Journal of Network and Computer Applications 33 (2010) 588–602

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Moving routing protocols to the user space in MANET middleware ☆

Pedro García López *, Raúl Gracia Tinedo, Josep M. Banús Alsina

Universitat Rovira i Virgili, Department of Computer Engineering and Maths, Av. Paisos Catalans 26, 43007 Tarragona, Spain

ARTICLE INFO

Article history:
Received 13 November 2009
Received in revised form
10 February 2010
Accepted 18 March 2010

Keywords:
Middleware
Group communication
Mobile Ad Hoc Networks
Routing protocols

ABSTRACT

Mobile Ad Hoc Network (MANET) middleware must be aware of the underlying multi-hop topology to self-adapt and to improve its communication efficiency. For this reason, many approaches rely on specific cross-layer communications to interact with the network protocols in the kernel space. But these solutions break the strict layering of the network stack and hinder the portability of middleware and applications.

The main argument of this paper is to move the routing protocols to the user space to simplify the development, testing, deployment and portability of middleware and applications. If routing is just another software component in the user space, cross-layering can be elegantly solved using advanced software engineering techniques like component frameworks and explicit APIs. As a consequence, a slight performance cost must be paid to achieve portability and easy deployment. But we will demonstrate that the performance obtained by a user-space routing protocol is satisfactory for a wide range of applications.

We have implemented the unicast MANET OLSR protocol in Java (jOLSR) and, on top of it, we have created a novel overlay multicast protocol (OMOLSR). We have then integrated both routing protocols (jOLSR, OMOLSR) as software components in a well-known group communication toolkit (JGroups). Modifying the JGroups toolkit, we have devised a topology-aware group communication middleware for MANETs (MChannel). In our MChannel middleware, group membership is obtained directly from OMOLSR multicast trees and failure detection is obtained from jOLSR active probing. We have validated our approach in several real testbeds to demonstrate the feasibility and efficiency of our middleware.

1. Introduction

In the next years we will see a boost in mobile terminals including wireless connectivity (802.11X). As a consequence, Mobile Ad Hoc Networks (MANETs) could represent an interesting substrate for many types of applications not requiring a fixed network infrastructure (Access Points). When every participant is a routing node in the network, several interesting multi-hop scenarios may arise. For example, spontaneous collaborative applications are of particular interest for settings where many users can interact with close-by participants (conferences, campus, stadium, popular events, cities and games).

But the reality is that MANET networks are seldom used in everyday life. As stated in Bouckaert et al. (2008): "A possible explanation is the fact that only few of the numerous theoretically promising proposals lead to practical solutions on real systems". Authors outline several causes like hardware issues regarding heterogeneous devices, cross-layer dependencies, and the existing gap between simulation and real test scenarios.

Besides, application development in this domain is quite complex. Application developers must face challenges like mobility, heterogeneity, dynamicity or topology awareness to name a few. A clear trend is to rely on advanced MANET middleware (Hadim et al., 2006) capable of dealing with the aforementioned problems and thus *simplifying application development*. In these settings, being aware of the underlying routing protocols and topology is very important for MANET middleware and applications. If the middleware just uses the underlying transport protocol as a black box, it can incur in communication inefficiencies due to the *multi-hop nature* of the medium.

A typical solution has been to implement cross-layer designs (Conti et al., 2004; Goldsmith and Wicker, 2002; Lin et al., 2006) enabling the construction of more efficient middleware (using topology information). In this line, cross-layering is required between user-space MANET middleware and kernel-space MANET protocols. However, we believe that this approach is flawed because it implies improper ad hoc hacks that break the clean network layering model. These approaches solve specific problems but are *difficult to port or adapt* to heterogeneous devices and operating systems.

☆ A previous publication related with this article can be found at the program of COMSWARE'2009.

* Corresponding author. Tel.: +34 657534626; fax: +34 977 55 97 10.
E-mail addresses: pedro.garcia@urv.cat (P. García López), raul.gracia@urv.cat (R. Gracia Tinedo), josepm.banus@urv.cat (J.M. Banús Alsina).

1084-8045/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2010.03.018

Another recent trend in MANET settings is to follow a framework design to the construction and deployment of ad hoc routing protocols (Ramdhany et al., 2009). The complexity of testing and development of ad hoc protocols is simplified thanks to advanced software engineering approaches like component frameworks. These frameworks combine routing components in the user space with network tools (netfilter) in the kernel space. These recent works are in line with the argument of this paper. However, we go one step further than these protocol development frameworks.

We propose to completely move routing protocols to the user space avoiding dependencies with kernel components. This design decision entails the two major contributions of our approach: portability and topology awareness. If routing is another software component, the middleware and applications are self-contained and portable to heterogeneous environments. Besides, applications can access the routing information to be aware of the multi-hop topology and optimize their communications.

In this paper, we will present the advantages and drawbacks of moving the routing protocols to the user space. As a proof of concept, we focus on the application domain of group collaboration in MANET scenarios (MChannel). Particularly, in the context of the European project IST-POPEYE (Peer to Peer Collaborative Working Environments over Mobile Ad Hoc Networks), we have developed a Java implementation of the OLSR protocol using UDP sockets (jOLSR). To address the group communication issues of the collaborative middleware, we have created OMOLSR, an overlay multicast channel that directly benefits from the jOLSR routing protocol and failure detection. Finally, using jOLSR and OMOLSR we have modified an existing group communication middleware (JGroups) to support MANET settings. We will demonstrate by extensive experimentation in real testbeds that the system is efficient and that it benefits from the knowledge of the underlying topology.

The structure of this paper is as follows: in Section 2 we present related work, then in Section 3 we present some design decisions and in Section 4 we describe the overall architecture. In Sections 5 and 6 we present validation results using experimentation, and finally we draw some conclusions in Section 7.

2. Related work

It is widely accepted in MANET settings that *cross-layer solutions* are required for achieving performance and adaptation in many scenarios (Conti et al., 2004; Goldsmith and Wicker, 2002; Lin et al., 2006). Nevertheless, many researchers also recognize that strict layering enables controlled interaction among layers (each layer is independent developed and maintained). Besides, many cross-layer solutions generate highly coupled code that is impossible to maintain efficiently because unexpected dependencies between layers may arise.

To avoid this coupled cross-layer solutions, authors in Conti et al. (2004) proposed a vertical component named Network Status that would avoid improper inter-layer communication. All the layers (MAC, Network, Transport, Middleware and Application) would then communicate directly with Network Status and thus avoiding direct coupling between layers. This design is more elegant than the classical cross-layer solutions but still faces an important problem: it is not a standardized operating system component so it is difficult to port and maintain.

Concerning the middleware and application layers, it is also obvious that cross-layer solutions clearly present better performance (Delmastro, 2005; Conti et al., 2005). But as stated before, cross-layer solutions break the strict layering of the network stack and hinder the portability of middleware and applications.

Regarding *Protocol Development Frameworks*, we outline three main works: MANETKit (Ramdhany et al., 2009), ASL (Kawadia, 2003) and PICA (Calafate and Manzoni, 2003). All of them try to simplify protocol development and testing by moving routing protocols to the user space. While PICA and ASL only offer design-time and implementation-time facilities, MANETKit also offers runtime reconfiguration thanks to its component framework approach. These frameworks achieve performance numbers close to the transport protocol kernel implementations thanks to their interaction with kernel tools like netfilter. Nevertheless, their interaction with such kernel tools restricts their portability to heterogeneous settings. Furthermore, such frameworks are not focused on providing high level middleware services to applications.

Concerning *MANET communication middleware*, other works also followed a purely user-space routing solution. In Mottola et al. (2008) authors present COMAN, a protocol to organize the nodes of a MANET in a tree-shaped network providing content-based routing functionalities. COMAN is implemented at the application layer and it benefits from topology information to self-repair and reconfigure the middleware. In order to provide multicast communication, COMAN does not use per-group trees. Its strategy is based on connecting all the brokers in a tree-shaped network. In consequence, a multicast message will be forwarded to all brokers with subscribers interested in.

In Yoneki and Bacon (2005) authors also present a content-based publish/subscribe system for MANETs that extends the ODMRP (On-Demand Multicast Routing Protocol) to construct an optimized dynamic dissemination mesh. Authors in Yoneki and Bacon (2005) stress the importance of cooperation between the middleware-tier and routing components.

Both COMAN and ODRMP present purely user-space solutions that create their own specific topologies. They show the advantages of self-adaptation and portability of middleware components. Thus, they are clearly aligned with our proposed trend for MANET middleware of moving routing logic to the user space. But COMAN and ODRMP, like many overlay multicast protocols, focus on adaptive tree construction and one-to-many communication. They do not build multicast protocols that benefit from an underlying unicast routing protocol. As we will explain later, we present a complete communication model for MANETs that offers both unicast and multicast services without implying redundant control messages.

Furthermore, they do not provide topology-aware high level services to applications. Our position is that many applications explicitly require topology awareness. For example, a data replication application could require topology awareness to distinguish between close-by and far nodes in term of hop distance. This is not possible with COMAN and ODRMP, since they make the network transparent to their applications.

Regarding *Group Communication services* (*GC*) over MANETs we must outline two research works: JazzEnsemble (Friedman, 2004) and MobileMan (Conti et al., 2006). The authors of JazzEnsemble devised lightweight membership protocols based on fuzzy membership and gossip-based failure detection protocols. MobileMan implemented a cross-layer solution using the FreePastry overlay over the OLSR protocol. Nevertheless, none of them construct their GC primitives over standard MANET unicast and multicast routing protocols (OLSR) like us. As we will explain later, our decision entails important communication savings.

Finally, we outline an important survey entitled "Trends in Middleware for Mobile Ad Hoc Networks" (Hadim et al., 2006). Authors classify MANET middleware into six categories: event-based and message oriented middleware, component-based and

mobile agents middleware, peer-to-peer middleware, tuple spaces middleware, data sharing middleware, and virtual machine based middleware. They compare these categories under an evaluation framework that considers six aspects: power awareness, openness, scalability, mobility, heterogeneity and ease of use.

We draw an important conclusion from this study; while four out of the six variables—power awareness, scalability, mobility and ease of use—depend on the specific design of the MANET middleware, both openness and heterogeneity require a solution that implements the routing logic in the user-space. On the one hand, supporting heterogeneous devices and operating systems is seriously handicapped if the middleware relies on cross-layer solutions or depends on specific kernel components. On the other hand, openness, or the possibility to extend and modify the system easily, also relies on clean software engineering architectures (like component models) that avoid dependencies with the network stack.

As we will explain in the following sections, our approach for MANET middleware is simple: moving the routing protocols to the user space. This is probably not a solution for all kind of problems and we neither aim to replace existing transport layer MANET protocols. But we will explain how this simple solution can be appropriate for many scenarios where cross-layer information flows are required by middleware and applications.

3. Motivation and design decisions

In MANET settings, moving the routing logic to the user space can be specially interesting for several applications. For example, context-based or location-based applications may use the topology information to communicate with close-by nodes while moving. Local advertisements can thus be disseminated only to nodes at one or two hops. In this case, users may discover and contact services that are located at a given range. A concrete example can be a mobile tourist guide application for museums or cities. This application may offer specific information to the user based on the location of the nodes. For example, the user can obtain information about a close-by monument or even interact with other tourists.

Other target applications are spontaneous collaboration tools or interactive games. They may use topology information to store or disseminate data depending on their specific locations. If these applications ignore the underlying topology (broadcast) they will be rather inefficient transmitting information to inappropriate nodes. For example, multi-user games can employ hop scoped multicast to limit traffic to specific zones. Furthermore, collaborative tools like shared whiteboards or file sharing can benefit from the stability of nodes or from their strategic location in the graph to store and retrieve their required information.

In general, our approach will be specially interesting in the following years when mobile terminals like smart phones with wireless connectivity will be widespread. The heterogeneity of these devices will require portable solutions for infrastructure-less communications. The openness and flexibility of our approach is more suitable than ad hoc cross-layer solutions. As we will explain, portability is one of the key aspects of our proposed solution.

Before delving into our main technical contribution, we first present the advantages and drawbacks of moving the routing logic to the application layer, and then we explain the design decisions behind the chosen unicast and multicast routing protocols.

3.1. Advantages and drawbacks of application-layer MANET routing

Let us first present the major advantages of this approach:

Portability to heterogeneous environments: Portability to heterogeneous devices and operating systems is ensured thanks to the use of portable programming infrastructures (virtual machines) in the user space. Cross-layer solutions are tied to specific environments and thus prevent the support for heterogeneity. The middleware can be developed transparently from the operating system and thus simplifying kernel compilation, library dependencies and hardware settings. Mobile phones could easily install an application that creates its own MANET adapted to its specific requirements. It is far more difficult to motivate users to install new transport protocols because their limited knowledge and technical skills. Besides, it is more complex for developers to develop and maintain OS specific protocol distributions.

Topology-aware overlay: In MANET environments, nodes are connected in a peer-to-peer network created in the transport layer using protocols like OLSR (Jacquet et al., 2001) or DYMO (Chakeres and Perkins, 2004). If we just move these protocols to the application layer (OLSR for example), the topology does not change. The application layer peer-to-peer network will then reflect exactly the physical connections. Furthermore, the application layer protocol will send packets using UDP unicast connections in the same way that the transport protocol would operate.

Flexibility, openness and adaptation to specific requirements: Different applications may have completely different requirements for the routing layer and underlying topology. If it is a matter of changing a software component, it is feasible to develop specific adaptive routing layers for different settings. Furthermore, the use of components and composition technologies may provide design-time, compile-time and runtime reconfiguration. For example, emergency scenarios, military applications or collaborative work settings may develop specific underlying routing components adapted to their particular requirements.

Simplicity of development, testing and deployment: It is easier to develop and test routing protocols in the application layer than in the transport layer. It is also possible to develop and test MANET applications using software emulation in a single machine and afterwards test them in real testbeds with multiple devices. Protocol development frameworks are already offering emulation environments for testing and developing new protocols. MANET middleware can offer the same advantages to MANET applications.

Clean interaction between middleware and routing protocols: When MANET routing is just another middleware component we can use well-established software engineering practices to specify module interaction. With standardized APIs in the application layer it is not necessary to break the strict layering of the network stack for MANET middleware.

On the contrary, we find these major *drawbacks*:

Specific routing service: At the time present, our approach does not offer a generic service for all the applications running in an OS (like transport protocols do). Only the application using this middleware will benefit from the MANET routing services. If we have for example existing videoconferencing or desktop sharing tools using standard transport protocols, they will not be able to benefit from our middleware.

Performance: It is obvious that working in the user-space implies a penalty over kernel transport protocols. Besides this, the implementation in Java may also damage the performance of the protocol compared to native kernel implementations. However, we justify in the validation that our middleware achieves performance numbers that are reasonable for a large number of applications. We sacrifice some performance to obtain portability and flexibility of MANET applications.

3.2. MANET protocols

Here we include a brief summary of recent unicast and multicast protocols considered in our design. Concerning unicast

protocols, we considered two recent protocols (one reactive and one proactive):

DYMO (Dynamic MANET On-Demand) (Chakeres and Perkins, 2004) is a reactive unicast protocol, successor of the popular AODV (Ad Hoc On-Demand Distance Vector) and shares many of its functionalities. Routes are created on-demand by sending request and response control packets. This means that when nodes stop sending messages, there is no overhead traffic in the network. Furthermore, no global topology information is available. We can see that DYMO is more suitable for sparse communications.

OLSR (Jacquet et al., 2001) is a proactive unicast protocol, so it maintains routing table information up to date continuously. Topology information is exchanged by means of controlled flooding of Topology Messages (TC messages). HELLO messages provide information about the two-hop neighborhood in a way that each node selects a neighbor as MPR (Multi-Point Relay). These MPRs are in charge of sending topology messages to the entire network performing controlled flooding. With the topology information, each node can build the routing table in order to be able to send messages to the other nodes. OLSR performs well in small-medium sized networks where node density is relatively high. The knowledge of the topology, together with its good performance with dense communication patterns, turns OLSR in a good candidate for performing group communication in MANETs.

Concerning MANET multicast protocols, we reviewed two overlay multicast protocols (ALMA, PAST-DM) and the MOLSR network multicast protocol.

ALMA (Application Layer Multicast Algorithm) (Ge et al., 2006) creates a tree of logical links between group members. The aim of this protocol is to reduce the cost of each link in the tree by reconfiguring the tree under mobility and congestion situations. When a node joins the network it must select a node as a parent, so as to become part of the tree. If tree performance drops below a defined threshold, the node must reconfigure the tree by switching the parent or freeing children.

PAST-DM (Progressively Adaptive Subtree in Dynamic Mesh) (Gui and Mohapatra, 2003) is an overlay multicast protocol based on the construction of a dynamic virtual mesh. The mesh is maintained dynamically through the exchange of link state packets, thus adapting to network topology changes. With the topology information extracted from the mesh, nodes compute a source-based Steiner tree to deliver information to all members in the multicast group.

In conclusion, both protocols, as most overlay multicast protocols, need to send control packets to keep their structures up to date using periodic exchange of link packets or communicating with other nodes in the tree. In both cases, the protocols ignore the underlying unicast network routing protocol so they perform redundant communication and, in consequence, bandwidth misuse. We have been inspired in ideas from both of them to create our new OMOLSR protocol.

Because our group communication middleware must provide both unicast and multicast services, it is preferable to use multicast protocols that benefit from the OLSR unicast protocol. For example, MOLSR (Multicast extension for the Optimized Link State Routing protocol) (Jacquet et al., 2001) is a multicast routing protocol for MANETs designed to work on top of OLSR nodes. This protocol provides multicast routing that benefits from the topology knowledge of OLSR (MPRs). This strategy reduces drastically the number of control messages compared to other simple flooding strategies. However, we believe that the amount of control messages sent by MOLSR could also be avoided. Such protocol could benefit directly from the topological information given by the underlying unicast routing protocol to avoid redundant control traffic. For this reason, we decided to implement our own multicast protocol over OLSR. We will further explain our OMOLSR protocol in the following section.

4. MChannel: group communication middleware for MANETs

In the context of the European project IST-POPEYE (Peer to Peer Collaborative Working Environments over Mobile Ad Hoc Networks), we have designed and implemented a complete group communication middleware for MANETs.

Instead of constructing an entire GC toolkit from scratch we decided to modify a well-known Java toolkit like JGroups. JGroups is a toolkit for reliable multicast communication that offers a JChannel abstraction providing group membership, methods to send messages to one or to all members in a group, and event listeners about the channel state (member joins, leaves and received messages). Furthermore, JGroups offers a flexible protocol stack with many existing protocols for group membership, failure detection, flow control, fragmentation of packets, message ordering and reliability over IP multicast.

But JGroups uses IP unicast and IP multicast to send messages to group members. For this reason, we have created two MANET routing protocols in the user space: jOLSR (implementation of the OLSR unicast protocol) and OMOLSR (a novel multicast protocol over jOLSR). Both jOLSR and OMOLSR have been integrated as standard components in the JGroups protocol stack.

Furthermore, JGroups membership protocols, failure detection and flow control are not designed for multi-hop settings. They generate excessive traffic communication and acknowledgments because they ignore the underlying topology. For this reason, we have implemented a novel abstraction, namely MChannel, where group membership is obtained from the OMOLSR protocol, failure detection is obtained from the jOLSR protocol, and our modified flow control protocol benefits from jOLSR topology information. Besides, the MChannel also offers an added API to third-party applications with information about the underlying topology. This is very important for applications that must be aware of the topology to optimize their behavior.

In addition, we have created an emulation environment for MANET applications. We have short-circuited the JGroups protocol stack to emulate the multi-hop communication in a single node. Using a network definition file in Pajek format (Batagelj and Mrvar, 1998), it is possible to create a multi-hop topology. The emulator supports simple mobility and packet loss. The interesting aspect is that it is possible to develop and test collaborative applications in a single machine using this emulator. Without modifying the application, it is then possible to move it to a real testbed. The same applies for the development and testing of the MANET protocols (jOLSR and OMOLSR).

To explain the overall architecture (see Fig. 1), we will proceed now bottom-up describing the three building blocks of our platform: jOLSR, OMOLSR and MChannel.

4.1. jOLSR

jOLSR is an application level implementation of the OLSR routing protocol (Jacquet et al., 2001) written in Java. jOLSR implements nearly all components of the core functionality of the OLSR standard. Although the core functionality also includes support for multiple interface addresses, this feature is not provided in the current version of jOLSR in order to simplify the implementation.

jOLSR stores network information in different tables similarly to OLSR specification: Neighborhood information base (NIB) stores neighbor information; Local Link Information Base (LLIB) keeps updated information about the state of links to the neighbors; Topology Information Base (TIB) maintains information of the network topology to perform routing calculation.

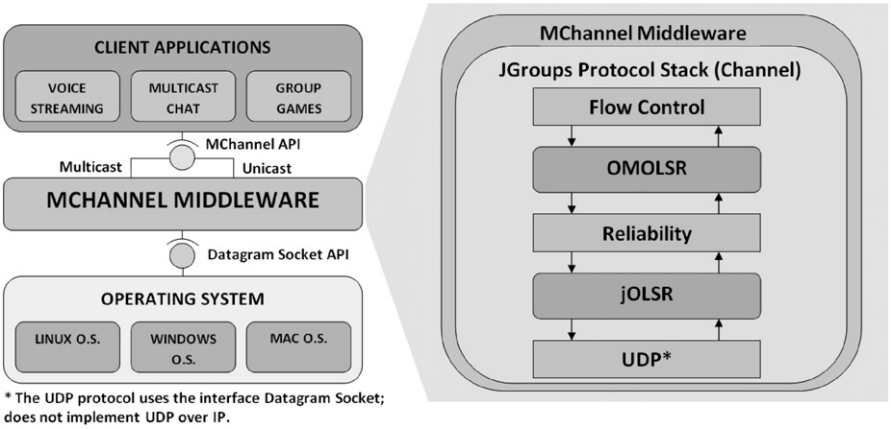

Fig. 1. Middleware placement and architecture.

Like OLSR, jOLSR is a proactive link state protocol that maintains the topology information thanks to periodic Topology Control (TC) messages between nodes. In both OLSR and jOLSR, TC messages are sent periodically from one node to the rest of the network, so all nodes can compute its own topology graph. We have realized that by adding little information in TC messages, we can disseminate information easily to all nodes in the network. This is in fact really useful since our multicast routing protocol needs information about the multicast groups joined by each node. We will explain it in detail in the following section.

TC message modification: jOLSR send two types of control messages, HELLO and TC (Topology Control) messages. HELLO messages permit a node to know its one-hop and two-hop neighbors. Based on this information, the node can select its multipoint relays (MPR) which will be in charge of performing controlled flooding. TC messages are sent to all the nodes in the network thanks to this controlled flooding mechanism, and disseminate topology information of the local node to all nodes in the network. Therefore, we attach the multicast address of the groups joined by the local node in TC messages, as Fig. 2 depicts.

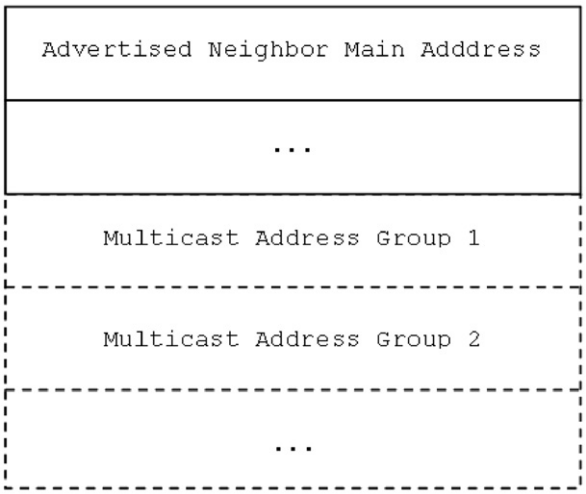

Fig. 2. TC message modification.

4.2. OMOLSR

OMOLSR (overlay multicast over OLSR) is a new application level multicast routing protocol, designed to work on top of jOLSR. OMOLSR computes locally Minimum Spanning Trees by benefiting from the topology information gathered by jOLSR. The main characteristic of OMOLSR is that it does not need to send additional control packets to perform multicast delivery. The unicast routing protocol already provides all necessary information.

Thanks to the TC message modification, OMOLSR propagates to jOLSR the multicast addresses of groups joined by the local node. jOLSR also notifies OMOLSR—Topology Events—of group changes received from TC messages. With this information, OMOLSR updates its Multicast Group Table and computes the Minimum Spanning Tree.

Multicast Group Table: The Multicast Group Table keeps the information about the multicast groups joined by each node. This information is updated when the local node decides to join a new group or when it receives a new TC message. The information in this table is used by the multicast routing protocol and changes in the table are reported as membership events to the application. When a change is detected in the neighbor table, in the topology table or in the multicast groups table, a graph containing the members of the group is computed. The multicast protocol will receive a new event with the information of this graph.

In order to calculate the graph with the members of the group, we obtain an approximate representation of the network by creating a network graph from the information stored in the topology table. Then we check in the Multicast Group Table which nodes belong to which group, so we create an event for each different group in the table. The graph is used by the multicast protocol for retrieving updated membership information.

Basic operations: The basic operations of OMOLSR are the dynamic computation of the Minimum Spanning Tree and the routing of multicast packets. In first place, OMOLSR computes a virtual mesh that connects all members of the multicast group. This mesh is a contraction of the network graph that jOLSR generates from the Neighbor Table and the Topology Table. By using this contracted graph, which contains only the members of the group, the local node can easily know which members are located at one hop. These nodes located at one logical hop are also known as virtual neighbors of the local node. The procedure of computing the contraction of the graph is performed for each multicast group the local node is member of. For example, in Fig. 3a nodes C and H are at two hops from the point of view of the routing table. However, they belong to the same multicast group and they are at one logical hop in the multicast spanning tree (Fig. 3a): they are virtual neighbors.

Then, once the graph is ready, node A computes a Minimum Spanning Tree with the local node as the source of the tree, as

depicted in Fig. 3(b). The tree will now be used for routing packets to all the members of the group.

Multicast routing: In order to route multicast packets, OMOLSR uses an explicit multi-unicast scheme. When the application generates a new multicast packet, OMOLSR routes the message based on the tree computed for that multicast group. Note that this tree is only recomputed in each node when a topology change is notified from jOLSR. The multicast algorithm proceeds as follows: a copy of the packet is sent to each virtual neighbor, which is responsible for delivering the message to a certain forwarding subset of nodes. This subset consists of all the nodes that are in the subtree of each virtual neighbor. This information is then attached to the header of the data packet.

Algorithm 1. *forwardMulticastMessage(Message msg)*

```
1.  Set ⟨Node⟩ forwardingTableEntry ← getForwardingTableEntryFromHeader(msg,localNode)
2.  if forwardingTableEntry is not empty then
3.    Set ⟨Node⟩VN ← getVirtualNeighbors(localNode,forwardingTableEntry)
4.    Set ⟨Node⟩NVN ← getNonVirtualNeighbors(localNode,forwardingTableEntry)
5.    /* Build a new table collecting all the virtual neighbor's children
      existing in the Network Graph */
6.    Map ⟨Node,Set⟨Node⟩⟩forwardingTable ← buildForwardingTable(VN,NVN)
7.    msg.updateForwardingTableOnHeader(forwardingTable)
8.    msg.updateSourceAddressOnHeader(localAddress)
9.    for all neighbor in VN do
10      sendMessage(msg, neighbor)
11.   end for
12. end if
```

As we can see in Algorithm 1, when a node receives a multicast message, it first obtains its own forwarding subset (line 1). Then, it computes the new forwarding table with the nonvirtual neighbors found in its network graph (line 6). It is worth saying that the network graph is used to adapt the forwarding table to topology changes. This forwarding table is then appended to the header of the new message (line 7) and sent to all its virtual neighbors (line 10). The process is repeated until the node's entry of the forwarding table is empty.

4.3. MChannel

We have extended an existing abstraction (JChannel) of a well-known toolkit for reliable communication (JGroups). The key feature of JGroups is its flexible protocol stack, which can be configured and extended depending on the communication needs. Each protocol in the stack provides different functionalities: ordering, reliability, membership, state transfer, etc.

On top of the routing protocols we have developed a channel abstraction which enables flexible group communication over mobile ad hoc networks: the MChannel. The main characteristic of MChannel is that users can send messages to a single member or to all the members in the group even if they are not in range. In consequence, a MChannel is bound to a single group, so if we want to communicate in two groups, we should create two different channels.

As we can see in Fig. 4, MChannel offers methods to send one-to-one or one-to-many messages to group members (send), a method to send messages to one-hop neighbors (sendToNeighbors), a method to obtain the topology information (getNetworkGraph), and methods for subscribing to events in the group (registerListeners).

Our GC middleware for MANETs provide the following services: group membership, failure detection, flow control, reliability and a local emulation environment.

Group membership: GC toolkits like JGroups maintain membership and failure detection pinging frequently or using keep-alives to all members in the JChannel. Whereas this approach works fine in local area networks, it can severely harm the overall MANET network creating unnecessary traffic. This happens because JGroups is unaware of the multi-hop nature of the medium. We provide a lightweight group membership protocol that directly benefits from the group information of the OMOLSR multicast tree. In our case, membership changes in OMOLSR (Topology Events) are an extension of JGroups membership events and are injected in the same way.

Failure detection: As explained before, failure detection implies pings or keep-alives to all group members. Again, this works in flat groups, but it causes a real burden in a multi-hop network. Our solution is to rely on the jOLSR topology detection algorithms. jOLSR is already checking the availability of nodes and continuously repairing the topology graph. Because of that, it is optimal to benefit from this information to detect leaving partners or failing nodes. It

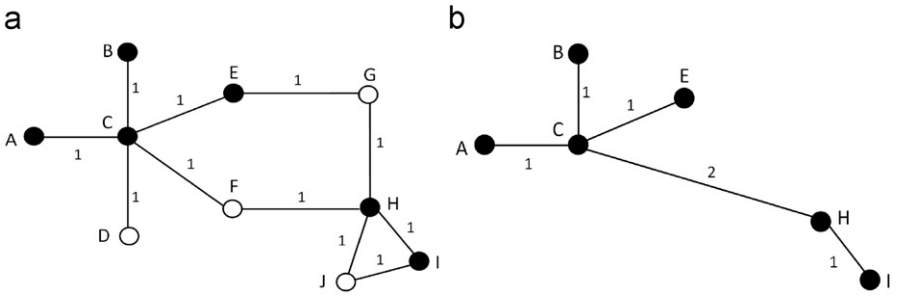

Fig. 3. The spanning tree for the node A. (a) Actual network graph; colored nodes belong to the multicast group. (b) Multicast tree seen from node A.

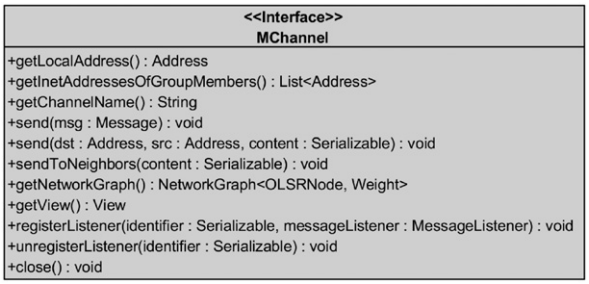

Fig. 4. MChannel API.

is nonsense to duplicate the communication overhead if jOLSR is already doing that job in an efficient and decentralized way.

Flow control: JGroups provides a simple flow control protocol based on a credit system. Each sender has a number of credits (bytes to send) and when the credits have been exhausted, the sender blocks. Each receiver also keeps track of how many bytes it has received from a sender. When credits for a sender fall below a threshold, the receiver sends more credits to the sender.

Again, the existing algorithm does not take into account the underlying multi-hop setting so data flows are not optimized for the underlying network. We have modified the JGroups FC (flow control) protocol to benefit from topology information. Our strategy is to assign credits to nodes in different proportions depending on the hop distance. The closest nodes will obtain more credits whereas distant nodes will get less credits. With this decision we adjust the flow to the possible throughput between nodes.

Reliability: We have implemented both routing protocols (jOLSR and OMOLSR) as JGroups protocols so we can benefit from unicast reliability and ordering by adding the UNICAST protocol to the JGroups stack. This unicast reliability layer uses a point-to-point acknowledgement scheme to provide lossless transmission of unicast messages. Therefore, multicast reliability is also ensured by adding this layer between both protocols: OMOLSR splits each multicast packet into several unicast packets that will be sent under the acknowledgement scheme.

Emulation environment: Finally, a clear goal of our work is to simplify the development, testing and deployment of MANET applications and protocols. For that reason, we provide an emulation layer for both MANET collaborative applications and routing protocols. It is thus possible to test applications in a single machine and later on move the same applications to a real testbed. This layer offers a smooth transition from local emulation to real experimentation without changing at all the application and routing code. To implement the emulation layer, we replace the UDP layer of the JGroups protocol stack for a virtualized layer: socket message delivery is replaced by local message queues.

In this section we have exposed our middleware architecture. In the following two sections we will report the validation results we have obtained with our implementation. First, in the next section, we detail several experiments that demonstrate that our implementation is adequate for a wide range of scenarios. Second, in Section 6 we compare the overheads that are introduced by routing at user space to those produced by network layer routing protocols.

5. Middleware validation

Many research works in MANET protocols perform their validation in simulation environments. As pointed out in Tschudin et al. (2005), there is a serious lack of real-world experiences in MANET research. Furthermore, there is a big gap between simulation and experimentation. Instead, since our ready-to-use middleware is intended to be used by real applications, we believe that a real-world test should be done to verify the middleware's feasibility and the performance of the routing protocols.

Therefore, in this section we describe several real tests which have been conducted to verify that our middleware is able to be the basis of group applications over MANET networks. We designed four test categories: the first three ones were intended to measure the middleware's reliability and resiliency with static nodes whereas in the fourth one mobility was tested.

Different hardware and software configurations were used. The computers used for the static tests were heterogeneous, including desktop PC (P4 3.0 Ghz Atheros AR5001X+ with 802.11b/g), laptops (Asus Z53S-2 Gb-e/Wlan 802.11 g) and also netbooks (Asus Eee PC 901 with 802.11n). Operating systems also were heterogeneous: Microsoft Windows XP, Vista and Linux. In contrast, mobility tests were carried out with a single laptop model (HP NX6125 with a Broadcom 4318 802.11b/g and Windows XP), in order to unify radio ranges. jOLSR timers were configured following the standard defined in RFC3626 (Clausen and Jacquet, 2003) (Hello_interval = 2 s, TC_interval = 5 s). All tests used the MChannel's standard protocol stack (emulation disabled, flow control and reliability enabled) except mobility tests, where the reliability mechanism was disabled to measure packet loss ratios. Every middleware test was repeated three times and collected data were averaged.

5.1. Test 1: real working group

The first aspect we wanted to test was the overall behavior of our middleware with a real application and common users. For this reason, this test is a *qualitative analysis of MChannel*. It should demonstrate that (i) the middleware is feasible; (ii) it works correctly with heterogeneous hardware and operating systems; and (iii) users were ready to use it in less than 1 min. Concretely, both (ii) and (iii) points are really important to verify because they are direct benefits from moving routing protocols to the user space.

With this purpose, we developed a multicast chat application capable of maintaining several chat groups at same time. This application also keeps track of all sent and received messages as well as it shows periodically the underlying topology graph obtained from the middleware.

Therefore, 40 computer science students were told to participate in this test with their own laptops. In this way, heterogeneous devices were involved using different O.S. (Windows XP, Linux) and different wireless cards. Nevertheless, *installation was very easy*, as both the middleware and the application were developed with Java. Notice that it would be a complex task if routing protocols were not at application layer.

Students were distributed in several groups along the campus. The resulting network was almost static and had a diameter of four hops. The students established correctly different multicast groups and were able to communicate successfully with other participants. We measured in the biggest multicast group 20 laptops participating at the same time.

As a qualitative analysis, we conclude that the real world experiences were successful in the sense that chat application was possible and satisfactory in real-time, proving the middleware's feasibility. This was corroborated during the posterior analysis of the logs generated by the chat application. The mentioned analysis showed that both unicast and multicast messages were sent and received correctly.

5.2. Test 2: reliability

With this test category, we wanted to evaluate the performance and behavior of the multi-hop forwarding mechanism. In this context, we focused in the analysis of how the number of hops affects to the throughput and retransmissions.

In order to carry out this test we used a file transfer application over the unicast service of the middleware.

We evaluated our middleware using a twofold metric: (i) *throughput*—i.e. total amount of application level information sent per unit of time—measured in MBps, and (ii) *ratio of retransmitted packets* over the total sent.

Hence, we disposed five computers aligned statically in the same room. Because all the computers were in range, we forced the hop sequence through setting up the proper rules for IPTables local firewalls. The leftmost computer sent a 15 MB file to other nodes, at different hop distances. Wireless cards were configured at 11 MBps transmission ratio. We executed three times the test and the averaged results are depicted in Fig. 5.

As we can see in Fig. 5, we obtained a reasonable performance ranging from 650 kBps at one hop, and decaying until 150 kBps at four hops. The throughput decreases in average about 200 kBps per hop. The rate of retransmissions is low in proportion, ranging from almost 0% at one hop to 14% at four hops. These results were possible because the middleware used the flow control and reliability protocols.

It is important to remark two points to understand this behavior: in first place, the maximum throughput is determined by the slowest node—in this case a netbook. Secondly, all nodes were in range and competing for the medium. This fact reduces the overall throughput, delays the acknowledgments and, as a consequence, increases the number of retransmissions.

In conclusion, these results demonstrate that our middleware is feasible and reliable for short and medium diameter networks. Specially, when the applications involved require unicast reliability and certain degree of throughput. According with the results, our middleware could provide a good service to a large number of synchronous applications like *shared whiteboards*, *tele-pointers or file sharing*. Furthermore, this throughput is reasonable for voice communication applications and even more if the reliability protocol (unicast retransmissions) is disabled.

5.3. Test 3: resiliency

In this third test category, we aimed to prove the routing tolerance to node failures. As the previous test, the application used was a file transfer over the unicast service of the middleware. We selected three metrics for this test: (i) *throughput* of the intermediate nodes, (ii) *retransmissions per second* in the source node and (iii) *route recovery time*.

The scenario consisted of four nodes making up a rhomboid topology without mobility, as depicted in Fig. 6. Nodes 1 and 4 could not see each other because IPTables filter rules. A file was sent from nodes 1 to 4 through one intermediate hop (see Fig. 6). It is worth mentioning that node 2 was a netbook, with much less computing power. During the file transference, we halted the intermediate node (node 3) to verify that the routing system would self-repair.

As we can see in Fig. 7, at a certain moment of the file transfer (time = 14 s) the intermediate node (node 3) stops forwarding packets. When this happens, jOLSR detects this failure via the HELLO messages sent to node 3 that are not answered. While jOLSR is processing this new event, some retransmissions are sent from nodes 1 to 3 because the reliability protocol is still trying to deliver data packets to the old forwarder. As soon as jOLSR finishes processing the failure of node 3, two things happen: first, jOLSR recomputes and updates its data structures (LLIB, NIB, TIB and Routing Table) and sends a Topology Event to the upper layers. In second place, the protocol propagates this information via TC messages to all other nodes in the network. Thus, jOLSR is sure that the neighbors are aware of this topological change.

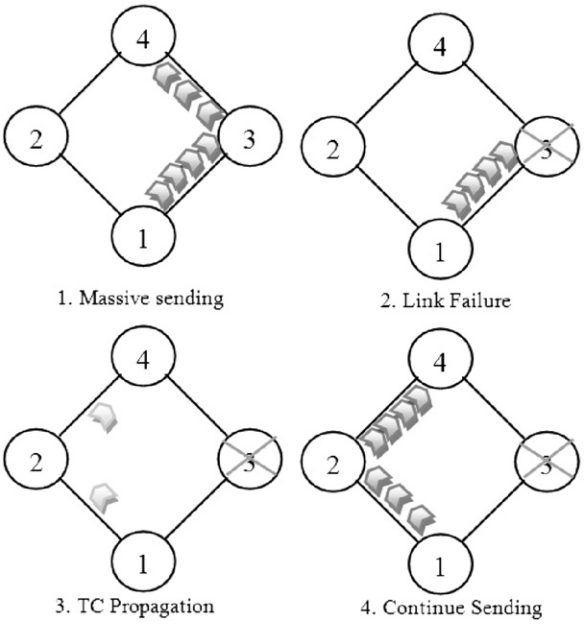

Fig. 6. Node failure scenario and self-repair process.

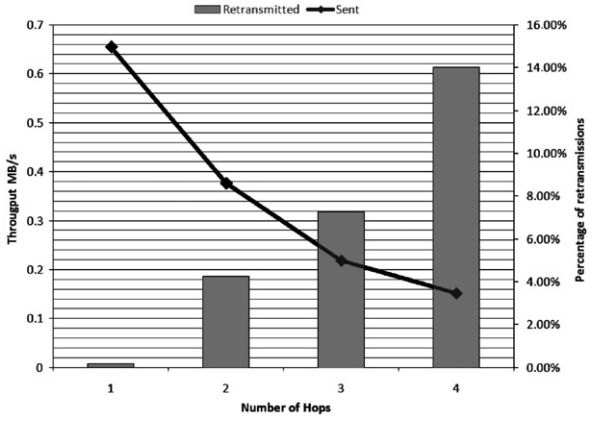

Fig. 5. Rate of retransmissions and throughput by hops.

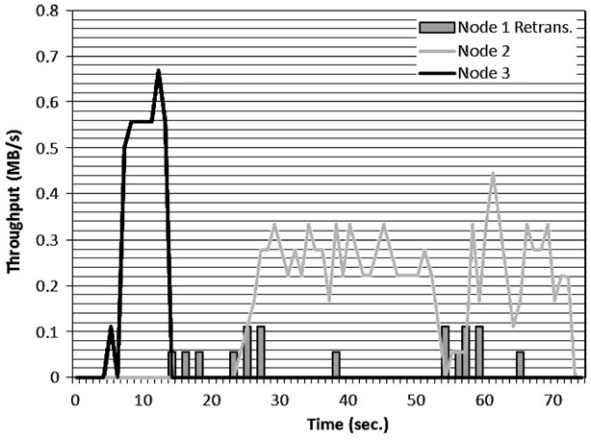

Fig. 7. MBps retransmitted by the sender and received coming from nodes 2 and 3.

This process lead to a route recovery time of about 9 s, due to (i) jOLSR TC and HELLO intervals, (ii) the medium access was shared and (iii) the Java processing time. The new path is less powerful (via a netbook) than the previous one. This can be observed in Fig. 7, as node's 2 throughput is almost the half of node's 3 throughput, but finally all the data arrived to the destination node (node 4). As a conclusion, the middleware is able to recover from node failures and to self-adjust the routes.

5.4. Test 4: mobility

One of the main challenges for a MANET protocol is to tackle with topology changes due to mobility. This fourth category of tests is intended to point out the flexibility of jOLSR for self-adapting to mobile topologies.

To facilitate user's mobility we developed an automatic multicast chat application. So, the user could walk while in its node a multicast message was generated every second. In addition, the application gave instructions to users in order to induce nodes to move as *Random Waypoint* mobility pattern.

The metric used was the *Packet Delivery Ratio* (PDR). Concretely, we measured the number of received packets over the total sent from the entire group. This metric is directly related with the time a node losses connectivity. Obviously, to measure this metric we had to disable the reliability protocol to avoid resending lost packets.

Due to the inherent complexity of mobile trials, we decided to simplify the testbed. On the one hand, we chose a testbed area without external wireless interferences—from other networks—which also was a typical MANET scenario: a leafy park (see Fig. 8). On the other hand, we used a homogeneous hardware (HP NX6125 laptop whose internal wireless card was a Broadcom 802.11b/g). Furthermore, the node's wireless card transmission power was set to 25% in order to reduce its radio range and therefore create hops between nodes easily (21 dBm provides a radio range of about 111 m in free space and 44 m in a low density forest).

The mobility tests were conducted in four different phases:

- Nodes were spread across the area with a distance among them about 30–50 m. As we empirically tested, this is a long enough distance for losing the signal of farthest nodes. In consequence, the routing protocol was induced to create hops and update the routing table often.
- Once the nodes were placed correctly, users joined the same multicast group and each node started to send *one multicast message per second*.
- People carrying laptops started to walk following the instructions from the software. These instructions simulated a *Random Waypoint* mobility pattern in order to give to the user the path to follow. The instructions were generated randomly and the possibilities were: remain quiet or walk in one of four directions. People walked at a speed of about 1.7 m per second (6 km/h). In case of obstacles, people could change direction and speed at their will. The interval between two instructions was a random value between 5 and 9 s. This led us to a movement ranges from 8 to 15 m per instruction. Hence, two nodes could increase its distance in 30 m, enough to become out of range.
- After 90 s, all the software was shut down and all the logs collected for the future analysis.

In the following subsections we describe three mobility test scenarios. In the first one all nodes moved according to the aforementioned mobility pattern, while in the others some nodes moved and others remained static.

5.4.1. Total mobility topology

This test represents a MANET scenario where all the nodes are mobile. The nodes were distributed over the test ground covering about 75% of the defined area (Fig. 8). In consequence, some of the links between nodes had two hops length.

The objective of this scenario was to test the worst mobility case, where every node moves and connectivity changes frequently.

Collected results are presented in Fig. 9. As we can see, the average PDR is quite positive (*about* 80%) keeping in mind the distances between nodes, the wireless card power and the absence of any kind of reliability protocol. At different points of the test links were broken and until the protocol's recuperation, several packets were lost. This fact represented a 20% of lost packets, varying for every node according to its location and obstacles nearby. This results are better than we expected, taking into account the high mobility.

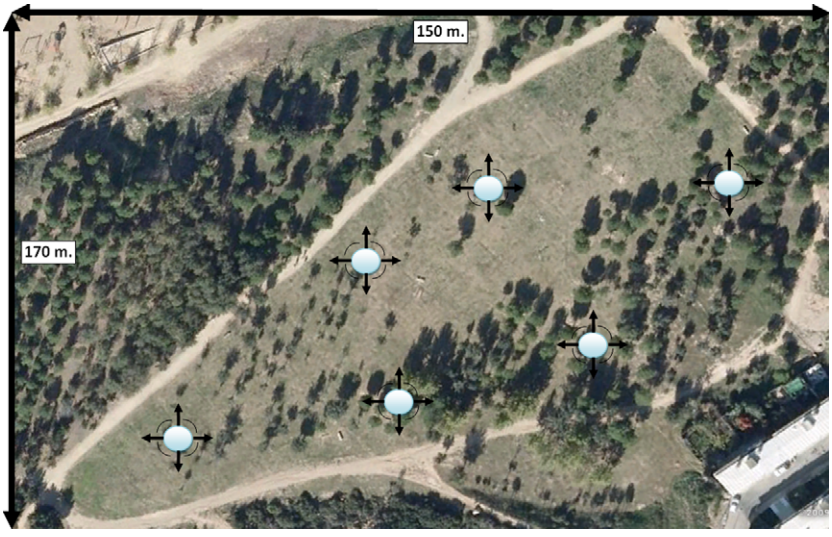

Fig. 8. Mobility test scenario (41°7′57.40″N, 1°14′56.30″E). The play ground was almost free of external wireless interferences. The main handicap for the wireless signal reception was the young forest over the area and the distance between nodes.

5.4.2. Static extremes with central mobility

This test forced the routing protocol to work in a situation where the nodes' links were as weak as possible.

Two nodes (nodes 1 and 6) were placed statically at the extremes of the test area. The rest of nodes were placed in the middle area, like the previous experiment, but covering the 100% of the ground. Nodes in the middle zone started to move randomly as described before. In such scenario, paths of three hops length were built. Because the signal weakness, the ground extremes' nodes lost the communication with the group more often than in the previous test. This can be observed in Fig. 10, where extreme nodes achieved a 43% PDR. The rest of nodes have also dropped its performance about a 12%, due to packet loss with nodes 1 and 6. PDR between central nodes should be the same as in the previous test. Now we have reduced the mobility and the results are worst, but this is because we have forced the longest geographical distance. Nevertheless, path recovery is achieved whenever a node becomes in range.

5.4.3. Static center with external mobility

In this last test, the majority of nodes were static except one node—satellite node—which had a high mobility. Therefore many links were available and working all the time.

The setup was as follows: five nodes were spread statically conforming a sparse ring shape and occupying almost 100% of the test area. Every node in the ring had two nodes in radio range. On the contrary, node 6 moved clockwise outside the ring as a *satellite* in order to route messages through a different node frequently.

From the results shown in Fig. 11, we can conclude that topology changes of the satellite node were done well enough because its PDR is similar to the other nodes. It is worth mentioning that PDR variation observed in the different nodes can be due to trees attenuation and signal weakness. The results

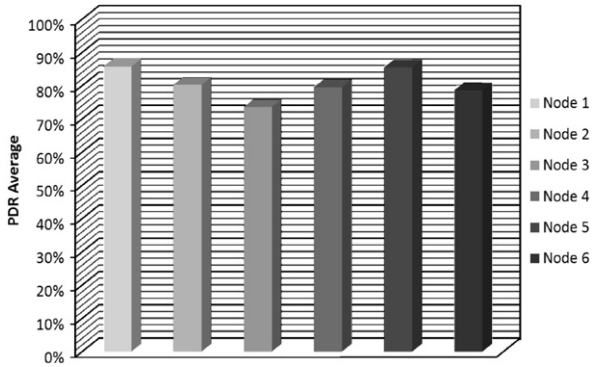

Fig. 9. PDR evaluation in a total mobility scenario.

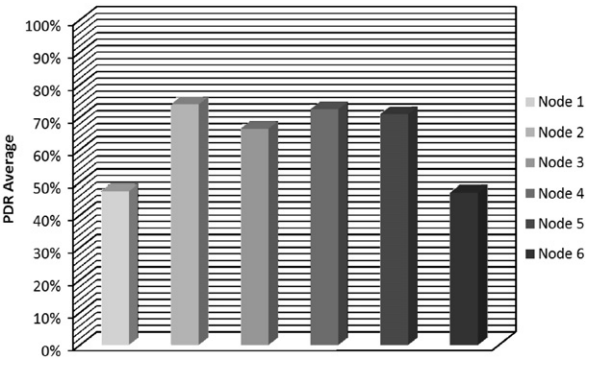

Fig. 10. Results obtained with static extremes and mobility in the middle area.

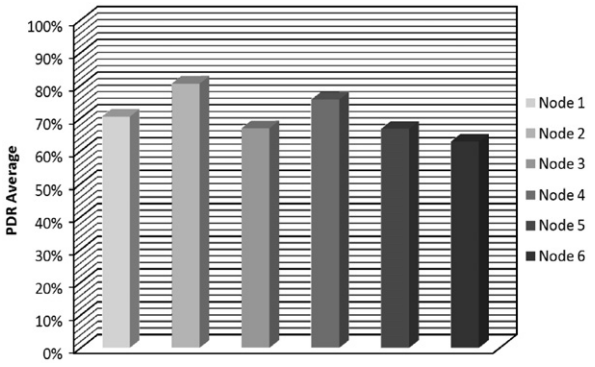

Fig. 11. PDR evaluation for the static center with external mobility.

obtained in this scenario are better than in the previous one, because the rotation of the satellite facilitates a rapid recuperation through the next neighbor, while a random walk can isolate fixed nodes during longer periods of time. When comparing to the first mobility test, results are worse (average PDR of 70% versus a PDR of 80%). There are two main causes: (i) in the first test nodes covered about 75% of the test area, and (ii) every node moved randomly and therefore nodes remained randomly spread through the test area, reducing chances to lose coverage.

We can conclude that the three mobility tests demonstrated the feasibility of MChannel in mobile scenarios. This complete validation of the middleware has proven its fault tolerance and adequate performance numbers in both static and mobile tests. Applications received a good service and topology information but, as pointed out in the architecture description, it has some cost. So, in the following section we analyze in more detail the overhead and performance of routing at user space compared with routing at network layer.

6. Routing layers comparison

In this section, we evaluate some aspects of our routing protocols (jOLSR and OMOLSR) and we compare experimentally and analytically some important metrics against their natural counterparts (OLSRd and MOLSR). The aim of these tests is to compare the user space routing with the network level routing in terms of overhead and throughput.

6.1. jOLSR versus OLSRd

In our middleware we have moved the routing logic to the user space. Therefore, an exhaustive and detailed comparison with a network layer protocol as OLSRd is compulsory. Regardless there are a plenty of metrics that are used for measuring different aspects of a protocol performance, we focused our efforts in three main metrics: *Average Packet Size*, *Average Bandwidth Consumption*—overhead tests—and *throughput*—performance tests. All the experiments were done exactly equal for OLSRd and jOLSR in order to compare fairly and obtain determinant results (see Fig. 12).

Test Hardware. Each node in the network was an HP NX6110 laptop. The internal wireless card was an Intel PRO/Wireless 2200 BG card for IEEE 802.11b/g. During the experiments, wireless cards on laptops were set to channel 1 at the 802.11b/g band, using auto rate (54 MBps), transmission power 1 dBm and RTC/CTS off.

Test Software. All nodes used Ubuntu 9.04 as Operating System. We configured the IPTables firewall of this O.S. to create virtual low-level hops, so that messages of nonvirtual neighbors were dropped by the firewall working below the routing protocols. The objective of these *virtual hops* was to create a controlled and

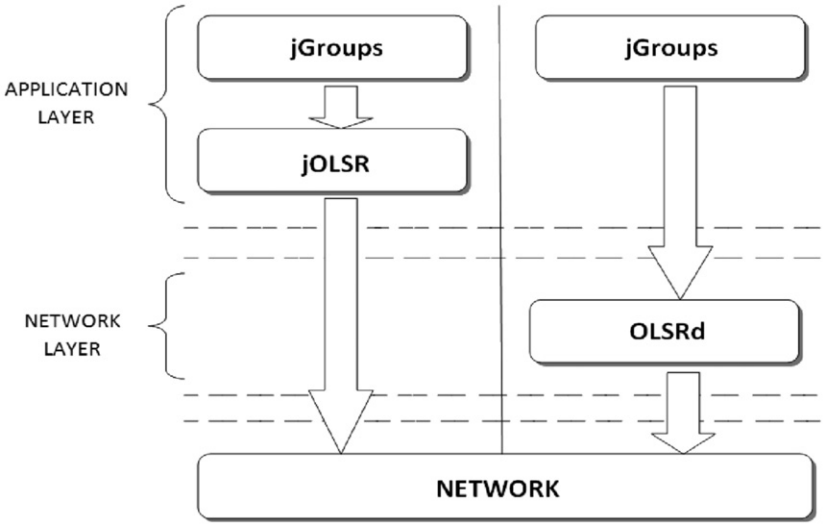

Fig. 12. Protocol placement of our experiment. All tests were executed from JGroups in order to obtain the same software overhead.

defined topology without placing physically the computers far from each other. OLSRd version used was OLSRd 0.5.6-r5. We also used MChannel over JGroups 2.5.2 and Java 6 Runtime Environment. Finally, we used Wireshark to monitor and capture all the network traffic and we compared its captures with our own logs.

6.1.1. Protocol overhead test

This test consisted of capturing and logging the traffic generated by the MANET protocols and without any other application traffic. We increased the number of members involved into the group progressively. In order to ensure the results reliability each test was repeated twice. The duration of each test was also the same for all tests, being 1 min the selected test duration. This time is long enough to interchange several control packets.

There are two analyses about the protocol overhead: the first one illustrates the *bandwidth required* in order to maintain the group communicated (see Fig. 13) and the second one describes the growth in *size of control packets* related with the group size (see Fig. 14).

In Fig. 13 it can be observed that the jOLSR's bandwidth consumption is higher than that of OLSRd. The growth observed in OLSRd's bandwidth consumption is irregular but lower all along the experiment compared with our protocol. jOLSR maintains in its bandwidth consumption a linear growth with an initial overhead higher than OLSRd. The jOLSR's overhead ranged from 4.4 (2 nodes) to 1.7 (10 nodes) times higher than OLSRd. In other words, when the number of nodes increases, the difference is reduced drastically. In reference to the previous fact, we foresee that the bandwidth consumption differences will get smaller as the group becomes bigger.

We must take into account that the difference between both protocols is less than 1.3 kB/s in the worst case. If we consider that in an 802.11b/g wireless card it represents from 0.02% to 0.1% of bandwidth consumption for jOLSR (depending on the transmission rate), we conclude that this extra overhead can be assumed. Recall that this slight extra overhead is the price to pay for providing a plenty of high level services to the top level applications.

Packet size metric is another way to measure protocol overhead (see Fig. 14). It is useful in order to know whether control messages grow in a scalable way when the group becomes bigger or not.

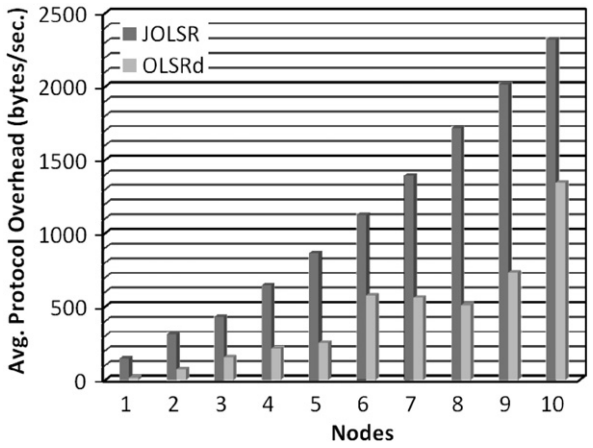

Fig. 13. Average Bandwidth Consumption overhead of OLSRd and jOLSR as the number of nodes increases.

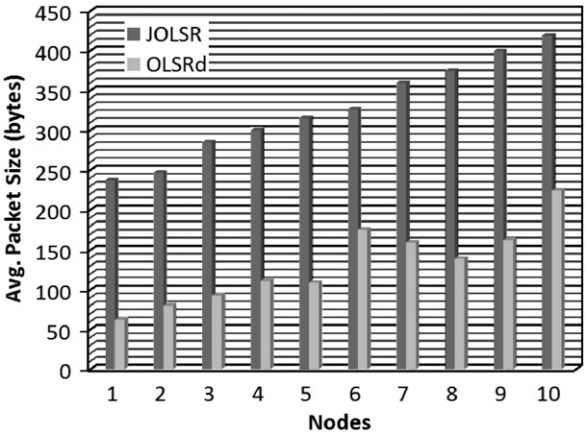

Fig. 14. Average Packet Size of OLSRd and jOLSR as the number of nodes increases.

In Fig. 14 we can see a noticeable difference between the default (1 node) communication packet sizes of both protocols. The explanation is easy: MChannel's architecture needs more headers—Application Layer, JGroups Protocols—in order to

encapsulate a single packet. The average difference between both protocols is approximately 2.7 times higher the size of the jOLSR's control packet than the OLSRd's one. As before, a strength of jOLSR is that this difference gets smaller while the number of nodes grows. It means that membership information is efficiently managed. Again, as in the previous analysis, in the context of nowadays wireless networks, a difference of almost 200 bytes in each packet—in the 10 nodes case—does not represent a considerable overhead. Specially, if we consider the benefits for upper layers and protocols.

6.1.2. Throughput test

This test consisted of monitoring and measuring the amount of user information per second that both protocols were able to send. This test extends and complements the test in Section 5.2. Nevertheless, in this test was used a hardware homogeneous scenario—without a bottleneck node.

For this second test we defined one of the most difficult topologies to maintain the bandwidth: the *in-line topology*.

The user application transferred a 20 MB file to a certain destination node measuring the achieved *throughput*. The mentioned destination node was further—increasing number of hops—each time. This suffices to stress the protocols and to achieve their maximum performance. The results obtained with both protocols are represented in Fig. 15.

The performance difference between jOLSR and OLSRd is not as bad as could be expected from a priori theoretical prediction, due to the Java execution overhead and user space routing. The average difference in throughput is −18.5%—on average 170 kBps—for jOLSR compared with OLSRd. However, from this experiment we can conclude that the throughput reduction is somewhat affordable. Furthermore, with the obtained results we can ensure that our middleware is able to provide a good service to a wide range of collaborative applications.

As expected, jOLSR introduces more overhead than OLSRd does, but this is not really significant to the available bandwidth. This is corroborated by the fact that differences in throughput are low. We assume that the mentioned drawbacks are the price to pay for providing a set of high level services to client applications.

6.2. OMOLSR versus MOLSR

One of the main goals of our middleware is the topology awareness of all layers above the routing protocol. For this reason, we developed OMOLSR as the default application level multicast of MChannel. OMOLSR is intended to manage efficiently multicast groups without any additional control messages overhead, just taking advantage of the topological information from jOLSR. This is possible because both are placed at application level.

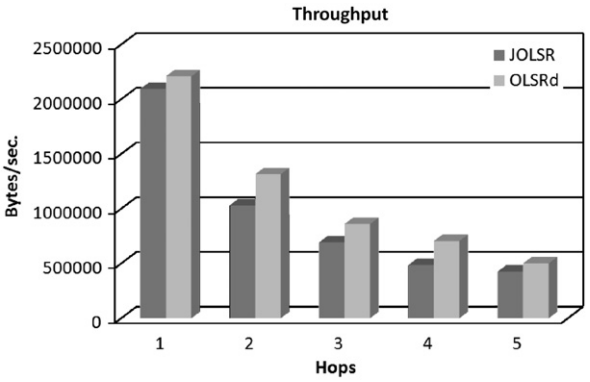

Fig. 15. Average throughput comparison.

In this section we evaluate analytically how MOLSR would work in the Application Level as it is defined in Jacquet et al. (2001). This example will give us an argument strong enough for considering OMOLSR as a good choice in order to give multicast communication in our middleware. Again, the main objective of this evaluation is to point out and prove the motivation for creating OMOLSR and reasoning why it is suitable for our architecture.

Simple Flooding and MPR Flooding Strategies. As is deeply detailed in Qayyum et al. (2002), the MPR Flooding Strategy *reduces drastically* the number of control messages sent and outperforms in many other metrics the Simple Flooding Strategy. Although this fact demonstrates that MOLSR is more efficient than SMOLSR, in our opinion the amount of control messages sent by MOLSR could be avoided using a multicast protocol that benefits directly of the topological information given by the routing protocol.

Assumptions:

- All nodes of the hypothetical topology are multicast capable with one wireless card per node.
- The number of messages that we will count for the multicast behavior is the theoretical for the Application Level. It means that there is not real multicast: each multicast data packet forwarded is split into unicast packets, corresponding to the forwarder's children.
- In order to reduce the complexity of the example we suppose just one multicast group in the network whose source is the root.
- If both protocols create the multicast tree correctly, we can assume that the cost in messages for delivering a multicast data message is the same for both at the Application Level.
- The hypothetical topology (Fig. 16) is defined as follows: the tree has as root the source node which relies in its own MPR. From the first MPR onwards, the tree grows regularly being a group of four nodes (2 MPR and 2 Not MPR) the set of nodes which each MPR is responsible for.

Given a protocol which relies in the MPRs for flooding the network, we can define that the overall number of messages needed for disseminate a message to the whole network is: $f_{MPRBroadcast}$:

$$\sum_{l \in L} |MPR(l)|$$

where L represents *Levels of the topological tree* and $MPR(l)$ represents *MPRs for the tree level l*.

Note that in the case of the hypothetical topology of Fig. 16 the overall number of messages needed to send a broadcast message through tree levels grows binary exponential.

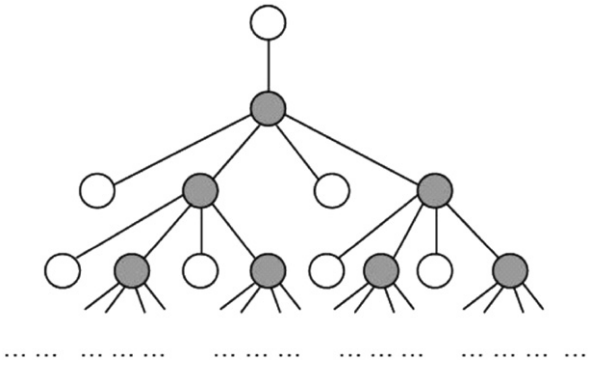

Fig. 16. Hypothetical topology used in order to exemplify the behavior of multicast protocols.

Given a network which contains a set of multicast members joined to a group, the overall number of messages needed for delivering a single multicast message to the entire group is:

$f_{Multicast}$:

$$\sum_{l \in L} |GM(l)|$$

where *GM(l): Multicast Group Members* (*participants or subscribers*) *for the level l.*

MOLSR and OMOLSR use the multicast tree generated for delivering the multicast messages to the entire group efficiently. Fig. 17 represents the overall amount of messages generated in order to deliver a multicast message depending on the percentage of subscribers. Furthermore, this graphic represents the worst case possible for the multicast data delivery in a network structured as Fig. 16: the subscriber nodes are not MPR as possible and homogeneously distributed in the network.

MOLSR uses flooding in order to deliver two types of control messages (*mc_claim* and *source_claim*) which are described in the protocol's definition. Moreover, MOLSR also needs to send messages periodically in order to receive the confirmation of the children nodes which still being part of the multicast tree (*parent_confirm*) for building it. This kind of messages has the same bandwidth consumption than sending a multicast data message. Finally, we can define the overhead in messages that MOLSR generates for building and maintaining a *single multicast tree* as follows:

$$f_{Overhead\ In\ Messages} = (f_{MPR\ Broadcast} * 1/t1 * \mu) + (f_{MPR\ Broadcast} * 1/t2) + (f_{Multicast} * 1/t3) + ((f_{MPR\ Broadcast} + f_{Multicast}) * \alpha)$$

where

$1/t\,1$: *Frequency of mc_claim broadcasting in function of period* $t1$.
$1/t\,2$: *Frequency of source_claim broadcasting in function of period* $t2$.
$1/t\,3$: *Frequency of parent_confirm multicast sending in function of period* $t3$.
α: *Number of topology changes triggered by OLSR which implies a source_claim + confirm_parent messages sending.*
μ: *multicast capable nodes.*

We use the above definition of $f_{Overhead\ In\ Messages}$ as a metric to count the *number of control messages* generated by the protocols.

With the previous definitions we can evaluate the hypothetical behavior of MOLSR multicast protocol. We depict different parameterizations of frequencies and number of topology changes in Figs. 18–20 using the settings specified in Table 1. The metric represented is the calculation of the overall messages sent into the network classified by message type.

As we can observe in Figs. 18–20 the proactive nature of MOLSR has a non-negligible overhead in messages across the network. The amount of control messages sent by MOLSR depends specially on the parameterization of the protocol and the timers (t 1, t 2, t 3). In all three figures OMOLSR Data and MOLSR Data (first two columns to the left) remain the same and are a good reference. Keeping in mind that this overhead is calculated just for building and maintaining one multicast group, we observe that *source_claim* and *confirm_parent* messages would grow linearly depending on the number of nodes.

On the other hand, Fig. 20 shows that when a group becomes bigger and the timers are short to improve the multicast information convergence, the amount of control messages sent through the network is quite high (about 15,000). This means that the number of *mc_claim* messages has grown up 4 times, being this number higher than the number of data packets, almost 3 times higher. It is very important to keep in mind that in wireless networks medium access is shared and is a real bottleneck.

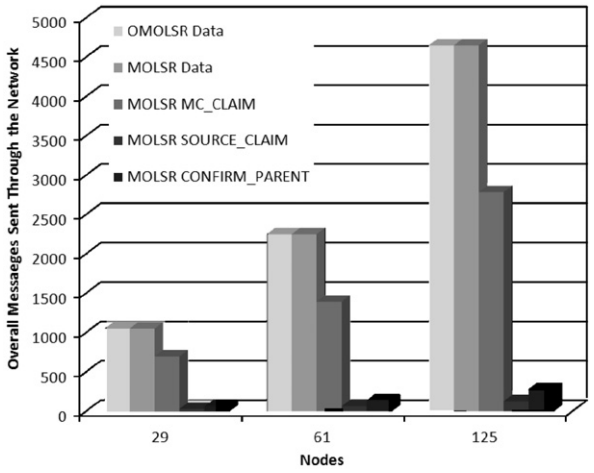

Fig. 18. First case where the timers are long and there are not topological changes in the network.

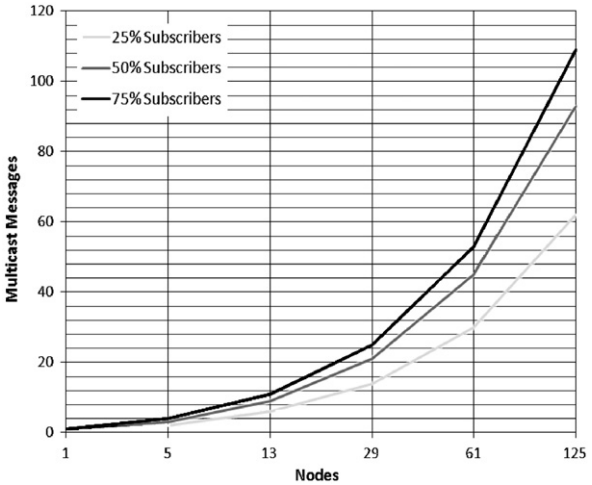

Fig. 17. Amount of messages sent through the network in order to deliver a single multicast message depending on the percentage of subscribers and the number of nodes given a topology as Fig. 16.

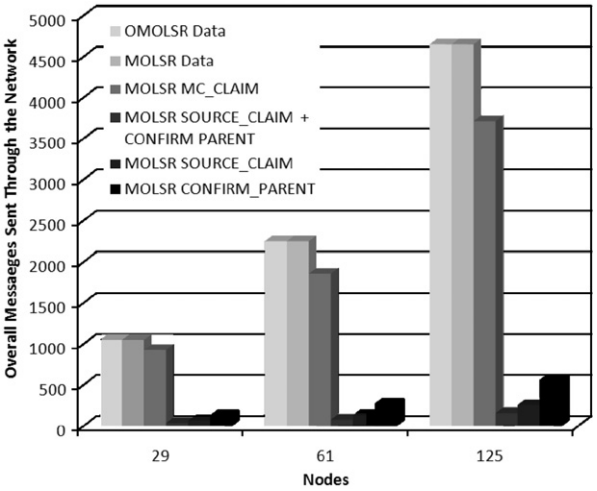

Fig. 19. Second case where the timers are shorter and there is one topological change.

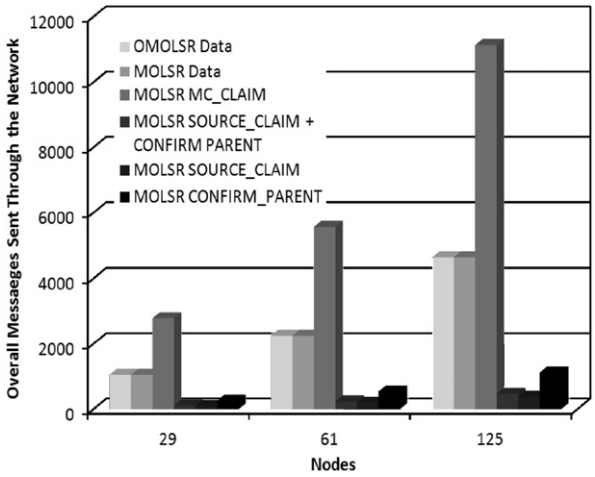

Fig. 20. Worst case: the timers are the shortest parameterized and there are three topological changes.

Table 1
Evaluation settings.

Parameter	Fig. 18	Fig. 19	Fig. 20
Execution time (s)	60	60	60
Data messages	50	50	50
Topology changes (α)	0	1	3
Subscribers percentage (%)	50	50	50
mc_claim period (t 1) (s)	40	30	20
source_claim period (t 2) (s)	30	20	15
confirm_parent period (t 3) (s)	20	10	5

Although *mc_claim* messages are much shorter than data packets, but to access the medium to send these short messages can cause a deep drop in available throughput.

Obviously, OMOLSR needs information in order to build its multicast tree. The membership information of all nodes is attached in each jOLSR TC message and it is disseminated periodically. Regardless this approach has an extra data overhead, the growth of the control packet size depending on the number of nodes is absolutely affordable as is shown in Section 5.1.

It is noticeable the fact that while the amount of MOLSR control messages grow linearly depending on the number of groups, the number of TC messages does not. This is so because the growth of the TC message depending on the number of a node's groups adds few bytes for each group joined: just the size of the group identifier. The computation of the multicast tree in OMOLSR is done by an implementation of the Dijkstra's algorithm. This algorithm is widely used in link-state routing protocols. The CPU overhead that OMOLSR generates in order to calculate the multicast tree in a MANET group is not significant. Nevertheless, the OMOLSR overhead is not just for building a multicast tree with a partial view of the network. The fact that the membership of each node is attached in the TC messages implies that *each node has a complete view of the network*. This is a *high level service* that could be used by the top level applications.

7. Conclusions

A MANET middleware with dependencies in the kernel space (cross-layering) cannot be easily ported to heterogeneous environments. The main argument of this paper is, thus, to move the routing protocols to the user space to simplify the development, testing, deployment and portability of middleware and applications. In particular, we extended a popular group communication toolkit (JGroups) to support MANET settings. We implemented a Java version of the OLSR protocol (jOLSR) and a novel multicast protocol (OMOLSR) that benefits from jOLSR. Furthermore, we created a MChannel group communication abstraction where group membership is obtained from multicast membership information and failure detection from jOLSR probing. Our Java middleware provides complete portability and simple development of collaborative MANET applications.

To evaluate the feasibility of our approach we have performed an extensive validation of our middleware in real testbeds. More concretely, we measured the throughput, fault tolerance, mobility and overhead of our middleware. We obtained reasonable throughput numbers of 1 MBps (two hops) or 600 kBps (three hops) for our Java middleware. We also obtained good Packet Delivery Ratios (close to 80%) and strong resiliency in mobile scenarios. As we can see, in MANET networks the throughput considerably decays with the number of hops. If applications ignore the topology and just broadcast information, they can really harm the overall performance of the network. For this reason, we believe that many MANET applications require topology-awareness in order to optimize their data transmissions.

The performance penalty paid by the Java middleware is thus justified for the portability and easy deployment of applications. The throughput results make our approach feasible for a wide range of applications like spontaneous collaboration, interactive games or context-based and location-based services. In particular, our approach will be specially interesting in the following years when mobile terminals like smart phones with wireless connectivity will be widespread. The heterogeneity of these devices will require portable solutions for infrastructure-less communications. As future work we plan to make jOLSR energy-aware and thus supporting the constraints of mobile terminals.

Acknowledgements

This work was partially funded by the Spanish research project P2PGRID, ref. TIN2007-68050-C03-03, of the Spanish Ministry of Science and Innovation. This work was also partially funded by the European research project POPEYE, ref. IST-2006-034241. Special thanks to Jordi Pujol, Roc Messeguer and Esunly Medina for their work and energy.

All the source code is freely available for download at http://ast-deim.urv.cat/mchannel. In this site the reader can find a link to a YouTube video of the *real working group* MANET experiment.

References

Batagelj V, Mrvar A. Pajek: program for large network analysis. Connections 1998;21:47–57.

Bouckaert S, Naudts D, Moerman I, Demeester P. Making ad hoc networking a reality: problems and solutions. Journal of Telecommunications and Information Technology 2008;1:3–11.

Calafate CMT, Manzoni P. A multi-platform programming interface for protocol development. In: Proceedings of the eleventh euromicro international conference on parallel, distributed, and network-based processing; 2003. p. 243–9.

Chakeres I, Perkins C. Dynamic manet on-demand (dymo) routing. Available: 〈http://tools.ietf.org/html/draft-ietf-manet-dymo-04〉; 2004.

Clausen T, Jacquet P. Optimized link state routing protocol (OLSR). RFC 2003;3626: 1–75.

Conti M, Crowcroft J, Delmastro F, Passarella A. P2P support for group-communication applications: a cross-layer approach for manet environments. In: 25th IEEE conference on computer communications; 2006.

Conti M, Gregori E, Turi G. A cross-layer optimization of gnutella for mobile ad hoc networks. In: Proceedings of the 6th ACM international symposium on mobile ad hoc networking and computing; 2005. p. 343–54.

Conti M, Maselli G, Turi G, Giordano S. Cross-layering in mobile ad hoc network design. Computer 2004;37(2):48–51.
Delmastro F. From pastry to crossroad: cross-layer ring overlay for ad hoc networks. In: Third IEEE international conference on pervasive computing and communications workshops; 2005. p. 60–4.
Friedman R. Jazzensemble: a group communication middleware for manets. In: Proceedings of the second MiNEMA workshop; 2004.
Ge M, Krishnamurthy SV, Faloutsos M. Application versus network layer multicasting in ad hoc networks: the alma routing protocol. Ad Hoc Networks 2006;4(2):283–300.
Goldsmith A, Wicker S. Design challenges for energy-constrained ad hoc wireless networks. Wireless Communications 2002;9(4):8–27.
Gui C, Mohapatra P. Efficient overlay multicast for mobile ad hoc networks. Wireless Communications and Networking 2003;2:1118–23.
Hadim S, Al-Jaroodi J, Mohamed N. Trends in middleware for mobile ad hoc networks. JCM 2006;1(4):11–21.
Jacquet P, Mühlethaler P, Clausen T, Laouiti A, Qayyum A, Viennot L. Optimized link state routing protocol for ad hoc networks. In: Proceedings of the first IEEE international multi topic conference; 2001. p. 62–8.
Jacquet P, Minet P, Laouiti A, Viennot L, Clausen T, Adjih C. Multicast optimized link state routing. Available: 〈http://tools.ietf.org/html/draft-jacquet-olsr-molsr-00〉; 2001.
JGroups Project. Available: 〈http://www.jgroups.org/〉.
Kawadia V. System services for ad hoc routing: architecture, implementation and experiences. In: Proceedings of the first international conference on mobile systems, applications and services; 2003. p. 99–112.
Lin X, Shroff NB, Member S, Srikant R. A tutorial on cross-layer optimization in wireless networks. IEEE Journal on Selected Areas in Communications 2006;24:1452–63.
Mottola L, Cugola G, Picco GP. A self-repairing tree topology enabling content-based routing in mobile ad hoc networks. IEEE Transactions on Mobile Computing 2008;7(8):946–60.
Olsrd: The OLSR daemon. Available: 〈http://www.olsr.org/〉.
Qayyum A, Viennot L, Laouiti A. Multipoint relaying for flooding broadcast messages in mobile wireless networks. In: Proceedings of the 35th annual Hawaii international conference on system sciences; 2002. p. 3866–75.
Ramdhany R, Grace P, Coulson G, Hutchison D. Manetkit: Supporting the dynamic deployment and reconfiguration of ad hoc routing protocols. In: Proceedings of the tenth ACM/IFIP/USENIX international conference on middleware, vol. 5896; 2009. p. 1–20.
The Wireshark Project. Available: 〈http://www.wireshark.org/〉.
Tschudin C, Gunningberg P, Lundgren H, Nordström E. Lessons from experimental manet research. Ad Hoc Networks 2005;3(2):221–33.
Yoneki E, Bacon J. Distributed multicast grouping for publish/subscribe over mobile ad hoc networks. Wireless Communications and Networking Conference 2005;4:2293–9.