
	<p style="text-align: center;"><b>MOBILEMAN</b></p> <p style="text-align: center;">IST-2001-38113</p> <p style="text-align: center;">Mobile Metropolitan Ad hoc Networks</p>
-----------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<p><b>MOBILEMAN</b></p> <p><b>MobileMAN Functionalities – Enhanced Set</b></p> <p>Deliverable D11</p> <p>Contractual Preparation Date: October 2004  Actual Date of Delivery: 16 November 2004  Estimated Person Months: 18  Number of pages: 15</p> <p><b>Contributing Partners:</b> Consiglio Nazionale delle Ricerche (Italy), University of Cambridge, Eurecom Institut (France), Helsinki University of Technology (Finland)</p> <p><b>Authors:</b> Marco Conti, Franca Delmastro Giovanni Mainetto (CNR), Jose Costa-Requena, Jarrod Creado, Mohhammad Ayyash (HUT), Jon Crowcroft, Ziran Sun (Cambridge), Claudio Lavecchia, Pietro Michiardi (Eurecom)</p>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<p><b>Abstract:</b> The aim of this deliverable is to provide the software that implements, on the Linux operating system, the functions required to set up a MANET. Specifically, we are delivering the set of functionalities that have been extensively tested during the MobileMAN First Phase testing (see Deliverable D8) which enabled us to fix software problems/errors, and identify integration problems and errors. Specifically, we include in this deliverable i) the revised ad hoc routing framework that enables us to construct a multi-hop ad hoc network, ii) a p2p middleware platform based on Pastry (<i>FreePastry</i>) that enables us to run simple p2p testing applications on top of the multi-hop ad hoc network. In addition, we also deliver the additional functionalities that have been completed at the end of the second year for which preliminary tests have been reported in Deliverable D10. Specifically, these include a VoIP and a whiteboard application for the multi-hop ad hoc network, and a watchdog mechanism required to implement the cooperation enforcing mechanism. The software we are delivering is contained in the CD ROM associated with this deliverable and which is also made available in the Software web site <a href="http://keskus.hut.fi/tutkimus/MobileMan">http://keskus.hut.fi/tutkimus/MobileMan</a>.</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<p><b>Project funded by the European Community under the “Information Society Technologies” Programme (1998-2002)</b></p>
-------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------

## SUMMARY

The aim of this deliverable is to provide the software that implements, on the Linux operating system, the basic functions required to set up a relatively small campus-wide MobileMAN to be used during the MobileMAN preliminary testing phases (see Deliverables D8 and D10). To this end we mainly concentrated on the implementation and testing of the network and middleware layer functionalities. Specifically, the software we are delivering includes: i) the revised ad hoc routing framework that enables us to construct a multi-hop ad hoc network, ii) a p2p middleware platform based on Pastry (*FreePastry*) that enables us to run simple p2p testing applications on top of the multi-hop ad hoc network. After, the extensive tests performed during the MobileMAN First Phase testing the software was updated to fix software problems/errors, and integration problems/errors. In addition, we also deliver the additional functionalities that have been completed at the end of the second year for which preliminary tests have been reported in Deliverable D10. Specifically, these include a whiteboard and VoIP application for the multi-hop ad hoc network, and a watchdog mechanism required to implement the cooperation enforcing mechanism. In this document, we briefly present the main characteristics of software functionalities we are delivering. More details can be found in Deliverables D5 and D10. The software we are delivering is contained in the CD ROM associated with this deliverable and which is also made available in the Software web site <http://keskus.hut.fi/tutkimus/MobileMan> (maintained by HUT).

## CONTENTS LIST

1. AD HOC FRAMEWORK.....	4
1.1 Ad Hoc Framework.....	4
1.2 Common modules.....	5
2. MIDDLEWARE.....	8
2.1 FreePastry and the Distributed Messaging application.....	8
3. WATCHDOG.....	10
3.1 Architecture.....	10
3.2 Functional Description.....	11
4. VOIP.....	12
4.1 Signaling module.....	12
4.2 Data transport module.....	12
5. WHITEBOARD.....	13
5.1 Bamboo.....	13
5.2 Multicast Layer.....	13
5.3 Whiteboard Application.....	14
6. REFERENCES.....	15

## 1. AD HOC FRAMEWORK

The Ad Hoc routing framework is a software package, which can support different Ad Hoc networks routing protocols, such as proactive, reactive and also some hybrid solutions. The Ad Hoc routing framework can be installed in a node (PDA or Laptop) that runs Linux Operating System. With this framework, we could add new routing protocols and other functionalities, such as naming and service discovery. This section describes the existing components of the Ad Hoc framework and all the subsystems including interfaces and the basic functionalities implemented in the framework.

The framework has been designed in separate components with clearly defined interfaces. This allows an easy integration of these components and the possibility of adding new functionalities. The benefit of this component-based or plug-in design is that each individual component could change its internal implementation while all components can still be integrated together if the interfaces are kept consistent. This allows having a complex system, which can be divided into small modules that are easy to implement.

### *1.1 Ad Hoc Framework*

The framework provides general functionalities for both proactive and reactive routing protocols. The existing framework includes a reactive protocol (e.g. AODV [1]) and a proactive protocol (e.g. OLSR [2]). In order to test the framework implementation with constrained devices, in addition to laptops the framework is integrated into a small number of Personal Digital Assistants (PDA) nodes (iPAQ). The iPAQ is a mobile node where the original operating system is PocketPC 2002. The Operating system has to be changed to Linux in order to integrate the framework. Linux supports the ARM architecture, which is used in the iPAQ.

We use the Familiar [3] operating system, which is a Linux portable version used on iPAQ. It is a tailored Linux version to fit into limited resources of mobile devices. Because of very little space for the file system, we cannot install the Linux kernel source and a compile tool chain into iPAQ. This means we cannot build native executables on iPAQ directly. The default Familiar kernel does not include Netfilter (an operating system services for manipulating IP packets), but we need it for the framework. For these reasons, we have to compile everything in a Linux [4] PC to make ARM executables. Then, we transfer them to iPAQ and run them. This can be achieved by using a GNU/gcc [5] compile tool chain for ARM.

The Ad Hoc Framework consists of four subsystems; the Common Cache Registry Server, the Reactive modules, the Proactive modules and the Hybrid modules. The Common Cache includes all the modules that must be kept constantly running in the node since they store routing information and other data used by the other modules. The Reactive modules consist of the software modules that implement the reactive routing protocols (e.g. in this case the Reactive module consists of the module that implements AODV). The proactive modules consist of the software modules that implement proactive routing protocols (e.g. in the actual framework the proactive modules contain only a software module that implements OLSR). Finally, the hybrid modules include all modules that will implement hybrid routing protocol such as ZRP. The four modules of the Ad Hoc framework consist of independent software components that implement specific routing protocols and store routing information into a single cache. The Ad Hoc framework architecture is shown in Figure 1.

- Common modules: Common Cache Registry Server, Common Cache and Registry.
- Reactive routing modules: AODV module.

- Proactive routing modules: OLSR module.
- Hybrid routing modules.

## 1.2 Common modules

The common modules consist of a Registry, the Common Cache and the Common Cache Registry Server that communicates with the independent routing modules. The Common Cache keeps routing and other information collected by the routing protocols that are running simultaneously in the node. The Registry stores information of the routing protocols running in the node.

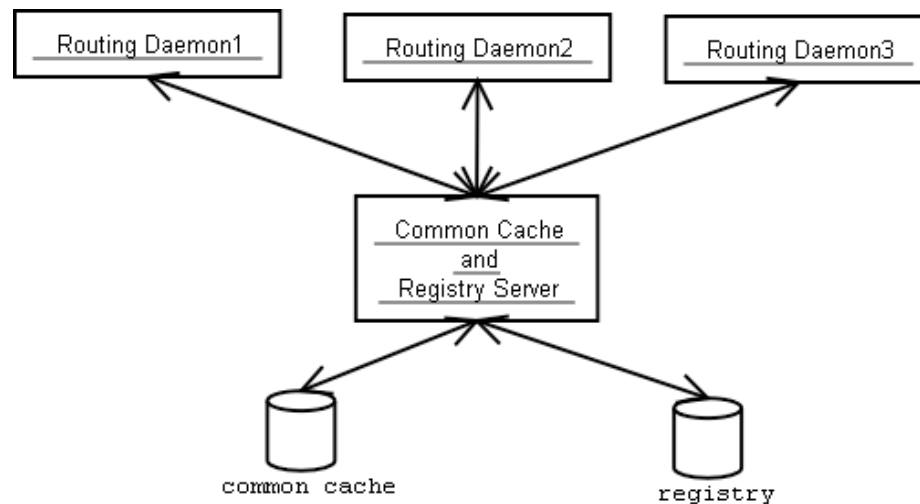


Figure 1 Ad Hoc framework architecture

The aim of the Ad Hoc framework design is to contain several independent routing protocols running simultaneously as daemons (the actual implementation of the framework contains two routing daemons: AODV and OSLR). The Common Cache and Registry Server (CCRS) act as the front end of two data repositories: the Common Cache and the Registry. The routing protocols running daemons act as clients to the Common Cache and Registry Server for accessing the Common Cache and the Registry.

The Common cache stores all the route information that the CCRS receives from all routing protocols running in the node as daemons. The Registry contains the state information of all routing protocols daemons (e.g. active, inactive, etc). The state information consists of configuration parameters of the protocol that is active but also other parameters for sending messages to the daemon in order to change its configuration during runtime. This allows the implementation of new routing algorithms by using existing routing protocols that are already running (e.g. hybrid routing protocols such as ZRP could be implemented using existing AODV and OLSR daemons).

### 1.2.1 Common Cache Registry Server

The Common Cache Registry Server (CCRS) is one of the most important common modules in the Ad Hoc framework. The CCRS keeps listening to specific messages coming from the routing protocols daemons that want to communicate either with the Common Cache or with the Registry.

Thus, when one routing protocol daemon starts running, it must upload its state information and other protocol parameters into the registry through CCRS. Moreover, during the lifetime of the

routing protocol it updates its own routing table and the routing information in the common cache through the CCRS. Thus, the Common Cache always stores the routes discovered by the separated protocols running in the device simultaneously. Meanwhile, the registry keeps the latest state information of the routing protocol daemons running in the node.

When one routing daemon wants to know the status of other daemons, it can ask the information from CCRS. Different daemons can also communicate with each other through CCRS.

The communication between the daemons and the CCRS is implemented by different messages defined in request/reply format. The CCRS server and the routing protocol acting as clients are running in different processes and they communicate through a well-known port.

### 1.2.2 Reactive modules

The reactive modules consist of the software components that implement reactive routing protocols (e.g. AODV).

#### **AODV Module**

This section describes a reactive module that implements the specific AODV reactive routing protocol. We select AODV because it is the most widely implemented reactive algorithm and might become the IETF standard.

The AODV module is based on UU-AODV [6]. The actual implementation of the Ad Hoc framework contains version 0.8 of the UU-AODV, which has been improved and is stable. The aim of the Ad Hoc framework is not to re-implement existing protocols but integrate existing implementations and reuse as much as possible existing implementations that are already tested and debugged into a single framework. The existing UU-AODV implementation has to include an additional component to communicate with the CCRS implemented as part of the framework. Thus, the UU-AODV is included into the framework as independent routing module that only provides the AODV protocol logic. An additional component named CCRS proxy is added within the UU-AODV to slightly modify the AODV behavior. The CCRS proxy will store the routes discovered by AODV into the CCRS so the routes are available to the Linux kernel routing table for the node communications.

### 1.2.3 Proactive modules

The proactive modules consist of the software components that implement proactive routing protocols (e.g. OLSR).

#### **OLSR**

OLSR is a proactive routing protocol that periodically sends control messages to maintain the knowledge of the network topology. OLSR protocol is a link state protocol where the nodes broadcast over the network the list of its neighbors. In this case all the nodes know the neighborhood of all the nodes. Therefore, the nodes have all the routes and thus the shortest path to all the destinations.

The OLSR module included in the Ad Hoc framework follows the same approach as the AODV module. Thus, the OLSR module is based on UNIK-OLSR [7]. The actual implementation of the Ad Hoc framework contains version 0.4.7 of the UNIK-OLSR. Similarly to the AODV, the existing UNIK-OLSR implementation has to include an additional component allowing the communication between the UNIK-OLSR and CCRS. Thus, an additional component named CCRS proxy is added within the UNIK-OLSR to slightly modify the UNIK-OLSR behavior. The CCRS proxy will store the routes discovered by OLSR into the CCRS so the routes are available to the Linux kernel routing table for the node communications.

#### 1.2.4 Hybrid modules

The hybrid modules consist of the software components that implement hybrid routing protocols (e.g. ZRP) but the actual Ad Hoc framework implementation does not contain any implementation.

## 2. MIDDLEWARE

The Ad Hoc framework also includes a p2p middleware platform based on Pastry (*FreePastry*) that enables us to run simple p2p testing applications on top of the multi-hop ad hoc network.

### 2.1 *FreePastry and the Distributed Messaging application*

FreePastry [8] is an open source implementation of the Pastry overlay network model. It is developed by the Rice University and it completely follows Pastry principles as described in [9]. Particularly it defines the internal data structures aimed at maintaining overlay's information, it defines the bootstrap node needed to establish and join the ring and related remote connections needed to recover those information.

In order to test FreePastry functionalities, we defined a simple application of *Distributed Messaging*. This service is an example of messages' exchange between peers, storing contents on distributed nodes of the network. Each node defines a "MailTable" data structure containing a variable number of records. These records are represented by (mailboxID, msgList) pairs. Each node can create more than one mailbox, specifying a unique identifier for each of them. They are created on demand through a "create" request message, sent on the overlay specifying the mailboxID as the key value. In this way a single node can maintain the mailbox of different nodes, depending on the distribution of the identifiers on the ring and they can send messages with the destination's mailboxID as routing key. The identifier of the first node is represented by the hash function applied to the IP address of the local node, while the identifier of the others nodes is autonomously calculated by FreePastry specifying only the IP address of a physical neighbor already present in the overlay. At the same time the identifier of each mailbox is represented by the hash function applied to a string chosen by the user as the identifier of the mailbox (mailboxID). Each node maintains messages belonging to mailboxes with ID logically closest to its logical address. In this way, specifying the mailboxID as the research key of the routing message, the subject-based routing is correctly implemented and mailboxes are uniformly distributed on the nodes joining the ring.

Once a mailbox is created and some messages received, a node can send a "get" request for this mailbox to download messages without deleting the related entry in the Mailtable, except the case in which a "delete" request is forwarded. When the node storing the mailbox receives the request, it directly sends the message list to the requiring node, forcing the Pastry routing protocol to a single peer-to-peer connection.

We can summarize as follows basic steps for the definition and the start up of the Distributed Messaging application. First of all, the application requires the user to specify if the starting node is the first node of the ring or not:

- 1) if the node is the first node that participates to the service, Pastry defines the ring starting from the IP address of the local node
- 2) if the requiring node has a knowledge of the ring, it has to specify the IP address of a known node (physical neighbour). In this case Pastry automatically recovers the ID of the specified node and initializes the routing tables of the new node directly connecting with other participants of the overlay.

A preliminary phase to test original FreePastry functionalities was conducted using a set of PC connected to a LAN, forming a classical P2P system. After an exhaustive test of FreePastry implementation on the wired network, we configured the application for an ad hoc scenario using a group of laptops equipped with PCMCIA wireless cards [10]. After setting up the connection



parameters for ad hoc mode and synchronizing nodes on the same frequency, it is necessary to explicitly set the correspondence between the hostname and the ad hoc IP address before running the application. At this point, to start the application and join the ring, is sufficient to use the local IP address if the node is the first of the ring, or the IP address of a known host in the range of the requiring node.

### 3. WATCHDOG

This section describes the watchdog module as part of the implementation of reputation-based cooperation enforcement mechanism for mobile ad hoc networks (i.e. CORE).

#### 3.1 Architecture

The cooperation enforcement mechanism proposed for the MobileMan architecture is the CORE mechanism. CORE is a collaborative monitoring mechanism based on reputation that strongly binds network utilization to the correct participation to basic networking function like routing and packet forwarding. CORE is being implemented as a Linux user-space daemon that runs on all the nodes of a MANET and can possibly be used with different routing protocols. In the future CORE will be able to store reputation information in a local storage accessible from different layers of the MobileMAN stack in order to help inter-layer optimization. CORE can be decomposed in three building blocks as shown in the CORE mechanism architecture depicted in Figure 2:

- 1) A monitoring mechanism implemented as a MAC layer sniffer. It monitors the packets that pass across layer 2 of the TCP/IP stack of a node and deduces whether neighbors are participating or not to basic networking functions. Monitoring of neighbors behavior is achieved by setting the WLAN card in promiscuous mode.
- 2) A reputation function that according to the output of the MAC layer sniffer calculates a reputation value for each neighboring node and marks neighbors as selfish when their reputation falls below a given threshold.
- 3) A punishment mechanism that punishes neighbors marked as selfish. According to a simple punishment model, a node punishes a selfish neighbor by refusing the forwarding of the selfish neighbor packets. A selfish node can be reintegrated in the MANET if it restarts performing packet forwarding function.

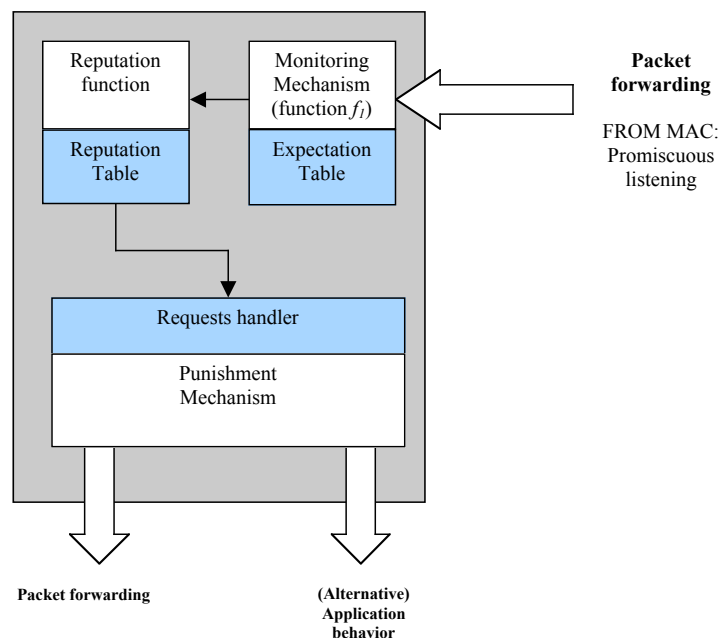


Figure 2 CORE architecture.

In the earlier stages of CORE design and implementation we consider the reputation as depending only on the participation to the packet forwarding function and concentrate on selfish nodes detection.

Further research and development will include integration of participation to the routing function as an input of the reputation function as well as design of the punishment mechanism.

Detailed description of the CORE implementation concepts such as the expectation table and the reputation function can be found in section 4.5 of MobileMAN deliverable D10.

### ***3.2 Functional Description***

CORE watchdog is implemented as a Linux daemon and packaged for installation on PDAs running Familiar Linux distribution.

Once it is installed following the instructions contained in the README file, it can be run simply by typing “watchdog” on the console.

CORE watchdog needs to be executed in cooperation with a WLAN card that works in promiscuous mode. It has been successfully tested on the Dell TrueMobile 1150 WLAN card.

CORE watchdog module monitors neighboring nodes behavior and writes output to the console. Some attributes of each packet captured within the transmission range are shown to the user, such as source and destination IP and MAC addresses, TCP sequence number and so on.

CORE uses MAC addresses to identify neighbors. Each time CORE watchdog detects a selfish behavior (i.e. a neighbor does not forward a packet coming from the node where the CORE watchdog is running), a line with the information about the selfish neighbor is printed to the console. Similar output is printed to the console when cooperative neighbors are detected.

The detection of selfish behavior is made by overhearing the packets that neighbors forward. When a watchdog waits for a neighbor to forward a packet, it temporarily stores the packet into the expectation table.

Operations on expectation table contents are printed on the standard output as well as other messages that help the user to understand the operations of the CORE watchdog.

## 4. VoIP

The Voice over IP is a real time application that is quite demanding for Ad Hoc networks. However, voice communications is a service that will provide added value to Ad Hoc networks since users will benefit from a communications without infrastructure support.

Following the same criteria and trying to re-use existing implementations we faced several problems when considering devices with limited resources such as PDA (i.e. iPAQ). The existing implementations of VoIP services required devices with enough processing power. Therefore, in order to develop a VoIP service for real scenarios including portable devices with low resources, we had to implement light version of VoIP application.

The VoIP application included in the Ad Hoc framework contains two main modules; signaling module and data transport module. The results show that a VoIP service can be provided in Ad Hoc networks with reasonable quality. However, since users have a good VoIP service with fixed networks the existing implementation requires a QoS module in order to enhance the service quality and meet user expectation.

### 4.1 Signaling module

The signaling module consists of the software component that will initiate the VoIP session with other peer nodes in the Ad Hoc network. This module has been implemented specifically for the Ad Hoc framework since existing implementations did require excessive resources (e.g. CPU, memory, etc). The signaling module implements the SIP signaling protocol and utilizes IP addresses for finding the peer nodes to initiate the VoIP session. The SIP signaling protocol can run on UDP or TCP protocol but in order to minimize the requirements for maintaining the session state in the nodes, the existing implementation uses UDP as the only transport protocol. The session initiation also requires negotiating the media parameters using SDP protocol. In order to minimize the negotiation process the signaling module uses the same codec for the VoIP session (i.e. GSM). Therefore, the signaling module is compliant with the SIP protocol but having a single codec optimizes the session set-up.

The SIP module is implemented specifically for the Ad Hoc network but the GSM codec is obtained from public source [11].

### 4.2 Data transport module

The data transport module consists of the software component that after the VoIP session is set up, takes care of exchanging the voice packets coded with the selected media format (i.e. in this case GSM is the only codec used in the session).

The data transport module implements a RTP client for exchanging the voice packets. The RTP client implements the functions for obtaining the audio samples from the microphone, encoding them using the selected codec (i.e. GSM) and then exchange the packets using the RTP protocol. The RTP client uses an publicly available RTP library for managing the RTP messages [12].

## 5. WHITEBOARD

This section describes the functionality of the whiteboard as a peer-to-peer multicast application suitable for Ad Hoc networks. The Whiteboard developed by Cambridge University is a Peer-to-Peer multicast application based on Bamboo substrate. It consists of two major function modules: the multicast layer and the whiteboard application. In order to make the design as flexible as possible, these functions were implemented independently. The whiteboard application could be introduced to other P2P systems, such as FreePastry, and a lot of other applications, such as video streaming, could be developed based on the same multicast layer developed in this work.

In this section, a brief introduction of the bamboo system, the essential building block for Whiteboard, is given. It is followed by the description of the functionalities of the multicast layer and the whiteboard application.

### 5.1 Bamboo

Bamboo is a peer-to-peer system primarily written by Sean Rhea of UC Berkeley, but it is based heavily on the OceanStore [13] and the libasync [14] projects. It is written in Java and is available as open source.

Bamboo distributed hash table (DHT) follows the same pattern of neighbour links as in Pastry. It assigns to each of its node a unique, 160-bit node ID. The set of existing node IDs is uniformly distributed, achieved by using a secure hash (e.g. SHA-1) applied to either the IP address and port number or to a public key. A message addressed to a certain key is then reliably routed to the node with the node ID numerically closest to that key among all live nodes.

Bamboo, on the other hand, re-engineers the joining or neighbour management algorithms in Pastry. New features were introduced for handling high levels of churn, especially in bandwidth-limited environments [15]. These include static resilience by which Bamboo can continue to perform lookups after node failures, and routing around them even before recovery begins; a combination of active probing and recursive routing allows Bamboo to make timely and accurate failure detection; using periodic algorithms to scale back maintenance periods automatically in response to congestion.

### 5.2 Multicast Layer

The multicast layer was implemented as a full Bamboo stage called *ComCore*. It is transparent to upper-layer applications and provides multicast service for multiple groups.

In *ComCore*, the multicast tree is created using a reverse path forwarding multicast algorithm [16], similar to Scribe in Pastry. It is proved to scale well for large groups of nodes [17]. The working procedure of this multicast protocol was described in detail in [18]. *ComCore* also provides two multicast tree maintenance mechanisms: *ChildAlarm* and the *MCastRepairAlarm*. The *ChildAlarm* makes sure that a *heartbeat* message is sent to each children of each group known by the local node. Also acknowledgements are passed to a parent if there was none already within the last time period. By default this alarm is initiated every ten seconds. The *MCastRepairAlarm*, as the name suggests, tries to detect failures of children or parents and to repair the multicast tree afterwards. It is run every 30 seconds by default.

The multicast layer offers several simple APIs to its applications, such as *create(topicID)*, *subscribe(topicID)*, *unsubscribe(topicID)* and *publish(topicID)* etc. Each topic has a unique topic ID which is associated with its own multicast group. An example of executing an application command is as follows: when a *create(topicID)* event is triggered, a create message is routed through Bamboo with the topic ID as the key. Bamboo will deliver the message to the node numerically closest to the topic ID. The chosen node adds the topic to its list of known topics and acts as from now as the root of the multicast tree for that topic. This node is furthermore the rendezvous point for all published messages for this group.

### **5.3 Whiteboard Application**

The whiteboard application developed provides the basic functionality like subscribing to an arbitrary topic and publishing changes on the canvas to the multicast group associated to the topic. The multicast group in this context could be dynamic, in which the members join and leave at anytime.

The whiteboard application can be further divided into two parts, one is the GUI and the communication interface to the multicast layer, the other is the canvas for drawing. The GUI was kept very simple: a canvas where you can draw at and several function buttons to choose between draw and erase mode, and to switch the topic. Whiteboard application only subscribed to the events between itself and the multicast layer, such as *PublishMsg* events to *ComCore*, and *IncomingMsg* events on the other way. The inter-node communication is completely done by the multicast layer.

The canvas extends from the *PanelTracker* class which is implemented to manage all user input, such as keyboard and mouse. It also provides two inline classes: one for drawing new strokes to the canvas and the other for managing the sending of new strokes to the network. If new data arrives at the whiteboard it calls the *newData()* function of the canvas and passes the container with the new data. In order not to block the whiteboard (and with it, the whole Bamboo node) a new thread is started to process that data and to draw it on the canvas finally.

## 6. REFERENCES

- [1] Charles E. Perkins, Elizabeth M. Belding-Royer and Samir Das, Ad Hoc On Demand Distance Vector (AODV) Routing, IETF RFC.
- [2] P. Jacquet et al, "Optimized Link State Routing Protocol for Ad Hoc Networks", Hipercom Project, INRIA Rocquencourt, BP 105,78153 Le Chesnay Cedex, France.
- [3] Familiar homepage, <http://www.handhelds.org>, March 2003.
- [4] Linux homepage, <http://www.linux.org/>
- [5] GUN's Not UNIX homepage, <http://www.gnu.org/>, February 2003
- [6] Uppsala University AODV homepage, <http://user.it.uu.se/~henrikl/aodv/>, March 2003
- [7] Olsr.org, <http://www.olsr.org/>
- [8] FreePastry, <http://freepastry.rice.edu>
- [9] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001.
- [10] D8 "MobileMAN First Phase", <http://cnd.iit.cnr.it/mobileMAN/pub-deliv.html>
- [11] GSM codec library, <http://kbs.cs.tu-berlin.de/~jutta/toast.html>
- [12] JRTPLib version 2.9, [http://research.edm.luc.ac.be/jori/jrtplib/jrtplib\\_old.html](http://research.edm.luc.ac.be/jori/jrtplib/jrtplib_old.html)
- [13] J. Kubiawicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells und B. Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), Banff, Canada, Nov 2000.
- [14] Nickolai Zeldovich, Alexander Yip, Frank Dabek, Robert T. Morris, David Mazières und M. Frans Kaashoek. Multiprocessor support for event-driven programs. In Proceedings of the 2003 USENIX Technical Conference, San Antonio, Texas, Jun 2003. S. 239–252, section 2.
- [15] Sean C. Rhea, Dennis Geels, Timothy Roscoe und John Kubiawicz. Handling Churn in a DHT. Technischer Bericht UCB//CSD-03-1299, University of California, Berkely and Intel Research, Berkely, Dec 2003.
- [16] Yogen K. Dalal und Robert Metcalfe. Reverse path forwarding of broadcast packets. Communications of the ACM 21(12), 1978, S. 1040{1048.
- [17] Antony Rowstron, Anne-Marie Kermarrec, Miguel Castro und Peter Druschel. Scribe: A large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in Communications (JSAC) 20(8), Oct 2002.
- [18] MOBILEMAN IST-2001-38113 Mobile Metropolitan Ad hoc Networks Deliverable D10: MOBILEMAN architecture, protocols and services. Oct 2004.