ÉPITA Final Internship Report

# Quality of Service Routing in Wireless Mobile Ad hoc Networks with IPv6 Support

Ignacy GAWĘDZKI <ig@lrde.epita.fr> — 13147 — CSI Promo 2003

01 February – 31 July, 2003

Within the
Networking research group of the

Laboratoire de Recherche en Informatique,
University of Paris-Sud, Orsay.

Under the direction of
Dr. Khaldoun AL AGHA <alagha@lri.fr>

# Contents

# Summary

This report describes my final ÉPITA[1]internship among the Networking group at the LRI[2], Paris-Sud University, France. The internship was directed by Dr. Khaldoun Al Agha and was entitled "Study and implementation of QoS routing protocol for wireless ad hoc networks with IPv6 support".

**QoS in Wireless mobile ad hoc networks**

Wireless mobile ad hoc networks (MANETs) are gaining inscreasing interest for several years now with the advent of widely available IEEE 802.11 [1] compatible wireless network interface cards. The potential uses are where a wired network infrastructure is not existing or not usable, ranging from military communication on the battlefield, rescue squads communications in a disaster area to commercial ambient computing.

The standard specification defines two operating modes for these cards: infrastructure and ad hoc. In infrastructure mode, the wireless link is merely a way to connect a computer to a classical wired network. On the other hand, in ad hoc mode, wireless interfaces can communicate with one another on the sole condition they are in the communication range (receiving and transmitting) of one another. With the use of a proper routing algorithm, a large group of mobile nodes with wireless interfaces operating in ad hoc mode and evolving in close geographical locations can form an autonomous and dynamic network of nodes that can reach one another using intermediate nodes as relays if they are not in direct communication.

**MANET routing with OLSR and IPv6**

Several routing protocols have been proposed within the Manet working group at the IETF[3]but few of them, if any, had existing working implementations which supported the IPv6 address family.

The Safari project, in which the internship tool place, required the use of a routing protocol that supported IPv6. Thus after the study of previous work on quality of service (QoS) routing in wireless mobile ad hoc networks, the goal was to write an implementation of the OLSR protocol with IPv6 support in order to provide an experimental platform for the later implementation of QoS methods.

**The ¶olyester** [ˌkɒlɪˈestəʳ] **project**

The implementation of the OLSR protocol was done in the C++ programming language[2] and was released as a free software project called **¶olyester**. The implementation is made in a modular way that allows easy extension for further evolution.

[1]École Pour l'Informatique et les Techniques Avancées, `http://www.epita.fr`
[2]Laboratoire de Recherche en Informatique, `http://www.lri.fr`
[3]Internet Engineering Task Force, `http://www.ietf.org`

# Chapter 1

# Introduction

## 1.1 The subject

The subject of the internship was "Quality of service routing in ad hoc networks with IPv6 support", which is a pretty large field of research at this time. This is because it contains three important keywords: "ad hoc networks", "routing" and "quality of service".

### 1.1.1 Ad hoc networks

By "ad hoc networks", one usually means more precisely "wireless mobile ad hoc network", often called **MANET** for short.

Wireless networks are formed by **nodes** (or **stations**) equipped with a radio network interface capable of transmitting data packets to some other node within its radio range. Today's widely available standard IEEE 802.11 cards can be set to operate in two distinctive modes called **infrastructure** and **ad hoc**.

The infrastructure mode considers two kinds of nodes: **client stations** and **access points**. The former differs from the latter in that it can communicate only with an access point and that the latter is necessarily connected to some wired infrastructure. Thus in infrastructure mode, the wireless links are merely an extension of the wired network and a station cannot communicate unless it is associated with an access point in its communication range. Infrastructure mode also supports extended coverage where several access points are spread over an area (overlapping or disconnected) and are wired together to form one large logical network.

In ad hoc mode, each station can potentially communicate with any other in its transmitting or receiving range, thus requiring no pre-existing wired infrastructure. Hence a set of stations located near one another form a network. If the intersection of all the communication ranges does not contain all the stations, there are some pairs of stations that cannot communicate with one another and would thus require relaying by other stations. For this reason, every station can be considered a host as well as a router.

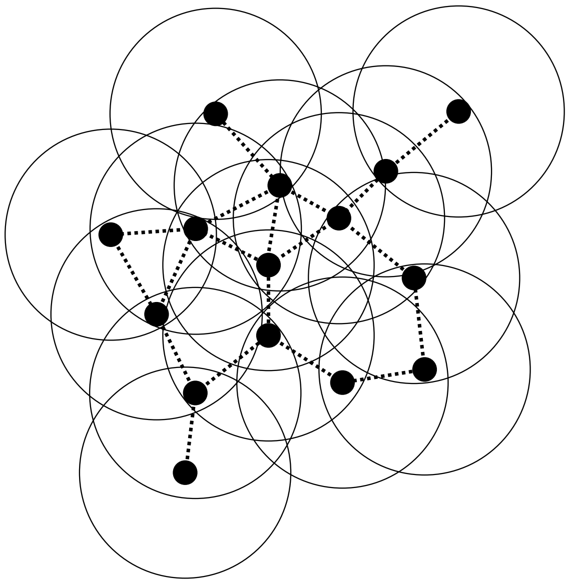


Figure 1.1: A wireless ad hoc network

Ad hoc networks bear some resemblance to wired networks in that stations in the communication range of one another form a network link and the whole ad hoc network can be represented by a graph. But the similarity stops here for two obvious reasons: the graph is not static, for stations are not spatially fixed and links are prone to interference from other stations, other radio equipment or simply shielding by some obstacles. Moreover, the stations being often battery powered, they have limited resources in power and transmission range. For these reasons, mobile ad hoc networks are dynamic.

### 1.1.2 Routing in MANETs

Since ad hoc networks are so-called multi-hop, we need routing algorithms to let stations relay packets towards a given destination.

Routing algorithms for static wired networks are not well suited for ad hoc networks because of their dynamic nature. In wired networks, the topology is known in advance and links are stable, whereas in ad hoc networks the topology changes all the time and stations may appear and disappear at any time.

The field of routing in ad hoc networks has been active for at least the past ten years. There are already many applications of multi-hop ad hoc networks, mainly because of its lack of infrastructure need: battlefield, disaster area, etc.

Routing algorithms for ad hoc networks can be separated into two families: **re-active** and **pro-active**. In the former, a route is calculated on demand, when a node needs to send a packet to a destination. In the latter, each node maintains a routing table at each moment. Re-active algorithms often necessitate the broadcasting of a **probe** packet to find a suitable path towards the destination, whereas pro-active algorithms need **control** messages to be exchanged to learn the network topology.

Each one family has its advantages and drawbacks. Re-active algorithms add a delay whenever a new route is required (be it for a new communication or a change in the topology). Pro-active algorithms require broadcasting control packets to all the nodes at a regular pace to spread topology information.

### 1.1.3 QoS routing in MANETs

Be it in military, rescue or simply commercial contexts, there are applications that require some constraints to be fulfilled on the path towards the destination. For example, real-time audio- or video-conferencing require minimal delay and bounded bandwidth, but support packet loss to some extent, whereas file transfer must be loss-free but can stand bandwidth variations. In wired networks, **Quality of Service** (QoS) routing is used for some time now and there are several models that work pretty well. But these models rely on the static nature of wired networks and thus cannot be applied as such to MANETs. For example, no one can guarantee QoS constraints for some communications, since the network changes with time and two nodes can totally loose connectivity in the most extreme case.

Thus routing algorithms for ad hoc networks are an open research problem for the time being.

## 1.2 The LRI

The Laboratoire de Recherche en Informatique is a CNRS[1] UMR[2] which means that it is co-directed by the CNRS and the University of Paris-Sud[3]. It has been created in 1974 and is located on the University's campus of Orsay, in a region densely packed with research and academic institutions as well as industrial complexes.

The purpose of the laboratory is to conduct research and academic activities in a wide range of theoretical as well as applied fields. It currently employs 15 CNRS, 58 university and 34 invited researchers, 59 Ph.D students, 13 administrative personel, engineers and technicians. It publishes 200 publications and 18 Ph.D theses each year. It is currently operating on 29 public, 25 european and 5 private contracts and in 30 industrial collaborations.

The laboratory is composed of 10 research teams:

- Graph Theory
- Algorithms and Complexity
- Programming
- Databases
- Proofs and Programs
- Parallel Architecture
- Parallelism
- Artificial Intelligence and Inference Systems
- Inference and Learning
- Bioinformatics

Each team is further divided into research groups and each of them may conduct several distinct research activities.

The internship took place among the Programming team, which is divided into three groups:

- Software Engineering
- Human-Computer Interaction
- Networking

[1]Centre National de la Recherche Scientifique `http://www.cnrs.fr`
[2]Unité Mixte de Recherche
[3]Université de Paris-Sud, `http://www.u-psud.fr`

The internship took place among Networking group led by Dr. Khaldoun Al Agha. The research topics of the group are third and fourth generation mobile networks, IP mobility, mobile ad hoc networks and group mobility.

The context was that of the Safari[1] project of the RNRT[2] which is aimed at the conception, combination and realization of a protocol and software infrastructure for transparent access, automatic configuration, integration and adaptation of services on an IPv6 network in ad hoc mode with wired connections.

The Safari project is thus a collaboration between many industrial and research partners: FTR&D[3], Alcatel[4], INRIA[5], LIP6[6], LRI[7], LSIIT[8], LSR-IMAG[9], SNCF[10]and ENST[11].

## 1.3 State of knowledge

The relation between the subject of QoS routing and the Networking group of the LRI is self-evident. Wireless networks are the main research interest of the group for some time now. Khaldoun Al Agha is a specialist of the field, after many years of research and publication.

Routing in MANETs without QoS has been actively studied for the last several years among the Manet IETF group. Four experimental routing protocols emerged from this effort:

- Re-active protocols:
  - Dynamic Source Routing (DSR)[3]
  - Ad hoc On Demand Distance Vector (AODV)[4]
- Pro-active protocols:
  - Topology dissemination Based on Reverse-Path Forwarding (TBRPF)[5]
  - Optimized Link State Routing (OLSR)[6]

The decision was made that the Safari project would be using the OLSR protocol, developed by the HIPERCOM[12]team at INRIA Rocquencourt.

At the beginning of the internship, the OLSR protocol was at the draft stage. The current version was 7, to be shortly superseded by version 8 in March 2003, version 9 in April 2003, version 10 in May 2003 and version 11 in July 2003. At the time of writing, the OLSR protocol is published as an Experimental RFC[13], RFC 3626[6].

Nevertheless, the OLSR protocol could not be used as such because it does not support QoS routing and no implementation available at that time supported the IPv6 address family, though OLSR was designed to be independent of the protocol family.

[1]`http://www.telecom.gouv.fr/rnrt/projets/res_02_04.htm`

[2]Réseau National de Recherche en Télécommunications, `http://www.telecom.gouv.fr/rnrt`

[3]France Telecom Recherche et Développement, `http://www.rd.francetelecom.fr`

[4]Alcatel, `http://www.alcatel.fr`

[5]Institut National de Recherche en Informatique et Automatique, `http://www.inria.fr`

[6]Laboratoire d'Informatique de Paris 6, `http://www.lip6.fr`

[7]Laboratoire de Recherche en Informatique, `http://www.lri.fr`

[8]Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection, `http://lsiit.u-strasbg.fr`

[9]Laboratoire Logiciels Systèmes Réseaux de l'Institut d'Informatique et Mathématiques Appliquées de Grenoble, `http://www-lsr.imag.fr`

[10]Société Nationale des Chemins de Fer, `http://www.sncf.fr`

[11]École Nationale Supérieur des Télécommunications, hrefhttp://www.enst.fr`http://www.enst.fr`

[12]HIgh PERformance COMmunication, `http://www.inria.fr/recherche/equipes/hipercom.en.html`

[13]Request For Comments

## 1.4 My own state of knowledge

My knowledge of IPv4 networking was that of a young student having toyed with his Linux computer(s) for several years. I had very basic routing notions and absolutely no experience with wireless networks. I also had very basic knowledge of IPv6 specifics.

What I knew pretty well was how to develop on a GNU/Linux platform with free software development tools. In addition, I made my research debuts at the LRDE[1]during my engineering years at ÉPITA and had notions about how research activities were usually conducted.

I found the perspective of doing this internship very interesting not only because of the subject, but also because I knew I wanted to work in a research environment in some laboratory. The perspective of working on a research subject, requiring to keep contact with several research teams at once, composed of Ph.D students as well as doctors and professors, on a project that would end up in a concrete realization appealed me very much.

My aspirations were at the crossing of applied research and innovative development, so this internship was all-the-most suited.

## 1.5 Working environment

Since the very beginning of the internship, I was provided everything I needed to work efficiently. I was assigned a desk in an office where two Ph.D students were already working. I was offered to receive a computer to put on my desk, but I proposed to bring my own workstation that I was to take away from the LRDE and that was accepted without any problem. I was provided complete network connectivity as soon as I brought it and connected it. I was given an account with the login name I desired on the laboratory's network to allow me to receive mail and connect from the outside.

I also received all special access badges and permission cards to allow me in my office at any time of day or night, which pleased me very much since I sometimes like to work late at night.

In addition, since it was believed that I could work in cooperation with the HIPERCOM team at INRIA Rocquencourt, I was shortly introduced there and given similar access as at the LRI (account, mail and late night access).

Working on concrete experimentations with wireless networks implied the use of PDAs[2] with wireless network interfaces which were provided to me since the very beginning. I was also offered to receive an additional laptop computer as soon as the budget for the Safari project would be available.

For matters related to documentation, I was able to use the LRI library which happened to be located right at the end of my corridor. Although the library is very well supplied, I made little use of it, since most of the information I required was available publicly on the Internet. Of course, I was able to print anything I needed on fast laser printers.

Over the course of the internship I kept regular contact with my director and the Ph.D students of the group. I also had pretty regular contact with Ph.D students of the teams working on Safari at the LIP6[3]as well as the HIPERCOM team at INRIA Rocquencourt, though to a lesser extent. For all network-related or scientific questions, the group members including my director were very helpful, be it by direct discussion or email.

[1]Laboratoire de Recherche et Développement de l'ÉPITA, `http://www.lrde.epita.fr`

[2]Personal Digital Assistant

[3]Laboratoire d'Informatique de Paris 6, `http://www.lip6.fr`

# Chapter 2

# Organization and time-line

The internship started shortly after the beginning of the Safari project which was planned to last 33 months. The end of my internship coincided with the first deliverable of the project which was bibliographical: a state of the art of QoS routing in MANETs. Of course it was clear that my internship would not be only composed of bibliographical work as I was hired to help the group get a working implementation of the OLSR protocol with support of IPv6 to subsequently implement the QoS models that were worked out theoretically and tested in simulators.

In fact this was the only constraint on the time-line of the internship. The rest was left for me to organize as I wished.

## 2.1 Setting up of the working platform: 1 month

Prior to any development work, I had to set up the development platform which was composed of my working station and several PDAs based on StrongARM processors and able to run a Linux kernel.

So to be able to develop for such computers, I had to set up a whole cross-compilation environment on my workstation. Direct compilation is impossible since the PDAs have pretty limited memory resources (16 Mb of flash mass memory and 32 Mb of SDRAM for the models that were able to run Linux at that time).

## 2.2 Bibliographical research: 2 months

Since my knowledge in the field of QoS routing in MANETs with IPv6 was very limited, I had to spend a lot of time studying the existing publications on the matter.

I started with the IETF RFCs and books[7] about the IPv6 protocol family. Then I had to familiarize with the way IEEE 802.11 protocols work, since I was to work with this kind of widely available hardware. Subsequently I had to study existing and experimental best-effort routing protocols for MANETs and lastly I had to study those that have been designed to address QoS requirements. Meanwhile, I also studied QoS support in wired networks, since existing working solutions in wired networks preclude any reading about QoS in MANETs, for the latter is much influenced by the former.

At the end of this part of the work, I was asked to write a state-of-the-art report on the matter, which was to be used as a starting point for the first deliverable for Safari. I was also convinced to submit this paper to IEEE Wireless Communications, which appeared to be calling for just this kind of papers at that time for a special issue.

## 2.3 Development of the OLSR implementation: 3 months

This part in fact really started at the same time as the bibliographical work, but had to be delayed for several reasons. After having first tried to reuse an existing OLSR implementation and adapt it from IPv4 to IPv6, I had to resign myself to re-implement it from scratch, since the state of the existing code was terrible and would require too much work. Besides, the existing implementation was written in the C programming language, which was far from being a first choice language for large projects which make use of pretty abstract notions as sets and reusable components.

I managed to convince, though without much trouble, my director to allow me to re-implement the OLSR protocol from scratch using the C++ language which I knew pretty well from my time at the LRDE. It has much better suited features for set processing and generic components than C and can be used in a way that generates binary code is as efficient as code generated from C code.

Of course this was a compromise because if it were only a matter of language and efficiency, I would certainly have chosen Objective Caml. But I had to take into account the fact that people in the networking field are seldom programming theorists and don't know many modern languages. If the project had to be taken over by somebody else from the field at a later time, it would have been written in some language understood and mastered by her, otherwise all the effort of writing it from scratch would have been wasted.

## 2.4 Comments on the time-line

This time-line was not completely planned from the beginning of the internship since the work was not obvious at all. If it had been possible to use an existing implementation of OLSR and the adapting work had been easy, the third part would have been shorter and there certainly would have been a fourth part about QoS extensions and testing on the existing OLSR implementation. This was not necessarily what was intended at the beginning and was pretty much the unknown in the equation. It would have balanced the work more towards research than development. I must admit I do not feel less pleased by the actual turn the events took, because it allowed me to express my software engineering skills with a language much less frustrating — although with its own load of frustration — than a low-level language like C.

The third part of the internship took me more time that planned for several reasons. The most important one was certainly that of my own perfectionism which led me not to accept certain compromises which could have spared me considerable amounts of time and effort. I wanted to find the best possible solution taking account of the clarity of the model and efficiency of the code. This implied several re-workings of some parts when new ideas sparked along the development. This was certainly not the best method to honor deadlines, but gave really interesting results at the end. Another reason of the delay was that new versions of the draft documents that described the OLSR protocol were released in the meantime and that subsequent descriptions of the inner working of the protocol exhibited some unexpected aspects that were not clear at all from previous versions.

# Chapter 3

# Technical details

## 3.1 Development and testing platform

The first thing to do was to make the tools work. This meant to install a cross-compilation environment on the workstation to be able to generate executable binaries to be run on the PDAs.

### 3.1.1 The cross-compiler

To build a cross-compilation environment from scratch is a very tedious task, since theoretically to build the cross-compiler, one needs a cross-compiled standard library which in turn needs a cross-compiler to be compiled. It is a bit like the chicken and egg problem. In fact the cross-compiler is compiled twice, once without the standard library and once again when the standard library is available.

Luckily enough, there is a lot of people working with cross-compilers for the ARM architecture, a great deal of whom is working with the very kind of PDAs I was supposed to work with. So there were already several other means to install a cross-compiling tool-chain than to compile it from scratch. After several attempts to do a clean installation from scratch, I resigned myself to use an existing compiled package. This was mainly because “vanilla” GCC sources needed patching for correct operation as a ARM cross-compiler and these patches were available only for a limited range of GCC versions. Of course I wanted to use the latest major release of the free compiler, which was 3.x and patches were available only for 2.9x versions.

### 3.1.2 The Familiar distribution

Hopefully, I had not to compile a whole GNU/Linux OS from the ground up, since there are pretty well working distributions available to download and install. It comes with a kernel all patched for correct operation of all the specific components of the PDAs as the flash memory, the touch-screen, the sensors, the sound and the most important: the PCMCIA add-on sleeve and the Orinoco wireless IEEE 802.11b PCMCIA network card.

One of the iPAQs was already running some older version of Familiar[1], but another one was freshly taken out of the box and qualified for total conversion. This comprised the installation of a boot-loader, released by the Hewlett-Packard (formerly Compaq and even earlier Digital Equipment Corp.) Cambridge Research Laboratory and the installation of the Familiar distribution.

[1]http://familiar.handhelds.org

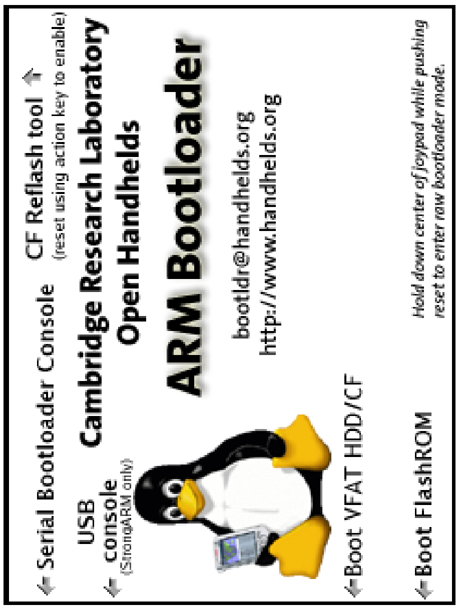
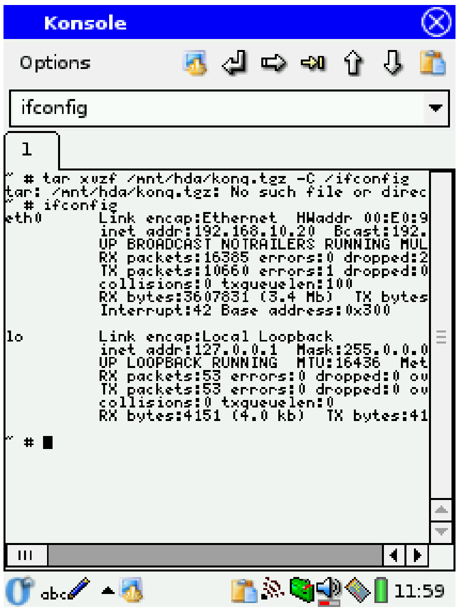

Serial Bootloader Console
CF Reflash tool
(reset using action key to enable)
USB console
(StrongARM only)
Cambridge Research Laboratory
Open Handhelds
ARM Bootloader
bootldr@handhelds.org
http://www.handhelds.org
Boot VFAT HDD/CF
Boot FlashROM
Hold down center of joypad while pushing reset to enter raw bootloader mode.

Figure 3.1: The boot-loader splash screen

The boot-loader image had to be sent over the serial line to the flash partition on the PDA. Then a flashing utility also had to be sent the same way. Once the new boot-loader was installed, the PDA had to be rebooted and the rest of the operation had to be conducted using the command line of the boot-loader on the serial port, using some terminal emulation software as Minicom. The distribution had then to be sent as an image to be written over the flash boot partition with the help of the boot-loader. This operation took some considerable time, since the image has to be uploaded using XMODEM or YMODEM (an old but pretty simple file transfer protocol over serial lines).



```
~ # tar xvzf /mnt/hda/konq.tgz -C /ifconfig
tar: /mnt/hda/konq.tgz: No such file or direc
~ # ifconfig
eth0      Link encap:Ethernet  HWaddr 00:E0:9
          inet addr:192.168.10.20  Bcast:192.
          UP BROADCAST NOTRAILERS RUNNING MUL
          RX packets:16385 errors:0 dropped:2
          TX packets:10660 errors:1 dropped:0
          collisions:0 txqueuelen:100
          RX bytes:3607831 (3.4 Mb)  TX bytes
          Interrupt:42 Base address:0x300

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Met
          RX packets:53 errors:0 dropped:0 ou
          TX packets:53 errors:0 dropped:0 ou
          collisions:0 txqueuelen:0
          RX bytes:4151 (4.0 kb)  TX bytes:41

~ # █
```

Figure 3.2: The shell in the OPIE environment

Once the transfer finished, the system could be booted and later either used with the stylus and the graphical interface, or by the way of the command line on the serial console. The command line on the console is a plain GNU/Linux login prompt and a shell. The distribution bears some resemblance with Debian in the way software packages are managed. Once the wireless network card is configured, new software packages can be installed from the network, which simplifies and speeds up things very much.

### 3.1.3 The USAGI IPv6 stack

The problem with IPv6 — apart from it not being yet widely used — is that the state of development in "vanilla" Linux kernels is poor and not up to date with standards. So anyone willing to do serious work with IPv6 needs to use a package with a much more up to date version developed by a team in Japan and dubbed USAGI (UniverSAl playGround for Ipv6)[1]. This package contains a kernel patch and a set of tools for network management. It is available in two versions: stable and experimental, the latter being released in snapshots every two weeks.

Of course, the patch is generated against "vanilla" kernel sources and so applies cleanly to this flavor. The set of tools allows to take into account some IPv6 specifics that are not possible with standard IPv4 traditional tools (like routing table management, etc).

But as far as the ARM architecture is concerned, the task gets a bit more tricky. One has to merge the heavy patch-set required by the iPAQs with the USAGI patch. The patches are not applicable as such since the iPAQ patch-set already makes its own modifications to the IPv6 stack. In fact the solution lies in a straight replacement of sources files in the kernel source tree, but the way of doing it properly has to be dug up from some deeply hidden mailing list archives.

### 3.1.4 Other specifics of cross-compilation for ARM

Once I had a working kernel with top-of-the-shelf IPv6 support, I was faced with other annoying little problems. The version of the standard C library used by Familiar was not the same as the one available in source form, so the binaries had to be statically linked with the version on the workstation. This produced larger executables which could be problematic at some later point, if debugging using GDB on the iPAQ was necessary.

At a later time when C++ code had to be cross-compiled, the fact that I had only GCC-2.9x version as a cross-compiler meant that all the compatibility problems between GCC-2.9x and GCC-3.x had to be worked out before any testing. This meant that some C++ code that is compiled without problems with GCC-3.x would not compile with GCC-2.9x. Fortunately, these were pretty trivial language issues that could be worked out pretty easily at the beginning. At some later point in time, I found a way to install a GCC 3.2.3 cross-compiler, so the compatibility problems vanished… Or so until I tried to compile using GCC 2.9x again and it appeared that the incompatibility issues have become impossible to solve, mainly because version of GCC earlier than 3.x use a non-standard STL implementation.

## 3.2 QoS routing in mobile ad hoc networks with IPv6 support

To familiarize myself with the field, I had to conduct bibliographical research along four principal axes.

### 3.2.1 IPv6 specifics

Today's Internet relies on a protocol engineered nearly thirty years ago. Though very visionary, its designers could not foresee all the problems that were to arise in the meantime. Among others, the most obvious and also recurrent problem was that of underestimating future needs: the Internet address space appears to be too tight for tomorrow's applications. Hence in 1994, the

[1] `http://www.linux-ipv6.org`

IPng IETF working group was formed [8] [9] to design what was to become the Internet Protocol version 6 (IPv6) [10] address family, destined to supersede its predecessor, namely Internet Protocol version 4 (IPv4) [11] [12] [13]. This subsection illustrates main IPv6 features.

**Larger address space** The main goal of the promotion of IPv6 was the new address space [14] [15], which was to address the imminent lack of IP addresses in the currently used IPv4. In fact, IPv4 addresses are 32 bits integers and thus can code at most $2^{32}$ different addresses. The point is that this address space is divided in classes [16] in such a way that today many portions of the address space are unused as much as unusable, because they were wasted at early times when they were not a scarce resource. IPv6 addresses are 128 bits long and can thus code $2^{128}$ different addresses, which seems to be enough for some time to come.

**Auto-configuration of nodes** One bad thing about large addresses is that no one really wants to manage then by hand, so along with the new address space, IPv6 comes with a clever way of making the network interfaces configure themselves automatically [17], replacing the need for DHCP servers and the like.

The address space is also divided in many classes, differentiated by their prefix. In fact one interface would have many different addresses used alternatively depending on the purpose of the communication. There are so-called link-local addresses, which prefix is `fe80::/10` [1], site-local addresses which prefix is `fec0::/10` and the global addresses which prefix is `0200::/3`.

After a network interface comes up, it assigns itself a link-local address automatically using its MAC address or any other scheme, depending on the underlying MAC architecture. Then an address conflict resolution protocol is followed on the link to avoid ending up having two interfaces on the link having the same link-local address. If an address collision is detected, other address attribution schemes are applied and the operation is repeated.

After having configured its link-local interface, a node sends control packets called **router solicitations** to ask whether there is a router on the link. If a router is present, it answers with a **router advertisement** to announce its existence to the new node and to provide it with a site-local or global prefix. When a node receives the prefix, it configures its site-local or global address and is able to talk to the outside world. This implies that routers are forbidden to forward link-local packets between interfaces and that site-local packets between sites.

**Multicasting** Broadcasting does not exist anymore in IPv6 and has been replaced with multicasting [18]. There are several address prefixes used for multicasting. The `ff01::/16` is the host-local multicast prefix and `ff02::/16` and `ff05::/16` are respectively the link-local and site-local multicast prefixes. There exist other multicast prefixes, each provided for a well-defined purpose. Following the 16 bits prefix, comes the 112 bits multicast group which in turn is standardized. For example, the `::1` group means all-nodes, whereas `::2` means all-routers. So when a node wishes to multicast to all nodes on the local link, it has to use the `ff02::1` multicast address.

**Anycasting** This is a new way of addressing nodes on a network. Its purpose is to address the first reachable node in terms of network routing distance. It can be used for services discovery, etc.

### 3.2.2 IEEE 802.11 Wireless networking

The widely available IEEE 802.11b wireless network cards make it a good candidate for implementation and experimentations. From an operating system standpoint, they behave very much

[1]This notation exhibits two characteristics of the currently widely adopted IPv6 address notation: the `::` shortcut and the `/nn` notation. The former means an arbitrary number of zeroes and must be used unambiguously, while the second indicates the length of the prefix in bits

like ethernet interfaces in that they have a MTU[1] of 1500 bytes and have 48 bits MAC addresses.

In the physical layer, the modulation is quite different, since the medium is not a copper wire anymore but ambient air, water, void, etc. The details of the physical modulation are outside of the scope of the internship and are not presented here, please refer to [19] for more information.

In the data link layer, the used medium access scheme is ethernet's cousin: CSMA/CA[2] [19]. Whereas ethernet interfaces can sense the medium and transmit at the same time and hence detect collisions, wireless interfaces have only one antenna and cannot sense the medium and transmit at the same time. Thus collisions on the wireless medium cannot be detected and have to be avoided. This implies the use of strategies like random back-off timers and inter-frame silence plus positive acknowledgements.

To allow medium reuse, the interfaces label their packets with identifiers to distinguish then from other networks in the vicinity. In addition, some amount of security is provided by the use of keys, ciphers and authentication schemes in the data link layer. Meanwhile, the cryptographic scheme used in wireless interfaces, called Wired Equivalent Privacy (WEP), has been proven to have serious flaws that make it a poor security measure.

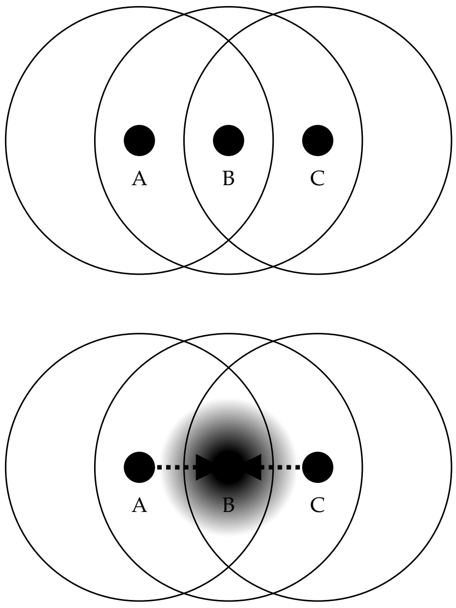


Figure 3.3: The "hidden node" problem. B "hears" both A and C, but neither does A "hear" C nor the opposite. If A and C send a packet to B at the same time, there is a collision.

A last specific interesting point of wireless networking comes from the fact that the stations have limited transmitting range. The so-called hidden-node problem appears when two nodes that are too far apart to "hear" each other communicate with a third node in between. Then any one of the extreme nodes sensing an empty medium would send a packet to the middle one which would be not be able to do anything about the interfering packets. Off course, each node would wait for the positive acknowledgement packet (ACK) and not receiving it for some time would try to transmit again. But this proves to be very inefficient since the data packet can be in fact quite long. To cope with that, flow control packets can be used for large data packets to detect that the middle node is communicating with the other node: before sending the packet, a node sends a Request To Send (RTS) control packet and waits for the destination to reply with a Clear To Send (CTS) control packet. The size of the intended data packet being advertised

[1]Maximum Transfer Unit: the maximum length of data in a packet.

[2]Carrier Sense Multiple Access with Collision Avoidance. Not to be confused with ethernet's CSMA/CD: Carrier Sense Multiple Access with Collision Detection.

in the RTS and CTS packets, a node not hearing the sender would infer its presence from the CTS reply of the receiver and would thus know the amount of time the medium has to be kept clear. This phenomenon has an important impact on QoS routing in that the available bandwidth at one node depends on the bandwidth of the neighboring nodes. However, the flow control mechanism can be only used for unicast packets (i.e. as opposed to multicast or broadcast), since the destination needs to be unique.

### 3.2.3 Routing in mobile ad hoc networks

The particularities of wireless mobile ad hoc networks that make routing not trivial are mainly node mobility (i.e. a node moves geographically) and link unreliability (i.e. a link can break at any time). They make the MANET a dynamic network for which the topology is neither known nor stable.

In wired networks, the topology is not only known but even often planned to exhibit some interesting properties for the purpose of concrete applications (trees, rings, etc). Automatic routing at larger scales has become necessary as the Internet evolved from its simple initial architectures of one backbone connecting several local networks. Thus, protocols for local network routing information gathering like RIP[1] [20] or later OSPFOpen Short Path First, as well as external network routing like BGP[2] [21] are widely used.

Unfortunately, existing routing protocols for wired networks do not suit wireless mobile networks very well. Nevertheless, some of them inspired new routing protocols for MANETs.

Routing protocols for dynamic networks can be divided into two categories, namely re- and pro-active algorithms.

#### Re-active algorithms

In this algorithm family, a route towards a given destination is looked for on demand, when a request comes from a new flow. To be effective, and avoid repeating the operation at each intermediate node, it needs caching of routes and use of a source-routing scheme, in which the sender provides the route to be followed inside the packet header.

There are several ways to find a route towards a destination when the topology is unknown. The most obvious one consists in broadcasting a special probe control packet with the address of the destination. Each receiving intermediate node forwards the packet broadcasting it to all of its neighbors, and so on until the packet reaches its destination. At each node, the probes are marked with the route they followed to this point, so the destination receives the probes with their paths. The destination sends one of the received probes back towards the sender using source routing along the same path but in reverse.

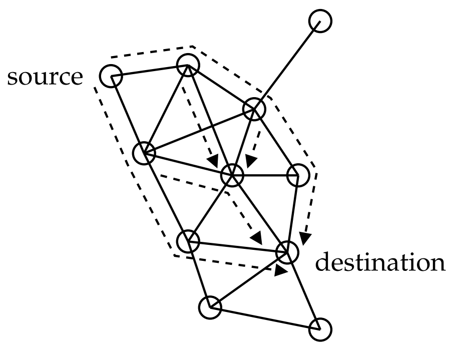


Figure 3.4: Sending probes in re-active protocols

[1]Routing Information Protocol
[2]Border Gateway Protocol

The use of this kind of flooding mechanism has been shown not to scale well in wireless ad hoc networks since the number of control packets sent for one flow grow exponentially with the size of the path. Additional use of TTL[1] is essential to avoid bringing the whole network down to its knees.

Another problem with re-active routing algorithms is that they do not implicitly adapt the route to the changing topology and hence need a signaling scheme to inform the sender about the need to find a new route.

#### Pro-active algorithms

Pro-active routing assumes that each node has at least a partial knowledge of the network topology and maintain routing paths to any destination at any time. Thus a intermediate node can forward the packet without the need of the latter to carry path informations along.

Network topology information is spread throughout the network in two possible ways:

**Link State** Nodes spread information about the links to their surrounding neighbors. The routing table is then populated using a SPF[2] algorithm on the reconstructed topology graph of the network.

**Distance Vector** Nodes exchange information about the distance (in number of hops) towards all the destinations they know. The routing table is thus updated continuously, with fresh information from the neighbors. If, for one destination, a neighbor advertises such a distance that, when incremented, is still lower than the distance known so far, the latter is replaced with it and the entry in the routing table is changed with the advertising neighbor as next hop towards the destination.

Although a node needs not send a probe to find a route, it has to spread topology information throughout the network. In this sense, link state and distance vector methods are equivalent since the former spreads information explicitly by forwarding other's topology control packets and the latter exchanges whole distance vectors with length proportional to the total number of nodes. The flooding problem has to be dealt with and so spreading information must be optimized to remain scalable.

For link state methods, the idea is to allow only a subset of the direct neighborhood to forward control packets. Of course, this subset should be minimal with total coverage of the 2-hop neighborhood, but the problem of finding such a minimal set of neighbors has been shown reducible to the Minimal Dominant Set problem which is NP-Complete. Therefore heuristics have to be used instead of an exact algorithm to allow scalability.

### 3.2.4 QoS considerations

Quality of service (QoS) is the set of performance properties of a network that can include bandwidth, delay, packet loss probability, etc.

By QoS, one generally means the set of measures to satisfy or even guarantee some constraints on the quality of service of a network. In fact there are several different aspects of QoS one needs to consider:

[1]Time To Live, often expresses in a maximum number of hops a packet is allowed to go.
[2]Short Path First, like the Dijkstra or Bellman-Ford algorithms.

| | |
|---|---|
| **QoS model** | This is the goal one wants to achieve, be it guarantees of hard bounds on network parameters or flexible bounds with quality reporting, etc. |
| **QoS-aware routing** | This is a routing protocol that supports finding of routes using other criteria than simply the minimum number of hops. These criteria may possibly be mixed together which can lead to intractable problems. |
| **Packet scheduling** | This is a way to treat packets at each forwarding node, in order to prioritize them properly to comply with QoS requirements. This implies some differentiation scheme in order to classify the packets into correct scheduling categories. Among others, possible differentiation are on a per-flow or per-class basis. |
| **Reservation scheme** | This is a way to ensure that each intermediate node is aware of the new-coming flow and that two simultaneously started flows do not end up using the same resources (and thus collide). |

When tied up together, QoS measures allow network service providers to offer the features of virtual circuits that allow them to sign Service Level Agreements (SLA) with customers. These are contracts in which the provider agrees to pay fees to the customer in case the QoS is not provided for some reason.

In wired network applications, routers and links are not supposed to break often. This assumption leads to existing protocols that rely on the fact that a route will not change once it has been reserved. Among others, the most used two are IntServ[1] [22] and DiffServ[2] [23] [24] [25].

The assumption does not hold anymore when the network topology changes unpredictably. Early models based on classical models for wired networks have off course failed to achieve their goals for this precise reason.

Other early attempts of QoS support in MANETs were based on close cooperation with the MAC layer. An example is the DSDV+ [26] protocol that was in charge of time slots assignment for the underlying TDMA[3] medium access scheme. While this approach can be very interesting at one time when a given MAC layer is used it can be obsoleted by the later wider adoption of another incompatible scheme. This is the case today for DSDV+ since IEEE 802.11 standards use the CDMA[4] scheme.

Strong QoS constraints cannot be guaranteed on MANETs anyway, since nothing prevents the topology graph to become partitioned at some point in time. The implications thus surface up to the application layer and this suggests that a correct way to go is to create variable-QoS-aware applications that can adjust their requirements to the network conditions. For example, an audio-conference application could reduce sound quality to increase compression ratios in order to require less available bandwidth when capacity of links become scarce. This would require the applications to communicate with the routing daemon to find a good compromise.

## 3.3 Internals of ¶olyester

### 3.3.1 Introduction

The last part of the internship was the implementation written in the C++ programming language of the OLSR protocol with IPv6 support.

[1]Integration of Services
[2]Differentiation of Services
[3]Time Division Multiple Access
[4]Code Division Multiple Access

**Brief description of the OLSR protocol**

The Optimized Link State Routing protocol has been developed by the HIPERCOM team at INRIA Rocquencourt [27]. This subsection is a brief overview of its principles. For a detailed description, please refer to the authoritative specification in RFC 3626[6].

OLSR is based on the idea already used in OSPF[28] to spread link state information throughout the network. Of course plain broadcast flooding of control messages does not scale at all in MANETs, since many nodes then receive redundant information and the number of transmitted packets grows exponentially, which dramatically increases packet collision probability.

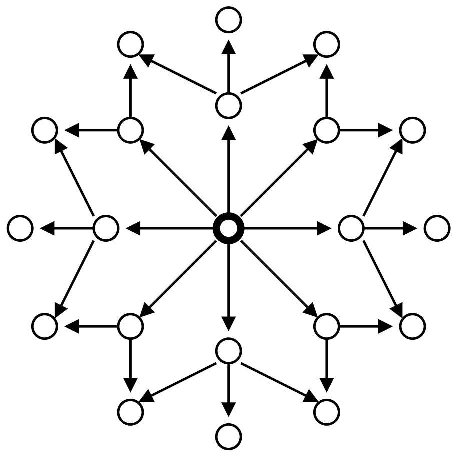


Figure 3.5: Dumb message flooding

**Neighbor sensing** OLSR is a pro-active protocol, in that it maintains a knowledge of the network topology at any time. To achieve that, a node must first of all sense its immediate 1-hop links by the exchange of **HELLO** messages. A node populates its **link** and **neighbor sets** by using information provided in HELLO messages received from neighboring nodes. Thus, a neighbor exists if at least one link with that neighbor exists. HELLO messages then contain information about the status of already known neighbors and the associated status of the link on the interface the HELLO is being sent. A node extracts from a particular HELLO message the information about its 1-hop neighbor but also about the 2-hop neighbors that are reachable through it.

**Multi-point relays** OLSR uses another broadcasting technique based on **multi-point relays** (or **MPR**s for short): every node elects some of its 1-hop neighbors to relay information to be broadcasted on the network. It must choose them so that they best cover its 2-hop neighborhood. Since the problem of finding a such a minimal set of MPRs is reducible to the Dominant Set problem, it is NP-Complete and thus an approximation based on heuristics is used. The MPR set is recomputed as soon as change in the node's symmetric neighbor set or 2-hop neighbor set is detected.

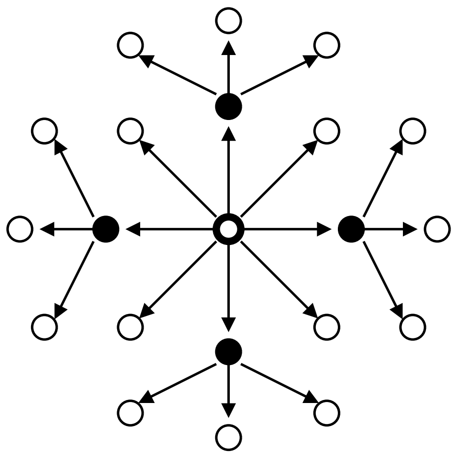


Figure 3.6: Message flooding using the MPRs

When a node has computed its MPR set, it declares its MPR neighbors in the neighbor status field of HELLO messages. This allows one node to see what neighbors have selected it as one of their MPRs and thus populate its **MPR selector** set.

When a node receives an OLSR control message other than HELLO message, it has to forward it further only if the sender node is contained in its MPR selector set. This way redundant flooding of the network is avoided to allow better scalability. Duplicate forwarding is avoided by the use of sequence numbers in messages and duplicate sets in each node.

To support the use of gateways and power saving, a node declares in its HELLO messages its degree of **willingness** to be selected as a MPR. Thus a node gatewaying some external network may declare a willingness of `WILL_ALWAYS` to be selected as MPR by all its 1-hop neighbors, whereas a node short on battery power my declare a willingness of `WILL_NEVER` to avoid being selected as a MPR at all. Varying degrees of willingness in between these two extremal values are supported and a node i supposed to advertise a willingness of `WILL_DEFAULT` during normal operation.

**Topology control messages** As soon as one node's MPR selector set is not empty, a node must emit **Topology Control** (**TC**) messages to be broadcasted on the network. TC messages advertise a node's MPR selector set and thus allow to spread network topology information to every node. Although TC messages are not emitted as often as HELLO messages, each message bears an additional sequence number that is incremented each time a change in the advertised neighbor set is detected (the **ANSN**: Advertised Neighbor Sequence Number). Upon reception of a TC message, a node populates its **topology set** by adding new information and discarding that information that was added by the reception of a previous TC message originating from the same node with a smaller ANSN.

**Routing table calculation** As soon as a change in the 1-hop or 2-hop neighbor set or topology set is detected, the routing table must be recalculated. Symmetric 1-hop neighbors must be reachable directly without the need of relaying. Routes for nodes that are farther away are computed using a shortest path algorithm (Dijkstra or Bellman-Ford) on the graph formed by topology information.

**The multiple interface extension** This extension adds a new type of message: the **Multiple Interface Definition** (**MID**) to allow a node to advertise its additional interfaces. If a node has more than one network interface participating in the OLSR protocol, it has to choose a main address among its network interfaces. For all interfaces which address is not the main address, the node has to advertise the interface address in a MID message that is to be forwarded using the MPRs.

Interface addresses are only used in HELLO messages. All other message type that advertises some node's address must use its main address. Thus MID messages are only useful in populating one node's 2-hop neighbor set to find the main address of a node given the address of one of its interfaces.

The routing table calculation is a bit modified in that a node must add routes to all the known interfaces (in terms of received MID messages) in addition to main addresses. Therefore, a routing table recalculation is triggered each time the MID set is changed.

**The external networks extension** If a node is connected to one or more networks not taking part in the OLSR protocol, **Host-Network Association** (**HNA**) messages can be sent to advertise them. A HNA message contains all the network addresses and prefixes the originating node is willing to gateway.

Routing table calculation must be extended to add routes to external networks advertised in received HNA messages. In addition, a routing table calculation is triggered each time the HNA set is changed.

**Link layer notification** This part provides the possibility to use information extracted from the MAC layer to populate (or rather depopulate) the link set. This part has not been implemented in **Qolyester** which remains independent of the MAC layer totally (in fact it can be used on plain ethernet or even serial links).

**The link hysteresis extension** The relative quality of a link can be expressed by the frequency at which its state changes and this can be taken into account in MPR selection. This part has not been implemented in **Qolyester**.

**Redundant topology information extension** Instead of advertising only its MPR selector set, a MPR node can also advertise its MPR set and even its whole symmetric neighbor set. This is provided to add topology information to allow for more routes to be selected in the routing algorithm.

**MPR Redundancy information** This extension allows adding a stronger coverage criterion to the MPR selection algorithm, requiring to choose MPRs so that each 2-hop neighbor is reached by more than one (this is parameterizable) MPR node whenever possible.

### 3.3.2 Motivations

The whole development process was directed by two principal motivations: clarity and efficiency. The code had to be evolutive because the project was to be the experimentation platform for the next two years of work for the Safari Project. Clarity was thus required as a means to work efficiently and not having to compromise large parts of the project by small changes. Besides, efficiency was required to make the project ready for definitive exploitation at each compilable stage.

The idea was that since the model can be simply expressed formally, the code should be expressed in simple concepts that would hide somehow the intricacies of the inner functioning away from the programmer. Nevertheless, clarity could not be preferred at the expense of efficiency.

Existing C++ techniques for static generic programming are a way to satisfy these two requirements if applied correctly.

According to the draft OLSR specification, the protocol can be expressed in a way that abstracts the notion of address and thus the model could be made independent of the address family used. So providing the model with a pluggable IP family was a motivation since the beginning.

The name of the actual project was found at the end of the internship. At that time I asked whether the code could be released as free software under the GPL licence, a question I have not thought about asking at the beginning, but which was positively answered as soon as property issues were clarified with other members of the Safari project. So I had then to find a suitable name for the project if I wanted it to be made available publicly. So I applied the same method that was used for Samba: I grepped the dictionary for words containing the four letters O L S R. I finally found "polyester" and decided to flip capital P, which then looks very much like a lower-case Q shifted upwards. This was fortunate, since the project was aimed to be carried on beyond my internship and was destined to have QoS features included in the following months.

### 3.3.3 General architecture

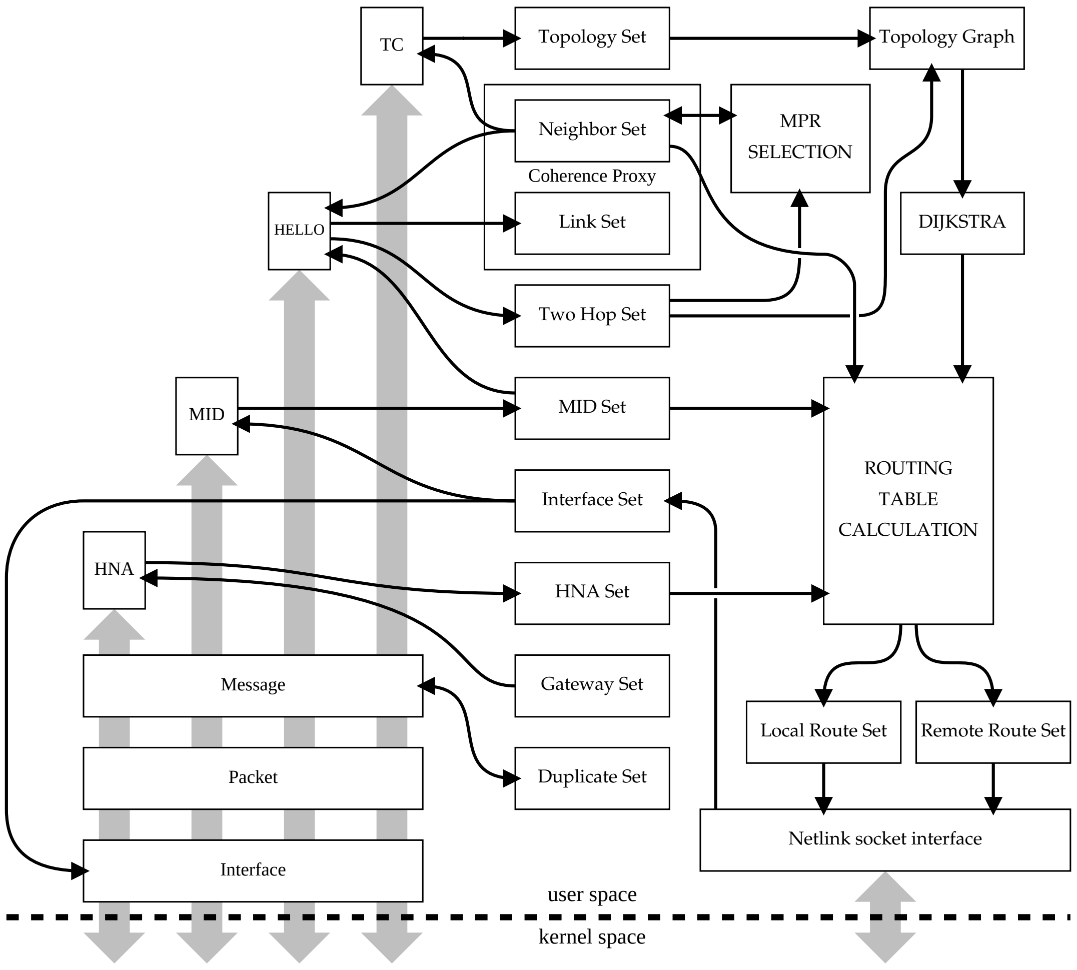


Figure 3.7: The general architecture of **ꟼolyester**. The part on the left represents message exchange, the thick grey arrows show the way data travels from and to packets. In the remaining part, boxes with capitalized text represent procedures, whereas the other boxes represent actual data sets. The thin black arrows indicate the flow of information between messages, sets and procedures. The dashed line on the bottom shows the limit between the user and kernel spaces and exhibit the system-dependent interface.

The whole project is about sending and receiving messages in packets and managing sets that affect route calculation. Hence it is a mix of functionality of high level of abstraction (elements and sets) and low level of abstraction (sockets, network interfaces and routes).

All the framework evolves around a scheduler that triggers timed events and input/output events (though only input events are used for the time being).

**Source organization**

This conceptual segmentation implied the division of the source directory into the following subdirectories:

| | |
|---|---|
| `alg/` | General algorithms. |
| `cst/` | Preprocessor, OLSR-specific and compile-time defaults. |
| `gra/` | Graph data structures. |
| `msg/` | Message hierarchy. |
| `net/` | IP addresses and interfaces. |
| `pkt/` | Packet class. |
| `sch/` | Scheduler and event hierarchy. |
| `set/` | All set classes. |
| `sys/` | System interface for sockets, interfaces and routing. |
| `sys/linux/` | Linux kernel specifics. |
| `utl/` | General purpose utility classes. |

Particular classes are grouped together into a separate source file, to ease code maintenance. For each class, at most three files are used.

For example, take `set::DuplicateSet` class and its surroundings:

| | |
|---|---|
| `set/forward.hh` | This contains all pre-declarations of the `set/` directory along with `typedef` aliases for convenient use outside of the `set` namespace. |
| `set/duplicate.hh` | This contains all the declaration part of `set::DuplicateEntry` and `set::DuplicateSet` along with the definition of the global `dup_set` instance. |
| `set/duplicate.hxx` | This contains all the methods' and static attributes' definitions. |

This way of organizing the source C++ code minimizes the impact of intra-code dependencies. Indeed, as soon as classes are pre-declared, their types can be used in method prototypes. As soon as a class is declared (the implementation code of the methods is not known yet), it can be instantiated and used inside any other class's methods implementations.

Nevertheless, the declarations are still not fully independent, since a class's attribute which type is incomplete cannot be declared. A way to circumvent this problem is to declare the attribute to be a reference to that type, so that the knowledge of that type's constructors is not necessary. This trick is dangerous, since the attribute's existence must be ensured throughout the class's lifetime. That could be achieved by explicit instantiation in the heap at construction time, but then it may inhibit some kind of optimization the compiler could apply if the instantiation of the attribute was statically known.

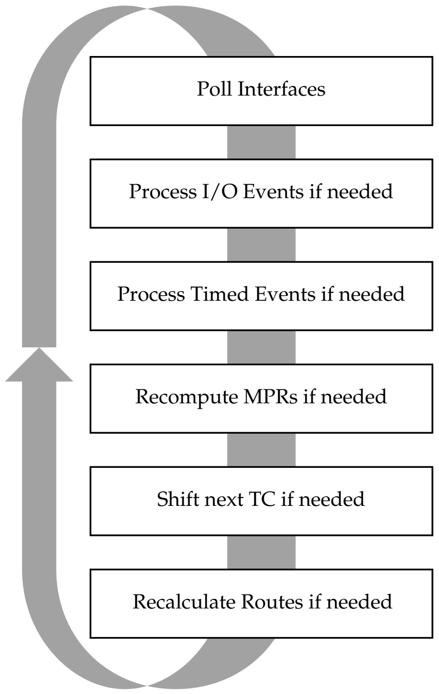
In addition, there are obscure dependence between template parameters declaration.

### 3.3.4 Scheduling

The need for a scheduler is implied by the use of periodic events like message sending and perishable information. There are many sets which elements need to be removed after their validity of time has elapsed.

Since Linux provides each process with only one settable timer functionality — namely the setting of a timer which sends a SIGALRM upon expiration — the use of several simultaneous timers needs to be emulated.

This is done simply by maintaining a set of events ordered by next time of triggering (STL's std::multiset is a great way to implement time ordered sets) and setting the timer to the next time of triggering.



Poll Interfaces
Process I/O Events if needed
Process Timed Events if needed
Recompute MPRs if needed
Shift next TC if needed
Recalculate Routes if needed

Figure 3.8: The main scheduler loop

At the same time, the scheduler uses the poll(2) system call to listen on interfaces for incoming packets. In normal operation, most of the time is spent in the poll(2) system call. When a SIGALRM is received, the associated signal handling function is called. The sole purpose of the signal handler is simply to mark the timed event set for processing. This marking is an atomic operation which is important since signal handling can virtually happen while the process is doing anything. When a signal is delivered and the process was inside a system call, the latter returns an error indicating that it was interrupted by a signal (EINTR). This way the poll(2) call returns and scheduler can check the mark for timed events and process all timed events which time of next triggering has elapsed. After having processed a timed event, the scheduler removes the event from the set, resets its time of next iteration and inserts the event back into the set. An input/output is triggered simply by poll(2) returning a flag indicating that there is information to be received or that some file descriptor is ready for writing.

There are two kinds of timed events: OnceEvent and PeriodicEvent. The first is intended for those timed events that are to occur only once, whereas the second is for those that are to occur periodically. Periodic events support the so-called **jitter** in their triggering times, i.e. random amount of time that the event is triggered ahead of its normal time of triggering. This functionality is required for the sending of packets to avoid nodes to end up sending packets synchronously, since that would make the packets collide.

### 3.3.5 Message handling

#### Receiving

When a packet is received, it is read from the file descriptor and fed into a `pkt::Packet` which along raw data contains also information about the sending interface. The packet is then parsed with `pkt::Packet::parse()` for message extraction.

For each message, after having checked its integrity (mostly against the message size declared in the message header), the packet parser forwards the message for further parsing to the message parser `msg::Message::parse()` along with the addresses of the sender and receiver interfaces.

The message parser then makes some integrity checks of its own. At this point, a special case is made for HELLO messages which cannot be forwarded. If the message is a HELLO message, further processing is handed over to `msg::HELLOMessage::parse()` and processing stops. If the message is of another type, the parser checks if the packet is not a duplicate, i.e. that is has not been already processed earlier. The duplicate check is achieved thanks to the duplicate set. If the type of the message is known, further processing is handed over to the respective specific parser.

Whether the message type is known or not, the message is further considered for forwarding on all interfaces according to the rules described in the RFC. If the message is to be forwarded and the message type is known, the forwarding is handed over to the specific forwarder. Otherwise, the message is handed over to the default forwarder `msg::Message::forward()` which implements the default forwarding algorithm as described in the RFC. For now, any known message that is forwarded is done so in the default way, so every specific forwarder in fact directly calls `msg::Message::forward()`.

#### Sending

Sending and forwarding is done pretty much the same way. Messages are never sent explicitly from the core of the system. Since messages must be sent with a jitter correction in time, messages are first scheduled to be sent later. The same holds for forwarding.

The events handlers for message sending or forwarding do not perform packet sending either. They only put the message in the pending queue that is checked by the scheduler after every timed events have been processed.

If there are pending messages to be sent, the scheduler then triggers the packet generation mechanism in `set::InterfaceSet::send_messages()`, which builds packets one interface at a time, for message generation is dependent on the interface on which the message is to be sent.

This is a way to control the extent of message aggregation in packets. The construction of the content of locally generated messages is deferred until the packet is filled with messages. This allows messages to be generated according to the available room in the packet currently being generated. For each pending message, the respective dump method is called (for example `msg::HELLOMessage::dump()` for HELLO messages). The return value of the method allows to decide whether the message needs to be regenerated/resent immediately because it did not fit in the current packet.

The reason of the retarded message generation is due to the possibility of the messages to be partial, i.e. contain only part of the information they advertise. This allows to have very large amounts of information to be advertised in several messages. Each entry in a message has an associated validity time (usually given in the message header and common to all the entries) that indicates how long the information is to be kept valid in the neighbors' internal sets. Therefore, one node must ensure it advertises each entries often enough to forbid expiration in neighbors' sets. This is achieved using time-stamps on local advertised elements. When a new message is generated, the entries are inserted in the order of increasing last time-stamp and are stamped as they are added to the message. When a node detects that there is no room left and there are entries remaining to be advertised, it checks whether their stamp will be too old at next regular generation. If those entries cannot wait for the next round, an additional packet is created and filled with the remaining entries.

After all the pending queue has been processed, the packet is sent on the interface. If there are messages remaining in the queue, another packet is generated and so on, until the queue is empty.

Packet generation is done in such a way that all the packets are composed of the same proportion of each message type, but message contents can differ[1], since they depend on the interface for which each message is generated (a good example of this are HELLO messages.

### 3.3.6 Sets handling

#### The delete set

This is a special set aggregating instances of subclasses of `set::Deleter` that are specialized versions for any concrete type of object they are supposed to act upon. In fact the delete set is mainly used for garbage collecting. Each time a perishable object is inserted to its respective set, a deleter tagged with its date of removal is inserted into the delete set. This deleter instance contains an iterator pointing to the element just added. Then once in a while, a "deleting" event is triggered that iterates over the delete set in order of increasing date of removal and triggers the action defined in the deleter (namely the `set::Deleter::execute()` virtual method).

When a perishable element of a set is removed in any other way from the set, the corresponding deleter is removed from the delete set.

#### The link and neighbor set

These are the two most complicated sets to manage. This is due to the fact that they are correlated in the following way: a neighbor exists if and only if it has one or more associated valid links. Hopefully neighbors are not inserted explicitly into the neighbor set but must appear when a corresponding link appears in the link set. On the other hand, the link set is populated explicitly upon reception of HELLO messages. Besides, a neighbor's status changes when certain properties on its links change.

Therefore, access to those two sets is shadowed by a proxy object that takes care of all the coherence maintenance between the two sets. Any operation on one of the sets is performed through the use of an instance of the `set::CoherenceProxy` which contains instances of `set::Link` and `set::Neighbor`.

The proxy class offers many ways of accessing the links and neighbors for several different purposes. HELLO message generation requires iteration on all the valid (but not necessarily symmetric) neighbors in increasing last HELLO stamp order. TC message generation requires iteration on all the valid and symmetric neighbors in increasing last TC stamp order. The MPR selection algorithm needs to iterate over the symmetric neighbors in no particular order. All these variants of the neighbor set are implemented using subsets and indexes.

#### The 2-hop neighbor set

This set contains all the nodes that are symmetric to a symmetric neighbor. It is populated from the HELLO message parser and its elements are perishable but can be explicitly removed by the HELLO message parser if it detects that the connectivity is not actual anymore or that the interface address was not the 2-hop neighbor's main address. There may be several entries for one given 2-hop neighbor, because it may be symmetric to several different symmetric 1-hop neighbors. This redundancy is wanted since it is a key aspect exploited in the MPR selection algorithm.

[1]This point was not obvious from the draft version 11 but has to be inferred from the way packets are forwarded on multiple interfaces

#### The HNA set

This set contains all the Host-Network Association entries to be taken account of during routing table calculation. It is populated by the HNA message parser and its elements are perishable (the garbage collector is the only way to purge invalid elements).

#### The MID set

This set contains all the Multiple Interface Declaration entries to be taken account of during routing table calculation and main address calculation for some 2-hop interfaces. It is populated by the MID message parser and its elements are perishable.

#### The topology set

This set contains information about the network topology contained in the TC messages received from other nodes. It is populated by the TC message parser and its elements are perishable but can be explicitly removed by the TC message parser if a TC message with a greater ANSN is received.

#### The duplicate set

This set contains information about the messages already processed and already forwarded. It is populated by the message forwarder and its elements are perishable (the garbage collector is the only way to purge invalid elements).

#### The gateway set

This set contains information about the locally advertised external (non-OLSR) networks in HNA messages. It is populated at startup and its elements are not perishable. It is possible to explicitly remove some of its elements, but this feature is not used at this time.

This set contains an index to allow iteration in increasing stamp order, provided for HNA message generation.

#### The interface set

This set contains information about all the OLSR interfaces used. It is populated at startup and its elements are not perishable. It is possible to explicitly remove some of its elements, but this feature is not used at this time.

This set contains an index to allow iteration in increasing stamp order, provided for MID message generation.

### 3.3.7 Linux kernel specifics

This subsection presents all the specific problems I had to solve in order to make the things actually work. Whereas all the core features of OLSR are system independent, since OLSR is a routing protocol, the program must dive into the system specific dirt at some time.

#### Network interface information retrieval

First of all, at initialization, network interface information must be retrieved for two purposes: minimum interface MTU calculation and IP address retrieval.

Packet construction requires to know in advance the maximum data size that a network packet can contain to avoid fragmentation by the IP layer. If a broadcasted packet is fragmented, chances to loose it are multiplied and we cannot rely on flow control to avoid collisions.

On Linux kernels, there are several ways to retrieve a given interface's MTU, but I chose to use the newest way which is the use of Netlink communication. Netlink sockets are a way for user-land processes to communicate with parts of the kernel. It is already used for interface parameter setting and retrieval, interface address setting and retrieval, firewall parameter setting and retrieval, ARP table management, IPv4 and IPv6 routing table management and increasingly many others. This was thus the best thing to use in order to spare efforts, since all three system dependent operations we needed were possible by the use of Netlink sockets.

Since Netlink messages must be built and parsed in a very specific way, I decided to make use of a well known design pattern in object oriented modelling: hierarchies of message and visitor classes.

So to extract interface parameters, one has only to build a request using predefined request message classes and define a specific visitor that extracts the relevant informations from the kernel response messages.

#### Routing table management

Routing table management is done in the same way interface information is retrieved. An information retrieval or setting request is instantiated and sent to the Netlink socket. In fact, retrieval is not used at this time, since the program maintains its own version or OLSR route set, to avoid having to parse the routing table entries each time an operation is performed. So only good completion of the setting operation is checked after a request has been sent.

When one is manipulating the routing table, one has to be careful not to try to set a new route with a gateway for which no route actually exists. Therefore, routes are separated into two kinds: local routes and remote routes. A **local route** is one that concerns a destination directly reachable through one of the node's interfaces and is defined by its destination address, a prefix length and an output interface. On the other hand, a **remote route** is one that concerns a destination which is not reachable directly but only through a relaying node that is accessible directly (i.e. for which a local route exists) and is defined by its destination address, a prefix length and a gateway address. So to avoid incorrect route insertion, local routes need to be added before remote routes. In the case of route removal, the order is reversed: remote routes are removed first.

To maintain maximum connectivity while updating the routing table, a node maintains an internal copy of its routes (which as said before, avoids having to parse the routing table). When a new version of the routing table is calculated, only the difference between the old table and the new table is processed. The difference is calculated in both ways, to get routes to be removed and routes to be added separately. So the routing table updating scheme goes as follows:

1. Add new local routes
2. Add new remote routes
3. Remove old remote routes
4. Remove old local routes

### 3.3.8 Various utility features

#### Time calculation

Each time a time duration or date is needed, the `utl::TimeVal` class is used. This class implements all needed time operations such as getting the current time, addition and subtraction of dates, multiplication and division by a float, etc.

At many places, current time is required for comparison or time-stamping purposes. To avoid having to call the `gettimeofday(2)` system call each time the current date is needed as well as to allow checking for simultaneity and because that date would not be exact anyway (since Linux is not a real-time kernel), the current time is took from a static attribute of the class that is updated once in the scheduler loop. Thus every operations in one scheduler loop appear to be simultaneous. A static method returning the effective real time is provided for future use (e.g. for time-stamping received and emitted packets) and for the use in the `SIGALRM` handler.

#### Validity and holding time calculation

In packet and message headers, time durations are often expressed in a special two-byte floating point format which operations are described in the RFC. This is implemented in the `utl::Vtime` class and supports construction from and coercion to a raw two-byte word for use in message and packet construction. Of course, conversion from and to `utl::TimeVal` is provided.

#### Sequence numbers

Packet and message sequence numbers are defined with specific arithmetic and comparison operations to allow overlapping. The `utl::Seqnum` meta-class allows the use of concrete sequence number classes instantiated for a particular internal representation.

#### Shared data buffers

To allow easy memory management of temporary buffers used in packets, special `utl::Data` and `utl::ConstData` classes are provided. They implement buffer sharing that minimizes whole buffer copying in message forwarding, etc.

#### Stampable components

The `utl::Stampable` component class is provided for construction of time-stampable objects to be used in sets indexed by time-stamp order.

#### Static iterator decorators

The management of perishable elements in sets required the use of several special design patterns that would hide all the underlining checks for validity. Thus a particular set can implement a plain set of elements internally and provide access to a so-called "masked" set for external use. Such a masked set is just a special iterator that gives access only to elements that verify a given predicate (for simple perishable sets, this predicate is true if and only if the element had not yet expired) and when incremented, "jumps" over other element until the predicate is true or the end of the set is reached. Thus `utl::MaskIterator` is a static decorator that is instantiated with a particular iterator, set, access predicate and action functor to form a new iterator class. The action functor is provided for the case when an element that does not verify the predicate must be acted upon (e.g. time-stamped).

For ordered sets that index elements on one particular attribute, there is no real reason why an iterator of a mutable set should dereference to a constant element, forbidding modification of "non indexing" attributes. For this purpose, there is a `utl::DeconstIterator` static decorator that dereferences to mutable objects. The only thing that the programmer has to do to keep safety is to declare indexing attributes to be constant in the element class. This could even be ensured statically using meta-programming techniques (but the subject has not been studied exactly yet).

There are many places where a set of elements needs to be iterated on in different orders. To ensure the fastest possible iterations, different ordered sets would have to be kept at all time. But to avoid keeping multiple sets of the same element type and thus multiple copies of the same element, there had to be one set of the elements *per se* and the other sets would be composed of iterators to elements contained in the first set. But to hide this trick from external use, the iterators on the other sets would have to automatically dereference correctly to the object itself. For this purpose, the `utl::DerefIterator` static decorator was created along with a type traits system to allow correct dereference if the decorator decorates an already decorated iterator.

All these iterator decorators automatically add information in the `std::iterator_traits` for compatibility with the STL.

#### Static set decorators

To simply implement indexes, `utl::Index` is provided with all correct methods and `utl::DerefIterator` iterator definitions. The internal representation (`std::set` or `std::multiset`) can be chosen at instantiation time along with the element and original iterator type.

To simply implement masked sets, the `utl::Subset` set decorator is provided with all correct methods and `utl::MaskIterator` definitions. All parameters required for `utl::MaskIterator` are parameters of `utl::Subset` with reasonable defaults, to allow maximum versatility.

### 3.3.9 Particular aspects with respect to RFC 3626

#### Routing table calculation

The RFC document describes some kind of greedy shortest path algorithm based on the topology set and the symmetric neighbor set to compute routes towards every node in the network. The algorithm is similar to the Dijkstra algorithm which find shortest paths in terms of number of hops.

Since **Ꝑolyester** is aimed at experimentation of other routing metrics not necessarily expressed in number of hops (e.g. transmission delay, available bandwidth, packet loss probability, etc), the shortest path algorithm had to be extracted and rendered independent of the core OLSR data structures. This was done by writing a real Dijkstra algorithm and by maintaining a real graph structure along with OLSR internal sets. This way, any graph operation can be later performed on the topology graph without interfering with the core.

The routing table calculation is done in two steps, local routes are calculated first and only remote routes are calculated using the Dijkstra algorithm. There are numerous reasons for this distinction:

- Local routes are different from remote routes as far as routing tables are concerned.
- The routing calculation algorithm must provide a gateway address for each remote route and that's not easy if one only known the output interface.
- A direct destination has to be reached directly and not through another node. Even considering other QoS metrics, this assumption cannot be false since sending to one node affects the medium all around the sending node, so direct sending to a 1-hop neighbor is always the most efficient way of reaching it.

### 3.3.10 Aspects of the free software project

Since the project was to be released under the GNU GPL[29], I wanted it to have all friendly features of free software projects that make them easier to port and compile.

#### Autotools

The code as it has been first written was relying on the kernel's providing the Netlink socket functionality. Since as far as I know only Linux 2.4 offers it, the code is highly dependent on it.

Nevertheless, system-dependent features being clearly separated from the core features of the project, porting should be easy.

To help managing system and compiler particularities, the use of autoconf/automake [30] comes in handy. It provides user-friendly tools to configure and compile the project. It allows to easily configure the source tree to be compiled with a cross-compiler, which in this case was very interesting for experimentations. Another handy feature is the ability to configure the sources in two different ways at once using two separate build directories. This allows making modifications to the sources and compiling it immediately for the workstation and the PDAs, without making two separate configurations one after another.

### Doxygen

Doxygen is a nice tool that extracts code documentation from specially formatted comments in the source files and uses them to generate either electronic documentation in HTML or printable documentation in LaTeX or PDF.

### Web site

The need for a minimal web site was obvious since the first snapshot release of **Qolyester** and has been created quickly[1], in order to present the project to collaborating groups from the Safari project.

The site still lacks some interesting features as can be provided by sites such as Savannah[2] or SourceForge[3] like mailing lists with public archives, a bug-tracking system, public CVS tree, etc.

The opening of mailing lists have been delayed due to the current impossibility to use GNU Mailman[4]. The only available mailing list system on the LRI network is Majordomo which does not provide such convenient web interface, so there are ongoing efforts to find a good solution. But this is not (yet) a dramatic issue since current number of known users can be reached by mail individually without much trouble. The email address provided as a contact[5] forwards incoming mail to the four people currently involved in the development and experimentation.

[1]QoS support in OLSR `http://qolsr.lri.fr`
[2]`http://savannah.gnu.org`
[3]`http://www.sourceforge.net`
[4]`http://www.gnu.org/software/mailman/`
[5]`qolsr@lri.fr`

# Chapter 4

# Conclusion

## 4.1 Interest of the internship for the LRI Networking group

#### An original point of view

My personal experience being very heterogeneous, I hope I have some interesting thoughts at the crossing of fields and bringing original points of view.

#### Strong Linux skills

My rich experience with Unix-like operating systems allowed me to be quickly operational for an efficient use of the tools that were provided to me for experimentations. I had no particular problem with dealing with incompatible patches or kernel recompilations. I was not afraid by specific tasks of Unix and network administration and I familiarized quickly with new tools for IPv6 network configuration.

#### Strong working independence

Through the years, including before beginning studies at ÉPITA, I was a self-taught computer technician and engineer. I know where to look for the information I need[*], so I am very independent in my work. This was a key point during my internship because the work I did most of the time was quite different from other workers'. Nevertheless, interaction was more evident during the bibliographical research period, since this was exactly the field of the people surrounding me.

#### A starting point for the first deliverable

My bibliographical work ended with the writing of a state-of-the-art paper on the subject of QoS routing in mobile ad hoc networks which was used as a starting point for a more general state-of-the-art deliverable on routing models for mobile ad hoc networks required by the Safari time-line.

#### A working experimental platform

The main interest for the group has been to get a working implementation of the OLSR protocol with IPv6 support for the in-the-field experimentations of QoS-aware routing algorithms. These methods were previously worked out and formally described in scientific publications. But theoretical proof of routing algorithms is not of much interest, since there are lots of uncontrolled parameters in real world applications. Routing algorithms are usually tested in network simulators that approximate physical conditions with simpler models and spit out statistics about extreme

[*]Yes, I do actually RTFM.

cases like huge numbers of participating nodes or harsh physical conditions. But nothing beats real world, in-the-field, down-the-street, deep-inside-the-reinforced-concrete-jungle experimentations. Sometimes certain unobvious features appear only with a handful of nodes.

Another way for the group to get a working OLSR implementation with IPv6 support would be to wait until the HIPERCOM team at INRIA Rocquencourt gets IPv6 support into its own implementation. That is obviously not an easy task, since it seems that they are still in the process of doing it.

The ability to re-implement the protocol from scratch using C++ allowed faster development and still a high degree of efficiency. If the same had to be achieved using the C language, it would require much more time, since all the core parts that are dealing with sets would have to be coded by hand (or using some kind of dynamic library which would have slowed down the code). Unfortunately C++ static genericity programming techniques are not widely known (yet) in the networking community and C is believed to be the most efficient language.

## 4.2 Personal interest of the internship

My personal interest of the internship is manifold.

#### Social interest

The ability to confront by vision and my working methods with those of people from multiple backgrounds was a very exciting experience. I think diversity is the most important source of creative wealth. This was the occasion for me to meet people from a quite different field, although academic it was more linked to industrial concrete applications. While I happened to have slightly negative feelings regarding industrial applications, I convinced myself that they can be very interesting in academic working contexts, as long as commercial profit does not taint the technical challenge.

#### Professional interests

This professional experience was a good way for me to meet another scientific research working environment. The whole field of networking appeared to be really fascinating and at the same time very rewarding when I could see the practical result of my work.

It appeared to have pleased my employer as well, since I have been hired as a part time engineer to continue working with the team, during my master year. I would very much like to work further theoretical matters involving QoS in wireless networks during the master's internship.

#### Free software engineering experience

I am very pleased with the fact that my work ended up in a free software project. Although this takes me a lot of time, I would like to maintain it as long as I could to see it evolve and gain interest as new features are added along.

For the time being, a few people have contacted me with question regarding its use and web page statistics show that people continuously monitor it and download the snapshot packages.

I think that mobile wireless ad hoc networks, though not yet in common use should gain more and more interest over the years to come. Besides, I think that **Ꝗolyester** will greatly benefit from the current experimental work with QoS routing when first QoS-featured releases will hit the FTP server(s).

## 4.3 Looking back and perspectives

I am continuously asking myself whether that work could not have been done in a more efficient way. Although I have hit bumps along my way, I think that code needs some maturing time and that although modelling techniques exist to quickly built a project architecture, I don't think any of them would have given a solution for a problem that is both abstract and constrained by concrete requirements (memory consumption and speed).

If I knew since the beginning that I would have to re-implement the whole protocol from scratch, I would certainly begin the internship doing that part in order to have more time later to experiment with QoS routing methods. On the other hand, I think that without the theoretical knowledge I acquired during my bibliographical study, I would maybe have missed some important features that have been directing the model since the beginning. The implementation would maybe be not as modular as it is today.

Now my perspectives related to this are simply the continuous work on the project, though maybe not alone anymore, since a fourth year student is going to assist me during his internship starting mid-January. I would very much like to bend my work towards the study QoS routing models including QoS metrics measurements which appear to be quite a challenge at this time.

# Bibliography

[1] IEEE 802.11 Wireless. `http://standards.ieee.org/getieee802/802.11.html`.

[2] Programming languages — C++. International Standard 14882, ISO/IEC, September 1998.

[3] David B. Johnson, David A. Maltz, and Yih-Chun Hu. The dynamic source routing protocol for mobile ad hoc networks (DSR). Internet-Draft Version 09, IETF, April 2003.

[4] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561, IETF, July 2003.

[5] R. Ogier, F. Templin, and M. Lewis. Topology dissemination based on reverse-path forwarding (TBRPF). Internet-Draft Version 11, IETF, October 2003.

[6] T. Clausen and P. Jacquet. Optimized link state routing (OLSR) protocol. RFC 3626, October 2003.

[7] *IPv6, Théorie et pratique*. Third edition, March 2002.

[8] S. Bradner and A. Mankin. IP: Next generation (IPng) white paper solicitation. RFC 1550, IETF, December 1993.

[9] S. Bradner and A. Mankin. The recommendation for the ip next generation protocol. RFC 1550, IETF, January 1995.

[10] S. Deering and R. Hinden. Internet protocol, version 6 (IPv6) specification. RFC 2460, IETF, December 1998.

[11] J. Postel. Internet protocol. RFC 791, DARPA, September 1981.

[12] R. Braden. Requirements for internet hosts — communication layers. RFC 1122, IETF, October 1989.

[13] F. Baker. Requirements for IP version 4 routers. RFC 1812, IETF, June 1995.

[14] R. Hinden and S. Deering. IP version 6 addressing architecture. RFC 2373, IETF, July 1998.

[15] R. Hinden. Proposed TLA and NLA assignment rules. RFC 2450, IETF, December 1998.

[16] S. Kirkpatrick, M. Stahl, and M. Recker. Internet numbers. RFC 1166, IETF, July 1990.

[17] S. Thomson and T. Narten. IPv6 stateless address autoconfiguration. RFC 2462, IETF, December 1998.

[18] R. Hinden and S. Deering. IPv6 multicast address assignments. RFC 2375, IETF, July 1998.

[19] *IEEE Std 802.11-1999 Information Technology— Telecommunications And Information exchange Between Systems— Local And Metropolitan Area Networks— specific Requirements— Part 11: Wireless Lan Medium Access Control (MAC) And Physical Layer (PHY) Specifications*, 1999.

[20] C. Hedrick. Routing information protocol. RFC 1058, IETF, June 1988.

[21] Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4). RFC 1771, IETF, March 1995.

[22] J. Wroclawski. The use of RSVP with IETF integrated services. RFC 2210, IETF, September 1997.

[23] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. RFC 2475, IETF, December 1998.

[24] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (DS field). RFC 2474, IETF, December 1998.

[25] R. Braden, D. Clark, and S. Shenker. Integrated services in the Internet architecture: an overview. Technical Report 1633, IETF, 1994.

[26] Chunhung Richard Lin and Jain-Shing Liu. QoS routing in ad hoc wireless networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1426–1438, August 1999.

[27] T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, a. Qayyum, and L. Viennot. Optimized link state routing protocol. In *IEEE INMIC Pakistan*, 2001. Best paper award.

[28] J. Moy. The OSPF specification. RFC 1131, IETF, October 1989.

[29] The GNU General Public License. `http://www.gnu.org/copyleft/gpl.html`, June 1991.

[30] G. V. Vaughan, B. Elliston, T. Tromey, and I. L. Taylor. *GNU Autoconf, Automake, and Libtool*. New Riders, October 2000.

# Appendix A

# Glossary

| | |
|---|---|
| **Boot-loader** | A small piece of software that usually resides on read-only memory (though often erasable) that is in charge of loading the kernel image into memory and start the operating system. |
| **CDMA** | Code Division Multiple Access. |
| **CSMA/CA** | Carrier Sense Multiple Access with Collision Avoidance. |
| **CSMA/CD** | Carrier Sense Multiple Access with Collision Detection. |
| **DiffServ** | Differentiation of Services. |
| **Hop** | One network link as a part of a **route**. |
| **IEEE 802.11** | The family of standards published by the Institute of Electrical and Electronics Engineers, describing the Medium Access Control (MAC) and physical layer specifications. |
| **IntServ** | Integration of Services. |
| **IPv6** | The next generation of Internet Protocol that supports a much greater address space and several other interesting features like auto-configuration, security, etc. |
| **Iterator** | The abstraction of a pointer in object-oriented programming. Most of the time used to iterate on each elements of a set. |
| **Link** | The relation between two **nodes** that represents their ability to exchange information directly. |
| **Node** | An autonomous system with one or more network interfaces. |
| **Packet scheduling** | The ability to process incoming and outgoing data packets in a better way than in a First In First Out (FIFO) fashion. |

| | |
|---|---|
| **QoS** | Quality of Service. |
| **Route** | The sequence of **nodes** that connects one **node** to another in a network. |
| **Routing** | The ability to forward data packets not destined to the current **node** to the correct next **hop** towards the destination. |
| **Static decorator** | A generic design pattern that adds a set of new features to an already declared class. |
| **Static genericity** | The ability to program using elements of reusable software that are specialized and optimized at compile time. |
| **Station** | $\longrightarrow$ **Node**. |
| **TDMA** | Time Division Multiple Access. |