with real ad hoc networks. Instead, a large portion of protocol development is done in simulation settings only. In fact one of the main approaches in system performance evaluation is based on a representation of the system behavior via a model [Lav83] [KM88], while measurement techniques can be applied only when a real systems, or a prototype of it, is available. For this reason, most results on the behavior of MANET protocols have been obtained by defining a system model, and solving the model using analytical and/or simulative techniques. Analytical models are often not detailed enough for the ad hoc networks evaluation. On the other hand, simulation modeling is a more standardized, mature, and flexible tool for modeling various protocol and network scenarios. By running the simulation model, it allows researchers to collect and analyze data that fully characterizes protocol performance in most cases. However, as pointed out in [GLNT], simulations have not been conclusive for selecting MANET protocols among the several available solutions. Furthermore, simulation models often introduce simplifications and assumptions that mask (in simulation experiments) important characteristics of the real protocols behavior [ABCG04] [LNT02], see for example, the so-called communication gray zones problem [LNT02].

Specifically, this problem was revealed by a group of researchers at the Uppsala University, while measuring the performance of their own implementation of the AODV routing protocol in an IEEE 802.11b ad hoc network. Observing an unexpected large amount of packet losses, mainly during route changes, it was found that the increase in packet loss occurred in some specific geographic areas called "*communication gray zones*"[1]. It is important to point out that the communication-gray-zone problem was not revealed by commonly used simulation tools (e.g., NS-2, Glomosim) as in their 802.11 models both unicast and broadcast transmissions are performed at 2 Mbps, and hence have the same transmission range.

To avoid these modeling approximations, simulations have to be complemented by experiments on real prototypes. In addition, the availability of prototypes will also make possible to start creating communities of MANET users that, by experimenting with this technology, will provide feedback on its usefulness and stimulate the development of applications tailored for the ad hoc environment. At the end, user interaction can drive to identify possible ad-hoc-network killer applications making MANETs a success beyond the academic world.

[1] This phenomenon is due to the different transmission ranges between unicast (data) and broadcast frames (i.e., routing information) in 802.11 networks. A station inside a gray zone is considered using the routing information reachable by a neighboring station, while actual data communication between the stations is not possible.

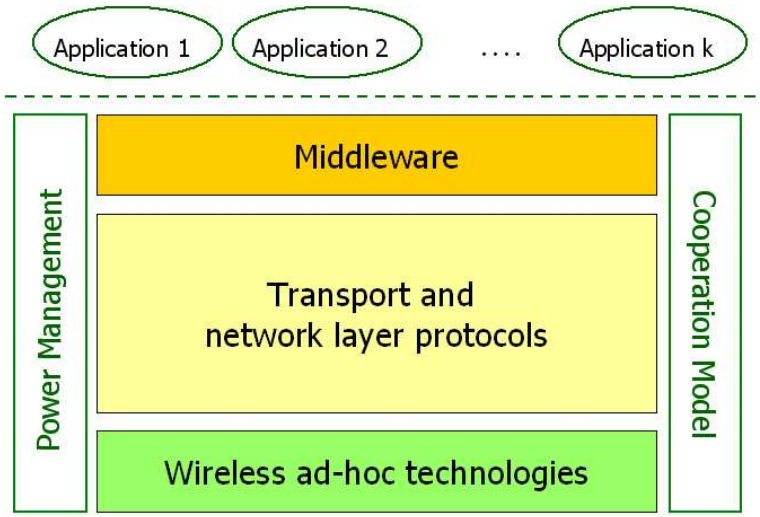



Figure 7.1.: Reference Architecture

Currently, only a few measurements studies on real ad hoc testbeds can be found in literature, see e.g., [BMJ00] [oCSaUS]. The Uppsala University APE testbed [oCSaUS] is one of the largest, having run tests with more than thirty nodes. The results from this testbed are very important [GLNT] and point out that more research in this direction is required to consolidate the ad hoc networking research field.

In the framework of this research activity, in order to investigate potentialities and limits of the ad hoc networking paradigm, we set up a MANET prototype on which we performed several sets of measurement. In a previous work [ABCG04] an experimental analysis of single layers of an ad hoc network (mainly studying 802.11 performance in ad hoc networks) has been done. In this chapter measurement results obtained by implementing a full MANET protocol stack are presented, mainly focusing on classical routing and middleware solutions. Specifically, a performance evaluation of a fully functioning prototype implementing a p2p middleware (Pastry) on top of a multihop ad hoc network is carried out.

## 7.1. Test-bed reference architecture

The novel aspect of this study is to investigate a full ad hoc network architecture, from the Wi-Fi ad hoc network up to the peer-to-peer middleware platform (Pastry) on top of which a simple testing application (distributed messaging) is running. For this experimentation we used the reference architecture shown in Fig.7.1.

More precisely, we integrated solutions for the main building blocks that makes our testbed representing a realistic MANET:

- ✧ Wireless Technologies
- ✧ Networking

✧ Middleware and Applications

In this work important issues like power management, and security and cooperation have not been addressed. In this way, we implemented a first simplified MANET that enabled us to perform realistic experiments still maintaining the problem complexity at an acceptable level. We believe that using an incremental approach is the best way to contribute to the consolidation of the MANET world. As pointed out in [GLNT], having realistic prototypes increases the attractiveness of the technology and provides a platform for applications development. These are key elements for the success of this technology.

Specifically, at the networking layer we adopt one proactive and one reactive routing protocol. They enable us to compare these two approaches in a realistic scenario. In literature, it is a common understanding to consider that on-demand reactive protocols are more efficient than proactive ones. As previously explained in Chapter 5), on-demand protocols minimize control overhead and power consumption since routes are only established when required. On the contrary, proactive protocols require periodic route updates to keep information current and consistent. In addition, maintaining multiple routes, which might never be needed, causes unnecessary routing overheads. On the other hand, proactive routing protocols provide better quality of service than on-demand protocols. As routing information is constantly updated in the proactive protocols, routes to every destination are always available and up-to-date, and hence end-to-end delay can be minimized. Instead, for on-demand protocols, the source node has to wait for the route to be discovered before communication can happen. This latency in the route discovery might be intolerable for real-time communications. One of the aim of this testbed is to compare and contrast AODV and OLSR as two implementations of these approaches, analysing their efficiency and QoS.

However, solving MANET routing and forwarding issues is only a first step towards deployable MANETs. Integrating applications on top of an ad hoc network is fundamental to stimulate the users interest in this technology, and hence to better understand its possible usage scenarios. In fact, integrating p2p systems on top of ad hoc networks makes the variety of p2p applications and services available to MANET users as well, and hence it would be an advantage for this emerging technology. However, it is not clear how these overlays should be ported, and how they will perform on ad hoc networks. To this aim, in this first part of our experimental analysis, we focused on investigating the performance of Pastry on top of our ad hoc network. Results from this investigation provide the basis for defining efficient ways to port Pastry on ad hoc environments.

Pastry differs from the other p2p routing substrates such as CAN and Chord due to the structure of information on nodes and the way messages are passed between peers in order to distribute data and workload. As previously said, we selected Pastry for our testbed because it scales with the network size, and a free implementation (FreePastry [fre]) with a rich set of services is available.

Referring to main principles of Pastry, the limited dimensions of its internal data structures, aimed at maintaing information of other nodes of the system, are such that the subject-based routing may require a multi-hop routing at the middleware layer.

As a preliminary result, we noted that in ad hoc networks consisting of a small number of nodes, all peers generally know all the others, and thus it is not necessary to execute a multihop routing at the middleware layer. However, at the same time, operations needed to create and maintain the

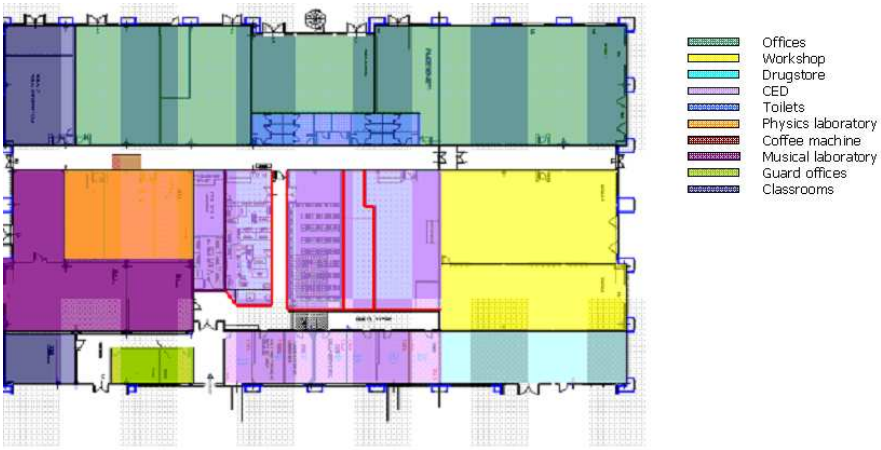



Figure 7.2.: Experiments Area

overlay represent a high cost for ad hoc networks. In fact, when a node enters the Pastry network, and it is not the first one, it has to contact one of its physical neighbors already present in the overlay in order to collect routing table information. The contacted node sends a message on the ring using the new node ID as the key of the message that thus will reach the node logically closest to the specified ID. At this point, the new node can initialize its routing tables receiving the Leafset from the logical closest node, the Neighborhood set from the specified physical neighbor, and the Routing table as a join of the routing tables of nodes that forwarded the original message. Each of these operations requires several remote connections that, in FreePastry (during this phase version 1.3 had been used), are implemented by TCP connections (thus introducing a high cost).

In addition, to maintain the overlay structure, each node has to execute a polling procedure needed to discover neighbors status, so that a node is considered disconnected from the overlay if it does not answer to a polling message before a timeout expiration. If this occurs, the sender of the polling message has to update its routing tables contacting other remote nodes, which thus implies opening other remote connections. FreePastry implements this polling procedure using UDP connections, and the exchange of routing table data using TCP connections. In this way it is possible that a node has to maintain several TCP connections to different nodes only to manage the ring, introducing a high overhead in ad hoc networks, in particular when links are unstable and there are many topology updates.

## 7.2. Experiments Environment

All the tests were conducted at the ground floor in the CNR campus in Pisa (Fig.7.2). At this level there is the computing center (CED) together with some companies' offices and measurement laboratories with several kinds of instrumentations. The structural characteristics of the building, and particularly of this floor, strictly determine the transmission capabilities for the nodes of a wireless network situated within. Rooms (offices, laboratories, etc.) are generally delimited by masonry

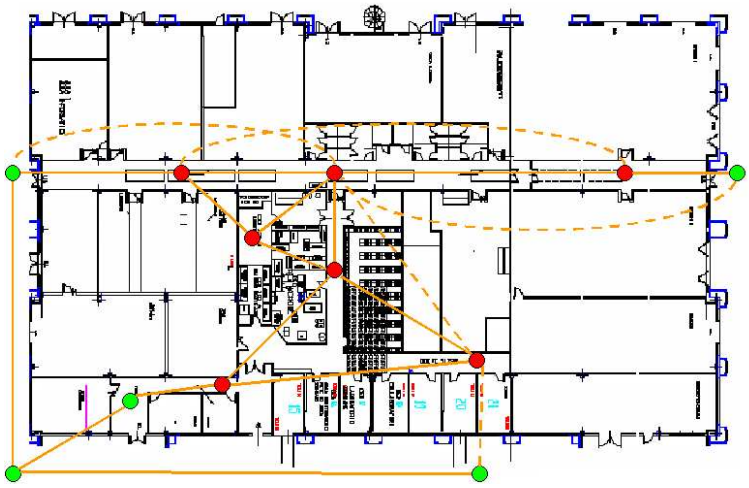


Figure 7.3.: Experiments Scenario

padding walls situated between reinforced concrete pillars. In the CED area, instead, 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. These generally cause minor impediments to waves compared to masonry walls or reinforced concrete pillars. Wireless links are also influenced by the nearby presence 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 varying in a continuous and unpredictable manner. As a result, the whole place can be considered quite a realistic environment for testing an ad hoc network. Fig.7.3 presents the detailed map of the place together with the static transmission coverage characteristics of the area. Nodes are situated where devices were placed during the experiments and straight lines are used to point out the presence of wireless links (two nodes see each other at one hop distance if a single straight line joins them). Dashed lines are used instead to point out weaker wireless links wherever a couple of nodes see just sometimes each other and their communications are affected by a considerable packet loss. The devices used for the experiments were both laptops and PDAs (Compaq iPAQ 3950) equipped with different wireless cards. This caused some links appear/disappear in different experiments depending on the power of wireless cards used by the nodes at each side of the link.

Specifically, we used three different types of wireless cards: PCMCIA DLink DCF-660W for PDAs, and D-Link DWL 650 (15 dBm) and 3COM 3CRWE62092A (14 dBm) for laptops.

In the experiments we used a limited number of nodes ranging from 5 up to 12 nodes. These numbers may appear not meaningful respect to simulations scenarios using hundreds of mobile nodes. However, recent results pointed out the existence, with the current technology, of an ad hoc horizon of two-three hops and 10 to 20 nodes. Beyond these limits the benefit from wireless multi-hop ad hoc networking virtually vanishes [GLNT]. Indeed all the experiments presented hereafter fall inside this ad hoc horizon. This may represent a scenario consisting of few people forming an ad hoc network to share documents. The focus of our study is therefore to contrast

and compare ad hoc networking solutions within current technology limits. Currently, networks of hundreds of mobile nodes connected to several hops seem to be an unrealistic goal.

## 7.3. Experiments warm-up: a qualitative analysis

An extensive first experimentation phase was carried out during June 2004 in CNR campus in Pisa (Italian National Research Council). The aim of the experimentation was to test if the software at each layer of the protocol stack behaves correctly.

Specifically, the test concerned different layers:

- ✧ Routing: testing the selected implementations of proactive OLSR and reactive AODV routing protocols to check their state of implementation, validate their functionalities and conduct a comparative analysis on them all. The considered routing protocol implementations were the Unik-OLSR [Ton] by the University of Oslo (Norway), and the UU-AODV [aod] by the Uppsala University (Sweden).
- ✧ Middleware: testing the selected FreePastry implementation on top of proactive and reactive routing protocols and evaluating the heaviness of the resulting solutions. The middleware platform used in these experiments was FreePastry version 1.3 which is the open-source implementation of the original Pastry model developed by the RICE University [fre]. To this end we installed FreePastry on a set of laptops which had been previously equipped with the j2sdk-1.4.02 Java Virtual Machine. FreePastry implements the p2p common API [DZD+03] that has been proposed to guarantee a common interface for distributed applications developed on top of different middleware platforms based on structured overlay networks.
- ✧ Application: evaluating the impact of routing protocols on the Quality of Service (QoS) experienced by the application (e.g., the transmission delay). We used both legacy Internet applications (e.g., ping and ftp) working directly over the TCP/IP protocol stack, and a Distributed Messaging application running on top of FreePastry.

Hereafter, we report main results of this experimental phase, a detailed presentation about all the experiments can be found in [d8]. The experiments were organized in two steps. First we tested OLSR and AODV in isolation to verify that they behave correctly. Then we integrated FreePastry to investigate the behavior of our full MANET.

### 7.3.1. Unik-OLSR Testing

Due to the proactive nature of the protocol the test was based on observing the status of the network routing tables while nodes were added/removed to/from the ad hoc network. This testing was further divided in two steps. In the first step we used a 5-node network. In this case the kernel routing tables were small and they could be read in real-time, hence it was possible to follow configuration changes while in progress. Upon the beginning of the experiments, node

insertions and removals were provided to check that configuration updates effectively took place. Moreover, by changing the time lag duration between successive node insertions and/or deletions, it was also possible, to some extent, to measure the configuration-update delays after the appearance/disappearance events. In all the experiments the protocol showed a correct behaviour. The routing tables quickly updated upon node insertion and removal. Then we considered a 12-node network. The increased number of nodes led to the increase of the number of protocol packets exchanged. This allowed the validation of Unik-OLSR behaviour in a more congested context. Also in this set of experiments, all the routing-forwarding operations were correctly performed. After this analysis, we investigated the ability of an OLSR-based network to transfer data between nodes at a distance of few hops. To this end, the Unik-OLSR protocol was started on all the nodes at the same time, then after a little delay to let the routes stabilize, an FTP transfer was started between two nodes. The destination was 3-hop distant from the source. The aim was to transfer a 34 megabytes (MB) file. Several problems were experienced in this case. Intermediate nodes along the sender-destination path stopped working correctly after a while. This was due to the wireless card not properly working. It seemed that the excessive traffic they have to manage caused problems to their cards' drivers. In these experiments the routing protocol still behaved correctly by selecting alternative routes to avoid the out-of-service nodes. The file continued until a network partition occurred. At this time the destination host had received only the first 15MB of the file. The throughput during the transmission had just been about 180Kbps. We repeated the experiment and similar problems were observed. Specifically, we observed that the file-transfer started correctly but while the transfer proceeded the throughput of the connection reduced. This type of behaviour can be explained with problems produced by the interaction between TCP and the 802.11 MAC extensively investigated in literature, see Chapter 3 in [BCGS04] for a summary [2].

### 7.3.2. UU-AODV Testing

As this protocol is reactive, some application-level traffic was introduced in order to observe the route creation process. To this aim we used the simple `ping` utility. Specifically, each node sent periodically a set of `ping` operations to different destinations and this forced the routing protocol to set up a route towards each destination. In this case, each sender was always able to discover the correct path, however the route discovery was very time-expensive. In the next section, some estimates of these delays will be provided. After this, we tested the UU-AODV ability to support user-data transfer. We used again a file transfer application. The file transfer was started from a couple of nodes at a 3-hop distance, aimed at transferring a 5MB file. The transfer was definitely too slow and after 16 minutes only 140KB had reached the destination; the experiment was then interrupted. Even in this case, the problem seemed related to interaction of MAC and TCP mechanisms. Packet losses caused a TCP congestion-reaction that slowed down the connection throughput. In addition, in this case, the reactive nature of the routing protocol made the things worse.

[2] In these experiments we used default values for TCP parameters, e.g., advertised window. It is left for further studies to investigate the impact of TCP parameters on the ad hoc network performance.

### 7.3.3. FreePastry Testing

The last set of experiments was carried out in order to evaluate the overhead introduced by a middleware platform based on the original Pastry model [RD01] and to validate its features on ad hoc networks. We integrated on top of FreePastry a simple application of Distributed Messaging (DM), aimed at testing main Pastry features. Nodes participating to DM set up and maintain a Pastry overlay corresponding to this Pastry service. Specifically, each instance of the messaging application defines an Identifier (ID) for the local node on which it runs. The ID is then used as the key for implementing the Pastry distributed hashing operations. When the application starts, the overlay initialization depends on the existence, or not, of a Pastry ring already providing that service. In the first case the node performs the operation required to join the overlay contacting a bootstrap node, otherwise, the node is the first one, and it creates a new overlay. If the overlay already exists, the user has to specify the IP address of a known physical neighbor already connected, so that the local node can collect information about middleware routing tables from it and enter the overlay. Once the local node has created/joined the overlay, the application allows the user to create/delete a mailbox on the nodes of the overlay. A mailbox physical location is randomly selected applying the DHT to the identifier associated to the mailbox, i.e., the identifier represents the key of the message on which the hash function is computed. Once the mailboxes are created, the user can send/receive a message to/from each of them. These operations are based on the Pastry subject-based routing mechanism. The DM application also generates 100 mailboxes with random IDs. Using this application we tested FreePastry to verify the overlay construction, the workload it generates, and the data distribution on the overlay. Furthermore, it is also possible to evaluate the overhead introduced by this platform on ad hoc networks.

These tests showed that all operations were performed correctly. However, the produced overhead was quite high, particularly, due to the large number of remote connections needed to maintain the overlay data structures. This overhead will be quantified in the next section. The same set of experiments (i.e., the same network configuration) was performed by running Unik-OLSR and UU-AODV. The network topology was kept as much similar as possible to the one used for previous experiments. In the network we had 8 nodes: 6 nodes ran FreePastry and the others 2 just worked as routers. During the experiments the nodes started running either Unik-OLSR or UU-AODV. After a delay of a few seconds to have the network topology stabilized, they ran the DM application trying to build a single Pastry overlay.

From the performed experiments we noticed that rarely application messages had to execute a multi-hop middleware path. Hence, the real overhead introduced by FreePastry on ad hoc networks is due to the periodical remote connections needed to the overlay maintenance. In this set of experiments, the main problems were observed when running UU-AODV. In this case, we experienced a high number of Pastry connections' failures (i.e., the connection used by Pastry to build and maintain the overlay). This was mainly caused by the reactive procedure to discover a route towards a specified node. When a node tries to connect to another one, FreePastry generates a TCP connection in order to recover middleware routing table information. If the local node has no routes to the destination, AODV generates a Route Request and waits for the answer. In the meanwhile, if path discovery is slow, the timeout of the Pastry connection may expire causing the raising of a Java exception. As TCP remote connections are periodically executed by FreePastry

in order to maintain the overlay structure, the previous problem causes the wrongly notification of "*dead node*", even if it is still connected to the overlay [3]. Therefore, using Pastry, the overhead reduction introduced by reactive protocols is cancelled by periodic remote connections at the middleware layer. Unik-OLSR experiments did not suffer these problems thanks to the continuous update of the kernel routing tables.

To summarize, our preliminary evaluation (mainly qualitative) of a full MANET prototype indicated that:

1. Unik-OLSR and UU-AODV correctly behave. They discover multihop paths and reconfigure the paths upon node failure/insertion.
2. FreePastry correctly operates on top of the multi-hop ad hoc networks.
3. TCP data transfers on top of a multi-hop ad hoc networks show severe performance problems. The performance seem to decrease when using a reactive routing algorithm due to the delays caused by the route discovery process.
4. FreePastry overheads are mainly caused by overlay creation and maintenance operations. These operations are based on the TCP protocol. As pointed out at point 3, TCP operations exhibit poorer performance by using AODV. In FreePastry this causes problems in the correct execution of the overlay maintenance operations, as TCP delays cause the timeout of the Pastry connection to expire causing the raising of a "Connection Refused" message at the Pastry level.

## 7.4. Quantitative analysis

A second set of experiments was performed, in the same environment used so far (see Section 7.2), to provide a quantitative estimation of the most interesting phenomena observed. Again, we first measured the performance of OLSR and AODV in isolation, then we analyzed the testbed integrating FreePastry on top of the ad hoc network.

### 7.4.1. AODV and OLSR Performance

The experiments were made in the environment shown in Fig.7.3. For ease of reading, in Fig.7.4 we report the same scenario in which we label the MANET nodes in order to identify them in the following discussion. The figure shows the 8-node scenario on which results reported below have been obtained. A line among a couple of nodes indicates that a link exists among them[4]. The aim of the experiments was to compare the two routing protocols in terms of:

[3] The AODV implementation we used caches routes into the kernel routing table for 15sec only. Therefore, a new path discovery phase is generally required when Pastry performs its overlay maintenance operations.

[4] It can be noted that some links that were marked as unstable in Fig.7.3 are now marked with solid lines to point out that they are now stable links, since we removed some obstacles to signal propagation (e.g., fire doors, that previoulsy caused weak links, in these experiments were opened).

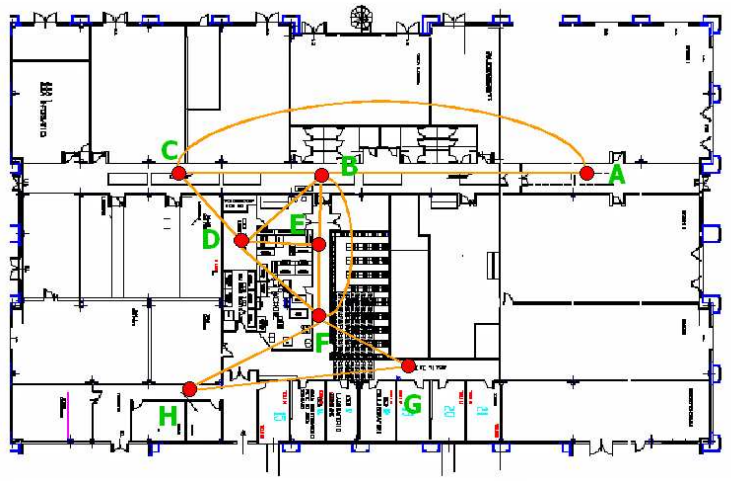



Figure 7.4.: Routing Network Topology

- ✧ overhead introduced in the network due to the routing messages;
- ✧ delay introduced for path discovery.

To have a meaningful comparison we used the ping application to generate some user-level traffic, otherwise the reactive protocol (AODV) discovers only 1-hop neighbors.

Two set of experiments were performed depending on the way the ping operation was performed.

#### 7.4.1.1. Experiment 1

In this set of experiments, the central node E performed a ping operation towards the other nodes of the network according to a randomly selected sequence: A,H,D,F,G,B,C. For each node in this sequence, the ping operation lasts for 1 minute. We performed several experiments that produced similar results using ever the same sequence. Main results are summarized in Fig.7.5 and Fig.7.6 for OLSR and AODV, respectively. In these figures the amount of traffic (expressed as number of Bytes per second) forwarded by each node of the network is reported. Specifically, this traffic includes both the routing traffic generated and forwarded by each node.

As expected, the traffic depends on the specific position of each node inside the network topology. In Fig.7.5 we can note that:

- ✧ nodes B and D measured the highest values (about 1.1 KBps);
- ✧ nodes C, E, and G measured intermediate values (about 0.8 KBps);
- ✧ nodes A and F measured a traffic load of about 0.4 KBps;
- ✧ node H is lightly loaded (its traffic, 300 KBps in average, is about 1/4 of node B and H traffic).

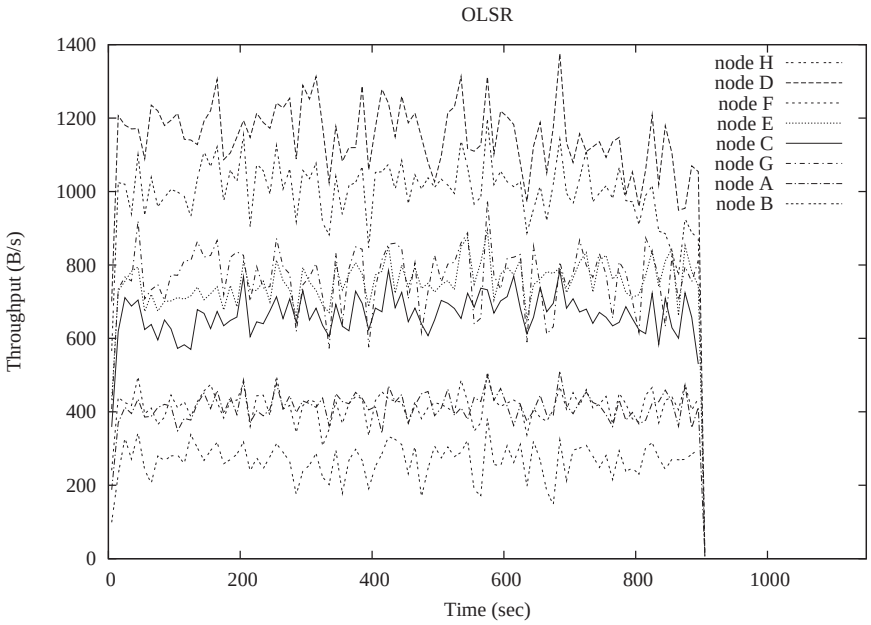



Figure 7.5.: OLSR overhead

These different traffic loads have a good correspondence with nodes positions and their degree (i.e. the number of links connecting to it). Specifically, the degree of H is equal to one since it is only connected to node G, nodes A and F have a degree of two, while the others located in the core of the network have several neighbors and they measured higher traffic loads. As expected, OLSR generated a traffic that was significantly higher than that produced by AODV. Specifically, while in case of AODV the traffic range is [100, 400] Bps, in case of OLSR it is [200, 1200] Bps. Even though their difference is very high, in both cases the impact on the utilization of the 802.11b bandwidth is almost negligible. In fact, a single node observed at most 1.2 KBps of routing traffic.

On the other hand, delay measurements pointed out possible severe problems on QoS when using AODV. Specifically, for completing a simple ping operation between a couple of nodes at 2-hop distance, we measured delays of about 19-20 sec involving the route discovery procedure. To perform the same operation, OLSR requires 1 second (or less) since routing tables are generally updated.

#### 7.4.1.2. Experiment 2

In this set of experiments, the external node H continuously pings node A for 400 seconds. Initially the shortest path is H-G-E-B-A, but after $x$ seconds from the beginning of the experiment, node B disconnects from the network and the shortest path becomes H-G-E-D-C-A. $x$ is equal to 250 seconds in case of OLSR and 180 seconds in case of AODV.

Figures 7.7 and 7.8 show the network load of the two routing protocols. In case of OLSR, the load distribution between different nodes is similar to that observed in the previous experiment (e.g., node B and node H experienced the highest and lowest load, respectively). The main difference

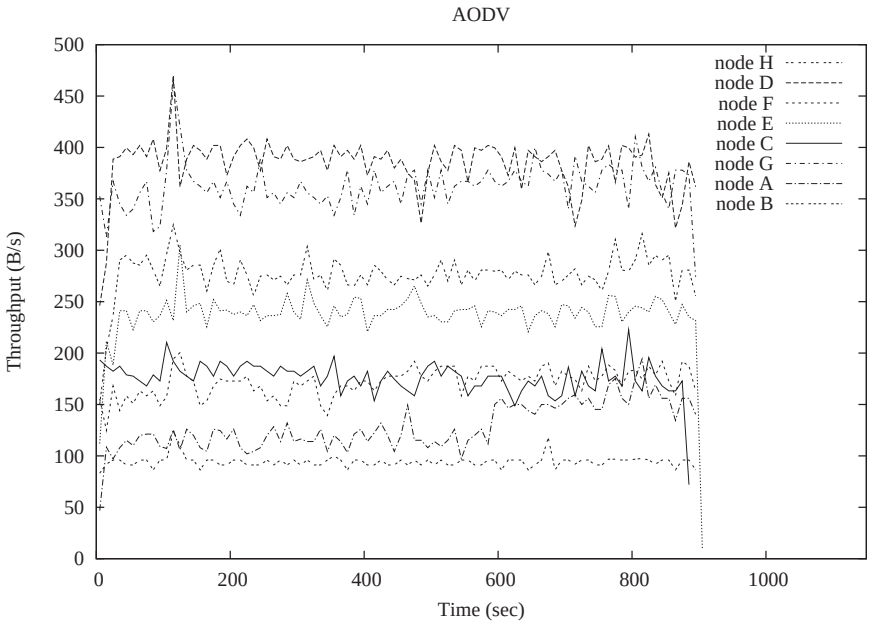



Figure 7.6.: AODV overhead

is observed after node B shut down. In fact, there is a transient phase during which the traffic load decreases due to some missing routes. Then a new steady state is achieved, and we observe a significant decrease of the traffic on nodes that were connected with node B (in particular nodes E, D, and C). Instead in case of AODV less marked differences can be observed. In fact, after the transient state, the active nodes almost observed the same load experienced before the shut down event.

Also analysing delays, we got similar results to those observed in the previous experiment:

- ✧ large delays with AODV (about 20 seconds) when the path is not already in the cache and a route discovery process is needed;
- ✧ small delays with OLSR before the shut down event; while after the routing table reconfiguration the ping operation performed a delay of about 6 sec to be completed.

### 7.4.2. Performance of FreePastry on Ad Hoc Networks

This set of experiments was made by adopting a network topology (see Fig.7.9) which slightly differs from that used for the analysis of the routing algorithms. We decided this modifications as experiments shown in Section 7.4.1 showed that central nodes in the network tends to become saturated. To avoid this, we moved one node towards the center of the network. In this way we increased the redundancy in this area. Among the eight nodes of the ad hoc network, six provided the Pastry service, while nodes B and G (the blue circles in the figure) were only involved in routing and forwarding operations.

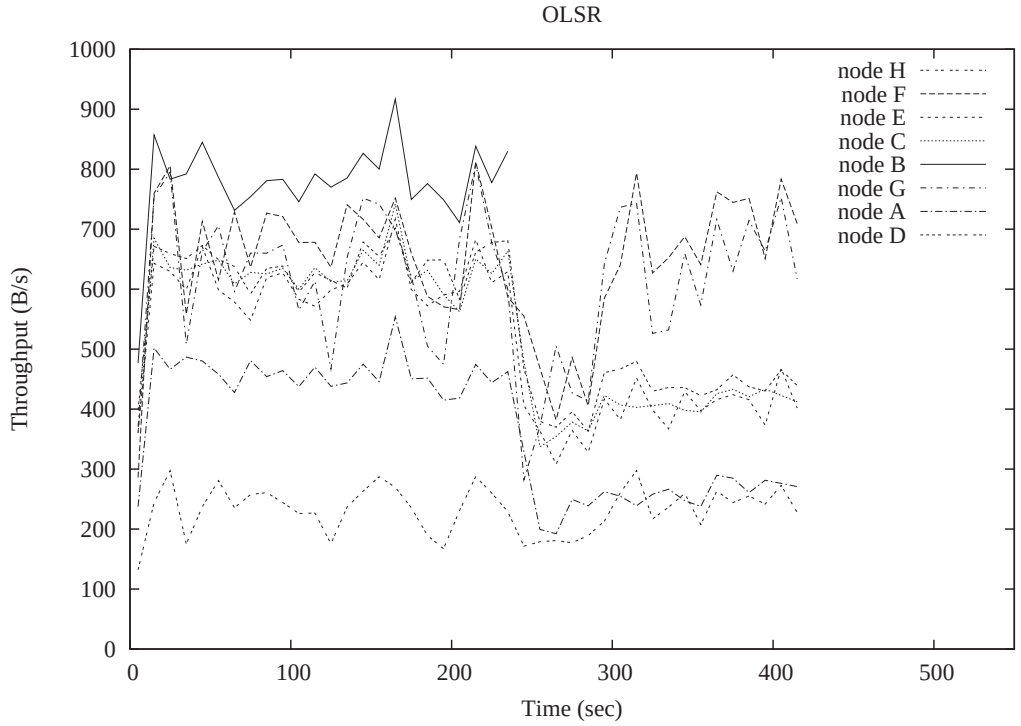



Figure 7.7.: OLSR overhead in case of disconnection event

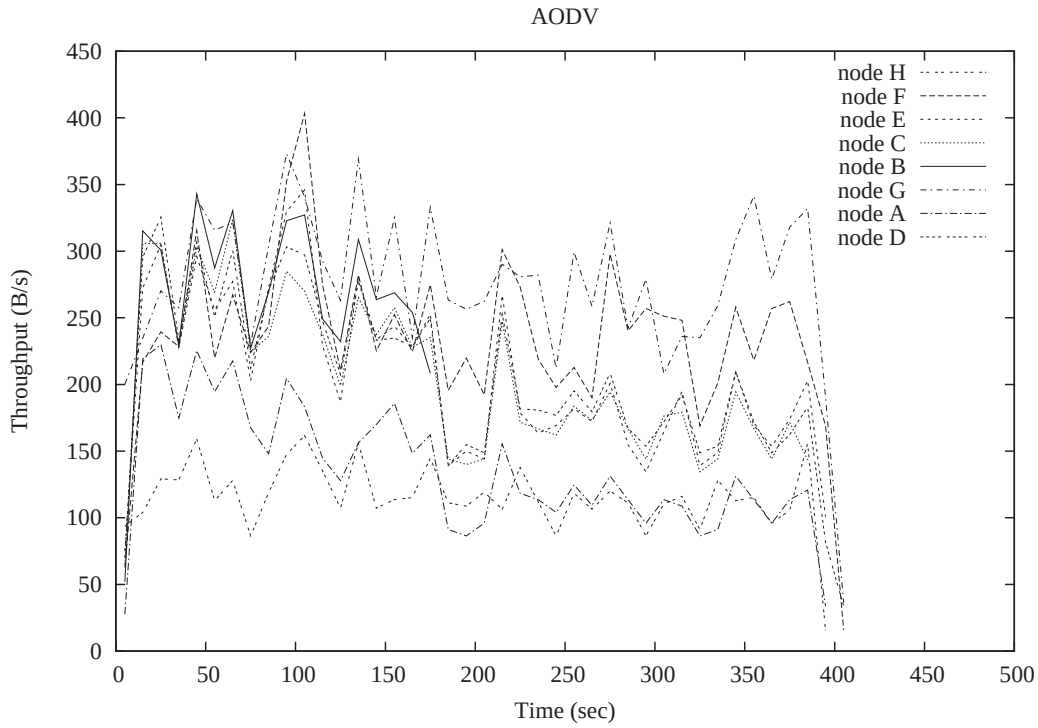



Figure 7.8.: AODV overhead in case of disconnection event

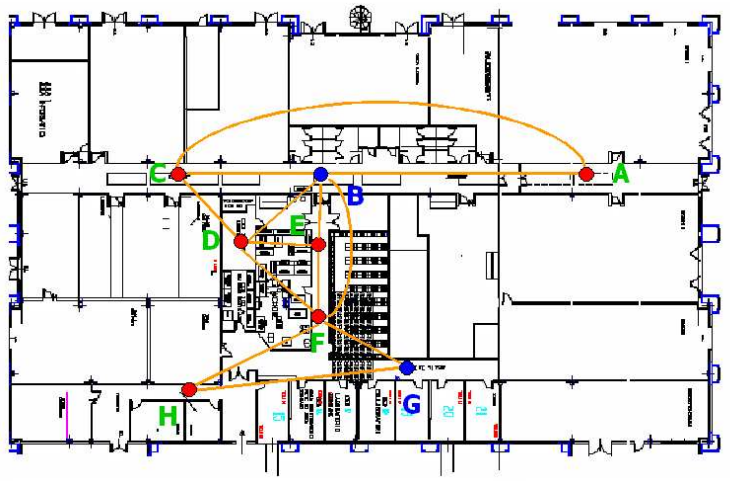



Figure 7.9.: Network Topology for FreePastry Experiments

Referring to Pastry operations to build the overlay network, we decided a bootstrap sequence to allow nodes to correctly join the system. Specifically, node E was the first to start and then it initialized the overlay related to the distributed messaging service. Then the following actions were performed in sequence:

1. F joined the overlay by connecting to E,
2. D connected to E,
3. C connected to D,
4. A connected to C,
5. H connected to F.

At this point all the nodes were connected to the overlay.

In next figures we investigate the costs (in terms of traffic load) required to maintain the Pastry overlay on top of our ad hoc network. It is worth remembering that Pastry generates a management traffic both when a node joins the network (and hence it needs to acquire the information about the other nodes belonging to the same service), and periodically to check the status of the overlay. This operation performed by each node is implemented by opening TCP connections from that node towards all the other nodes belonging to the overlay.

Figures 7.10 and 7.11 show the amount of traffic generated and forwarded by the network nodes using OLSR and AODV, respectively. As expected, nodes B and G, which only participate to the routing and forwarding operations, generate the same amount of traffic observed in Section 7.4.1 in which the traffic was mainly due to routing operations. On the other hand, nodes belonging to the Pastry overlay periodically experience peacks of traffic due to the overlay maintenance operations. To better investigate this aspect, in the following graphs we focus on a single node and analyse the type of traffic it generates.

Specifically, we identify four traffic classes:

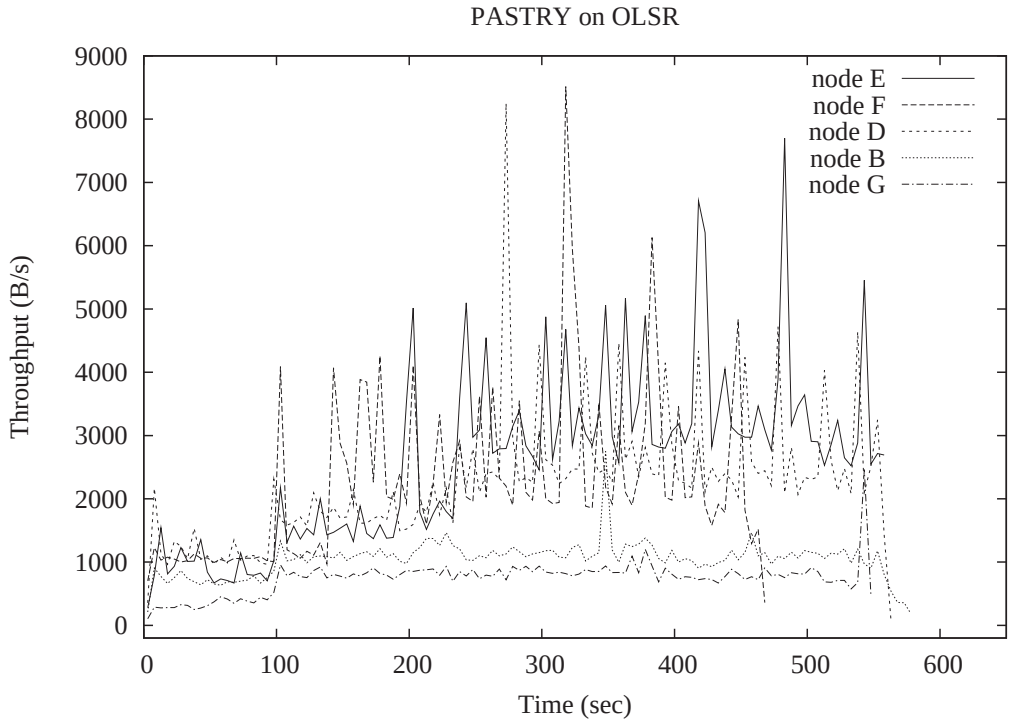



Figure 7.10.: Traffic Load on each node running Pastry on top of OLSR

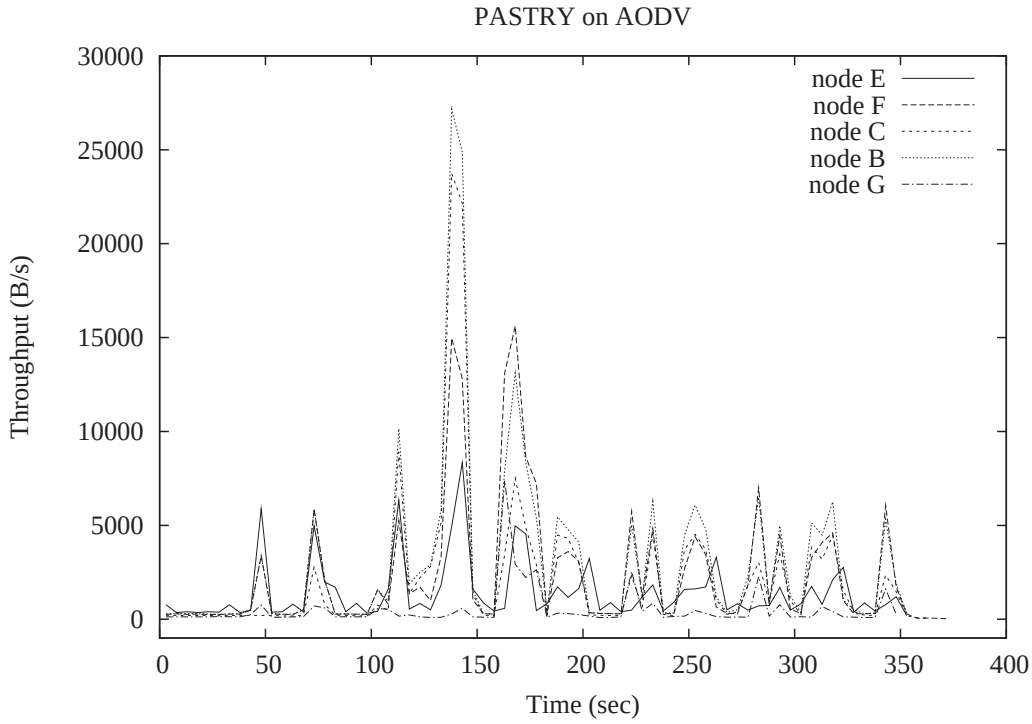



Figure 7.11.: Traffic Load on each node running Pastry on top of AODV

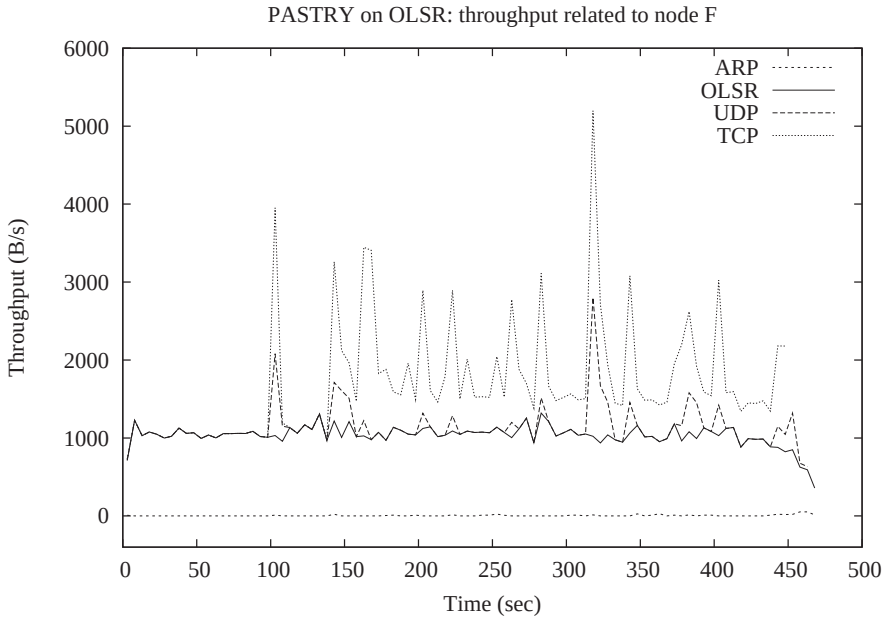



Figure 7.12.: Node F traffic on an OLSR network

- ✧ the traffic due to the ARP-protocol operations;
- ✧ routing traffic;
- ✧ the UDP traffic generated by Pastry overlay maintenance operations (e.g., ”ping” operations performed at the Pastry level by a node to verify the presence of other nodes in the overlay;
- ✧ the TCP traffic used by nodes belonging to the overlay to exchange their routing tables and other overlay-related information.

In the following figures we report the total traffic generated by a single node. More specifically, while the curve ARP indicates only the total traffic produced by the ARP protocol, the curve OLSR indicates the sum of ARP + OLSR traffic. The curve UDP denotes the sum of ARP+OLSR+UDP traffic, and the TCP curve is the total traffic (i.e., ARP+OLSR+UDP+TCP) measured by the node.

Figures 7.12 and 7.13 clearly point out that the traffic peacks are due to the overlay management (i.e., TCP and UDP traffic). On the other hand, from these figures we can observe that the ARP traffic is almost negligible, while routing traffic is quite regular and provides links utilization levels similar to those observed in Section 7.4.1 without FreePastry. Thus, the traffic burstiness is almost due only to TPC/UDP traffic required by the overlay maintenance operations [5].

These observations are confirmed by observing other network nodes. In particular, by observing node A, in case of AODV (see Fig.7.14), it clearly appears that Pastry operations may produce big traffic peacks, mainly during overlay initializations.

[5]The flat behavior in the first 100 seconds of the OLSR experiment is due to the schedule of the experiment. In fact, in this case the overlay starts only after a delay of 100 seconds in order to have the network topology stabilized with complete routing tables. We did not add any delay in AODV, due to the reactive nature of the protocol.

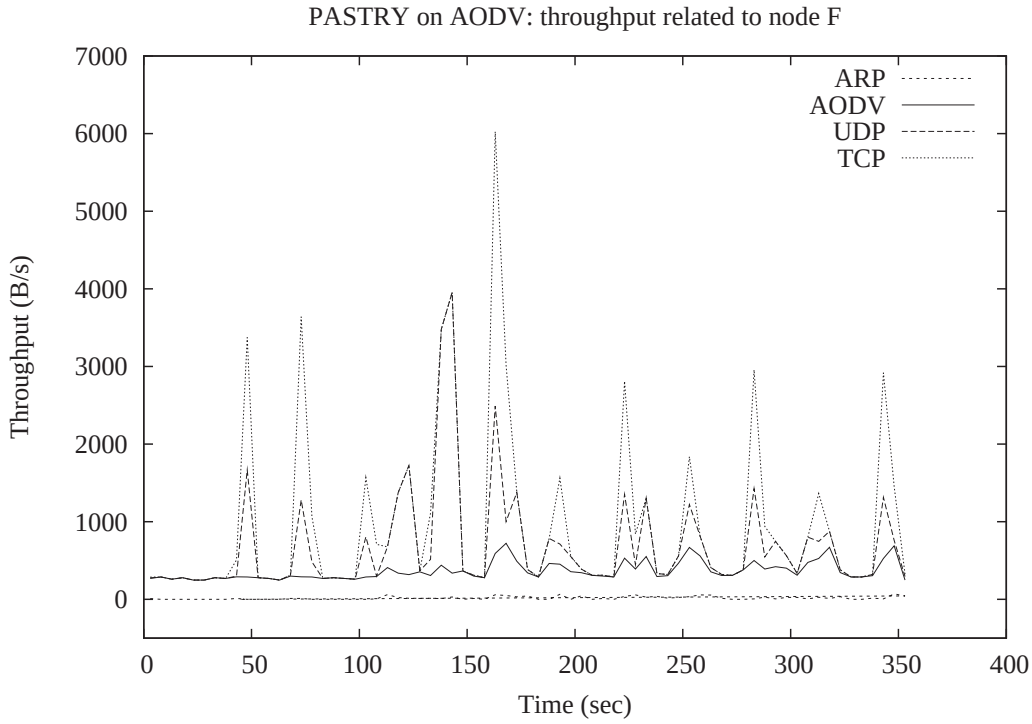



Figure 7.13.: Node F traffic on an AODV network

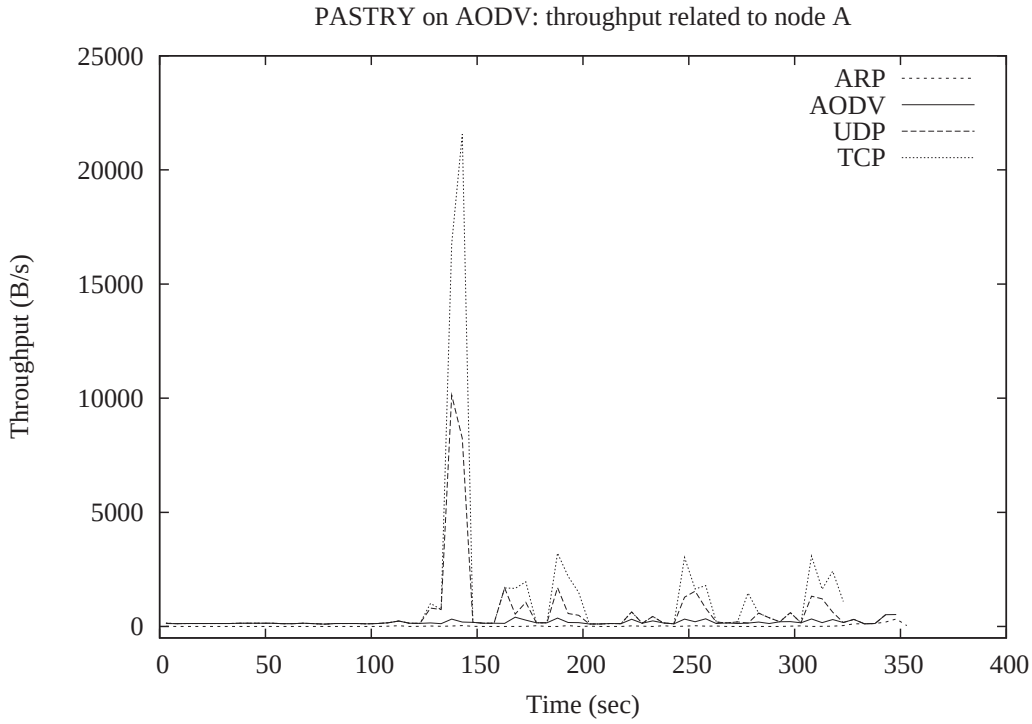



Figure 7.14.: Node A traffic profile (AODV case)

## 7.5. Lessons learned and further work

From this experimentation we can conclude that good pieces of software exist, correctly implementing single features required in a MANET. The implementation of AODV and OLSR we tested are quite robust. They are able to maintain updated routing tables even under frequent topology changes. However, their usage is not yet user friendly. Problems were experienced depending on the release of the LINUX kernel. The FreePastry implementation we tested properly operated on top of our multi-hop ad hoc networks. On the other hand severe problems have been identified from the performance standpoint. Such problems affected almost all MANET layers: network interface, routing and forwarding, TCP, and Pastry. The quality of the wireless links is highly variable. IEEE 802.11 operates in the ISM spectrum and hence experienced a lot of noise from external sources. In addition, the increasing success of WiFi hot spots tends to saturate all the available channels. In this experimentation we have often to switch our MANET on a different 802.11 channel to avoid the influence of existing WiFi access points.

Main routing and forwarding performance problems were experienced when using AODV and are due to the reactive nature of the protocol. Delays introduced by path discovery and maintenance have a strong negative impact on upper layer protocols that use "connection-oriented" operations. Specifically, with AODV, when no route is in the node cache, we measured a delay of about 20 seconds for completing a simple ping-operation between a couple of nodes at a 2-hop distance (values of this order were experienced several times). For the same operation, OLSR takes 1 second (or less) generally having routing tables updated. In rare cases, in which the ping operation was performed just after a change in the topology and hence no updated route was available, we experienced a delay of up to 6 seconds to complete the ping operation. At the upper layers AODV delays often caused timeout expiration (e.g., the FreePastry timeout related to overlay maintenance) that, as a consequence, declared failed an operation that is indeed only delayed due to the route discovery procedure.

From the overhead standpoint we observed that, as expected, OLSR produces a higher routing traffic in respect to AODV, but at least in the network we analyzed, the percentage of this traffic is small compared to the 802.11b available bandwidth.

At the transport level, we experienced TCP problems already pointed out in literature. Long TCP connections show a throughput that decreases with time. This aspect requires further investigation in future experimentation to verify if an appropriate tuning of the TCP parameters is needed (e.g., advertised window).

Finally, at the middleware layer, the FreePastry implementation, by operating its own routing overlay independently from the underlying ad hoc network, introduces a heavy overhead. In addition, the FreePastry maintenance operations exploit TCP services. Thus, poor TCP performances couple with FreePastry overhead to reduce the overall system performance.

From this experience we gained some indications for solving performance problems in our MANET. Specifically, these results encourage using a *cross-layer* architecture for a MANET as proposed in [CMTG04] [CCMT04].

Results related to the comparison between reactive and proactive routing protocols indicate that,

with a proactive protocol: *i)* the response times are much better, and *ii)* the protocols overhead, at least inside the ad hoc horizon, are not heavy.

Furthermore, results related to FreePastry indicate that significant performance benefits can be expected if routing information (extended with services information) can be used at the middleware layer to implement the overlay maintenance operations. In this way the big overhead connected with all remote connections is avoided.

According to this indication, a new optimized solution of overlay network for MANET is designed exploiting cross-layer interactions with a proactive routing protocol. In following chapters main principles of the cross-layer architecture are given, followed by the definition of the optimized p2p system called *CrossROAD: Cross-layer Ring Overlay for AD hoc networks*. The entire system is then validated and evaluated through several experimental sessions to highlight its advantages.

# 8. The Cross-Layer Architecture

One of the major challenges in the research on mobile ad hoc networks is to have them fully functional with good performance while, at the same time, make them able to communicate with the rest of the Internet. The IETF MANET WG proposes a view of mobile ad hoc networks as an evolution of the Internet. This mainly implies an IP-centric view of the network, and the use of a layered architecture. This paradigm has greatly simplified network design and led to the robust scalable protocols in the Internet. The use of the IP protocol has two main advantages: it simplifies MANET interconnection to the Internet, and guarantees the independence of wireless technologies [MC03]. However, current results show that the layered approach is not equally valid in terms of performance [GW02]. The layered approach leads the research efforts mainly to target isolated components of the overall network design (e.g., routing, MAC, power control). Each layer in the protocol stack is designed and operated independently, with interfaces between layers that are static and independent of the individual network constraints and applications.

However, as shown in Fig.7.1 in a MANET some functions cannot be assigned to a single layer. Energy management, security and cooperation and others cannot be completely implemented in a single layer but they are implemented by combining and exploiting mechanisms implemented in all layers. An efficient implementation of these functions can thus be achieved by avoiding a strict layering approach in which the protocols at each layer are developed in isolation, exploiting an integrated and hierarchical framework to take advantage of the interdependencies between them. For example, from the energy management standpoint, power control and multiple antennas at the link layer are coupled with scheduling at MAC layer, and with energy-constrained and delay-constrained routing at the network layer.

On the other hand, the layered approach was, and is, one of the key elements of the world-wide diffusion of the Internet, and a full cross-layer design is an extreme solution. In this case, control information is continuously flowing top down and bottom up through the protocol stack and a protocol behavior adapts both to higher and lower protocols status. For example, the physical layer can adapt rate, power, and coding to meet the requirements of the application given current channel and network conditions. The MAC layer can adapt its behavior based on underlying link and interference conditions as well as delay constraints and bit priorities. Adaptive routing protocols can be developed based on current link, network, and traffic conditions. Finally, the application layer can utilize a notion of soft QoS that adapts to the underlying network conditions to deliver the highest possible application quality [GW02].

The research community recognizes that cross layering can provide significant performance benefits, but it is also pointed out that a layered design has been one of the key element of the success and proliferation of Internet [KK03]. Supporters of the layered architectures point out that:

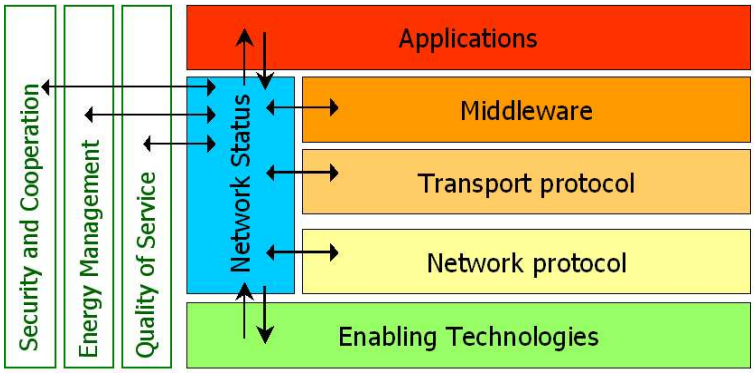



Figure 8.1.: Cross-Layer Architecture

- this design approach guarantees controlled interactions among layers, and hence designers of protocols at a particular layer do not need to worry about the rest of the stack. On the other hand, a cross layer design can produce unintended interactions among layers resulting in performance degradation;
- An "unbridled" cross-layer design can produce a spaghetti-like code that is impossible to maintain in an efficient way as every modification needs to be propagated to all protocols.

Our approach is to introduce inside the layered architecture the possibility of cooperation between protocols belonging to different layers by sharing *network-status* information still maintaining layers separation for protocols design.

## 8.1. The Network Status

Figure 8.1 shows the cross-layer reference architecture defined in the framework of IST-FET MobileMAN Project [mob]. As shown in the figure, the innovation of this architecture is a shared memory, called "*Network Status*" (NeSt) that is a repository of all the network status information collected by all protocols. Each of them can access this memory to write the information to share with the other protocols, and to read information produced/collected from the others. This avoids duplicating layers efforts for collecting network-status information, thus leading to a more efficient system design. In addition, inter-layer co-operations can be easily implemented by variables sharing. However, protocols are still implemented inside each layer, as in the traditional layered reference architecture. This has several advantages:

- Since core functions of each layer are not influenced, a full compatibility with standards is maintained.
- This solution is robust to upgrading, and protocols belonging to different layers can be added/removed to/from the protocol stack without modifying the operations at the other layers.
- It maintains all the advantages of a modular architecture.

To summarize the reference architecture tries to achieve the advantages of a full cross-layer design (i.e., joint optimization of protocols belonging to different layers) still satisfying the layer separation principle.

Layer separation is achieved by standardizing the access to the Network Status. This mainly implies defining the way protocols can read and write data from it.

Interactions between protocols and the Network Status represent enhancements to normal layers behavior, and provide optimization without compromising the expected normal functioning. Replacing thus a Network Status oriented protocol with its legacy counterpart allows the whole stack to keep working properly. For example, using the legacy TCP protocol as the transport protocol of the architecture implies that cross-layer optimizations do not occur at this layer. In addition, in this case the transport protocol does not provide any information to the Network Status but, even though in a degraded way, from the performance standpoint, the overall protocol stack still correctly operates.

The NeSt module stands vertically beside the network stack (as shown in Fig. 8.1) handling possible cross-layer interactions among protocols. In other words, the NeSt plays the role of intermediary, providing standard models to design protocol interactions. While the new component uniformly manages vertical exchange of information between protocols, usual network functions still take place layer-by-layer through standardized interfaces, which remain unaltered.

The idea is to have the NeSt exporting an interface toward protocols, so as to allow them to share information and react to particular events. In [d13] and [CCMT04] the specification of a general interface for the NeSt module is proposed. Specifically, it is specified that NeSt supports cross-layering implementing two models of interactions with protocols: *synchronous* and *asynchronous*.

Protocols interact synchronously when they share private data (i.e. internal status collected during their normal functioning). A request from a protocol for private data takes place on-demand, querying the NeSt to retrieve data produced at other layers, and waiting for the result.

Asynchronous interactions characterize the occurrence of specified conditions, to which protocols may be willing to react. As such conditions are occasional (i.e. not deliberate), protocols are required to subscribe for their occurrences. In other words, protocols subscribe for events they are interested in, and then return to their work. The NeSt in turn is responsible for delivering possible occurrences to the right subscribers.

Specifically, we consider two types of events: internal and external. Internal events are directly generated inside the protocols. For example, the routing protocol could notify the rest of the stack about a *broken route* event, whenever it discovers the failure of a preexisting route. On the other side, external events are discovered inside the NeSt on the basis of instructions provided by subscriber protocols. An example of external event could be a condition on the host energy level. A protocol can subscribe for a *batterylow* event, specifying an energy threshold to the NeSt, which in turn will notify the protocol when the battery power falls below the given value.

In addition, a common data representation inside the NeSt is fundamental for the correct interaction of protocols. To this end, the NeSt works with abstractions of data and events, intended as a set of data structures that comprehensively reflects the relevant (from a cross-layering standpoint) information and special conditions used throughout the stack. A straightforward example

is the topology information collected by a routing protocol. In order to abstract network topology information from implementation details of particular routing protocols, topology data can be represented as a graph inside the NeSt. Therefore, the NeSt becomes the provider of shared data, which appears independent of its origin and hence usable by each protocol.

Details on the interface specification can be found in [d13] while in this chapter we focus on some examples of cross-layer interactions to better understand advantages of this architecture.

## 8.2. Examples of Cross-Layer interactions

Hereafter we give some examples of cross-layer interactions that optimize protocols behavior in a MANET. Let us consider ad hoc routing, which is responsible for finding a route toward a destination in order to forward packets. With reference to the classifications reported in Chapter 5, the main classes of routing protocols are proactive and reactive. While reactive protocols establish routes only toward destinations that are in use, proactive approaches compute all the possible routes, even if they are not (and eventually will never be) in use. Typically, reactive approaches represent the best option: they minimize flooding, computing and maintaining only indispensable routes (even though they incur an initial delay for any new session to a new destination).

But what happens when we consider the cross-layer contribution that a routing protocol may introduce in a NeSt framework?

In the framework of this thesis we mainly focus on middleware and application layers, where the knowledge of the network topology from the routing protocol can extremely facilitate system performance. Thus we consider as the main example the interaction between a routing protocol and a middleware platform aimed at building an overlay network. In this case the routing protocol contributes exporting the locally collected knowledge of the network topology, while the other can exploit these information to maintain a correspondence between the logical and physical address space.

Building an overlay network mainly consists of discovering nodes in the network that provide a selected service or the same set of services (*peers*). Then it must establish and maintain routes toward them, as they will constitute the backbone of a distributed service.

As previously explained in Chapter 6, the overlay network is normally constituted by a subset of the network nodes, and a connection between two peers exists when a route in the underlaying (or physical) network can be established. The task of building and maintaining an overlay is carried out at the middleware layer, with a cost that is proportional to the dynamics of the physical network with degrading performance in MANETs, even in small-scale testbeds (as explained in Chapter 7).

Overlay platforms for the fixed Internet assume no knowledge of the physical topology, and each peer collects information about the overlay structure in a distributed manner with several remote connections. This is possible as the fixed network offers enough stability, in terms of topology, and bandwidth to exchange messages. Of course, similar conditions do not apply for ad hoc

environments, where bandwidth is a precious (and scarce) resource, wireless connectivity is often unstable and the topology is highly dynamic.

In a cross-layer architecture, information exported by the network routing protocol can be offered to the middleware layer. The key idea is that most of the overlay management can be simplified (and eventually avoided) on the basis of already available topology information [Del05]. In this case, the more information available, the easier the overlay management, and for this reason a proactive routing approach results more appealing. To support this claim, let us look at what is described in [SN03]. This paper describes a cross-layer interaction between a middleware that builds an overlay for peer-to-peer computing and a Dynamic Source Routing (DSR) at the network layer. In this work the DSR algorithm is forced to maintain valid routes toward the overlay peers, even if these routes are not in use. In other words, a reactive routing is forced to behave proactively, with the additional overhead of reactive control packets. The same cross-layer approach with a proactive protocol would probably represent the best joint optimization, and every node can maintain a complete view of the overlay exploiting the complete knowledge of the network topology.

At the same time, the network routing table contents are not sufficient to simplify the overlay management. In fact each overlay network has to be associated to a specific service, and only nodes providing that can take part to the overlay. For this reason a Service Discovery protocol is needed to identify all nodes providing the same service. Thus, avoiding the introduction of an additional protocol to send services information over the network, a further example of cross-layer interaction can be represented by an embedded solution of Service Discovery that exploits a proactive routing protocol.

In this case services information, specified by each node to publish a service on the network, can be added as optional field to routing packets and automatically sent on the network. Assuming that services information can be represented by small application identifiers, the hypothesis of a proactive routing protocol cannot negatively influence system's performance. In fact, as analyzed in Chapter 7, the overhead introduced by OLSR, as a proactive routing protocol, is not so heavy compared to performance obtained from AODV experiments. In addition, in order to collect information related to services provided by other nodes, each of them has to be associated to the IP address of the providing node in the NeSt abstraction of the network routing table, such that the middleware layer can directly access them and autonomously build its overlay.

All these concepts represent the building-blocks on which we based our optimized solution of a cross-layer p2p systems for ad hoc networks, called *CrossROAD (Cross-layer Ring Overlay for AD hoc networks)*. The detailed description of this new middleware solution and its software architecture will be presented in the next chapters.

# 9. From Pastry to CrossROAD: Cross-layer Ring Overlay for AD hoc networks

Experimental results showed that applying Pastry [RD01] to ad hoc networks based on a classical legacy architecture is not a valid solution to develop an optimized middleware platform for this kind of scenario. In fact, Pastry is designed for wired networks where thousands of nodes take part in the same service in order to share and exchange information. In this case, nodes have generally fixed positions and they have not power constraints or connection problems, while in ad hoc networks mobile nodes cause frequent topology updates due to their movements or possible coverage problems. In addition they also have to save energy correctly managing their resources.

The classical Pastry model does not care of these aspects, and particularly defines overlay management policies that can negatively influence ad hoc network performance (see Chapter 7). Specifically, join operations and monitoring overlay status require a lot of remote connections, not only to physical neighbors, producing a big overhead on ad hoc networks. In addition the distribution and recovery of information, can introduce a further overhead forcing the message forwarding to use additional path due to the subject-based policy.

CrossROAD represents a new overlay network for MANET, based on cross-layer interactions between middleware and network layers. It exploits the cross-layer architecture to directly interact with a proactive routing protocol, which guarantees a complete knowledge of the network topology, in order to maintain a correspondence between physical and logical address spaces, without increasing the network overhead. In this way CrossROAD can collect routing information in order to optimize the overlay's construction and management simply accessing shared information.

CrossROAD follows Pastry basic principles and, at the same time, it enhances them through the cross-layer interaction with the routing protocol, but the only knowledge of the network topology is not sufficient to define the optimized overlay network. In fact, a Service Discovery protocol is needed to associate to each node of the network the list of services they provide and to allow each node to autonomously build its own overlay. A detailed description of the entire system and its features is given in following sections.

## 9.1. Node Identifiers and Service Discovery protocol

As already described in Chapter 6, Pastry assigns a 128-bit logical address to each node willing to join a ring overlay. This address will represent it in the logical space and it is generally obtained hashing the IP address, the hostname or the public key of the local node, so that those nodes

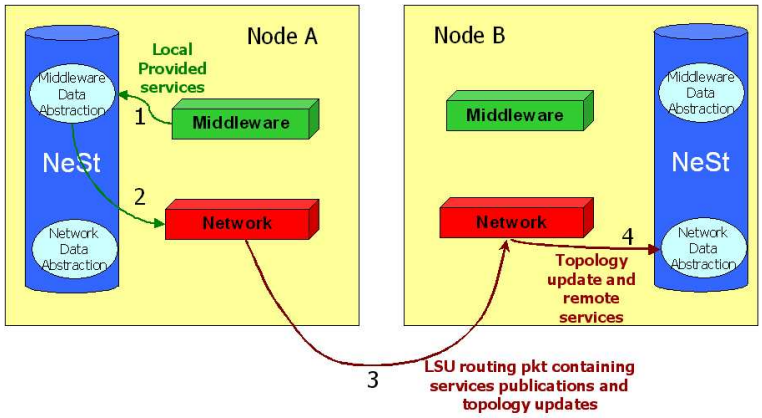



Figure 9.1.: Cross-layer interactions for the service discovery

that are physical neighbors are logically scattered on the overlay. The hash function used to compute logical identifiers uniformly and randomly distributes the logical addresses on the ring, thus guaranteeing a fair balancing of the amount of keys each node is responsible for. In FreePastry [fre], the open-source implementation of Pastry, each logical identifier is even represented by a 160-bit value obtained by the SHA-1 [EJ] hash function.

The same principle is applied in CrossROAD, but in this case, using the cross-layer architecture, each node can directly know the entire network topology from the routing table managed by the network routing protocol, based on a proactive approach. Since the network routing table contains the IP address of each node of the network, the node identifier can be represented by the hash function (SHA-1) applied to this information. However, the IP address of a node is not sufficient to select services provided by every single node. Since all nodes in an overlay network share a single service or a specific set of services, a mechanism to associate to each node the list of services locally provided is needed. To this aim, the common idea of a Service Discovery protocol developed at the application layer is not a good solution for MANETs, since it would introduce additional traffic to spread and collect services information.

The presence of the cross-layer architecture even in this case further optimizes system perfomance. In fact, considering the proactive routing protocol at the network layer, a new embedded solution can be defined. Specifically, the proactive flooding of routing packets can be exploited to broadcast service information on all nodes of the network and, at the same time, it allows nodes to recover the list of services provided by each node. Generally, this solution can be viewed as a software module that collect services information from upper-layer applications and interacts with the routing protocol through the NeSt.

In case of CrossROAD, each overlay network is associated to a single service. Thus, when a node decides to join the overlay, CrossROAD notifies to the NeSt the presence of a new service on the local node, then it informs the routing protocol to publish the new identifier through the first available packet.

Fig.9.1 shows an example of two nodes in the network that exploit the cross-layer interaction between the middleware layer and the network layer to publish a service identifier and consequently

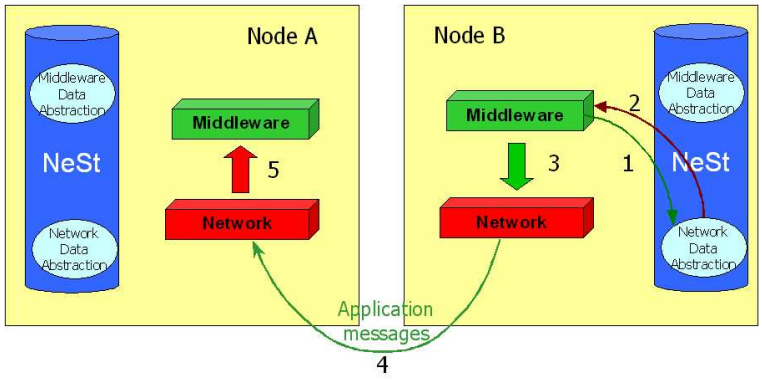



Figure 9.2.: Sending of an application message

retrieve it. In the first step node A starts providing a service and the overlay notifies the related identifier to the NeSt. Then, the NeSt stores the information in its local data structures that represent an abstraction of middleware routing tables, and in step 2 it forwards the same information to the routing protocol. This one not only encapsulates the service identifier as additional information in the first available routing packet, but it also provides a periodical reflooding of the same information through Hello packets until it receives a service disconnection event from the local node. When the routing protocol running on other nodes receives a packet containing additional information, it notifies its content to the NeSt that stores it in its data structure associating to each IP address the list of services identifiers provided by the related node (step 4). Thus, the local node can directly build the overlay network contacting the NeSt and simply applying the hash function to the IP address of the nodes providing the same service. In this way, all remote connections required by Pastry for each join operation are removed and there is no additional overhead to build the overlay network.

In addition, since a MANET topology is highly dynamic and nodes can be characterized by an intermittent connectivity and can also decide to temporary disconnect themselves from the service, the overlay network on the destination nodes is not notified by the NeSt of the presence of a new node. The local node gets back the state of the overlay from the NeSt only when it is going to send an application message. In this case the overlay must have an updated view of all participants to compute the best match for each application message. A simple example is shown in Fig.9.2. In this case node B wants to send an application message on the overlay, but it first requires to the NeSt the current state of the overlay to find the node whose logical identifier is the closest one to the message key (step 1-2). Once the best match has been found, node B directly sends the message to the selected destination (step 3-4-5).

## 9.2. Managing overlay data structures

In order to manage the overlay data structures consequently to join and disconnection events, Pastry requires several remote connections, increasing the overhead of ad hoc network communications. In addition it has to manage three different data structures to maintain the correspondence

between the physical and the logical topology. Specifically, it stores the node identifiers of strict logical neighbors in the Leaf set, those of physical neighbors in the Neighborhood set and the others sharing a common prefix with the logical address of the local node in the Routing Table.

CrossROAD avoids all remote connections for the overlay management exploiting the cross-layer interaction with the routing protocol. In addition it reduces the data structures used to define the overlay to a simple routing table while Leaf set and Neighborhood set disappear.

In fact CrossROAD Routing Table contains the logical addresses of all nodes taking part in the same service, organized following the same prefix-based metric of Pastry. Thus the subject-based routing principle is maintained. In this way the routing table size depends on the number of nodes providing the service and, in the worst case, it is equal to the network size if all nodes participate in the ring. This is a reasonable hypothesis for ad hoc networks characterized by a limited number of nodes. At the same time each node has a complete knowledge of the overlay network. Further information about nodes behavior (e.g. mobility, reliability) can also be inserted in CrossROAD Routing Table, enhancing the proactive routing protocol to maintain different metrics.

Once the CrossROAD Routing Table is locally built, consequently to the join operation of the local node, it has also to be maintained accordingly to the physical network topology and to additional connection and disconnection events. To this aim, Pastry defines a polling procedure limited to the physical neighbors, in order to discover their status (i.e. a remote node is considered disconnected from the Pastry network if it does not answer to a polling message before the timeout expiration). In this way Pastry requires additional remote connections that negatively influence network performance.

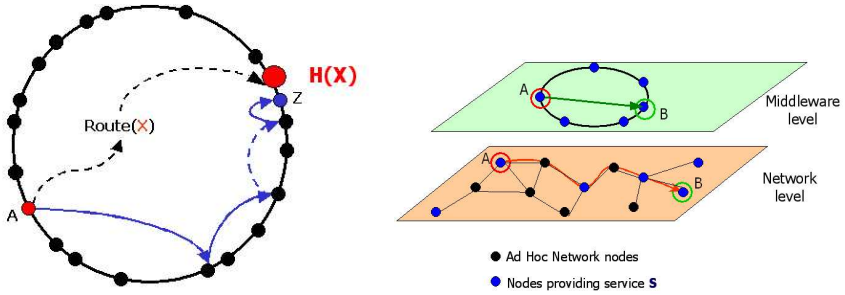
Even in this case CrossROAD does not require any remote connection thanks to the network routing protocol that, periodically sending its LSU packets, recovers all the topology updates and directly manages the routing table and its abstraction in the NeSt. Thus every time CrossROAD has to send a message on the overlay network, it has to verify in the NeSt if the content of its routing table is consistent with the current network topology. Otherwise it has to update its data structures before selecting the best destination and sending the message. In addition the system becomes aware of topology changes or services disconnections with the same delay of the routing protocol.

With this solution the overlay management is enormously simplified and there is no additional overhead on the ad hoc network.

## 9.3. CrossROAD Subject-based routing

The main characteristic of the structured overlay model based on Pastry is represented by the subject-based routing defined to distribute and recover data on the network.

Since the overlay data structures in Pastry have a fixed dimension depending on the network size, they cannot maintain the entire set of nodes taking part to the ring and, for this reason, the subject-based routing is represented by a multi-hop routing at the middleware level if the selected destination is not part of the logical neighbors of the sender (see Fig.9.3a). This implies that at




(a) Pastry subject-based routing

(b) Routing a message in CrossROAD and related network forwarding

Figure 9.3.: Overlays subject-basd routing

the network layer, the message forwarding is forced to send the message to intermediate nodes (logically nearer to the key of the message) extending the optimum path between the source and the destination. This creates an additional overhead to ad hoc network communications.

Instead, in case of CrossROAD this overhead is removed, thanks to the complete knowledge of the overlay. In fact, since each node knows all the others, the sender can directly recover the nearest destination for a selected key and send the message through a simple peer-to-peer connection. Thus, the forwarding protocol at the network layer can deliver the message through the shortest path (see Fig.9.3b). In this way in CrossROAD the logarithmic lookup cost depending on the network size is further reduced to a constant value, independently of the network dimension.

However, whether the local node has a not updated view of the network topology, due to routing delays, the node elected as best destination for the message checks whether in its data structure exists another node closest to the key value. In that case, CrossROAD adopts the proximity metric and forwards the message to the new destination. In this way, the system is reliable in case of routing errors, and the complexity of the lookup procedure is reduced with respect to Pastry.

# 10. Software Architecture

In the framework of this thesis a prototype of the entire system has been developed. Specifically the optimized p2p system has been completely developed and tested on real ad hoc networks, from a small-scale [BCDG05] to a medium-large scale testbed [d16]. In order to highlight CrossROAD advantages, the implementation of the cross-layer architecture has been limited to middleware and routing interactions. In addition, several distributed application have been developed on top of CrossROAD, implementing a common API defined for structured overlay networks [DZD+03], and subsequently enhanced with cross-layer features due to the interaction with the proactive routing protocol (*XL-CommonAPI* [CDG]). Technical details of the software architecture of CrossROAD, the new cross-layer Common API, and the implementation of the prototype of cross-layer architecture are given in the following sections. All applications developed on top of this system together with their performance evaluations will be then described in subsequent chapters.

First of all an overall view of CrossROAD and its interaction with the NeSt and the proactive routing protocol is needed to obtain a complete view of the system architecture. As is shown in Fig.10.1, the NeSt architecture is limited to two main packages aimed at managing services information, directly connected to CrossROAD and the proactive routing protocol through simple interprocess communications. Every time a new instance of CrossROAD is created, because a new service starts running upon it, it sends a *"PublishService"* request to the NeSt which forwards the service information to the routing protocol. This one encapsulates them in the next LSU packet, sending it through the network as soon as possible. Then, when the local node decides to send an application message, it has to check the consistency of CrossROAD data structures through a *"Network Topology Request"* to the NeSt, that recovers the state of the overlay from its topology abstraction.

For the development of this prototype of cross-layer architecture, we exploited an open source implementation of the proactive routing protocol OLSR [Ton] that allows developers to implement dynamic libraries (*plugins*) for the definition of additional information to be sent on the network through routing packets. In case of CrossROAD, this library has been called *XL-plugin*, because it implements cross-layer interactions between middleware and routing protocols and it is described in the next section.

## 10.1. XL-plugin: the NeSt implementation

As previously said, the main idea of CrossROAD is to exploit the cross-layer interaction with a proactive routing protocol in order to collect network topology information and to broadcast services information on the network through the flooding of routing packets. For this reason we

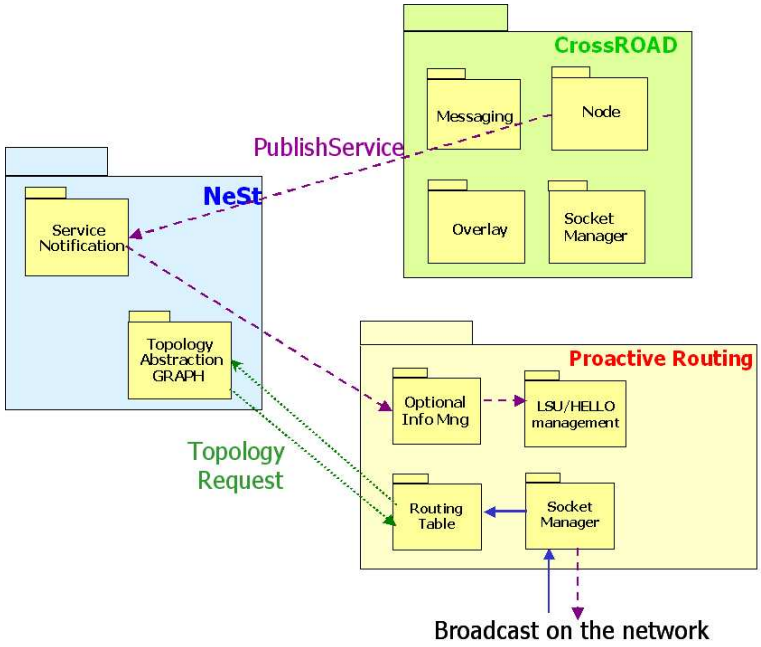



Figure 10.1.: Overview of the software architecture

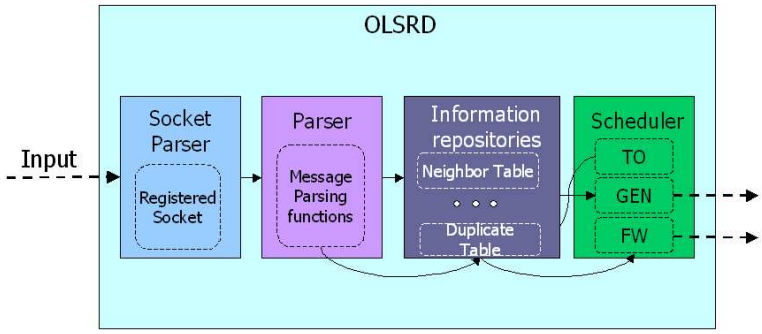



Figure 10.2.: UNIK-olsr software architecture

selected an open source implementation of OLSR (Unik-OLSR v.0.4.8 [Ton]), that we had already tested from the functionality and overhead standpoints as described in Chapter 7. OLSR provides a default forwarding algorithm that allows the distribution of additional messages of unknown types. A user may want to use the optimized flooding technique in OLSR to flood certain information, routing related or not, to all nodes that know how to handle this message. This particular version of Unik-OLSR allows the development of an internal plug-in for the definition of this additional information.

In the case of CrossROAD, the additional information is represented by services identifiers used to associate to each node the list of services locally provided. In fact, each overlay consists of all nodes providing the same service, and every node (in order to join the overlay) has to know all the nodes providing that service.

The software architecture of Unik-OLSR is described in Fig.10.2. It consists of four main packages:

- ✧ *Socket Parser:* this package waits for incoming traffic on a set of registered sockets. Since it is possible to define additional information to be sent on the network, different software modules may want to interact with OLSR and to do this they have to specify a local socket on which they can communicate. When data is received from one of these sockets, the socket parser calls the function associated with the specified socket. Sockets and their corresponding functions are registered at run-time.

- ✧ *Packet Parser:* it receives all incoming OLSR traffic. Particularly it assumes three different behaviors depending on the received packet:

  *i)* it discards the packet if it is found to be invalid;

  *ii)* it processes all messages contained in the packet if it recognizes valid message types;

  *iii)* it forwards the packet according to the default forwarding algorithm if it does not know the message type.

  A parse function is associated to each message type in order to process the related messages and update the information repositories.

- ✧ *Information repositories:* set of tables where information related to the current state of the network is kept. All calculation of routes and packets are executed based on these repositories. In addition all packet parsing functions update and check the content of these tables to process received messages. In particular, the forwarding functionality directly accesses to the duplicate table that is a cache of all recent processed and/or forwarded packets. Each entry of these tables is timed out.

- ✧ *Scheduler:* it runs different events at different time intervals. To transmit a message at a given interval, one can register a packet generation function with the scheduler. Timing out of tables entries is also triggered by the scheduler. To maintain an information repository that is timed out on a regular period, it is necessary to register a timeout function with the scheduler.

The Unik-OLSR implementation supports loading of dynamically linked libraries (DLL), called plugins, to generate and process private message types and any other custom functionality. One of the big advantages of DLLs is that they can be used simultaneously by multiple processes, maintaining only one instance of the library in memory. In this way, plugins provide new functions to an existing application without altering the original application. In addition the definition of plugins does not need to change any code in the OLSR daemon. Users are free to implement and license it under whatever terms they like, and they can be written in any language that can be compiled as a dynamic library.

In general the software architecture of a plug-in (see Fig.10.3) is mainly represented by its local data structures, and it can be organized in multiple threads in order to manage them and interact with the routing protocol, depending on the purpose of each single plug-in. It communicates with OLSR through the interprocess communication (IPC) using a local socket. When it is loaded by the routing protocol, it has to register the communication socket to the socket parser module of OLSR and to define the new message data structure and the related parsing function. In addition

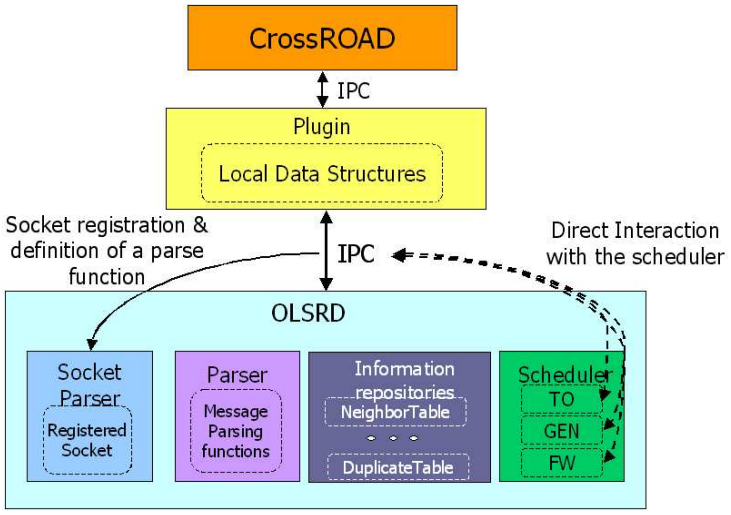



Figure 10.3.: CrossROAD and routing protocol interaction through the plug-in definition.

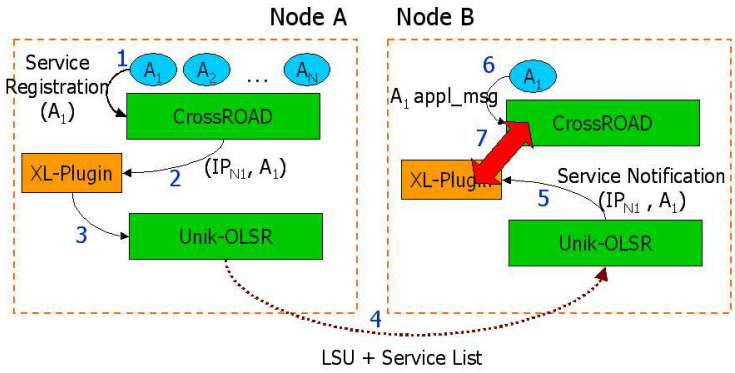



Figure 10.4.: Nodes communication example.

it has a direct interaction with the scheduler module to manage timeouts related to its internal data structures and the generation of additional messages. Finally, it has to define another socket on which it can directly interact with a specific application. Using the IPC model, there are no constraints on the programming languages chosen for the development of the application and the plug-in.

In our case, the plug-in exactly represents the cross-layer interaction between the routing protocol and the middleware platform (CrossROAD). For this reason it has been called *XLplugin*. At this point, CrossROAD and routing interactions shown in Chapter 9 can be presented by interprocess communications between CrossROAD, XL-plugin, and OLSR. Communications between any pair of nodes of the network participating in the overlay are summarized in Fig.10.4.

Since each overlay is associated to a single service, multiple instances of CrossROAD can be active on the same node, related to different services. Fig.10.5 shows how XL-plugin manages interactions between multiple instances of CrossROAD and the routing protocol.

In order to manage all the instances, the XL-plugin has been divided in two main threads:

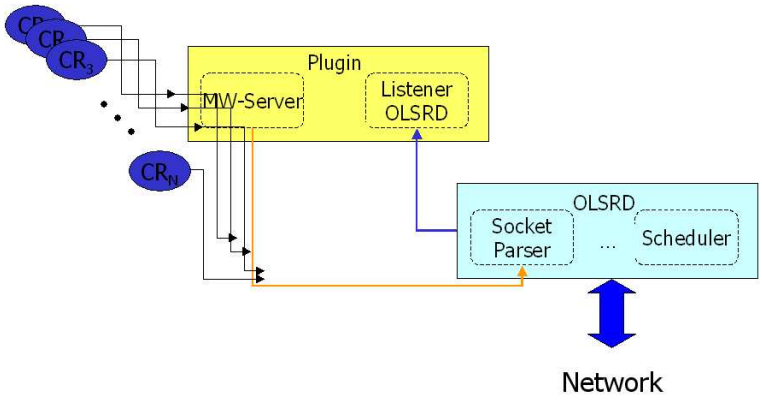



Figure 10.5.: XL-plugin as cross-layer interaction between CrossROAD and OLSR.

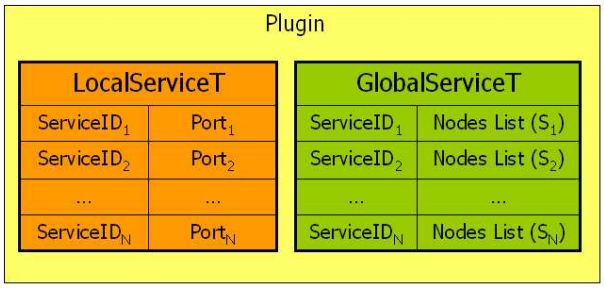



Figure 10.6.: XL-plugin internal data structures.

- *MW-Server:* it registers a local socket to the Socket Parser module to send all additional information. In addition it opens another socket on which it waits for requests from the middleware. When a new instance of CrossROAD is created, it opens a connection to the MW-Server in order to register the service identifier related to the upper-layer application. At this point the MW-Server generates a child thread to manage interactions with each CrossROAD instance. The child thread is responsible for processing CrossROAD messages, updating the local data structures and forwarding the additional information (all related to that particular service) to the routing daemon that will send it on the network through the next available packet.

- *Listener:* when the plug-in is loaded, it registers a local socket to the Socket Parser module on which it receives additional information processed by the plug-in parsing function when OLSR receives packets from other nodes.

Furthermore, since XL-plugin has been designed to implement a cross-layer Service Discovery protocol to optimize the overlay management, two main data structures have been defined. In Fig.10.6 they are represented as two tables: *LocalService Table* and *GlobalService Table*. Specifically, the LocalService Table maintains the list of services provided by the local node. Each entry of this table consists of a 32-bit service identifier (ServiceID) and the related port number on which it is served by CrossROAD. Instead the GlobalService Table maintains, for each service present in the network and currently running on CrossROAD, the list of nodes providing it. For this reason each

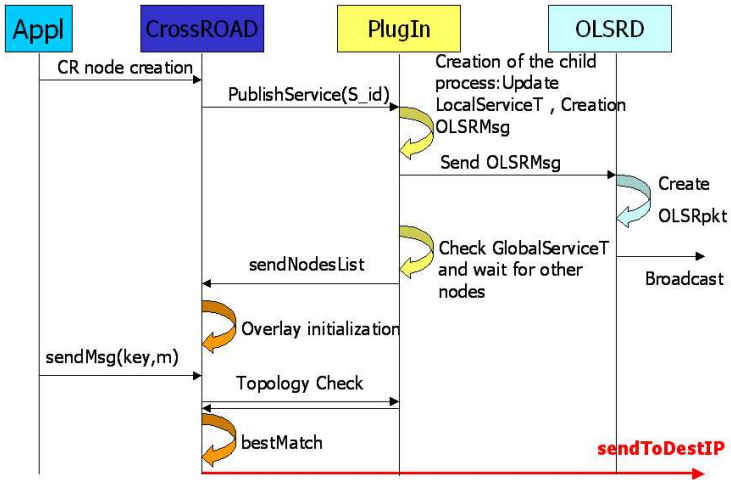



Figure 10.7.: Sequence Diagram

entry of this table is represented by the service identifier and a dynamic list of elements consisting of the IP address of the related node, and the port number on which the specific service is served. All entries are timed out in order to preserve the consistency of the service information.

Through the definition of this dynamic library, we developed a first prototype of the cross-layer architecture even though it represents only a subset of all cross-layer features presented in Chapter 8 and [CMTG04] [CCMT04].

To analyse the time-scale of the system interactions, a sequence diagram is reported in Fig.10.7. Note that XL-plugin, after receiving a PublishService request, waits for at least another node providing the same service to send the related NodeList to CrossROAD. Otherwise all CrossROAD features are useless since the local node cannot distribute information since the overlay is empty.

At this point, to better understand CrossROAD interactions with upper-layer applications, it is important to describe its software architecture. However, since CrossROAD implements the P2P CommonAPI, the architecture of its internal packages strictly depends on the internal structure of this interface. For this reason, before detailing CrossROAD architecture a brief description of this API is given in the following section. The detailed specification can be found in Appendix A.

## 10.2. The P2P CommonAPI and its cross-layer enhancements

The P2P CommonAPI has been designed to allow each distributed application developed on top of it to run on different middleware platforms based on the structured overlay network concept. Every p2p system that implements it maintains applications completely transparent in respect of the internal architecture of the system.

Actually, in literature another possible solution to group structured overlay networks under a common infrastructure have been presented [CDKR02]. It focused on the problem of finding a bootstrap node to join the overlay, and on the fact that every system requires that each node supports the same set of applications, pre-installed on each node. To solve these problems, the

authors proposed the use of an universal ring that only provides the bootstrap functionality while each service runs in a separate p2p overlay. The universal ring enables peers to advertise and discover services of interest, to find the code they need to run to participate in a particular service overlay, and to find a contact node to join the overlay. To this aim, the proposed infrastructure represents a higher level overlay network providing particular kind of services: an indexing service that enables users to find specific services and the list of contact nodes submitting particular queries. It represents a multicast service used to distribute software and related updates and to coordinate members of a single overlay. Since this solution requires to maintain additional data structures for the management of the universal ring, and the consequent increase of data exchange and peers communication in addition to the upper applications, this solution does not represent the optimal approach for ad hoc networks. In fact, in order to collect and maintain services information of all overlays used by nodes of the network, the introduction of selected nodes aimed at managing those information is needed, and temporary connections and mobility of ad hoc nodes do not represent a good prerequisite. For this reason we mainly focus on using a common API that does not require any infrastructure on top of the p2p systems.

However, the original specification of the P2P Common API [DZD+03] is meagre and current implementations of structured p2p systems have customized it reducing the portability of applications. In addition, in mobile environments, the possibility to exploit cross-layer interactions considerably improves overall performance. For this reason, in the framework of this work, this interface has been further enhanced with cross-layer features typical of CrossROAD, so that applications developed on top of it can either exploit cross-layer advantages or run on other legacy structured p2p systems. Specifically, each application developed on top of the XL-CommonAPI can collect network routing table information and the current state of the overlay (e.g. the complete list of nodes that participates to the system). In this way a large set of applications already developed for the Internet can also be ported on ad hoc networks and they can represent a great inheritance for the mobile environment.

Each application running on top of a structured p2p system can exploit a set of basic features that are represented by the assignment of logical identifiers and the subject-based routing. This was defined in [DZD+03] as "key-based routing API (KBR)", the basic layer of the commonAPI.

XL-CommonAPI represents the extension of KBR with cross-layer features. Further features developed to organize particular applications like multicast (Scribe [CJK+03]) or Decentralized Object Location and Routing (DOLR)[HKRZ02] can be implemented directly on top of this API as a simple distributed application. Then this kind of services export another interface to their upper layer, where specific applications can be developed.

XL-CommonAPI appears to the applications' programmer as a set of java interfaces. Most of them are implemented by the underlying p2p system, except the *Application* interface, which is implemented by each particular application.

Specifically, Fig.10.8 shows the UML class diagram of XL-CommonAPI and the specification of most important interfaces. We maintain the maximum consistency with the original definition of each interface to support all applications already developed. A brief description of single interfaces is given below, and a detailed specification is added only to the most important ones. Further technical details can be found in Appendix A.

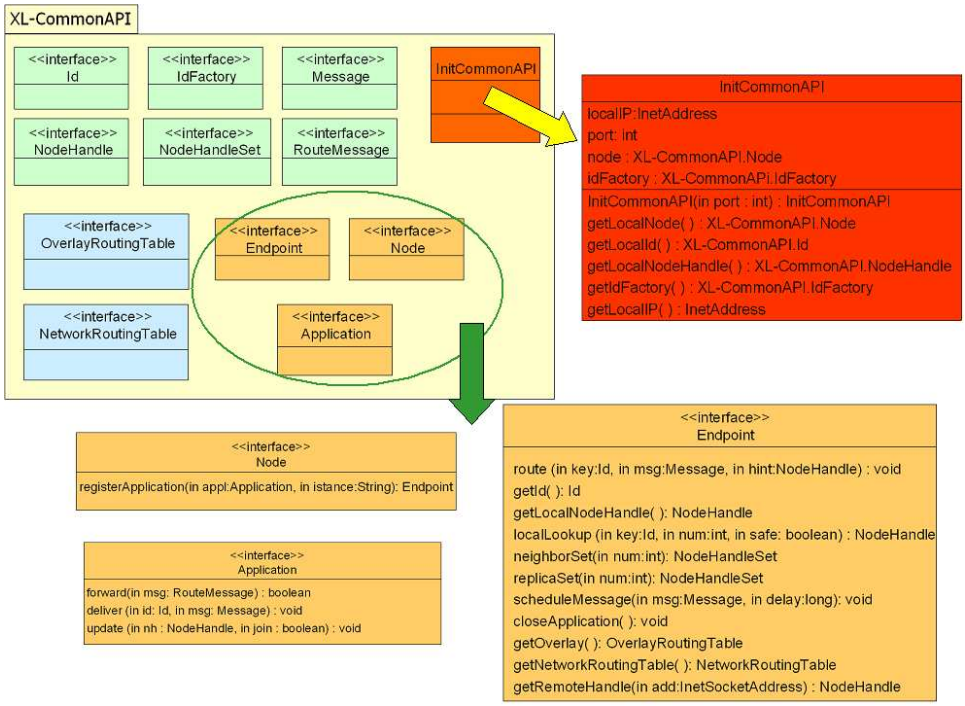



Figure 10.8.: XL-Common API class diagram.

*Id*: data abstraction of the logical identifier. In case of CrossROAD, it is implemented as a 160-bit value obtained from the SHA-1 hash function applied to the IP address of the local host. In Pastry a random quantity is added to the same value.

*IdFactory*: abstraction of the class that implements the arithmetic computation of the logical Id starting from an array of `byte`, an array of `int` values, or a `String`. The p2p system must choose the hash function to be applied to input values.

*NodeHandle*: abstraction of the pair of logical and physical identifiers associated to a specific host (Id, IP address and port where the specific service is provided).

*Message*: abstraction of each message that has to be sent on the network through the overlay. The data structure that represents an application message must implement this interface, that only represents a Serializable java object, to have a reference type for each message.

*RouteMessage*: it represents a group of data structures, i.e. the message to send to the overlay, the key specified for that message, and the optional next hop where the message would be forwarded. This interface can be used by the application to modify the contents of the message, the key or the next hop using specific functions detailed in Appendix A. If the next hop is specified by the application, the system directly connect to it without checking the current presence of the destination in the network.

*Endpoint*: represents the entity that allows the interaction of the programmer with the internal data structures of the p2p system. As shown in Fig.10.8, this interface provides the following functions to the application:

- ✧ `void route(Id key, Message message, NodeHandle hint);` it sends the specified message to the node logically closest to the `key` value. The `hint` value is optional, and it represents a node that should be used by the system to forward the message. Key and hint cannot be both set to null. If a key value is specified, in Pastry, Chord et al. this function represents a possible multi-hop routing of the message to reach the final destination, since each node maintains only a limited part of the overlay in its data structures. Instead, in case of CrossROAD, this function generates a single p2p connection to the final destination of the message, since each node should have a complete knowledge of the current state of the overlay. However, to increase the reliability of the system due to routing delays, the destination checks if it is the closest node to the key, otherwise it forwards the message following the proximity metric.
- ✧ `NodeHandleSet replicaSet(Id id, int maxRank);` this methods returns an ordered set of NodeHandles on which replicas of an object with a given id can be stored. The call returns a number of nodes up to maxRank.
- ✧ `NodeHandleSet localLookup(Id id, int num, boolean safe);` this method returns a list of a maximum number of `num` NodeHandles that can be used as next hops on a route towards the best destination for a given Id. The list is ordered by the logical proximity between the specified `id` and the logical Ids of nodes of the overlay. In FreePastry[1] [fre], if the safe flag is specified, the fraction of faulty nodes returned is no higher than the fraction of faulty nodes in the overlay. In CrossROAD it is equivalent to the previous function, because the subject-based routing is anyway a single p2p connection. However, if the safe flag is set to true, the cross-layer interaction updates the state of the overlay to remove the possibility of faulty nodes, otherwise the system exploits the last updated information stored in its internal data structures. This has been done to reduce the frequency of cross-layer interactions when unnecessary.
- ✧ `NodeHandleSet neighborSet(int num);` this method returns an unordered set of up to `num` NodeHandles that are logical neighbors of the local node.
- ✧ `void scheduleMessage(Message message, long delay);` it schedules a message to be delivered to the local application after the provided number of milliseconds.
- ✧ `Id getId(); NodeHandle getLocalNodeHandle():` they return respectively the Id and NodeHandle of the local node.
- ✧ `void closeApplication();` this function has been added in this version of the commonAPI in order to manage cases in which the application decides to notify its disconnection from the p2p system. In this way, CrossROAD manages the shutdown of the application by sending a *DisconnectMessage* on the network through the routing protocol, to notify other nodes of the disconnection of such node from the service. The receiving nodes update their internal data structures with the delay of the proactive routing protocol. On the other hand, in case of Pastry, the disconnection of a node from the overlay is detected by each node periodically monitoring the status of their logical neighbors, increasing the overhead on the network.

[1] An open source implementation of Pastry developed by Rice University. It implements a commonAPI that reflects basic concepts defined in [DZD+03], but it also extends it with many other specific functions.

Furthemore, functions explained below have been added to this interface as cross-layer enhancements to the interaction between the application and CrossROAD. At the same time, classic p2p systems can simply maintain the empty definition of these functions in their implementation. Specifically:

- ✧ `OverlayRoutingTable getOverlay();` it returns a list of Ids of nodes currently present in the overlay network. In order to return an updated view of the overlay to the application, CrossROAD implements this function as a reactive cross-layer interaction with the routing protocol, that returns the list of Ids of nodes that are providing the specified service. Some applications can be optimized directly knowing the current state of the overlay, managing the distribution of particular messages or becoming aware of overlay changes.
- ✧ `NetworkRoutingTable getNetworkRT();` it returns a list of entries consisting of: IP address of the destination, IP address of the next hop, and cost to reach the destination from the local node. CrossROAD implements this function directly requiring the network routing table to the XL-plugin. Also in this case the knowledge of the physical topology of the network can be exploited to optimize the behavior of some applications, e.g. defining the set of best nodes where to store replicas as the nodes whose logical ids are close to the key but, at the same time, are physical neighbors of the sender. We are also currently working on alternative metrics for the routing protocol (e.g. nodes mobility, link power). Such information can be exported by this function to applications as further cross-layer interactions.
- ✧ `NodeHandle getRemoteHandle(InetSocketAddress add);` it returns the NodeHandle of a remote node, starting from the IP address and the port where the service is provided. This function can be useful for applications that want to force the route of a message to pass through a specific node, knowing its address. To do this, they must know the NodeHandle of the related node but they cannot directly compute it through the *NodeHandle* interface.

In order to define standard data abstractions for middleware and network routing tables, two additional interfaces have been defined: *OverlayRoutingTable* and *NetworkRoutingTable*. They are represented by lists of single elements: Ids in case of OverlayRoutingTable, and (IP destination, IP next hop, cost) in case of NetworkRoutingTable (see Appendix A).

*Node*: entity that directly manages internal data structures of the overlay network (overlay routing tables, association between logical and physical identifiers, messages, and others). It is implemented by the p2p system and it appears to the applications' programmer just as a simple function that generates the Endpoint as a connection between the instance of the service and overlay data structures (`Endpoint registerApplication(Application application, String instance)`). This function allows the programmer to use and manage instances of overlay data structures through Endpoint functions, without knowing their specific implementation.

*Application*: abstraction of the service locally provided. The application's programmer has only to

implement this interface to exploit all features of the overlay network. Specifically it consists of 3 functions:

- ✧ `void deliver(Id key, Message message);` it is invoked by the p2p system on the node that is the best destination for the specified key upon the arrival of the message. The application has to interpret the content of the message and consequently operate on its data structures.
- ✧ `boolean forward(RouteMessage message);` this method is invoked at each node that forwards the message, including the source node. As the parameter is a RouteMessage, the application can decide to modify its contents, the key and/or the default routing behavior selecting a different next hop. If the next hop is modified from a valid value to null, the message will be terminated on the local node. Since CrossROAD always requires just a single p2p connection, this function is mainly maintained for other p2p systems.
- ✧ `void update(NodeHandle handle, boolean joined);` it is used by p2p systems to notify the application of a connection/disconnection event in the overlay. This upcall derives from proactive monitoring procedures that send neighbors discovering messages to recover their current status. In CrossROAD it is completely useless because, if the application needs to know the current state of the overlay, it can directly require it through the related function, and the consistency of the information is guarateed strictly depending on the network routing protocol. In addition, since it is a reactive operation, executed only if required by the application, it does not increase the overhead on the network.

An additional component of this new commonAPI is represented by a java class that initializes all data structures necessary to the service to interact with the p2p system. The application's programmer cannot initialize the overlay data structures only through their interfaces, because an explicit contructor is needed to build an istance of each object. This would require the knowledge of technical details of the overlay implementation in order to direclty access those data structures, making the application dependent on the overlay implementation.

To avoid this dependence, the *InitCommonAPI* class has been defined. In our case this class has been implemented for CrossROAD, and it provides a constructor to create instances of CrossROAD objects, referring to them through the related interfaces. Specifically, it creates an instance of CrossROAD node, the IdFactory that provides the generation of Ids, the local NodeHandle and the local Id. Thus, each application only needs to create an instance of the InitCommonAPI and retrieve from it all the references to the correspondent interfaces. In addition it has to directly implement the *Application* interface and to execute the `registerApplication` method of the Node interface to get the endpoint needed to interact with the overlay network. Once the service has got the reference to the instance of each single object, it can execute all functions declared in the interface. On the other hand, overlay networks have only to implement XL-commonAPI interfaces and modify the definition of the InitCommonAPI class, referring to their internal data structures.

The validation of this new commonAPI has been carried out implementing several distributed applications for ad hoc networks [DP05] [d13]. The description of these applications and the

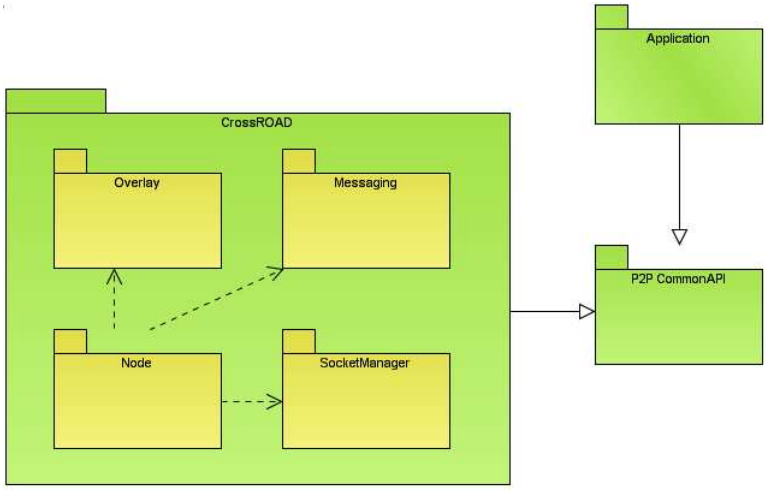



Figure 10.9.: CrossROAD package diagram

related performance results are detailed in Chapter 13, but first of all we must complete the system description with CrossROAD software architecture.

## 10.3. CrossROAD Software Architecture

Analysing the software architecture of CrossROAD we can better understand how the structure of the commonAPI influenced the system design to maintain the maximum correspondence with the other p2p systems like Pastry. As is shown in Fig.10.9, CrossROAD consists of four main packages aimed at implementing different features and managing related data structures. Specifically, we can describe them as follows:

- ✧ *Node:* it represents the kernel of this middleware platform. Not only it defines all objects necessary to interact with other packages, but it also generates all threads designed for the management of the overlay, for the establishment of remote connections towards remote nodes and for the maintenance of the cross-layer interaction with the routing protocol through the XL-plugin. It is also in charge of correctly managing disconnection and failure events.

- ✧ *Overlay:* it contains all data structures designed to collect the overlay routing table and the association between each node identifier and its IP address;

- ✧ *Messaging:* it defines all messages used by CrossROAD to communicate with the application and the XL-plugin;

- ✧ *SocketManager:* it manages all local and remote sockets used to communicate with the application, the XL-plugin, and other instances of CrossROAD running on different nodes for the same service.

Specifically, the *Node* package contains most of the classes that implement XL-CommonAPI interfaces. In particular, it contains the following classes:

- ✧ *Id:* implementing the related interface of XL-CommonAPI it represents a 160-bit logical identifier obtained as the results of the hash function on the IP address of the local node.
- ✧ *HashIdFactory:* class that defines the specific hash function (SHA-1) to build logical identifiers. It implements the IdFactory interface.
- ✧ *NodeHandle:* it implements the related interface collecting the IP address, port and Id of the local node.
- ✧ *NodeHandleSet:* it implements a data structure to collect a set of NodeHandles used to manage a set of possible destinations, or replicas.
- ✧ *CRendpoint:* implements all method defined in the related interface representing the entity that allows the interaction of the application with the internal data structures of the system.
- ✧ *CRendpointMsg:* represents the general structure of a CrossROAD message containing the key used for the subject-based routing together with the application message.
- ✧ *CRnode:* it defines all methods needed to manage internal data structures of CrossROAD and it implements all methods of the Node interface.

Following the previous order, the *Overlay* package contains a set of classes that implement main overlay data structures as represented by their names:

- ✧ *CrossROADRT:* implements the routing table of CrossROAD (logical Ids of all nodes of the overlay) organized following the prefix-based metric originally defined by Pastry.
- ✧ *TempRoutingTableEntry:* implements a single entry of the CrossROAD routing table, used to re-organize the previous data structure after a join/disjoin operation of another node of the overlay.
- ✧ *NetworkRT:* implements the middleware abstraction of the network routing table obtained from the cross-layer interaction with the routing protocol.
- ✧ *RoutingTableEntry:* implements a single entry of the network routing table, used to re-organize the previous data structure after a connection/disconnection event to/from the network.

The *Messaging* package contains all types of messages used by CrossROAD in the inter-process communication with XL-plugin and with other nodes of the overlay:

- ✧ *Message:* class that implements the related interface to define a common structure for all messages send/received to/from the overlay.
- ✧ *ApplicationMessage:* class that defines the common structure of all application messages that has to be passed to CrossROAD.

- ✧ *RouteMessage:* main argument of `forward` function from the XL-commonAPI. It notifies the forwarding of the application message to the next hop. A route message contains a message that has been wrapped to be sent to another node.
- ✧ *Service:* data structure that represents the pair (serviceID, port) used by CrossROAD to require information to XL-plugin.
- ✧ *ServiceNodeList:* list of pairs (IP address, port) of each node currently providing a specific service.
- ✧ *Message2Plugin:* general structure of messages used by CrossROAD to communicate with XL-plugin.
- ✧ *MessageFromPlugin:* general structure of messages that CrossROAD expects to receive from XL-plugin.

The *SocketManager* package contains the definition of all threads used by CrossROAD to manage either cross-layer interactions or the client-server protocol to distribute and recover messages to/from the overlay.

- ✧ *PluginSocketManager:* it defines and manages the socket connection between crossRoad and the plugin.
- ✧ *PublishLookupService:* class that implements the main thread of communciation with XL-plugin. First of all it is used to send a PublishService request and waits until at least another node takes part to the same overlay. When it receives XL-plugin reply, it waits for a subsequent Topology request from the main thread of CrossROAD in case the application wants to send a message on the overlay and the consistency of the internal data structures has to be checked.
- ✧ *PluginReader* and *PluginWriter*: classes that serve as a "reader" and "writer" for messages received/sent from/to XL-plugin via a local socket.
- ✧ *AliveServer:* server aimed at managing the "ALIVE" control message from XL-plugin. This message is periodically sent by the plugin to monitor the status of the CrossROAD instance.
- ✧ *DisconnectClient:* client aimed at sending a *Disconnection* event to XL-plugin in case the application explicity ends. XL-plugin when receives a Disconnect message forwards it on the network inside the first available OLSR packet.
- ✧ *GetRTclient:* client aimed at sending a *GetRoutingTable* request to XL-plugin in case the application requires to know the physical topology of the network.
- ✧ *RoutingServer:* server that manages all incoming requests from other nodes of the overlay.
- ✧ *RoutingServerManager:* it is the manager of a single incoming request.
- ✧ *RoutingClient:* it manages all messages sending from the local node to the others.

- ✧ *SocketReader* and *SocketWriter:* classes that serve as a "reader"/"writer" for messages received/sent from/to the CrossROAD overlay.

Fig.10.10 shows the activity diagram of CrossROAD, where concurrent operations of several threads are described. Specifically, after the initialization of CrossROAD data structures through a new instance of the InitCommonAPI class, the endpoint is created calling the `registerApplication` function. This is in charge of creating the AliveServer thread that is responsible for a specific control message from XL-plugin to monitor the status of CrossROAD. Concurrently the PublishLookupService client is created to manage the message exchange with XL-plugin to publish the service and recover the current list of nodes. Once at least another node takes part in the overlay, the CrossROAD routing table is initialized, and the overlay is active. At this point the RoutingServer thread is created to manage incoming messages from other nodes, while the system waits for local application messages to be sent on the overlay. For each application message the best destination is computed and the P2P connection is established. When the application sends a closeApplication message, the related Disconnect message is sent to XL-plugin to be flooded on the network and all active threads and sockets are closed ending the entire system.

After this exhaustive description of the software architecture of the entire system, a detailed analysis of its performance in small scale and medium-large scale ad hoc networks is given in next chapters.

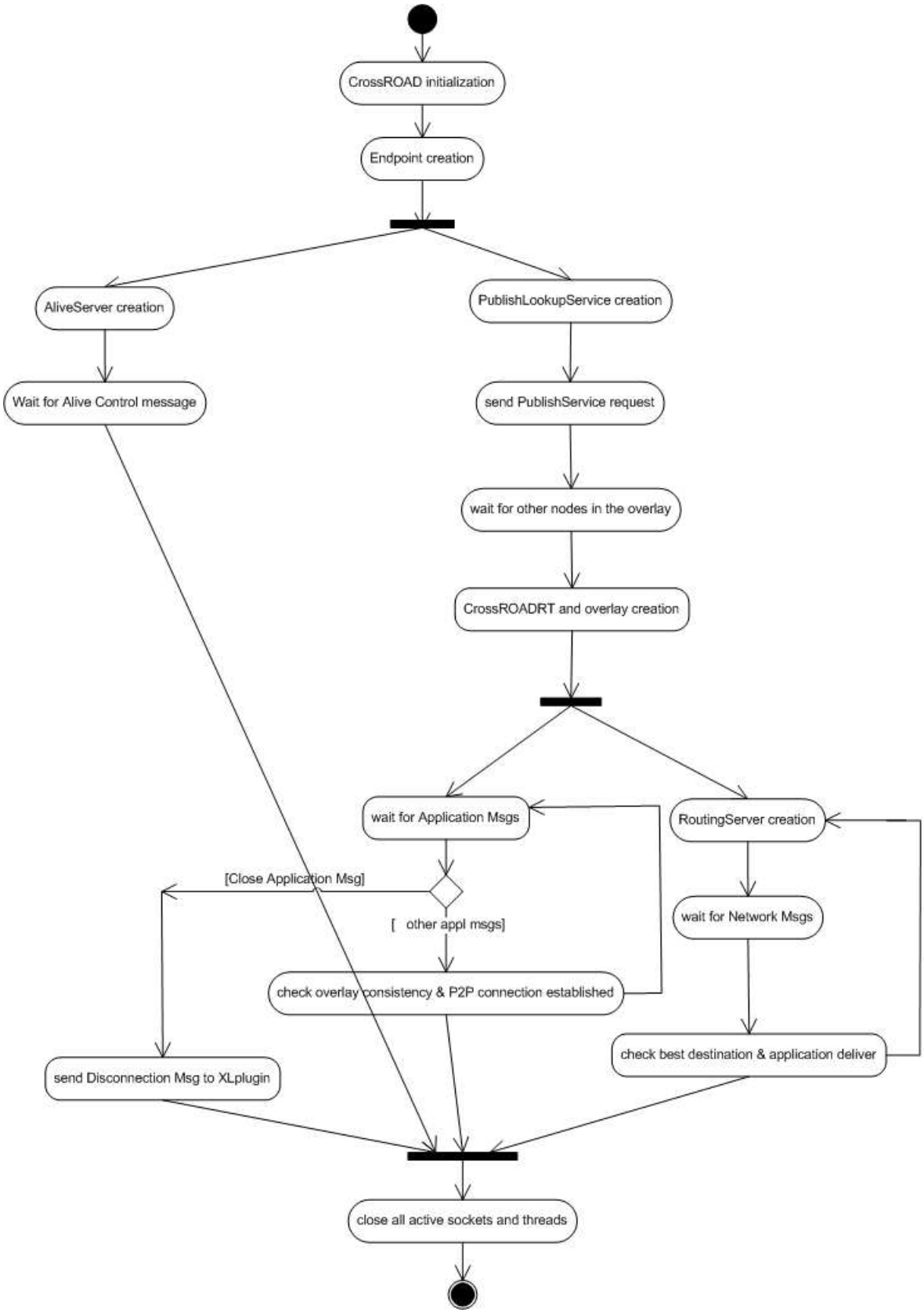



Figure 10.10.: CrossROAD activity diagram

# 11. CrossROAD on a small-scale ad hoc network testbed

In this chapter we report our experiences and results obtained by measurements on a real MANET implementing a full ad hoc network architecture as extension of our first experimental analysis (see Chapter 7). This part of our work provides an orginal contribution to the research community setting up a cross-layer MANET prototype. Specifically, we performed several sets of experiments and we focused the study on different solutions for routing protocols and middleware platforms. Specifically:

1. we investigate a full protocol stack with particular attention to routing and middleware layers;
2. we evaluate through experimental results the advantages of the cross-layer architecture, mainly focusing on routing and middleware interactions.

## 11.1. Testbed architecture and experiments environment

The novelty of this part of our study is the analysis of a full ad hoc network protocols stack, from the physical layer up to the application layer, comparing performance results of a legacy-layer architecture with those of a cross-layer architecture. Specifically, our prototype represents a cross-layer solution to exploit cross-layer interactions between middleware and routing protocols. A first phase of this experimental evaluation, presented in Chapter 7, was divided in the following steps:

- ✧ *Routing:* functional analysis of existent implementations of reactive (AODV [PR99]) and proactive (OLSR [rfc]) routing protocols and their comparisons in terms of traffic generation and configurations delays;
- ✧ *Middleware:* porting of Pastry on top of the selected routing protocols, in order to evaluate the performance of a structured p2p platform on a MANET. Specifically, we used FreePastry [fre] as open-source implementation of Pastry.

This chapter represents an evolution of the previous analysis, considering not only mobility scenarios and their impact on the network topology reconfiguration, but also evaluating CrossROAD performance.

Before analyzing system performance, a detailed description of the testbed architecture and the experimental environment is needed. All the experiments have been executed inside the same building of the CNR campus in Pisa used in previous experiments. In order to deploy a multi-hop ad hoc network in a small geographic area, physical characteristics of the building (masonry walls) and possible interferences (due to the presence of access points and measurement instrumentations), were exploited to limit transmission capabilities of nodes and obtain realistic wireless links (i.e., with variable quality due to external interferences). Taking into account the environment constraints we identified the initial positions for mobile devices and hence the starting topology of our MANET. For all the experiments a set of (up to 8) laptops (IBM R50) equipped with the integrated wireless card Intel PRO-Wireless 2200 have been used. Experimental scenarios cannot be compared with those used during simulations, where hundreds of nodes move inside areas arbitrarily wide. However, a small ad hoc network represents a more realistic scenario in which users, equipped with mobile devices, can decide to connect to each other, sharing information [GLNT]. The initial phase of the testbed was devoted to investigate the performance evaluation of network layer protocols, i.e., AODV and OLSR on static and mobile scenarios. Then, we focused on the performance evaluation of two different middleware platforms, Pastry vs CrossROAD, on top of the two routing protocols.

To compare and contrast the Pastry model with its cross-layer enhancement, we used the *Distributed Messaging (DM)* application, already developed on top of FreePastry, also on top of CrossROAD. Nodes running DM set up and maintain an overlay network related to this service. Once a node has created/joined the overlay, the application allows users to create/delete one or more mailboxes distributed on the other nodes and to send messages to them. A mailbox's physical location is randomly selected applying the hash function to the associated identifier, used as the key value of the related messages.

## 11.2. Experiments and performance evaluation

In order to obtain a complete performance evaluation of the entire system on a real multi hop ad hoc network, several sets of experiments had to be executed, focusing on different aspects of the MANET architecture. First of all we examined routing performance in case of mobility scenarios and topology changes. Then we analyzed CrossROAD in terms of functionality, overhead introduced on the network, and delays needed to build the overlay network and to become aware of overlay and topology changes.

### 11.2.1. Routing performance

In this section we present the performance evaluation of the routing protocols on a string topology. As evolution of a previous work [Bor05], in which only static networks were considered, we introduced mobile nodes in order to evaluate the impact of the mobility on routing protocols. To this end, we considered a string topology network of four nodes and we performed three sets of experiments increasing the number of mobile nodes (all the scenarios are shown in Fig.11.1).

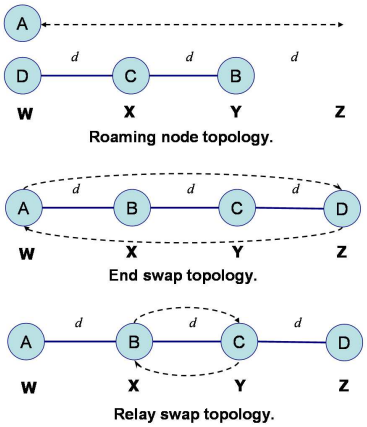



Figure 11.1.: String topology

In this scenarios the connectivity between the sender and the receiver changes from 1 hop to 3 hops and viceversa during the experiments. To have a comparison between OLSR and AODV, we studied the *Packet Delivery Ratio* (total number of packets received at the intended destinations divided by the total number of generated packets) and the delay needed to reconfigure the network consequently to the movements of nodes. 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 particular, in our scenarios, all the nodes start running the routing protocol and, after an initial period necessary for 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, thus we present an average of them. One may argue that similar set of experiments were already available in literature [Lun]. 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 underlaying 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 (see Chapter 7) and those presented in this chapter 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 [Lun], and they 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 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 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 11.1, we note that increasing the complexity of the proposed scenarios, 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 a 2-hop to a 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 are correctly received since the network is already stabilized when data transfer starts. In contrast AODV distributes uniformly its packet loss during the entire test-run.

As previously said, in the third set of the experiments the packet delivery ratio of OLSR and AODV decreases up to 66% and 60%, respectively. In particular, from the log files we note 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 tabl e for OLSR and to discover new paths to the destination for AODV.

In the first scenario, when node A moves towards position Z, OLSR requires $5sec$ to discover a 2-hop path to D after the direct link A-D is lost. 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

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

Table 11.1.: Overall Packet Delivery Ratio.

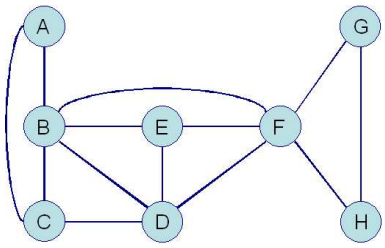



Figure 11.2.: Experimental network topology

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.

### 11.2.2. Overlay network performance

In order to evaluate CrossROAD performance, several sets of experiments were performed using 8 nodes (see Fig.11.2) but only 6 running the overlay. In fact nodes B and G just worked as routers. During the experiments all nodes were synchronized and started running OLSR enhanced with the *XL-plugin*.

In the first set of experiments, the overhead introduced by CrossROAD and FreePastry has been measured. The experiment consists of defining an order with which nodes start running the overlay, and no message generation is required by the application. In this way we can measure only the overhead introduced by the overlay construction and management. The main difference between these experiments is due to the self-organizing nature of CrossROAD and its complete independence on building and managing the overlay.

In fact, as is shown in fig.11.3, nodes B and G, that just work as routers, only perform routing traffic that is almost negligible (about $50Bps$). But also the other nodes, which start running the overlay with a delay of $10sec$ one from the other (interdeparture time), introduce an overhead less than $100Bps$. Comparing these results with Pastry performance on the same set of experiments (see fig.11.4), we can notice that nodes running the overlay introduce a much higher overhead than in case of CrossROAD, with several traffic peacks corresponding to TCP and UDP connections used to initialize and maintain the overlay data structures.

Another important feature of CrossROAD is represented by the timeliness with which every node at the startup becomes aware of the other participants. Since the *PublishService* message is sent on the network as soon as the next routing packet is ready, the delay introduced for each node to

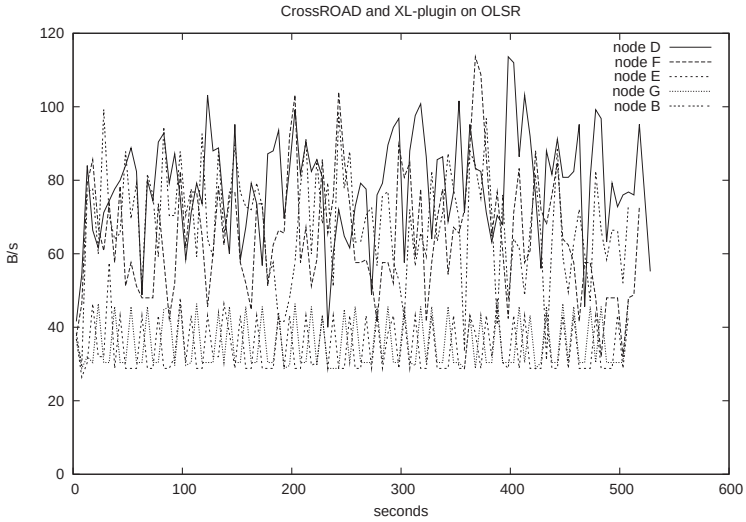



Figure 11.3.: CrossROAD: Throughput related to main nodes.

| Delays for overlay establishment | First node | Other nodes |
|---|---|---|
| $2^{nd}$ node at 1-hop distance | $180msec$ | $40msec$ |
| $2^{nd}$ node at 3-hops distance | $325msec$ | $30msec$ |

Table 11.2.: CrossROAD delays for the overlay construction.

collect information about all nodes taking part to the overlay, is lower than in case of establishing one or more remote connections.

In order to obtain these results, we set up two sets of experiments using an interdeparture time of $10sec$. In both cases the first node starts running CrossROAD and waits for being notified from the XL-plugin that some other nodes join the overlay. Specifically, in the first set the second node is 1-hop distant from the first, while in the second set the second node is 3-hop distant from the first node. As shown in table 11.2, the first node of the overlay experiences different delays in those experiments, due to the physical distance from the second node. When they are 1-hop neighbors, the first node experiences a delay of $180msec$, while in the second case it experiences a delay of $325msec$. The measured delay corresponds to the interval time between the sending of the *PublishService* from the second node to the notification of the list of nodes of the overlay to the first node. All the other nodes experience a delay of about $30 - 40msec$ because when they start the overlay, the plugin has already stored at least one other node providing the service in its local data structures. Hence, in this case, the delay represents only the processing time needed to exchange messages between CrossROAD and the plugin on the local node.

As the last set of experiments, we analyze a possible network partitioning and the consequent reaction of CrossROAD in the overlay management. To do this, a new network topology, shown in Fig.11.5a, has been set up. Specifically, the ad hoc network consists of 5 nodes, and only nodes in adjacent positions are in the transmission range of each other. All nodes start running OLSR enhanced with the XL-plugin, and after about $30sec$ to have the network topology stabilized, they start executing CrossROAD with a delay of $10sec$ from each other.

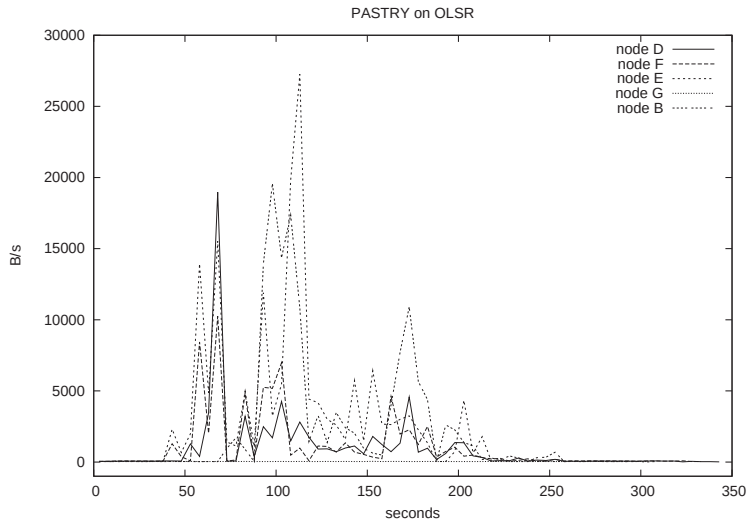



Figure 11.4.: Pastry: Throughput related to main nodes.

When all nodes are correctly connected to the overlay, node C starts periodically sending an application message with a specified key value. The period with which the message is sent on the overlay is set 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 $1m/s$, generating a network partitioning: nodes A and B create an independent ad hoc network as well as nodes C, D, and E (see Fig.11.5b).

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.

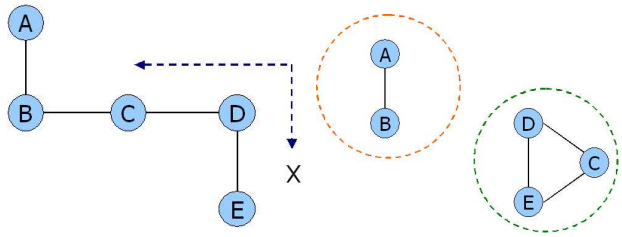
After 2 minutes, node C starts coming back to the initial position re-establishing a single ad hoc network. At this point, the following messages are sent again to node B. As shown in Fig.11.6, CrossROAD, thanks to the cross-layer interaction with the routing protocol, correctly manages data distribution in case of overlay and network partitioning. Specifically, the figure shows that:

*i)* during the first phase (a single ad hoc network), C's data are stored on node B;

*ii)* when the partition occurs, after a transient, C's data are delivered to node D;

*iii)* when the network is connected again, C's data are stored again on node B.

It is important to specify that CrossROAD, before sending each application message, requires to XL-plugin the list of nodes currently taking part to the overlay. At the same time, the XL-plugin has to maintan the consistency of its internal data structures with the processing of service discovery messages periodically received. For this reason, a validity timeout has to be associated to each entry of the *GlobalServiceTable* and a *ServiceMaintanance* message has to be sent periodically on the network.




(a) Initial Topology (b) Partitioned Topology

Figure 11.5.: CrossROAD mobility experiment

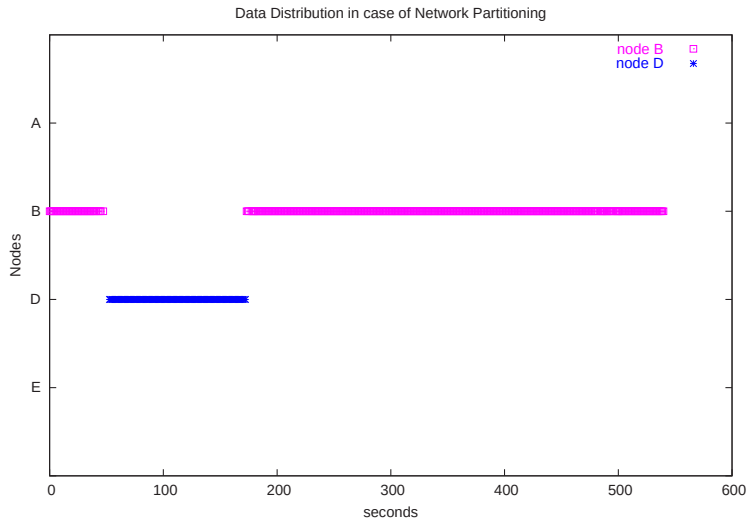



Figure 11.6.: CrossROAD reaction to network partitioning

In order to guarantee the maximum timeliness of CrossROAD to overlay topology changes, this period is set equal to that used by OLSR to flood `Hello` messages on the network $(2sec)$. This procedure does not negatively influence system performance because each service identifier is simply represented as 32-bit value. Hence, even changing the maintanance period, the total overhead does not considerably change.

In conclusion, cross-layer interactions between routing and middleware make every node of the overlay completely autonomous from the others, optimizing data distribution and recovery for upper applications without introducing additional overhead on the network. In this way, all the advantages introduced by the use of a structured p2p system are further optimized and all disadvantages introduced by Pastry on ad hoc networks are completely removed.

## 11.3. Conclusions

During this study a large number of experiments has been set up in order to obtain real measurements on a full ad hoc network architecture. In particular routing protocols and middleware

platforms have been evaluated in terms of overhead and reconfiguration delays in case of mobility scenarios and network topology changes. This real testbed demonstrated that:

*i)* the use of a proactive routing protocol (OLSR) does not penalize the system performance either in terms of PDR and reconfiguration delays in static and low mobility scenarios;

*ii)* CrossROAD drastically reduces the overhead introduced by a classical p2p system on ad hoc networks, and correctly manages cases of network partitioning and topology changes.

The next step of our study is to create a larger testbed in order to evaluate the overall system performance on a large scale ad hoc network, as is shown in the next chapter.

# 12. CrossROAD on a medium-large scale ad hoc network testbed

An extensive experimentation on our mobile ad hoc solutions involving up to 23 nodes 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.

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 adjoining building was also used (see Fig.12.1). The structural characteristics of these buildings strictly determine the transmission capabilities for nodes of a wireless network located within. 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.

A great number of experiments had been conducted during this experimentation. In following sections the most meaningful experiments are described and main results on system performance are detailed.

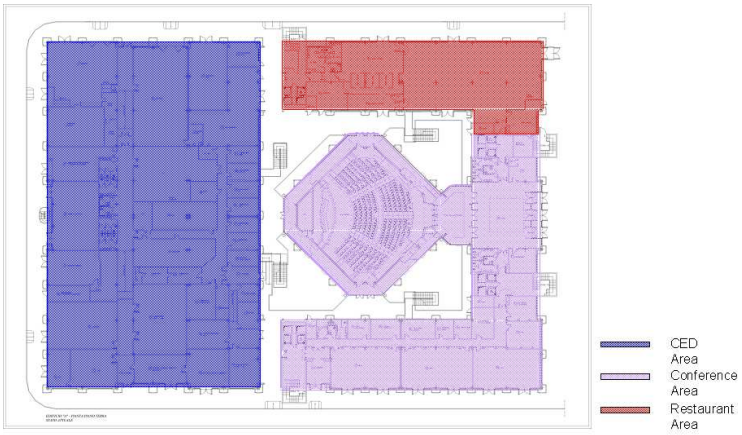



Figure 12.1.: Testbed area

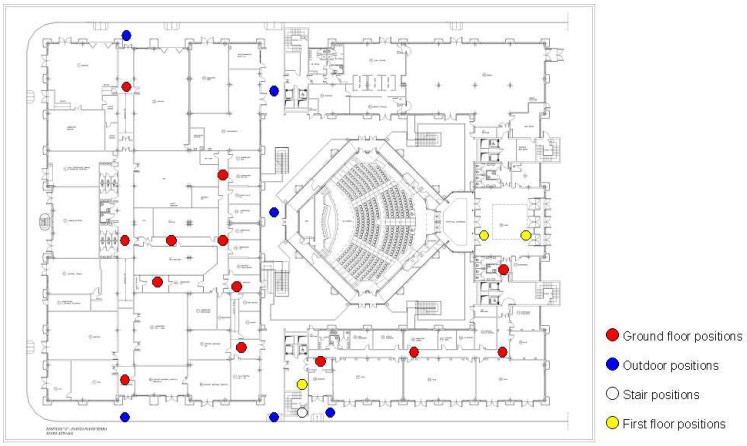



Figure 12.2.: Physical position of nodes

## 12.1. 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 Chapter 7. 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 Fig.12.2. Most part of nodes (17) was located inside buildings. In particular we placed 13 nodes at the ground floor (red circles), three at the first floor (yellow circles), and one on the stairs (the white circle). The last six nodes were located outside the buildings (blue 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.

Fig.12.3 shows the final network topology, where straight lines point out the presence of stable links (two nodes directly see each other), while 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

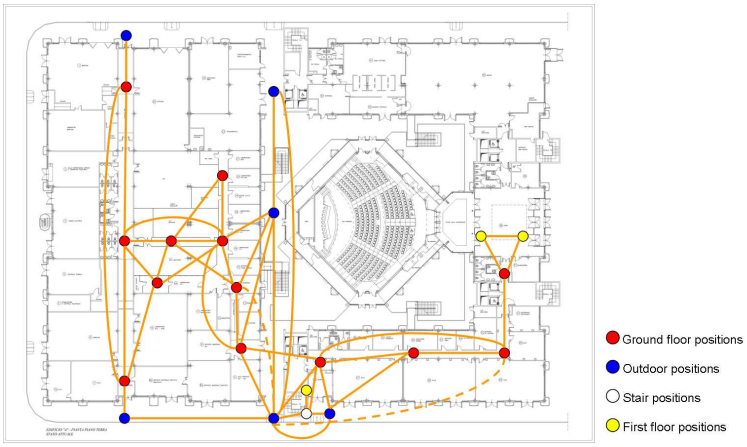



Figure 12.3.: Network topology

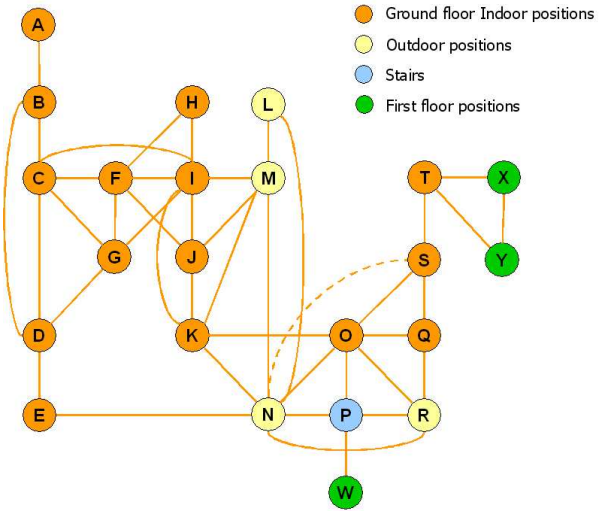



Figure 12.4.: Topology graph

illustrated in Fig.12.4.

The experimentation focused on the analysis of different layers of the protocol stack to compare a legacy architecture with those of a cross-layer architecture. Even in this case a comparative performance analysis of two routing protocols (OLSR and AODV) have been done both in static and mobile scenarios. The detailed analysis can be found in [d16], while in this chapter we mainly focus on middleware performance.

## 12.2. Middleware Experiments

In the middleware experiments we compared CrossROAD and FreePastry in static and mobile scenarios running the Distributed Messaging application on top of them. In static scenarios we mainly analysed the overhead introduced by the overlay management on the network in terms of

network load 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.

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 to avoid connection failures during this phase. To maintain a correspondence between the two different sets of experiments, the same bootstrap sequence was also used for CrossROAD experiments. Specifically:

- 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 neighbors;
- D, J, Q, S, P, started DM after 20 seconds from N, as its 2-hop neighbors;
- C, B, G, F, I, T, Y, W started DM after 30 seconds from N, as its 3-hop neighbors;
- A, H, X started DM after 40 seconds from N, as its 4-hop neighbors.

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 specified different start-up delays. They are all active after 60 seconds from the starter of the overlay.

In particular, to numerically evaluate system performance grouped by network load, 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 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 (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. In this case we could evaluate system performance with a different traffic load.

- ✧ Experiment 5: 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 Fig.12.5). 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 joined in the overlay, A and Y started sending a Get message every 200 msec for 6 minutes. This experiment allows us to evaluate system performance in case of merging of two networks and subsequently of two overlays.

A detailed description of each experiment can be found in [d16]. In the following sections the analysis of main performance results obtained by selected experiments are presented. In particular, the overhead to maintain these two different overlays in terms of network load is presented in Section 12.3. Then, measured delays to distribute and recover data are analyzed in Section 12.4 and, finally, the responsiveness of the cross-layer interactions between CrossROAD and OLSR to distribute data in a partitioned overlay is described in Section 12.5.

## 12.3. Network load 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 only the routing protocol.

We defined the average traffic load as the aggregation of the overlay management traffic, 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

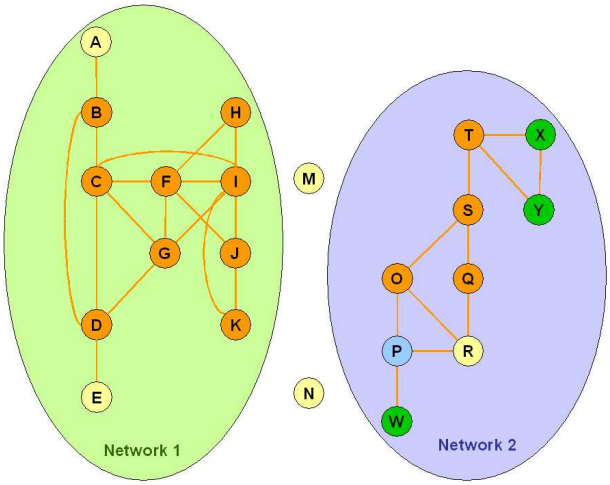



Figure 12.5.: Network topology for delayed joining experiment

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.

The average traffic load is shown in Fig.12.6 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 crosslayer 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).

Analysing Fig.12.7 in more detail, we can note that the average traffic load is negligible for the first 30 seconds of the experiment compared to high values in the second part. In fact, nodes spent 30 seconds running only the routing protocol to stabilize the network topology.

In this phase the AODV curve is the lowest one since AODV only sends Hello packets to discover 1-hop neighbors, while OLSR measures higher values even though they are negligible. In this phase, in case of CrossROAD the routing traffic coincides with the original protocol (OLSR) since the p2p system is not yet active.

On the other hand, from 30 to 90 seconds, the average throughput increases. In this phase nodes exchanged the information required to build the overlay from scratch. In case of Pastry, each node needs to bootstrap from another node already in the overlay, thus generating peacks of about 6KBps.

In case of CrossROAD, the traffic load is about 60% higher than the legacy OLSR (see Fig.12.8),

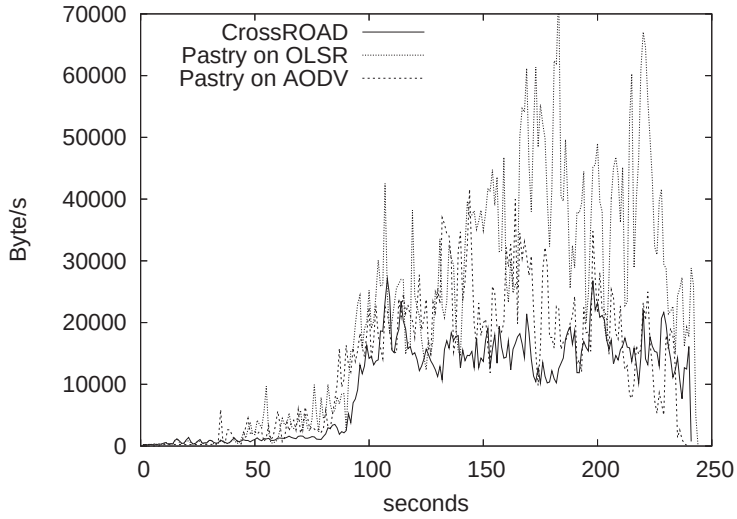



Figure 12.6.: Average aggregate network load

since nodes started running the system and they sent and received services information inside routing packets. After this phase, the overhead of the enhanced routing protocol approaches to the same values of OLSR, considering the periodical sending of service information on the network with the same frequency of Hello packets. For this reason, in the rest of the experiment CrossROAD overhead corresponds to the data traffic load introduced by the application, since the overlay management cost is almost negligible (see Fig.12.9). Instead, in case of Pastry, the overlay management is much higher than the data traffic load, identified by CrossROAD in Fig.12.6.

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 12.10 12.11 12.12 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 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.

In Table 12.1 different colours correspond to different overlays formed during the experiment. 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.

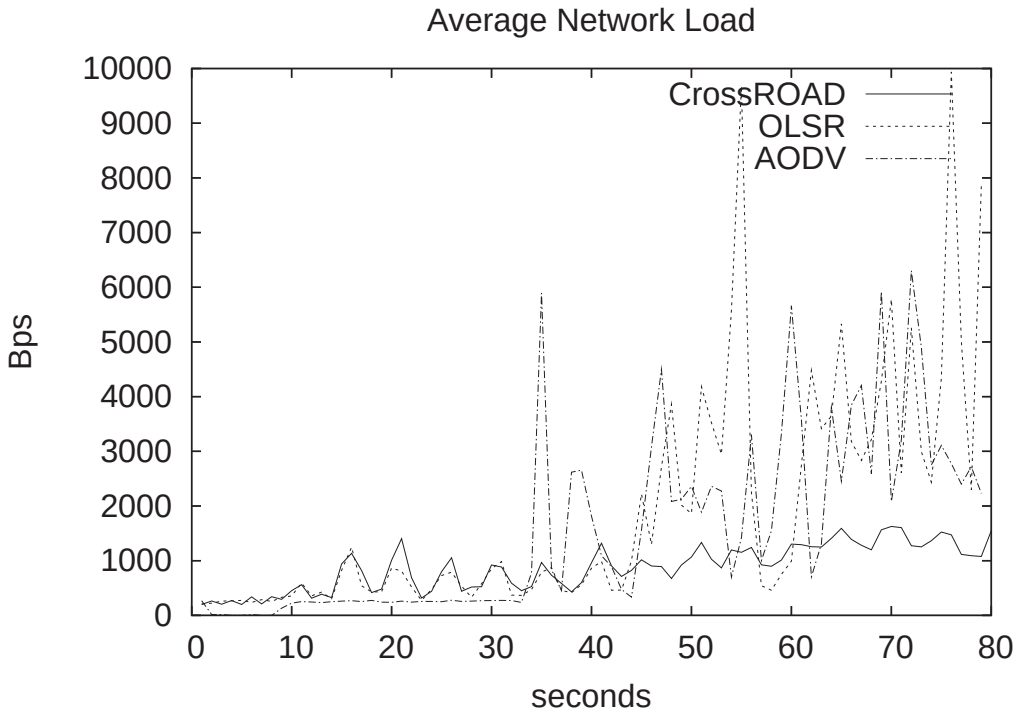



Figure 12.7.: Average aggregate network load, the first phase

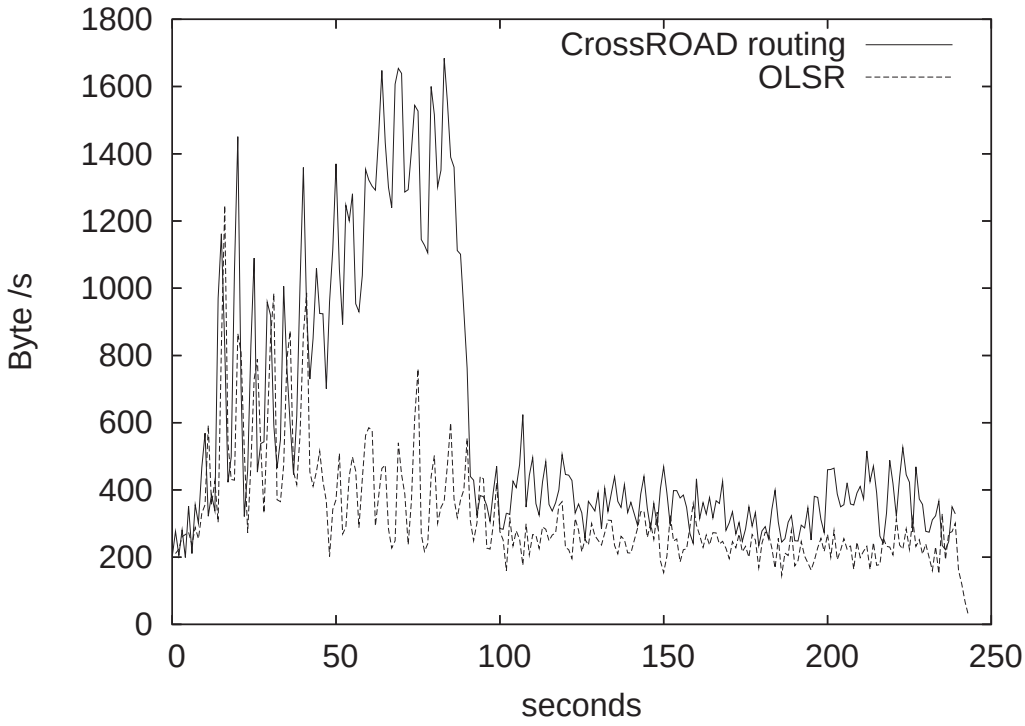



Figure 12.8.: Average network load

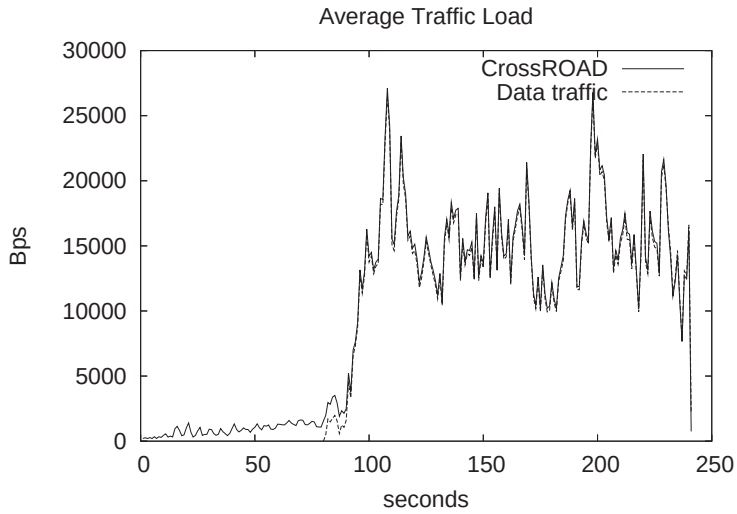



Figure 12.9.: Routing traffic load

| Nodes | CrossROAD (Bps) | Pastry on OLSR (Bps) | Pastry on AODV (Bps) |
|---|---|---|---|
| A | 7864.83 | 446.02 | 9818.63 |
| B | 1430.44 | 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 | 102.910.1 |
| 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 12.1.: CrossROAD delays for the overlay construction.

| Percentiles | CrossROAD | Pastry on OLSR | Pastry on AODV |
|---|---|---|---|
| $0,6$ | $599msec$ | $11.171sec$ | $9.138sec$ |
| $0,7$ | $2.306msec$ | $20.032sec$ | $16.055sec$ |
| $0,8$ | $4.692sec$ | $34.846sec$ | $28.823sec$ |
| $0,9$ | $10.648sec$ | $46.340sec$ | $75.475sec$ |
| $0,95$ | $23.025sec$ | $61.858sec$ | $88.701sec$ |
| $0,99$ | $60.468sec$ | $111.560sec$ | $105.649sec$ |

Table 12.2.: Delays distribution

In case of CrossROAD, the distribution of the average traffic load of single nodes mainly depends on the location of nodes in the network topology, since most connected nodes, that are also the 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 built an independent overlay, without other participants, measured a very low traffic load (e.g. node A in OLSR measured 446 Bps, while node G in AODV measured 550.56 Bps). In these cases the measured traffic load 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 the local node identifier is the nearest to each random key.

In addition, nodes that originated an overlay of two or three participants measured about the same amount of traffic either in case of AODV and OLSR. For example, nodes K and J in the OLSR experiment measured about the same traffic of nodes A and B in the AODV experiment. On the other hand, highest values of traffic were measured by nodes participating in the original overlay.

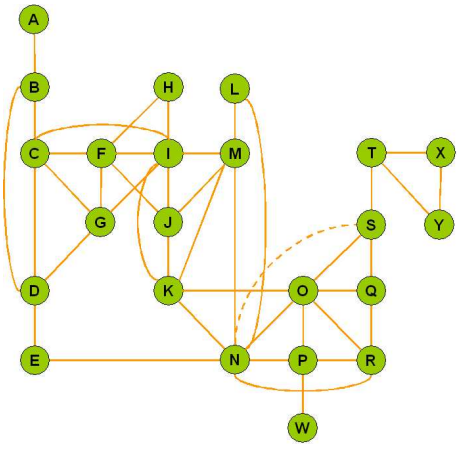



Figure 12.10.: CrossROAD overlay

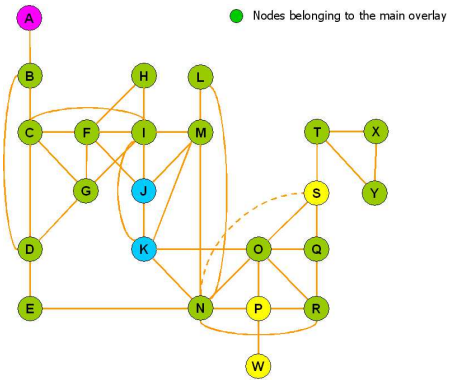



Figure 12.11.: Pastry on OLSR overlays

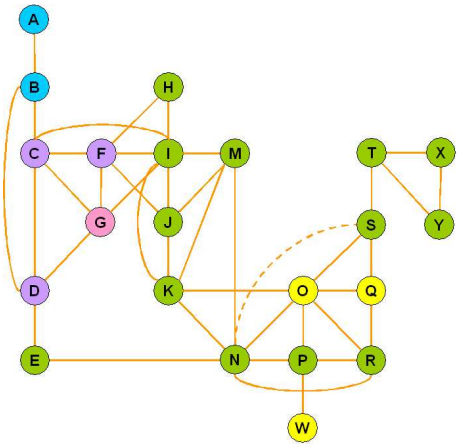



Figure 12.12.: Pastry on AODV overlays

## 12.4. Delays Analysis

Since the number of nodes in the original overlay in case of OLSR is greater than that in case of AODV, the average traffic load of Pastry on OLSR is higher than that of Pastry on AODV. In fact, values shown in Fig.12.6 represents the average on all nodes of the network, independently from the number of overlays.

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.

The delay distribution and related percentiles, shown respectively in Fig.12.13 and Table 12.2, highlight that delays reach the order of 100 seconds in case of Pastry and 60 seconds in case

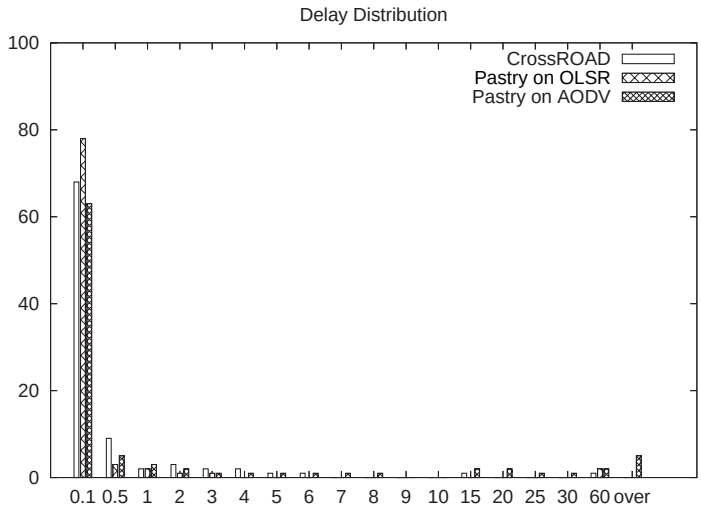



Figure 12.13.: Delay Distribution

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 by nodes of the main overlay were considered (i.e. nodes that correctly executed the bootstrap procedure, joining the right overlay). In fact, in case of smaller overlays, delays are reduced to few milliseconds because few packets have to be managed by the involved nodes. In addition, since the network topology is redundant and, in some cases, there are several unstable links, the distribution of data packets through TCP connections suffers many retransmissions, increasing the related timeout and transmission delay. Finally, in case of Pastry, processing data packets and concurrently managing overlay data structures further affects system performance.

## 12.5. 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 considerably influences the application behavior. We analysed results from experiment number 5, where, referring to the topology graph shown in Fig.12.5, all nodes, except for N and M, started running Unik-OLSR and XLplugin creating two different ad hoc networks. Thus, the initial network topology consisted of two ad hoc networks (i.e., nodes A, B, C, D, E, F, G, H, I, J, and K form MANET1, while nodes O, P, W, Q, R, S, T, X, Y form MANET2).

After the join of central nodes, the two networks merged in a single one, as the two overlays did. Note that Pastry is never able to merge two distinct overlays into a single one.

To better observe the system behavior, during the experiment nodes A and Y generated periodical

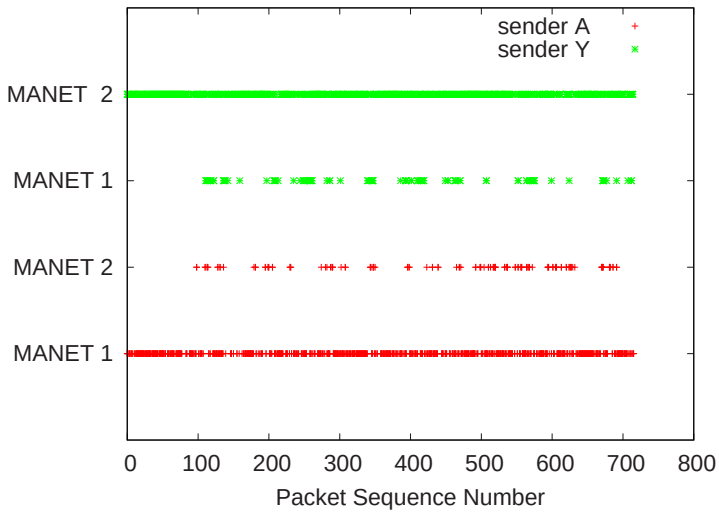



Figure 12.14.: CrossROAD data Distribution

messages with random keys. For this reason messages were originally sent to random destination inside the network area of the sender (i.e. node A sent messages to nodes of MANET 1, while node Y sent messages to nodes of MANET 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 in Fig.11.6, 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. This demonstrates the effectiveness of the cross-layer approach in a MANET, where supporting mobility and possible partitions should represent one of the main characteristics of network protocols.

## 12.6. 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. 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 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 system performance. To this aim, in the following chapter some application developed both on top of Pastry and CrossROAD are anayzed.

# 13. Distributed Applications on CrossROAD

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. However, starting from existent p2p systems developed for Internet, the large number of distributed applications developed on top of them can represent a great inheritance also for MANETs and a good incentive for users to adopt this technology in the daily use. In addition, there are several applications that originally do not require the presence of a p2p substrate, but that can exploit cross-layer optimizations of CrossROAD to increase their performance on MANETs.

Traditionally, MANETs have been considered as stand-alone networks, but extending the use of applications originally developed for wired networks, they can appear as flexible and low-cost extensions of them. In addition a new class of networks is emerging as a mix of fixed and mobile nodes interconnected via heterogeneous links (wireless and wired). This class has been called *Mesh Networks* [BCG05].

For this reason the interaction of Internet and MANET applications must be supported to guarantee the service extension and a global communication between users.

In this chapter we give some examples of existent applications that have been ported on MANET in their original definition to evaluate their performance and subsequently highligth advantages gaining from the cross-layer architecture, and specifically from CrossROAD.

## 13.1. A Group-Communication application: the Whiteboard

* Whiteboard application (WB) implements a distributed whiteboard among users. It allows users to share drawings, messages, and other dinamically generated content of interest. Such a self-organizing distributed application can be naturally supported by p2p systems.

Fig.13.1 shows a snapshot of WB running on two nodes. It provides the basic functionality of subscribe of an arbitrary topic, drawing or writing something related to the topic on a canvas, and sharing images with other members of the same topic. WB exploits an overlay p2p multicast algorithm as the engine to publish canvas changes to topic members. Scribe [CJK+03] is chosen as the multicast algorithm, since it performs better than other standard solutions (e.g. CAN-Flooding [CJK+03]). We use Pastry as the overlay substrate since Scribe has been developed on top of it.

It is woth pointing out that Pastry, scribe and WB are very loosely coupled to each other, thanks to a modular approach, and well-defined interfaces. It is therefore easy to substitute either Pastry or

*This work has been developed in the framework of the IST-FET MobileMAN project [mob] in collaboration with the Computer Laboratory of the University of Cambridge [cam].

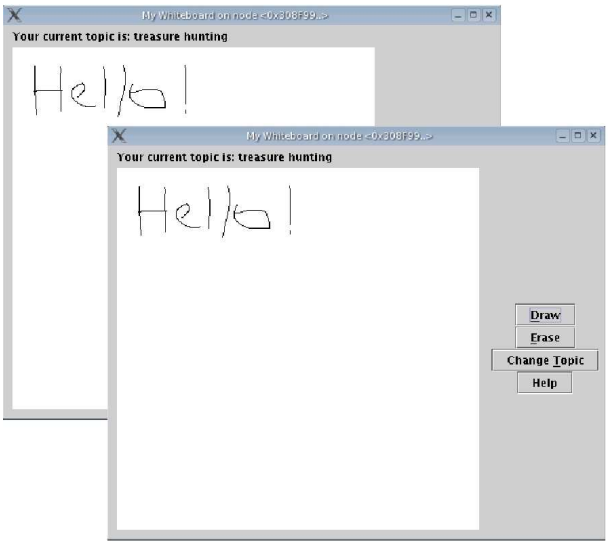



Figure 13.1.: Two nodes running WB

Scribe with their optimized solutions for MANETs since they implement the P2P CommonAPI and hence also the cross-layer enhancement.

WB is a very simple example of group-communication applications, which can be seen as a reference scenario for ad hoc networks. It is thus interesting investigating how such an application performs in a real ad hoc testbed.

### 13.1.1. Scribe Overview

Scribe is a subject-based, reverse-path forwarding, multicast algorithm implemented on top of an overlay network. For each WB topic a multicast group is defined at the Scribe level. Nodes subscribing for a topic (at the WB level) actually join the corresponding multicast group (at the Scribe level).

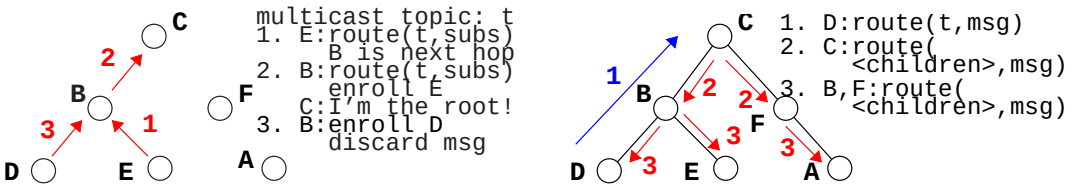
Scribe builds, on top of the overlay, a shared tree connecting the nodes subscribed to the same group. Each node may act both as sender and receiver of the group. Scribe simplifies the join operations and the multicast tree maintenance in respect of traditional network-level multicast protocols. Specifically, it exploits the subject-based nature of the underlaying overlay network to reduce the maintenance traffic (e.g., flood&prune messages or rendez-vous points dissemination are not required).

Scribe has been proved to scale well with large groups of nodes [CJK+03]. Finally, this protocol offers simple APIs to its applications, such as `create(topicID)`, `subscribe(topicID)`, `unsubscribe(topicID)` and `publish(topicID)`. Main operations of Scribe are described below.

- ✧ *Joining of a multicast group and content publishing*
  Scribe defines a rendez-vous point in the tree. Since it exploits the proximity logic of Pastry, the rendez-vous point of the group is the node numerically closest to the topicID. This is a

subject-based way of defining RV points, instead of a topology-based way, as in traditional network-level multicast. Therefore, the RV point can be reached by the overlay routing, and no mechanism for publishing the RV point(s) is required, introducing a low overhead on the system. In order to join the group, the Scribe module at the joining node sends through Pastry a join message to the rendez-vous point (specifying thus the topicID as the key of the message). The first node in the Pastry path towards the rendez-vous point gets the join message and delivers it to its Scribe module. If it does not recognize the topic since it is not yet registered to it, it discards the join message, creates a new children list for that topic, and adds the sender of the request to this list. Then, it tries to join itself to the group by generating a new join message. When the join message reaches a node that is already member of the multicast group (possibly, the RV point), it simply adds the sender of the join message to its children list, and the message is not further propagated. Since each node maintains a child list, the join message has just to travel up to the first branching point in the multicast tree (i.e., the first node in the path that is already registered to the topic). Thus the joining procedure scales well with the number of nodes.

Application messages are sent on the overlay with key equal to the topic ID. Hence, the reach the node logically closest to it, that represents the root of the shared tree, and it forwards the message to its children, which futher forward to its children, and so on. Figures 13.2 a) and b) show an example of building the tree and disseminating application messages.




(a) (b)

Figure 13.2.: Scribe building the tree (a), and disseminating messages (b).

✧ *Shared tree maintenance*
Scribe manages the tree as follows. Each parent periodically sends a *HeartBeat* message to each child. If a child does not receive any message from the parent for a given time interval (20 seconds is the default value), it assumes that the parent is disconnected from the tree and re-execute the join procedure. A publish message received by the parent implicitly acts as a HeartBeat. This procedure allows nodes to discover parent failures and re-join the tree.

### 13.1.2. Experimental Environment

In order to evaluate the application performance on a real MANET, a small-scale testbed has been set up in the CNR campus in Pisa. In a first phase we analyzed static scenarios highlighting limitations that originate from Pastry and Scribe design, rather than to mobility. The same sets

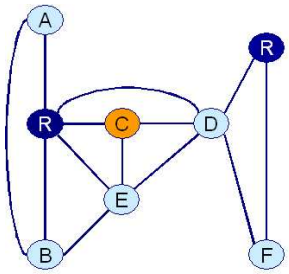



Figure 13.3.: Network Topology

of experiments were then repeated using CrossROAD as p2p substrate to point out cross-layer advantages.

As in previous small-scale testbeds, we set up a 8-nodes ad hoc network reproducing the network topology shown in FIg.13.3.

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.

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 alternates 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. 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 stops the application.

Burst sizes and idle phase lengths are sampled from exponentially distributed random variables. The average length of idle phases is 10 seconds, and it 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 account, each agent draws – on average – at least 50 strokes during an experiment. For the performance evaluation this represents a good trade-off between the experiment duration and the result accuracy, as is shown by following results.

Some final remarks should be pointed out about the experiment start-up phase. Nodes are synchronized at the beginning of each experiment. Since Pastry requires a bootstrap sequence to join the overlay, the same schedule was used also for CrossROAD, even though it does not require it. Specifically, 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 bootraps from E. Node A starts 5 seconds after B and bootraps from B. Finally, node F starts 5 seconds after D and bootraps from D.

After this point in time, the Scribe tree is created and, finally, WB instances start sending appli-

cation messages (herafter, WB messages). In this way, the Scribe tree is built when the overlay network is already stable, and WB starts sending messages when the Scribe tree is completely built.

To evaluate if the application provides an adequate Quality of Service to users in a MANET environment we used 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 $pl_i = 1 - \frac{R_i}{\sum_i S_i}$.

*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, aimed at quantifying 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 $NS_i = \frac{\sum_j n_j t_{ij}}{\sum_j t_{ij}}$.

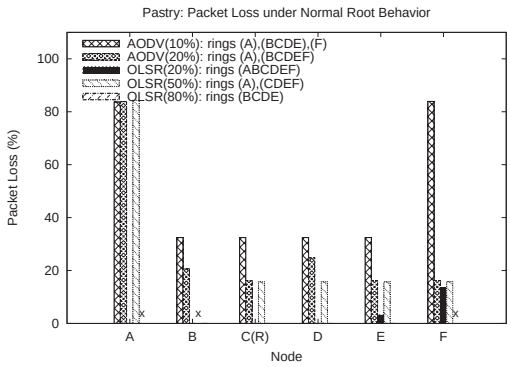
*Re-subscriptions:* for each node, we count the number of times (during an experiment) this node sends new subscriptions requests, because it can't communicate with the previous parent anymore.

### 13.1.3. Performance with Pastry

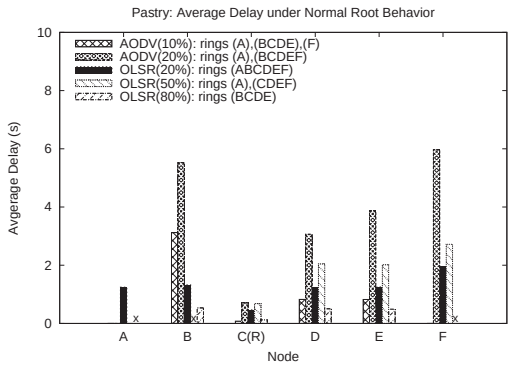
The results we report in this section are obtained by using Pastry as DHT, and either OLSR or AODV as routing protocol. Experiments are run by increasing the traffic load starting from 20% up to 80%.

Before presenting the results in detail, let us define what herafter will be referred to as "crash of the Scribe Root Node". In our configuration Pastry assigns nodeIds 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 13.3. 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 must be 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




(a) Packet Loss w/o MSRN crash




(b) Delay w/o MSRN crash

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.

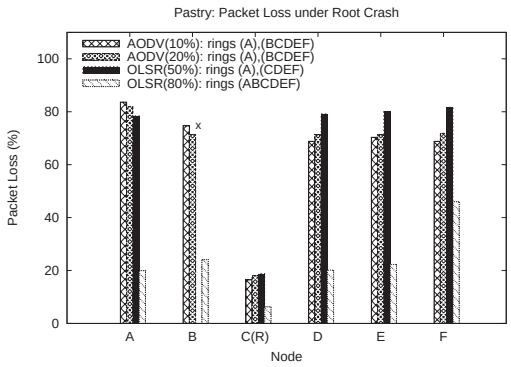
Figures 13.1.3 a) and b) show respectively the packet loss and the delay indices experienced by the WB nodes. We here consider experiments were there is no MSRN crash. Furthermore, we consider AODV experiments with 10% and 20% traffic load, and OLSR experiments with 20%, 50% and 80% traffic load, respectively. There is no point in running AODV experiments with higher traffic load, since performance when using AODV is quite bad even with such a light traffic load. In addition, 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). Finally, in the figure legend we also report the rings that Pastry builds during the bootstrap phase. Theoretically, just one ring should be built, encompassing all WB nodes, but it is not so easy in practice.

In fact, 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 during the bootstrap procedure 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. This is an important weakness of Pastry.

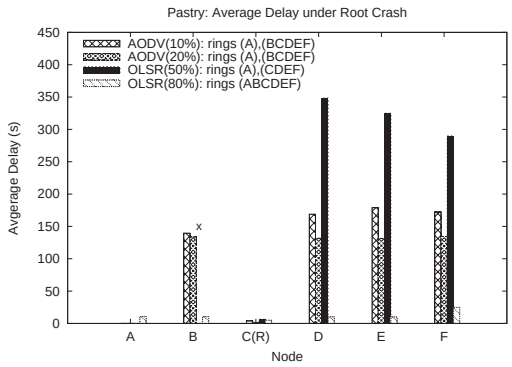
as is shown in fig.13.1.3a), in the "AODV 10%" experiment, nodes A and F are isolated, and create their own rings. This results in packet loss of about 80% at those nodes (i.e., they just get their own WB messages, which is about one sixth of the overall WB traffic), and about 32% at nodes B, C, D and 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 all to the same ring. As before, the packet loss measured by A is about 80%, and the packet loss at the other nodes due to A's isolation is about 18% (one sixth of the overall traffic) except for nodes B and D, that experience a little bit *higher* packet loss.

On the other hand, in the case "OLSR 20%" Pastry is able to correctly generate a single ring, and the packet loss is quite low (apart from node F). In the case "OLSR 80%" nodes A and F crash. However, the packet loss experienced by the other nodes is negligible.




(c) Packet Loss in case of MSRN crash




(d) Delay in case of MSRN crash

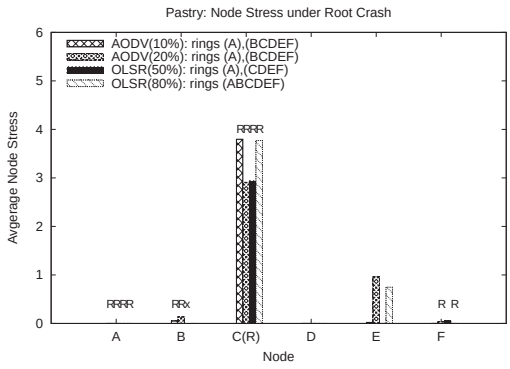
Similar observations can be drawn by focusing on the delay index (Fig.13.1.3b)). First of all, it should be pointed out that the delay measured by isolated nodes (e.g., A and F in the "AODV 10%" case) is obviously negligible. Even though the delay in this set of experiments is generally low, it can be noted that OLSR outperforms AODV. In addition MSRN (node C) always experiences a lower delay with respect to the other nodes in the same ring since all packets are first sent to it and then it forwards them to all the others.

Figures 13.1.3a) and b) show the packet loss and the delay indices in cases of MSRN crash. The packet loss experienced by nodes in the same type of experiment is much higher than in case of no crash of MSRN.

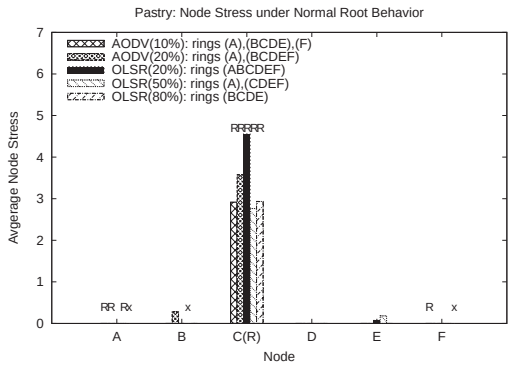
In the first three experiments, node A isolation causes a packet loss of about 18% at other nodes. Hence, MSRN crash is in charge of the remaining 60% packet loss. Quite surprisingly, OLSR with 80% traffic load shows better performance than OLSR with 50% traffic load. This can be also due to some temporary unstable links in case of 50% traffic load. However, generally we can note that the packet loss at MSRN is always lower than at other nodes in the same ring. This highlights that MSRN is not able to forward received WB messages over the Scribe tree.

Observing the worst cases in Fig. 13.1.3b), the delay experienced by nodes B, D, E and F reach high values in the order of *few* minutes, either by using AODV or OLSR.

To summarize, the above analysis allows us to draw the following conclusions. The Pastry bootstrap algorithm is too weak to work well in MANETs, and produces unrecoverable partitions of the overlay network. This behavior is generally exacerbated by AODV (compared to OLSR). Furthermore, MSRN is clearly a bottleneck for Scribe. MSRN may be unable to deliver WB messages also at 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 (i.e., 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 probability of WB users to be isolated from the overlay network makes the packet loss unacceptably high. These results suggest that Pastry and Scribe need to be highly improved




(e) Node stress in case of MSRN crash




(f) Node stress w/o MSRN crash

to support group communication applications such as WB in MANET environments.

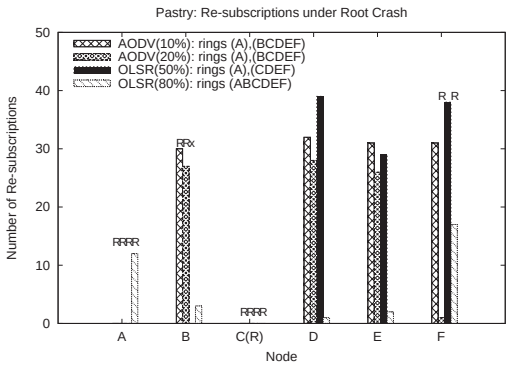
### 13.1.4. Multicast Tree Quality

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

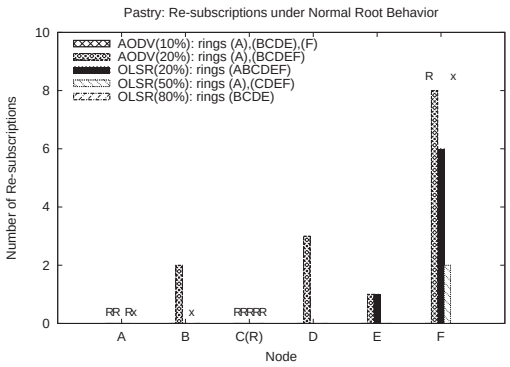
Figures 13.1.4a) and b) show the average node stress with and without MSRN crashes, respectively. In both cases, the node stress is significantly higher at MSRN than at every other node. This means that the Scribe tree is a one-level tree, and MSRN is the parent of *all* the other nodes. This behavior is expected, and can be explained by recalling the way Scribe works. In our small-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 the major limitation of the Scribe algorithm, and optimizations of the P2P system are clearly not sufficient to cope with it.

In Figures 13.1.4a) and b) we have added "R" labels to indicate nodes that occur to become Scribe Root during the corresponding experiment. When MSRN does not crash (Fig. 13.1.4b)) other nodes become Scribe root only as a side effect of a failed Pastry bootstrap. On an isolated WB node, Scribe builds a degenerate 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 Fig. 13.1.4a), it could be noticed that other nodes 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. A child node with id $n_1$ becomes root when it looses its previous parent and the Pastry routing table does not contain another node id $n_2$ such that $n_2$ is closer to the WB topic id than $n_1$. In addition in case of MSRN crash, the congestion at the Pastry level is so high that the Pastry routing table of some node becomes incomplete (i.e., MSRN disappears from other nodes' routing table). Thus, the Scribe




(g) Re-subscriptions in case of MSRN crash




(h) Re-subscriptions w/o MSRN crash

tree is partitioned in several isolated sub-trees. Clearly, this contributes to the high packet loss measured in these experiments.

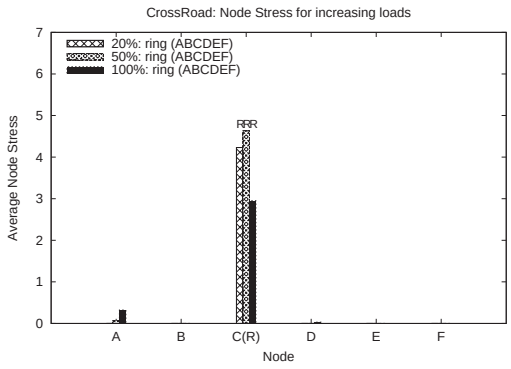
Another effect of Pastry congestion during MSRN crashes is a possible reshaping of the Scribe tree. Fig. 13.1.4a) shows that node E's average Node Stress is close to 1 in the "AODV 20%" and "OLSR 80%" cases. This means that MSRN disappears from the Pastry routing tables of some node, which – instead of becoming a new root – finds node E to be the closest node to the WB topic id. This phenomenon could be seen as beneficial, since it reduces the MSRN node stress. However, it stems from an incorrect view of the network at the Pastry level, originating from congestion.

Figures 13.1.4a) and b) show the re-subscription index for the same set of experiments. Figure 13.1.4b) shows that, when MSRN does not crash, the Scribe tree is quite stable. Most of the re-subscriptions occurs at node F, which is the "less connected" node in the network (see the network topology in Fig. 13.3). In these experiments, the performance in the AODV cases is worse than in OLSR cases. Furthermore, upon MSRN crashes (Fig. 13.1.4a)), the number of re-subscriptions increases dramatically, even at "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), thus increasing the number of re-subscriptions. It is interesting to point out that this is a typical positive-feedback cycle: the more MSRN is congested, the more re-subscriptions are sent, the more congestion is generated.

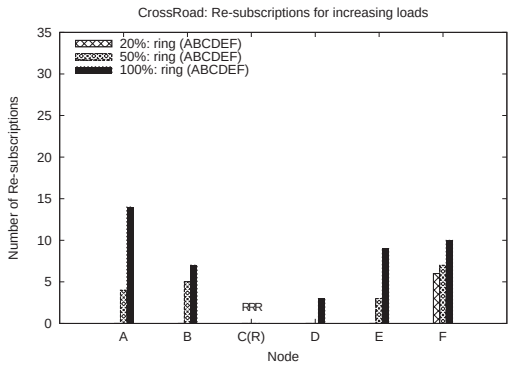
To summarize, the multicast tree generated by Scribe on top of Pastry is quite unstable, especially in cases of MSRN crashes. The tree may get partitioned in disjoint sub-trees, and many re-subscriptions are generated by all nodes. Furthermore, Scribe is not able to generate a well-balanced multicast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

### 13.1.5. CrossROAD Improvements

In this section we show that using CrossROAD is highly beneficial to the stability of the Scribe tree. In this experiments, we set the traffic load to 20%, 50% and 100%. We focus on the performance figures related to the quality of the multicast tree, i.e., the average node stress (Figure 13.1.5a))




(i) Node Stress with CrossROAD




(j) Re-subscriptions with CrossROAD

and the number of re-subscriptions (Figure 13.1.5b)). A complete evaluation of the User Satisfaction parameters, as well as further optimizations of the Scribe algorithm, are currently a work in progress.

The first main improvement achieved by using CrossROAD is that neither the overlay network nor the Scribe tree get 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 unique root of the Scribe tree. Therefore, CrossROAD is able to overcome all the partition problems experienced when Pastry is used.

Fig. 13.1.5a) 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 behavior is expected, since it stems from the Scribe algorithm, and cannot be modified by changing P2P system.

Finally, Fig. 13.1.5b) shows that the Scribe tree is more stable (i.e., requires less re-subscriptions) when CrossROAD is used instead of Pastry. To be fair, we have to compare Fig. 13.1.5b) with both Figures 13.1.4 a) and b). 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 Fig. 13.1.4b), 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 Pastry “no crash”. However, it should be noted that in the latter case the overlay network includes less nodes, and hence the congestion is lower. In addition, with the same nodes and traffic load in the overlay network, Pastry experiments suffer MSRN crashes (Fig. 13.1.4b)). 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 Fig. 13.1.4b), since the overlay network is the same in both experiments. CrossROAD achieves comparable performance, and at some nodes it outperforms Pastry, even though the application traffic is significantly higher.

### 13.1.6. 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.

Fig. 13.4 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[2]. The traffic load is sampled every 5 seconds. We take into consideration the traffic related to the whole network stack, from the routing level up to the application. For WB nodes (A and C), we thus include routing, transport, middleware (Pastry and Scribe) and application-level (WB) traffic. For "routing" nodes, we just include the OLSR traffic. We do not plot the network load related to other WB nodes, since it is qualitatively similar to that of node A.

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, the curves related to the two routers can hardly been distinguished in Figure 13.4. This means that the main load on WB nodes is related to Pastry, Scribe and the WB application.

Fig. 13.5 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 the two figures we can highlight that the network load of CrossROAD is much lower than the network load of Pastry. Specifically, while in the Pastry case the average load of C and A is 48.5 KBps and 16.5 KBps, respectively, in the CrossROAD case it drops to 21.1 KBps and 2.96 KBps, respectively. The reduction of the network load achieved by CrossROAD is 56% at node C and 82% at node A. Since the other stack components are exactly the same, CrossROAD is responsible for this reduction[3]. Furthermore, note 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.

Performance results previously shown, demonstrated that Pastry and Scribe are not 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 subnetworks, 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 optimized P2P system such as CrossROAD. Thanks to the cross-layer interactions CrossROAD is able to avoid all the partitioning problems experienced with Pastry.

However, CrossROAD cannot solve the problem of bottlenecks in the Scribe trees. Therefore, optimized version of Scribe are required for group communication applications such as WB to be really deployed on MANETs. There are mainly two ideas. The first one is related to existent

[2]We do not take into account AODV experiments, since OLSR has clearly shown to outperform AODV.
[3]The actual reduction is even higher, since the application-level traffic is 100% in the CrossROAD case.

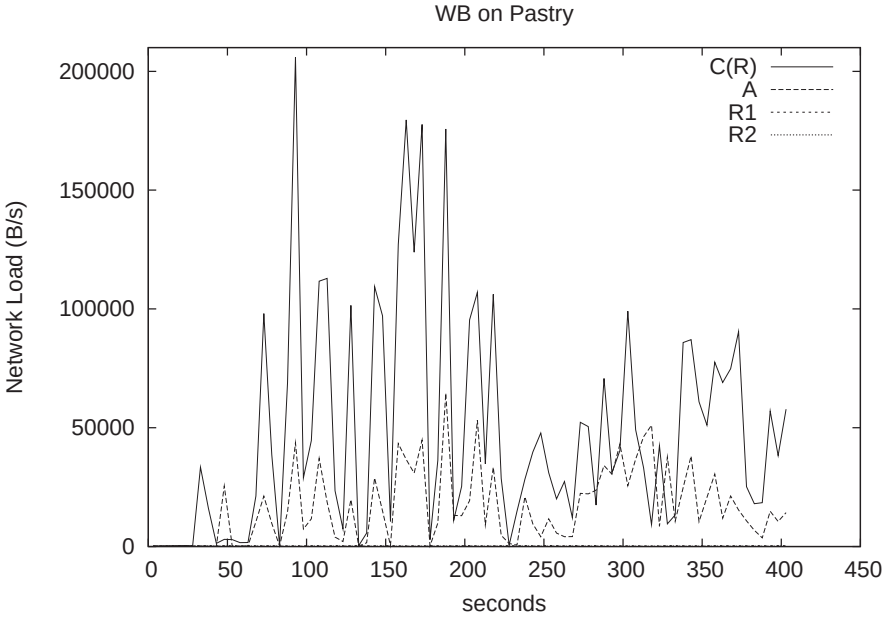



Figure 13.4.: Network Load with Pastry

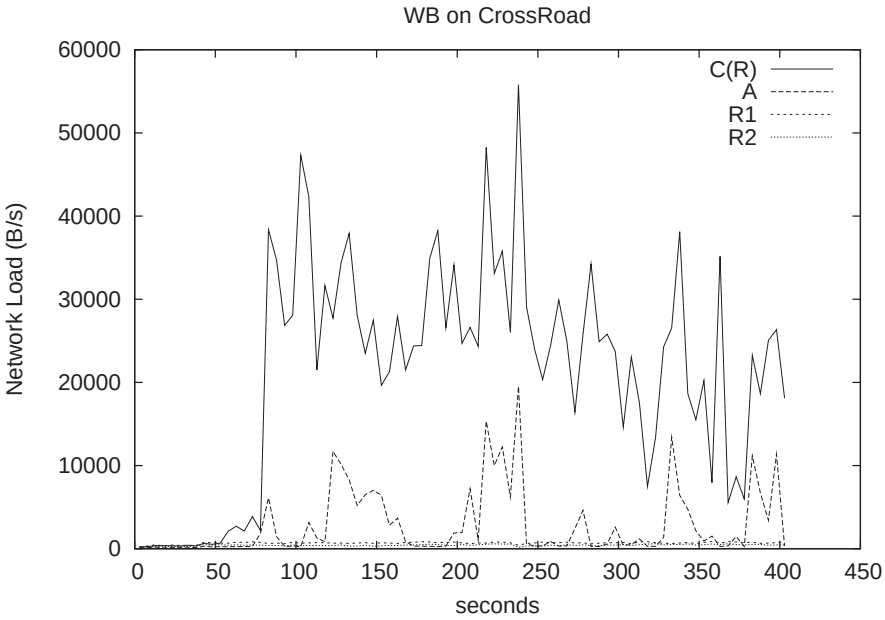



Figure 13.5.: Network Load with CrossROAD

multicast algorithms for MANETs independently of a p2p substrate, directly exploiting a cross-layer interaction between the application-level multicast and the proactive routing protocol.

The other approach is to allow the multicast algorithm to exploit cross-layer information through CrossROAD. In this case, Scribe could be enhanced recovering from CrossROAD the state of the overlay and the current network topology. However, in this case the information related to the service identifier needed to associate to each node the service it currently provides, is not sufficient. Each node has to publish also the list of topics it is interested in. In this way a shared tree is not necessary since each node has a complete knowledge of the overlay participants and it can autonomously maintain its multicast group selecting from CrossROAD the list of nodes currently interested in a specific topic. Thus, each node becomes "root" of its own messages, and it can directly forward them to all the others. The application becomes thus completely self-organising and distributed on all nodes of the overlay, avoiding bottlenecks of the original Scribe.

These cross-layer enhancements to multicast algorithms for MANETs are currently the subject of our research.

## 13.2. Content Sharing: The UDDI approach

[§] Service Discovery protocols represent an important set of applications for MANETs, where nodes are mobile and require to recover information on current available services.

Originally, the web services community has defined the UDDI (Universal Description, Discovery, and Integration [udd]) as a standard protocol to publish and discover information about web services on Internet with a centralized service management. The same approach has been followed also in ad hoc networks, but in this case resource costraints and mobility cannot guarantee a constant presence of nodes in the network to allow the use of centralized servers. For this reason a new service discovery protocol for MANETs, called *UDDI for manets (UDDI4m)* has been defined in the framework of the MobileMAN project. It inherits the basic principles of UDDI but it is enhanced with a cross-layer p2p substrate (CrossROAD) to optimize its behavior.

UDDI4m nodes are divided into two categories, characterized by different features:

- ✧ *UDDI4m clients* publish and recover services information, but they do not store related content;
- ✧ *UDDI4m servers* store information about published services in a local data structure and they can also assume the role of client to publish and require other services information.

One of the main hypothesis of the standard UDDI is represented by the knowledge of each node of a set of servers where publishing and recovering services information. In ad hoc networks, where the network topology is highly dynamic and nodes interactions are temporary, it is not possible to have a list of server nodes surely available. For this reason, the presence of an overlay network can

[§]This work has been developed in the framework of the IST-FET MobileMAN project [mob] in collaboration with NetiKos [net].

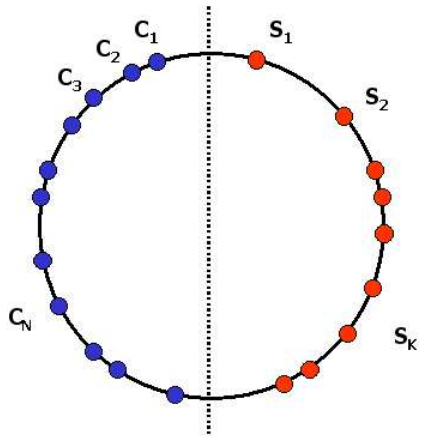



Figure 13.6.: UDDI4m nodes on CrossROAD overlay

optimize UDDI4m performance. In this way, each node has at least a partial knowledge of nodes taking part in the same service, and a policy to establish a possible destination for each request is defined.

However, in the overlay network all nodes have the same characteristics, physical constraints are not specified and they are uniquely identified by a logical address. This information is used by the subject-based routing to distribute data to the nodes that one logically closest to the specified key values. Each node is thus a possible destination for data that has to be distributed on the network.

In case of UDDI4m, client nodes are not able to receive a publish request from another node because they cannot store the related information. For this reason, all requests that involve data storage cannot be delivered to a client node, but they must be delivered to server nodes.

To solve this problem, also at the middleware layer a node categorization is needed. Since CrossROAD overlay network is represented by a 160-bit circular address space, it is possible to divide this logical space into two parts applying a mask to logical identifiers in order to distinguish server nodes from client nodes. This represents an enhancement to standard features of overlay network that can enormously optimize UDDI performance on ad hoc networks.

In Fig.13.6 an example of overlay network consisting of several client and server nodes is shown. In order to obtain this kind of configuration, it is necessary that a UDDI4m application, running on a particular node, specifies whether that node assumes the role of client or server. In this way, when the CrossROAD instance is created, it knows how to compute the local node identifier and which types of requests can be accepted from that node. Specifically, CrossROAD must implement the subject-based routing applying the same mask of the server nodes also on key values used to distribute data on the network. All requests of storage or recovery of services information are thus forwarded only to one of the server nodes, the closest one to the “masked” key value.

For this reason, also the definition of key values for data distribution is an important feature of UDDI4m. Since UDDI4m requests are mainly represented by publication and recovery of services information, the definition of services categories is necessary to exploit the subject-based policy of a structured overlay network. Hence, we define the key value as the type of service to be stored in the nodes databases or to be retrieved.

In this way, each server is in charge of maintaining information related to a specific service category, that one identified by the logical identifier numerically closest to its own. In addition, in order to guarantee the system reliability in case of nodes failures, a policy to store replica information on other server nodes has to be defined, and it is currently a work in progress.

We can suppose to define a set of macro-services as Content Sharing, Chat, Video-Conference, Devices, E-Commerce, Mail Server, Forecast Information, and others, foreseeing additional specializations in sub-services (e.g. a Content Sharing service can be divided in mp3, movies, games, documentation, and others). In this way, each UDDI4m node has not to know the server list a priori, but it has only to implement the XL-CommonAPI in order to exploit CrossROAD advantages. When a client node decides to publish a service, it specifies the category of the service as key value of the CrossROAD message, and the same for a request of update or recovery of services information. Then, CrossROAD selects the best destination for each message checking every time the consistency of its internal data structures with the current state of the network topology. Each message is sent to a destination currently available on the network, avoiding all attempts to connect to all server nodes looking for the one maintaining the required information, as in the standard implementation of UDDI.

In this way, this application is highly optimized and allows the interaction of etherogeneous devices, with different resource constraints.

# 14. Conclusions

Ad hoc networks are distributed systems composed of self-organized wireless nodes. As these systems cannot benefit from any centralized infrastructure, networking features, like packet forwarding, routing and network management, as well as application services, should be distributed among users devices. The distributed nature of ad hoc networking finds in p2p interactions 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 feasible on top of these overlays, suits also ad hoc scenarios. Thus, having them working in ad hoc environments would be an advantage for the MANET technology. However, it was not clear how to adapt these overlays to ad hoc networks, and how they perform on them.

For this reason, a preliminary experimental study has been conducted to evaluate Pastry performance on ad hoc networks and, first of all, two main routing protocols that must support the entire system.

Main results highlighted that the proactive protocol (OLSR) introduces a higher traffic load than the reactive protocol (AODV) but the percentage of this traffic is negligible compared to the 802.11b available bandwidth. In addition AODV measures higher delays and packet losses to deliver application packets since it exploits also unidirectional links. In this way it makes the system unreliable in case of mobility and network topology changes. On the other hand, OLSR guarantess a high responsiveness to topology changes through its proactive flooding lightly increasing the network overhead.

At the middleware layer, we experienced that Pastry, carrying out the overlay management through a high number of remote connections, introduces a heavy overhead on ad hoc networks, reducing the overall system performance. In addition, possible failures during the bootstrap procedure of Pastry generate different overlays that cannot rejoin subsequently. In this way, nodes belonging to different overlays are unable to communicate to each other, even though they run the same service.

In order to improve the system, each node should become aware of the network topology, maintaing a local correspondence between logical and physical address spaces.

[CDHR] defines a reorganization of Pastry's overlay exploiting the network proximity to improve application performance and the network usage. This solution is based on an additional location discovery protocol to recover physical distances between nodes, increasing the total overhead.

On the other hand, a cross-layer architecture, defining interactions between protocols belonging to different layers, allows us to exploit additional information to optimize the overlay management. From these principles the idea of a novel cross-layer p2p system for MANETs arises.

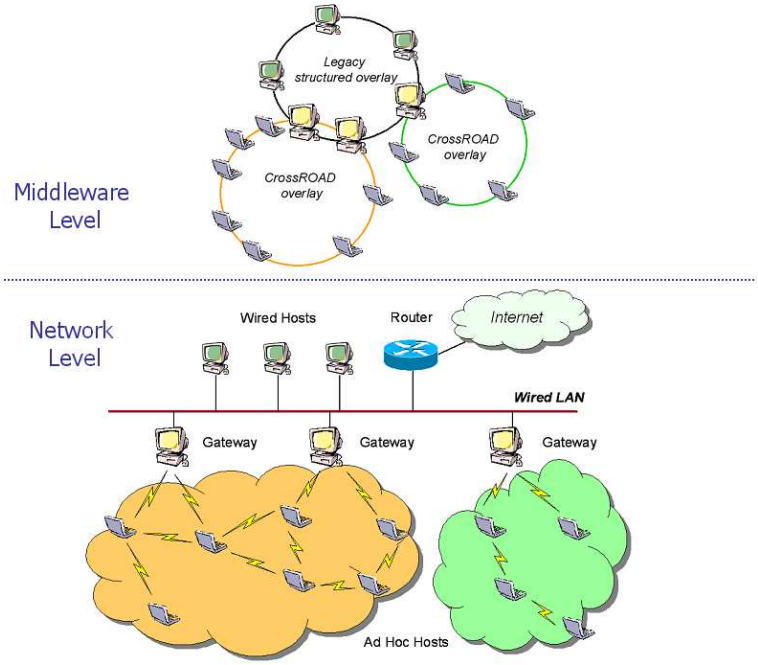



Figure 14.1.: Integration of mobile and fixed p2p systems.

CrossROAD exploits the cross-layer architecture to directly interact with a proactive routing protocol, which guarantees a complete knowledge of the network topology. Services information are added to the proactive flooding of the routing protocol developing a cross-layer service discovery protocol to allow nodes to associate topology information to provided services, lightly increasing the network overhead. In this way CrossROAD collects routing information optimizing the overlay construction and management through simple accesses to shared information.

The entire system has been developed and tested on small-medium scale testbeds. Main results highlithed that:

- ✧ the use of a proactive routing protocol does not penalize the system performance in terms of overhead, but it performs better than the reactive in terms of packet delivery ratio and reconfiguration delays in static and low mobility scenarios;
- ✧ CrossROAD drastically reduces the overhead introduced by a classical p2p system on ad hoc networks, and correctly manages cases of network partitioning and topology changes;
- ✧ several distributed application already developed for Internet have been successfully ported on CrossROAD, increasing their performance, and paving the way to new algorithms for data distribution and recovery (e.g. a cross-layer multicast protocol on top of the overlay network to develop group-communication applications).

In addition, the standardization of a Common API for structured overlay networks enhanced with cross-layer interactions in case of ad hoc networks, represents a point of contact between fixed and mobile p2p systems. Specifically, the use of services and applications already developed for legacy systems on MANETs can be a great incentive for users to interact with this new technology. For

this reason we cannot maintain this two worlds completely separated, but an effort to integrate them is needed.

We are currently designing a solution for the interaction between the CrossROAD "mobile" overlay and Pastry, Chord, CAN and others, running on wired hosts. As shown in Fig.14.1, from the network level, ad hoc nodes can be connected with wired hosts through one or more nodes that act as gateways for the routing protocol. In this way also nodes belonging to different MANETs connected to gateways can communicate. Specifically, a gateway for OLSR has been already developed and tested [ABC+05].

At the same time, supposing that ad hoc and wired hosts run the same application on a structured p2p system, each gateway could also represent an intersection point between different overlays. The gateway should have a wireless interface for the ad hoc network and a wired interface to forward packets to the wired network or to other ad hoc networks. To implement a fusion between the CrossROAD overlay and, for example, a Pastry ring on the Internet, the gateway has to run the two systems concurrently, and a particular service on top of them that elaborates messages, content, and decides on which ring each message should be forwarded. Instead, distributed applications must only implement the XL-CommonAPI on top of the selected p2p system.

A recent work [RGK+05] proposes a system (OpenDHT) that represents a common platform for several structured overlay networks on Internet. Specifically this new solution centralizes the execution of the p2p system on a set of servers, lightening client nodes. In this case the definition of the commonAPI is reduced to a simple PUT/GET api, and client nodes can interact with the overlay only through put/get requests to a specific server. Then it elaborates the related information forwarding it to the best destination.

The main idea of integrating mobile and fixed p2p systems can be applied also in this case, specifying on the gateway the translation from the XL-commonAPI requests to the PUT/GET interface of the OpenDHT system.

Nowadays, multi-hop ad hoc networks do not appear as isolated self-configured networks, but they emerge as flexible and low-cost extension of wired networks. This new paradigm is known as *Mesh Networks* [BCG05]. To this aim, the integration of mobile and fixed p2p systems is necessary to broaden the use of distributed applications on MANETs and to turn them into a commodity for all users, increasing their interest towards these new technologies.

# A. XL-CommonAPI: the complete specification

Several structured p2p systems exist in literature and many applications have been developed on top of them. To use these applications on top of different overlays without changing their implementations, a common API was proposed in [DZD+03]. However, since that specification is meagre, current implementations of structured p2p systems (e.g. Pastry, Chord, CAN and others) have extended this version reducing the portability of applications. In addition, in mobile environments, the possibility to exploit cross-layer interactions considerably improves overall performace providing cross-layer information also to upper-layer applications. In this way also their behaviour is further optimized. To define a common tool for fixed and mobile p2p systems that includes all these additional features, we report hereafter the detailed specification of the XL-CommonAPI.

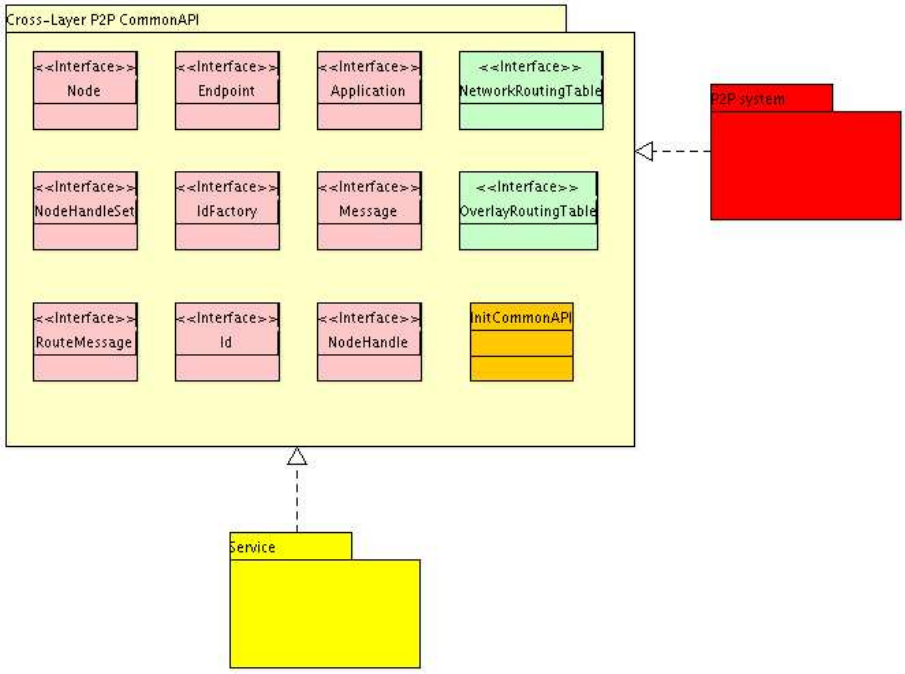



Figure A.1.: Packages Diagram: XL-CommonAPI and its interactions with p2p systems and applications.

Each application running on top of a structured p2p system can exploit a set of basic features that are represented by the assignment of logical identifiers and the subject-based routing. This was defined in [DZD+03] as "key-based routing API (KBR)". We maintain the same definition for consistency with the previous specification. Further features developed to organize particular applications like multicast (Scribe [CJK+03]) or Decentralized Object Location and Routing (DOLR) [HKRZ02] can be directly implemented on top of KBR as a simple distributed application. These

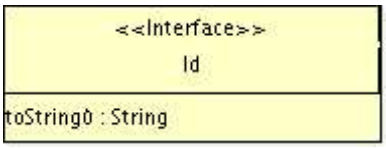



Figure A.2.: Class Diagram: Id interface

services export another interface to their upper layer, where more specialized applications can be developed.

What we have defined as XL-CommonAPI represents an evolution of the KBR adding to the subject-based routing some cross-layer features that can optimize both p2p systems and applications developed on top of them. This interface appears to the applications' programmer as a set of java interfaces. The p2p system must implement all interfaces except the *Application* interface, which is implemented by each specific service. It defines main methods used by the service to manage a received message, in case forward it, and eventually notify a change in the overlay data structures. Specifically, Fig.A.1 shows the UML packages diagram including XL-CommonAPI, a generic p2p system and a service. In particular, the p2p system has to define the InitCommonAPI class that represents the initialization of all its internal data structures and that returns references to related interfaces of the XL-CommonAPI. The implementation of the InitCommonAPI class has to be given by the p2p system, while the other interfaces have a standard definition as detailed below.

*Id*: data abstraction of the logical identifier (Fig.A.2). In case of CrossROAD, it is implemented as a 160-bit value obtained from the SHA-1 hash function applied to the IP address of the local host. In Pastry a random quantity is added to the same value.

- ✧ `String toString();` returns a String that represents the related Id in hexadecimal format.

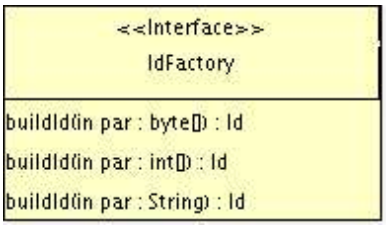



Figure A.3.: Class Diagram: IdFactory interface

*IdFactory*: abstraction of the class that implements the arithmetic computation of the logical Id starting from an array of `byte`, an array of `int` values, or a `String` (Fig.A.3). The p2p system must choose and implement the hash function to be applied to input values.

- ✧ `Id buildId(byte[] b);` returns an Id as the hash function applied to the specified array of `byte`.
- ✧ `Id buildId(int[] i);` returns an Id as the hash function applied to the specified array of `int`.

- ✧ `Id buildId(String s);` returns an Id as the hash function applied to the specified `String`.

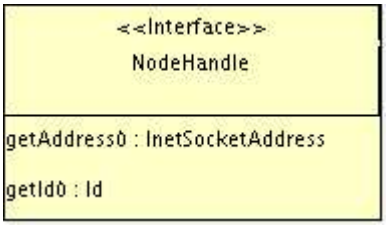



Figure A.4.: Class Diagram: NodeHandle interface

*NodeHandle*: abstraction of the pair of logical and physical identifiers associated to a specific host (Id, IP address and port where the specific service is provided) (Fig.A.4).

- ✧ `InetSocketAddress getAddress();` returns the IP address and port of the related node.
- ✧ `Id getId();` returns the Id of the related node.


Figure A.5.: Class Diagram: Message interface

*Message*: abstraction of each message that has to be sent on the network through the overlay (Fig.A.5). The data structure that represents an application message must implement this interface, that only represents a Serializable java object. No functions are specified. It is defined as common data type for the application and the p2p system.


Figure A.6.: Class Diagram: RouteMessage interface

*RouteMessage*: it represents a group of data structures. The message to send to the overlay, the key specified for that message, and the optional next hop where the message would be forwarded represent its components (Fig.A.6). This interface can be used by the application to modify the contents of the message, the key or the next hop through functions detailed below.

- ✧ `Id getDestinationId();` returns the Id of the node selected as best destination for this message.
- ✧ `NodeHandle getNexthopHandle();` returns the NodeHandle (Id, IP and port) of the node selected as possible next hop for this message.
- ✧ `Message getMessage();` retuns the related Message.
- ✧ `void setMessage(Message msg);` replaces the original message with that specified as parameter.
- ✧ `void setDestinationId(Id id);` set a new value for the destination Id.
- ✧ `void setNexthopHandle(NodeHandle nh);` set a new value for the next hop handle.

Generally the next hop is set to null, otherwise the application forces the message to be sent to a specific destinationgoing over the subject-based routing.


Figure A.7.: Class Diagram: Endpoint interface

*Endpoint*: represents the entity that allows the interaction of the programmer with the internal data structures of the p2p system. It defines main functions to initialize and maintain the overlay (Fig.A.7).

- ✧ `void route(Id key, Message message, NodeHandle hint);` sends the message, specified as parameter, to the node logically closest to the `key` value. The `hint` value is optional, and it represents a node that should be used by the system to forward the message. It is set to null value to use the original subject-based routing of the overlay, but it has to be set to a valid value in case no key is specified. Key and hint cannot be both set to null. If a key value is specified, in Pastry, Chord et al. this function represents a possible multi-hop middleware routing of the message to reach the final destination, since each node maintains only a limited part of the overlay in its data structures. In case of CrossROAD, since each node has a complete knowledge of the current state of the overlay, this function generates a single p2p connection to the final destination of the message.

- ✧ `Id getId();` returns the Id of the local node.
- ✧ `NodeHandle getLocalNodeHandle();` returns the NodeHandle of the local node.
- ✧ `NodeHandleSet replicaSet(Id id, int maxRank);` this methods returns an ordered set of NodeHandles on which replicas of an object with a given id can be stored. The call returns a number of nodes up to maxRank.
- ✧ `NodeHandleSet localLookup(Id id, int num, boolean safe);` this method returns a list of a maximum number of `num` NodeHandles that can be used as next hops on a route towards the best destination for a given Id. The list is ordered by the logical proximity between the specified `id` and the logical Ids of nodes of the overlay. In FreePastry[1] [fre], if the safe flag is specified, then the fraction of faulty nodes returned is no higher than the fraction of faulty nodes in the overlay. In CrossROAD it is equivalent to the previous function, because the subject-based routing is anyway a single p2p connection. However, if the safe flag is set to true, the cross-layer interaction updates the state of the overlay to eliminate the possibility of faulty nodes, otherwise the system exploits the last updated information stored in its internal data structures. This has been done to reduce the frequency of cross-layer interactions when unnecessary.
- ✧ `NodeHandleSet neighborSet(int num);` returns an unordered set of up to `num` NodeHandles that are logical neighbors of the local node.
- ✧ `void scheduleMessage(Message message, long delay);` schedules a message to be delivered to the local application after the provided number of milliseconds.
- ✧ `void closeApplication();` this function has been added in this version of the commonAPI in order to manage cases in which the application decides to notify its disconnection from the p2p system. In this way, CrossROAD manages the shutdown of the application by sending a *DisconnectMessage* on the network through the routing protocol, to notify other nodes of the disconnection of such node from the service. The receiving nodes update their internal data structures with the delay of the proactive routing protocol. On the other hand, in case of Pastry, the disconnection of a node from the overlay is detected by each node periodically monitoring the status of their logical neighbors, increasing the overhead on the network.

Furthemore, functions explained below have been added to this interface as cross-layer enhancements to the interaction between the application and CrossROAD. At the same time, classic p2p systems can simply maintain the empty definition of these functions in their implementation. Specifically:

- ✧ `OverlayRoutingTable getOverlay();` returns a list of node Ids currently present in the overlay network. In order to return an updated view of the overlay to the application, CrossROAD implements this function as a reactive cross-layer interaction with the routing protocol, that returns the list of Ids of nodes that are providing the specified service. Some applications can be optimized directly knowing the current state of the

[1] An open source implementation of Pastry developed by Rice University. It implements a commonAPI that reflects basic concepts defined in [DZD+03], but it also extends it with many other specific functions.

overlay, managing the distribution of particular messages or becoming aware of overlay changes.

- ✧ `NetworkRoutingTable getNetworkRT();` returns a list of entries consisting of: IP address of the destination, IP address of the next hop, and cost to reach the destination from the local node. CrossROAD implements this function directly requiring the network routing table to the XL-plugin. Also in this case the knowledge of the physical topology of the network can be exploited to optimize the behaviour of some applications (e.g. defining the set of best nodes where to store replicas as the nodes whose logical ids are close to the key but, at the same time, are physical neighbors of the sender). We are also currently working on alternative metrics for the routing protocol to be exploited by applications as additional information obtained by the cross-layer interaction (e.g. nodes mobility, link power).
- ✧ `NodeHandle getRemoteHandle(InetSocketAddress add);` returns the NodeHandle of a remote node, starting from the IP address and the port where the service is provided. This function can be useful for applications that want to force the route of a message to pass through a specific node. To do this, they must know the NodeHandle of the related node but they cannot directly compute it through the *NodeHandle* interface.

In order to define standard data abstractions for middleware and network routing tables, two additional interfaces have been defined:

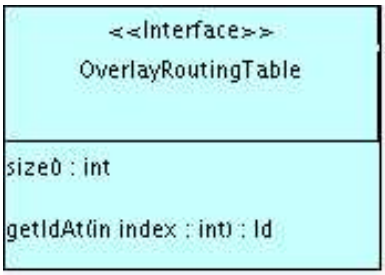



Figure A.8.: Class Diagram: OverlayRoutingTable interface

*OverlayRoutingTable*: list of node Ids that currently participates to the overlay (A.8).

- ✧ `int size();` returns the number of elements of the overlay.
- ✧ `Id getIdAt(int index);` returns the Id stored at the specified index of the list.

*NetworkRoutingTable*: list of entries of the network routing table consisting of IP address of the destination, IP address of the next hop, and the cost to reach the destination depending on the metric chosen by the protocol (Fig.A.9).

- ✧ `int size();` returns the number of elements of the network routing table.
- ✧ `InetAddress getDestIPAt(int index);` returns the IP address of the destination node at the specified index of the list.

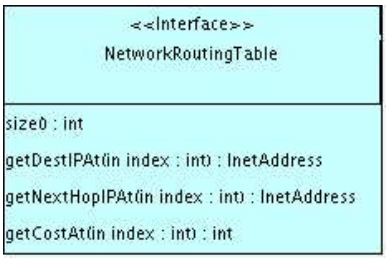



Figure A.9.: Class Diagram: NetworkRoutingTable interface

- ✧ `InetAddress getNextHopIPAt(int index);` returns the IP address of the netx hop node at the specified index.
- ✧ `int getCostAt(int index);` returns the cost to reach a specified destination at the selected index.

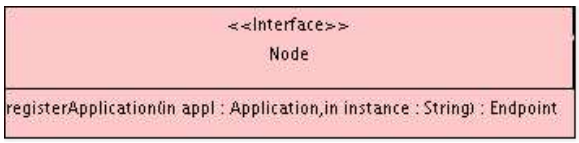



Figure A.10.: Class Diagram: Node interface

*Node*: entity that directly manages internal data structures of the overlay network (overlay routing tables, association between logical and physical identifiers, messages, and others). It is implemented by the p2p system and it appears to the applications' programmer just as a simple function that generates the Endpoint as a connection between the instance of the service and overlay data structures (Fig.A.10).

- ✧ `Endpoint registerApplication(Application application, String instance)` allows the programmer to use and manage instances of data structures of the overlay through Endpoint functions, without knowing their specific implementation.

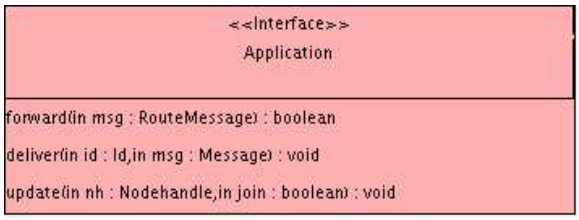



Figure A.11.: Class Diagram: Application interface

*Application*: abstraction of the service locally provided (Fig.A.11). The application's programmer has only to implement this interface to exploit all features of the overlay network. Specifically it consists of 3 functions:

- ✧ `void deliver(Id key, Message message);` is invoked by the p2p system on the node that is the best destination for the specified key upon the arrival of the message. The

application has to interpret the content of the message and consequently operate on its data structure.

- ✧ `boolean forward(RouteMessage message);` is invoked at each node that forwards the message, including the source node. As the parameter passed to the application is a RouteMessage, it can decide to modify its contents, the key and/or the default routing behaviour selecting a different next hop. If the next hop is modified from a valid value to null, the message will be terminated on the local node. Since CrossROAD always requires just a single p2p connection, this function is maintained mainly for other p2p systems.
- ✧ `void update(NodeHandle handle, boolean joined);` is used by p2p systems to notify the application of a connection/disconnection event in the overlay. This upcall derives from proactive monitoring procedures that send neighbor discovering messages to recover their current status. In CrossROAD it is completely useless because, if the application needs to know the current state of the overlay, it can directly require it through the related function, and the consistency of the information is guarateed strictly depending on the network routing protocol. In addition, since it is a reactive operation, executed only if required by the application, it does not increase the overhead on the network.


Figure A.12.: Class Diagram: InitCommonAPI class

An additional component of this new commonAPI is represented by a java class that initializes all data structures necessary to the service to interact with the p2p system. The application's programmer cannot initialize the overlay data structures only through their interfaces, because an explicit contructor is needed to build an istance of each object. This would require the knowledge

of the software architecture of the overlay in order to directly access those data structures, making the application dependent on the overlay implementation. To avoid this dependence, the *InitCommonAPI* class has been defined. In our case this class has been implemented for CrossROAD, and it provides the following functions:

- ✧ `InitCommonAPI(int port);` constructor that creates instances of CrossROAD objects. Specifically, it creates an instance of CrossROAD node, the IdFactory that provides the generation of Ids, the local NodeHandle and the local Id. The local IP address is specified in a configuration file to avoid problems selecting the right interface. The configuration file, called `service_config`, has been thought to maintain a set of information: the local IP and the pair (name of the service, port) for each service currently running on top of CrossROAD. This is necessary to associate a port to each service to allow the p2p system to know the port on which communicate with other nodes of the same overlay.
- ✧ `XL-CommonAPI.Node getLocalNode();` returns a reference to an instance of the local node initialized by the constructor.
- ✧ `XL-CommonAPI.getLocalId();` returns a reference to an instance of the local node's Id.
- ✧ `XL-CommonAPI.getLocalHandle();` returns a reference to an instance of the local NodeHandle.
- ✧ `XL-CommonAPI.Idfactory getIdFactory();` returns a reference to an instance of the IdFactory that implements the hash function chosen by the specific p2p system. This function is necessary to the application to create instances of message keys and node Ids, starting from their IP addresses.
- ✧ `InetAddress getLocalIP();` returns the IP address of the local node selected from the configuration file.

Thus, each application only needs to create an instance of the InitCommonAPI and retrieve from it all the references to the correspondent interfaces. In addition each service has to directly implement the *Application* interface and to execute the `registerApplication` method of the Node interface to get the endpoint needed to interact with the overlay network. Once the service has got the reference to each single object, it can execute all functions declared in their interface. On the other hand, overlay networks have only to implement commonAPI interfaces and modify the definition of the InitCommonAPI class, referring to their internal data structures.

# Bibliography


[ABC+05] E. Ancillotti, R. Bruno, M. Conti, E. Gregori, and A. Pinizzotto. Experimenting a Layer 2-based Approach to Internet Connectivity for Ad Hoc Networks. In *Proc. of IEEE ICPS Workshop on multi-hop Ad hoc Networks: from theory to reality (REALMAN 2005), in conjuction with IEEE ICPS 2005*, Santorini, Greece, July 2005. 119

[ABCG04] 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. 29, 30

[ADGS02] E. Anceaume, A. K. Datta, M. Gradinariu, and G. Simon. "Publish/subscribe scheme for mobile networks". In *Proc. of ACM Workshop on Principles of Mobile Computing 2002*, pages 74–81, 2002. 3

[aod] AODV: Ad hoc on demand distance vector routing. http://user.it.uu.se/ henrikl/aodv/. 16, 34

[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 E-WIND Workshop, in conjuction with SigCom 2005 Conference*, Philadelphia, August 2005. 4, 12, 61

[BCDP05] E. Borgia, M. Conti, F. Delmastro, and L. Pelusi. Lessons from an Ad-Hoc Network Test-Bed: Middleware and Routing Issues. In *Ad Hoc and Sensor Wireless Networks, An International Journal*, Vol.1, Numbers 1-2, 2005. 4

[BCG05] Raffaele Bruno, Marco Conti, and Enrico Gregori. Mesh Networks: Commodity Multihop Ad Hoc Networks. *IEEE Communications Magazine, Vol. 43, No. 3, pp. 123-131*, Mar 2005. 10, 101, 119

[BCGS04] S. Basagni, M. Conti, S. Giordano, and I. (Editors) Stojmenovic, editors. *Mobile Ad Hoc Networking*. IEEE Press and John Wiley and Sons, Inc., New York, 2004. 10, 35

[BCM05] P. Bellavista, A. Corradi, and E. Magistretti. "Lightweight Replication Middleware for Data and Service Components in DENSE MANETs". In *Proc. of 6th IEEE Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM 2005)*, Taormina, June 2005. 3, 12

[Bid01] C. Bidiskian. An overview of the Bluetooth Wireless Technology. *IEEE Communication Magazine*, December 2001. 9

[BMJ00] J. Broch, D. A. Maltz, and D. B. Johnson. Quantitative lessons from a full-scakle multi-hop wireless ad hoc network testbed. In *Proc. of the IEEE Wireless Communications and Network Conference (WCNC 2000)*, 2000. 30

[Bor05] E. Borgia. Experimental Evaluation of Ad Hoc Routing Protocols. In *Proc. of Workshop of Pervavise Wireless Networking 2005, in conjuction with the PerCom 2005 conference*, Kauai Island, Hawaii, March 2005. 78

[BR04] E. Belding-Royer. *Routing approaches in Mobile Ad Hoc Networks,*. S. Basagni and M. Conti and S. Giordano and I. Stojmenovic Ed., IEEE Press and John Wiley and Sons, New York, 2004. 12, 15, 16

[BRT99] E. M. Belding-Royer and C. K. Toh. A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks. *IEEE Personal Communication Magazine*, pages 46–55, April 1999. 15, 16

[cam] Computer Laboratory, University of Cambridge. http://www.cl.cam.ac.uk. 101

[CCL03] I. Chlamtac, M. Conti, and J. Liu. "Mobile Ad Hoc Networking: Imperatives and Challenges". *Ad Hoc Networks Journal, Vol. 1, N. 1, Jan-Feb-Mar*", 2003. 3, 10, 12

[CCMT04] M. Conti, J. Crowcroft, G. Maselli, and G. Turi. *A Modular Cross-Layer Architecture for Ad Hoc Networks*. Handbook on Theoretical and Algorithmic Aspects of Sensors, Ad Hoc Wireless, and Peer-to-Peer Networks,Jie Wu (Editor), CRC Press, New York, 2004. 46, 51, 66

[CD04] D. Cook and S. K. Das. *Environments: Technologies, Protocols and Applications*. IEEE Press and John Wiley and Sons, 2004. 9, 12

[CDG] M. Conti, F. Delmastro, and E. Gregori. P2P CommonAPI for structured overlay networks: a Cross-Layer Extension. Technical Report available at `http://cnd.iit.cnr.it/people/fdelmastro/`. 6, 61

[CDHR] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Exploting network proximity in peer-to-peer overlay networks. Technical report. `http://freepastry.rice.edu/PAST/`. 117

[CDKR02] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. One Ring to Rule them All: Service Discovery and Binding in Structured Peer-to-Peer Overlay Networks. In *Proc. of the Tenth ACM SIGOPS European Workshop*, Saint-Emilion, France, 2002. 66

[CDT05] M. Conti, F. Delmastro, and G. Turi. *Peer-to-Peer Computing in Mobile Ad hoc Networks*. Mobile Middleware, Computer and Information Sciences Series (coordinator Sartaj Sahni),P. Bellavista and A. Corradi Ed., Chapman and Hall/CRC Press, Nov. 2005. 4, 5, 12

[CGH+02] E. Callaway, P. Gorday, L. Hester, J. A. Gutierrez, and Naeve M. Home Networking with IEEE 802.15.4. *IEEE Communications Magazine, pp. 70-77*, Aug 2002. 12

[CGT05] M. Conti, E. Gregori, and G. Turi. "A Cross Layer Optimization of Gnutella for Mobile Ad hoc Networks". In *Proc. of ACM MobiHoc Symposium*, Urbana-Champain, May 2005. 12

[CJK+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. In *Proc. of INFOCOMM 2003*, San Francisco, CA, April 2003. 67, 101, 102, 121

[CMTG04] M. Conti, G. Maselli, G. Turi, and S. Giordano. Cross layering in mobile ad hoc network design. *IEEE Computer*, February 2004. 4, 5, 46, 66

[Coh03] Bram Cohen. Incentive Build Robustness in BitTorrent. http://www.bittorrent.com, 2003. 21

[d10] MobileMAN Deliverable D10. http://cnd.iit.cnr.it/mobileMAN/pub-deliv.html. 13

[d13] MobileMAN Deliverable D13: Architecture, protocols and services. http://cnd.iit.cnr.it/mobileMAN/pub-deliv.html. 4, 51, 52, 71

[d16] Deliverable D16: MobileMAN Technical Evaluation. http://cnd.iit.cnr.it/mobileMAN/pub-deliv.html. 4, 61, 89, 91

[d8] MobileMAN Deliverable D8: First Phase. http://cnd.iit.cnr.it/mobileMAN/pub-deliv.html. 34

[Del05] 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 cinference*, Kauai Island, Hawaii, March 2005. 4, 5, 53

[DP05] F. Delmastro and A. Passarella. On developing P2P Group-Communication Applications in Real-World MANETs: an Experimental Study. In *Proc. of IEEE REALMAN 2005, ICPS Workshop*, Santorini, Greece, July 2005. 4, 71

[DZD+03] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *Proc. of the the 2nd International Workshop on Peer-to-peer Systems (IPTPS'03)*, Berkeley, CA, February 2003. 6, 34, 61, 67, 69, 121, 125

[ECPS02] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1 (1):59–69, 2002. 9

[EJ] D. Eastlake and P. Jones. Us secure hash algorithm (SHA1). http://www.ietf.org/internet-drafts/draft-eastlake-sha1-02.txt. 56

[fre] Freepastry website. http://freepastry.rice.edu. 28, 31, 34, 56, 69, 77, 125

[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). 29, 30, 31, 33, 78

[GW02] A. J. Goldsmith and S. B. Wicker. Design Challenges for Energy-Contrained ad Hoc Wireless Networks. *IEEE Wireless Communication*, Vol.9, N.4:8–27, August 2002. 49

[Haa97] Z. J. Haas. "A New Routing Protocol For The Reconfigurable Wireless Networks". In *6th IEEE International Conference on Universal Personal Communications, IEEE ICUPC'97*, San Diego, CA, 1997. 16

[Her03] Klaus Hermann. "MESHMdl - A Middleware for Self-Organisation in Ad Hoc Networks". In *Proc. of IEEE Workshop on Mobile and Distributed Computing (MDC2003), in conjuction with ICDCS 2003*, May 2003. 3

[HKRZ02] K. Hildrum, J.D. Kubiatowicz, S. Rao, and B.Y. Zhao. Distributed object location in a dynamic network. In *Proc. of ACM SPAA*, Winnipeg, Canada, August 2002. 67, 121

[JM96] D. B. Johnson and D. A. Maltz. *Dynamic Source Routing IN Ad-Hoc Wireless Networks*. Mobile Computing, T. Imielinski and H. Korth (Editors), Kluwer Academic Publisher, 1996. 16

[kaz] http://www.kazaa.com. 21

[KK03] V. Kawadia and P. R. Kumar. A Cautionary Perspective on Cross Layer Design. *IEEE Wireless Communication Magazine*, July 2003. 49

[KM88] J. F. Kurose and H. Mouftah. Computer-aided modeling of computer communication networks. *IEEE Journal on Selected Areas in Communications*, 6, No. 1:130–145, Jan. 1988. 29

[KM02] T. Klinberg and R. Manfredi. Gnutella Protocol Specification v.0.6. http://rfc-nutella.sourceforge.net/src/rfc-0.6-draft.html, June 2002. 4, 21

[Lav83] S. S. Lavenberg. *Computer Performance Handbook*. Academic Press, New York, 1983. 29

[LNT02] H. Lundgren, E. Nordstrom, and C. Tschudin. "Coping with Communication Gray Zones in IEEE 802.11 based Ad Hoc Networks". In *Proc. of WoWMoM 2002*, Atlanta, GA, September 2002. 29

[Lun] H. Lundgren. Implementation and Experimental evaluation of Wireless Ad hoc Routing protocols, PhD. Thesis. Dept. of Information Technology "http://publications.uu.se/theses/abstract.xsql?dbid=4806. 79

[man] IETF Mobile Ad-hoc Networks (manet) working group. http://www.ietf.org/html.characters/manet-charter.html. 11, 16

[MC02] R. Meier and V. Cahill. "STEAM: Event-Based Middleware for Wireless Ad Hoc Networks". In *Proc. of 22nd International Conference on Distributed Computing Systems Workshops (ICDCDW'02*, 2002. 3

[MC03] J. P. Macker and S. Corson. *Mobile Ad Hoc Networks (MANET): Routing Technology for dynamic, wireless networking*. S. Basagni, M. Conti, S. Giordano, I. Stojmenovic (Editors), IEEE Press and John Wiley and Sons, Inc., New York, 2003. 49

[MCZE02a] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. "Middleware for Mobile Computing (A Survey)". *Advanced Lectures in Networking*, 2002. G. Anastasi, S. Basagni, E.Gregori (Editors). Springer. LNCS 2497. 3

[MCZE02b] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. "XMIDDLE: A Data-Sharing Middleware for Mobile Computing". In *Wireless Personal Communications, Vol. 21*, pages 77–103, 2002. 3

[MD02] K. M. Marina and S. R. Das. Routing performance in the presence of unidirectional links in multihop wireless networks. In *MOBIHOC 2002*, EPFL Lausanne, Switzerland, June 2002. 17

[mob] IST-FET MobileMAN Project. http://cnd.iit.cnr.it/mobileMAN/. 50, 101, 113

[MPR01] A.L. Murphy, G.P. Picco, and G. C. Roman. LIME: A Middleware for Physical and Logical Mobility. In *Proc. of the 21st International Conference on Distributed Computing Systems*, April 2001. 3

[net] NetiKos. http://www.netikos.com. 113

[NTCS99] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. "The broadcast storm problem in a mobile ad hoc network". In *The Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'99)*, Seattle, Washington, USA, August 1999. 15

[oCSaUS] Department of Computer Systems at Uppsala (Sweden). Ape: Ad hoc protocol evaluation testbed. http://apetestbed.sourceforge.net/. 30

[PC02] I. Pratt and J. Crowcroft. "Peer-to-Peer systems: Architecture and Performance". In *Networking 2002 Tutorial Session*, Pisa, Italy, May 2002. 4

[Per00] C. E. Perkins. *Ad Hoc Networking*. Addison-Wesley, Reading, MA, 2000. 15

[PR99] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *Proc. of 2nd IEEE Workshop on Mobile Computing Systems and Applications*, February 1999. 16, 77

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object localtion and routing for large scale peer-to-peer systems. volume 2218, pages 329–350, 2001. 4, 5, 22, 25, 26, 28, 36, 55

[rfc] Optimized link state routing protocol (OLSR): RFC3626. http://www.ietf.org/rfc/rfc3626.txt. 16, 18, 77

[RFH+01] S. Ratsanami, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM 2001*, San Diego, CA, August 2001. 4, 22, 23, 24

[RGK+05] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasanamy, S. Shenjer, I. Stoica, and H. Yu. OpenDHT: A Public DHT Service and Its Uses. In *Proc. of ACM SigCom 2005*, Philadelphia, August 2005. 119

[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. 3

[SMK+01] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM 2001*, San Diego, CA, August 2001. 4, 22, 24

[SN03] R. Schollmeier and F. Niethammer. protocol for peer-to-peer networking in mobile environments. In *Proc. of 12th IEEE International Conference on Computer Communications and Networks*, Dallas, Texas, USA, 2003. 53

[SW03] I. Stojmenovic and J. Wu. *Broadcasting and Activity-Scheduling in Ad Hoc Networks*. S. Basagni, M. Conti, S. Giordano, I. Stojmenovic (Editors), IEEE Press and John Wiley and Sons, Inc., New York, 2003. 15

[tbr] Topology dissemination based on reverse-path forwarding (tbrpf): RFC3684. http://www.ietf.org/rfc/rfc3684.txt. 16

[Ton] Andreas Tonnesen. OLSR: Optimized link state routing protocol. Institute for informatics at the University of Oslo (Norway), http://www.olsr.org. 16, 34, 61, 62

[udd] UDDI: Universal Description, Discovery and Integration. http://www.uddi.org. 113