Università degli Studi di Genova

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea Specialistica in Informatica

Anno Accademico 2006/2007

Tesi di Laurea Specialistica

Multi-platform architecture for routing protocol design for wireless and ad-hoc network

Candidato:

Valentina Gaggero

Relatori:	Correlatore:
Prof.ssa Gabriella Dodero	*Prof.ssa Marina Ribaldo*
Prof.re Pietro Manzoni	

Index

1 - Introduction 9

2 - State of art 10

2.1 - – Background 10

2.2 - – Introducing PICA 19

2.3 - – My objectives 20

3 - – Exploratory testing for the study of PICA 22

3.1 - Used methodology: Exploratory testing 22

3.2 - used tools 28

4 - – PICA library 30

4.1 - PICA architecture 30

4.2 - PICA primitives 39

4.3 - - test PICA’s functions analyzing their inputs and limits....... 44

4.4 - 4.4 – Integration test 50

5 - OLSRv1 with PICA 54

5.1 - Used tools 54

5.2 - OLSR project preview 55

5.3 - PICA in OLSR 56

6 - OLSR improvement 60

6.1 - Presenting OLSRv2. 60

6.2 - Adding feature: gateway and attached network 62

6.3 - Results 74

7 - Figure index 75

8 - References 77

Introduzione

Negli ultimi anni il mercato delle tecnologie wireless è cresciuto velocemente e offre agli utenti un ampia scelta sia in tipologie di device (cellulari, palmari, laptop) che in servizi.

Inoltre, il mondo delle applicazioni wireless è molto mutevole, e pertanto si ha la necessità di mettere rapidamente sul mercato applicazioni che possano essere utilizzate con differenti device e, quindi, essere eseguite in differenti ambienti di sviluppo.

Ciò ha portato a sentire la necessità di una libreria che nascondesse la diversità trai vari ambienti di sviluppo: da questa necessità è nata PICA, una libreria, sviluppata in ANSI C, che permette di sviluppare applicazioni per reti.

Solitamente i principali compiti di questo tipo di applicazioni sono tre: gestione dei processi, della memoria e della comunicazione.

Pertanto PICA offre un insieme di primitive che permettono di svolgere tali compiti nascondendo però le differenze tra un ambiente di sviluppo ed un altro: è possibile sviluppare un’ applicazione che utilizzi primitive di sistema, senza però preoccuparsi di quale sia il sistema target su cui essa verrà eseguita.

La gran parte delle primitive del sistemi operativi basati su Windows e quelle dei sistemi operativi di Unix differiscono, a volte solo nel numero e/o tipo di parametri, a volte anche nel comportamento. Di conseguenza PICA nasconde anche questo ultimo tipo di differenze omogeneizzando questi comportamenti in modo tale che la stessa funzione abbia lo stesso comportamento in su tutte le piattaforme. Per esempio la funzione *select()* è comunemente usata negli ambienti Unix per aspettare gli eventi associati a qualsiasi tipo di risorsa; invece, la famiglia dei sistemi operativi basati su windows offre tale funzione solo per i socket. In questo caso, in windows, PICA simula il comportamento della funzione *select()* in Unix utilizzando un thread, che attende sui socket, e le primitive della famiglia WaitFor per tutti gli altri tipi di risorsa. Quando avviene un evento sui socket, il thread lo comunica al processo principale tramite una pipe.

Tutto questo meccanismo rimane del tutto trasparente all'utente che quindi può chiamare la funzione *PICAselect()* sia su sistemi Unix che windows ottenendo lo stesso comportamento.

Questa tesi si occupa di aggiornare e documentare PICA e di utilizzarla nello sviluppo di alcune funzionalità della versione 2 del protocollo di incamminamento OLSR (Optimized Link State Routing) [8].

Quindi, la prima parte attività consistette nello studio del codice di PICA: data l'assenza di documentazione, questo studio si è ricondotto all'esecuzione di test che mi aiutassero a capire il comportamento delle funzionalità della libreria considerata.

Gran parte delle metodologie di testing presenti in letteratura prevedono la pianificazione di tale attività già nelle prime fasi di sviluppo del software.

Ovviamente in questa situazione, ciò non era possibile:era necessaria una metodologia che permettesse di studiare la libreria PICA dando delle linee guida nell'investigazione del suo codice (poco documentato e sconosciuto) e che nello stesso tempo permettesse di testarlo.

Per questi motivi si è deciso di utilizzare la metodologia "Exploratory testing". Essa fu proposta da Kaner, Falk and Nguyen [13] e si basa su una vera e propria esplorazione del software da testare seguendo l'istinto del tester e non seguendo delle precise e ben definite linee guida. *Exploratory testing* è una investigazione dinamica del codice effettuata a più livelli di dettagli: si inizia con uno studio superficiale del software e ogni volta si seleziona un'area particolare da investigare più accuratamente. La scelta di questa area viene effettuata dal tester in base ai risultati ottenuti nell'investigazione precedente e in base al suo istinto. È importante notare che la specializzazione del test non significa che esso viene eseguito in isolamento, ma che si verifica più in dettaglio il comportamento di quella parte del software. Il risultato di ogni investigazione devono essere riportati in schemi preparati dal tester. La struttura di tali schemi deve essere scelta dal tester in modo tale da aiutarlo nella sua attività e quindi anche in modo tale da poter utilizzare in modo incrociato i vari risultati ottenuti.

Kaner, Falk and Nguyen hanno suggerito la struttura di alcuni schemi in base al tipo di test effettuato, ma lasciano la libertà di rielaborarli in modo da adattarli alle esigenze personali.

La completa assenza di pianificazione, ma la presenza di linee guida generali hanno permesso di adottare questa metodologia per affrontare l' attività di studio di PICA.

La presenza di linee guida generali ha permesso di adottare questa metodologia per affrontare l'attività di studio di PICA, per il quale era doveroso affrontare la fese di testing senza poterlo pianificare precedentemente.

L'attività di test svolta consisteva principalmente in un test funzionale, cioè una primitiva di PICA veniva considerata come una funzione nel senso stretto matematico. Il suo comportamento veniva studiato dando differente valori ai suoi input; tali valori venivano scelti all'interno, all'esterno e sul contorno del dominio della funzione.

In alcuni casi, questo tipo di test perdeva il suo significato, in quanto alcune primitive di PICA consistono solo nell'invocazione della primitiva del sistema operativo; quindi effettuare su di esse un test funzionale significava testare la primitiva offerta dal sistema. Pertanto, in questi casi, si sono studiate tali primitive all'interno un di contesto concreto.

Il contesto scelto è il protocollo OLSR: esso mi permetteva di analizzare e di studiare alcune funzionalità in un contesto reale dando così l'opportunità di capre se era possibile aggiornare e migliorare PICA.

Questo studio ha permesso la stesura del manuale per l'utente e per il programmatore: mentre nel primo viene descritto come usare ogni primitiva offerta, nel secondo viene descritto l'implementazione delle primitive.

Successivamente a questo studio, mi sono dedicata ad analizzare la proposta della versione 2 del protocollo OLSR pubblicata nel draft "draft-ietf-manet-olsrv2" [12] .

Principalmente in questa proposta non si trovano grandi innovazioni e miglioramenti a livello di algoritmi e strutture dati per la memorizzazione delle informazioni necessarie al calcolo della tabella di incamminamento rispetto alla versione 1; il draft presenta feature aggiuntive e propone standard per la comunicazione e collaborazione tra routers delle MANET.

Tra le varie feature si è deciso aggiungere alla versione già esistente di OLSR basato su PICA quella che permette di diffondere informazioni sulla presenza di gateway all'interno della MANET.

Per esempio, se è presente un nodo che ha un'interfaccia di rete verso Internet, questa funzionalità permette a tutti gli altri nodi di usufruire della connessione ad Internet.

Lo sviluppo di questa feature è stata fatta seguendo la proposta del draft ed adattandola alla versione già esistente di OLSR.

1 - Introduction

I developed this thesis at "Universitad Politecnica de Valencia". I worked in a research group called "Grupo de Redes de Computadores" (GRC) [1]: even if GRC was founded recently, most of its members have a large experience in academic and research activities related with the computer networks field. One of their research topics concerns of design and evaluation of routing algorithms on wireless ad-hoc networks.

The development of communication networks was significant step for mankind, undoubtedly agilizing every day's task and improving the quality of life of many. In recent year, this trend has shifted towards wireless networks.

From commercial point of view, we ca also appreciate the advantage that wireless networks offer. The installation of the so-called "hot-spots" allows cafes, pubs and restaurants to attract more clients, it allows airport, hotels and trains to receive an extra income by offering Internet access and it can also improve the productivity of companies by allowing workers to access the internal network while moving through zones where wired connection are not possible.

Moreover, even the number of device types, is growing up quickly. Hence, it is possible meet and enter in heterogeneous wireless network, that is, network composed by different type of device.

This heterogeneity make more complicated and long developed phase of protocols for wireless networks: each device type has a different programming environment and, in order to develop a protocol network, it is necessary develop it for each programming environment.

With the aim of reducing this time and agilizing development phase, a API, called PICA was design and developed by GRC. This API allows to move to different programming environment without requiring any special porting procedures.

This thesis concerns of updating and testing PICA eliminating malfunctioning. Moreover PICA was ported to new operating systems available on market.

In order to evaluate PICA in a real context, this thesis analyzes OLSR implemented with PICA.

Since a new Internet-draft for OLSR version 2 was published in February 2007, this thesis update the existing version of OLSR based on PICA with features described in the Internet-draft.

2 - State of art

2.1 - – Background

By definition, a **wireless LAN** or **WLAN** is a wireless local area network, which is the linking of two or more devices without using wires. Two different types of basic transmission technologies are used to set up WLANs. One technology is based on the transmission of infrared light, the other one uses radio transmission. Other technologies exist, like Bluetooth and microwaves, but they are less used because they are only a wire replacement and therefore they can be used for connecting point-to-point devices only.

By its nature, wireless technology provides the following advantages and disadvantages.

WLAN are easy to put up because they do not need building construction plan: since radio waves can penetrate walls WLAN can be built without an extra masonry work. Moreover WLAN does not need inter networking units such as switches: this allows adding and removing devices from WLAN without extra efforts. This facility to put up a WLAN lets wireless network survive disaster: if the wireless devices survive, people can still communicate, while networks requiring a wired infrastructure will typically break down completely.

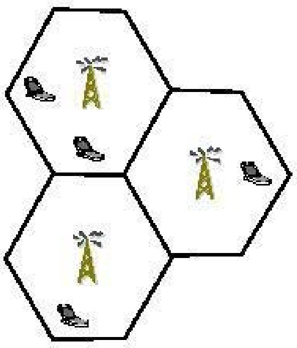
By its nature, WLAN suffers some problems triggered by wireless technology nature: devices can communicate if they are in waves range of other devices, and all devices inside the same range can hear all traffic, thus creating security problems. It is important to point out that wireless devices need a latency time to switch from listening to sending mode and vice versa, raising more collision.

Moreover, nodes mobility makes routing algorithms more complicated, because they have to adapt to frequent network topology alterations.

There are mainly two types of Wireless LAN: Basic Service Set (BSS) and Independent IBasic Service Set (IBSS).

2.1.1 Basic Service Set

In this kind of networks there is a node called “base point” or “access point”. The nodes participating in the network have to register themselves at it, and then all communication are done through these central coordinator nodes as Figure 2-1 shows.



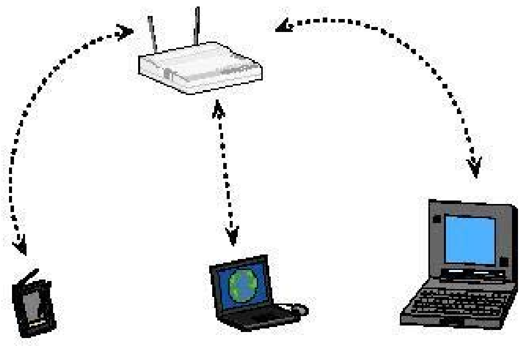


Figure 2-1: Left: Cellular mobile communication with infrastructure.

Right: WLAN insfrastructure mode

2.1.2 Indipendet Basic Service Set

In contrast to communication with infrastructure, communication in IBSS networks is organized completely decentralized. There is no central entity regulating or controlling network traffic. Two nodes can communicate only if exist a directed link between them like Figure 2-2 shows:

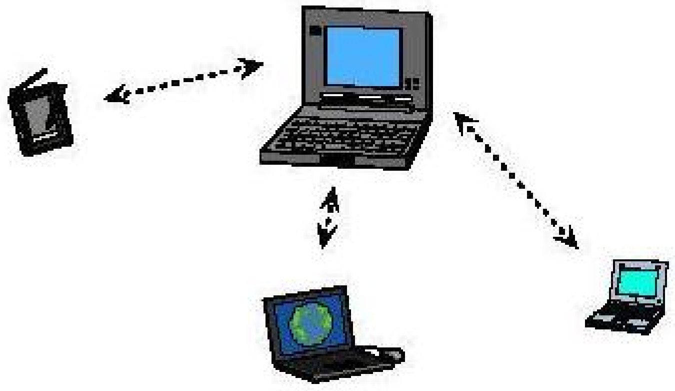


Figure 2-2: direct Communication of wireless node

Therefore, nodes residing sufficiently close to each other to be within radio range can exchange packets without any further measures.

If all nodes can be at the same time nodes originating and receiving network traffic as well as forwarding traffic for other nodes, the network is called “ad-hoc”.

In this kind of network, a node can exchange packets with nodes out of its range, because intermediary nodes forward its packets

In Figure 2-3 node A can establish a communication with node E as nodes B,C,D are forwarding packets exchanged from A to E and vice versa.

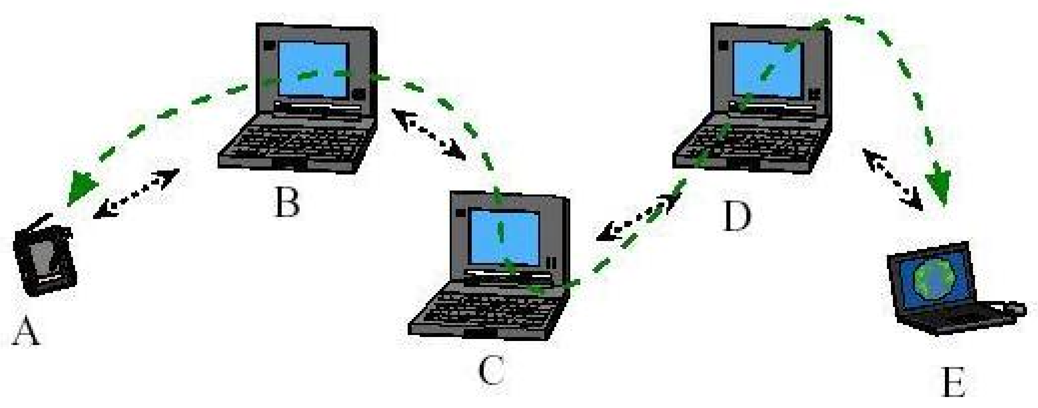


Figure 2-3: exchange of packets by forwarding

Every node thus simultaneously acts as communication endpoint and as a router for other nodes. Movements of nodes can lead to changing network topology. The used routing protocol has to be able to react to such changes.

The advantages of ad-hoc networks are especially based upon their decentralized self-organizing nature, not needing to setup any special infrastructure and to their flexibility with respect to changes. Typical scenarios for the use of ad-hoc networks therefore are mobile units within military operations , search and rescue operations in public safety, emergency and disaster applications, as for example in the WIDENS project [2], vehicular based applications, for example FleetNet [3] and Cartalk2000 [4] and wireless community networks, as for example Freifunk in Berlin [5] using the ad-hoc routing protocol OLSR for their community network.

2.1.3 Ad-hoc routing protocols

The self organizing decentralized control of ad-hoc networks as well as the potentially changing topology in mobile networks set the requirements for ad-hoc routing protocols.

Though communication with direct neighbours is rather trivial, communication with distant nodes requires some knowledge about network topology. The alternative is a pure flooding mechanism, in which all nodes retransmit every packet they receive as data traffic: in practice, flooding has to be avoided, as it makes very bad use of the available bandwidth and leads to a very high number of collisions.

The ad-hoc routing protocols generally are classified in two categories, the reactive protocols and the proactive protocols.

2.1.3.1 Reactive protocols

Reactive protocols determine the route to a destination on demand. If a communication is about to be set up and no route to the destination is already known, a route discovery is initialized. A route request packet is usually flooded through the network. When this packet either reaches a node with a route to the destination or the destination itself, a route reply is sent back to the source node either by link reversal or through flooding of the route reply packet. Routing occurs either in form of source routing or hop by hop.

Source routing

In source routing each data packet contains the entire route form source to destination. Intermediate nodes do not have to maintain up to date routing information for each active route, but rather forward the packets based upon the information stored in the header of the packet. An example for this type of routing protocol is the DSR [6] protocol which has become an internet draft of the mobile ad-hoc network (MANET) working group within the internet engineering taskforce (IETF).

Hop by hop routing

In case of hop by hop routing, each data packet only carries the destination address and the next hop address. Each intermediate node in the path to the destination forwards the packet towards its destination based upon a routing table which it has to maintain for each active route. The AODV protocol as described in RFC 3561 [7] belongs to this category.

2.1.3.2 Proactive routing protocols

Proactive Routing is based upon a table driven approach. Each node has to maintain routing information to the other nodes in the network. This information is usually stored in a number of different tables which are updated periodically and/or upon the detection of changes within the network. Which information is kept and how it is exchanged varies from the used routing protocols. Routing protocols in this category, which have been officially acknowledged by MANET working group, are Topology Based Reverse Path Forwarding (TBRPF) [8], Destination-Sequenced Distance Vector (DSV), and the Optimized Link State Routing (OLSR) [8] protocol.

TBRPF (Topology Based Reverse Path Forwarding)

TBRPF is a link state routing protocol providing hop by hop routing along minimum hop path to each destination. Each node running TBRPF computes a source tree based upon partial topology information stored in its topology table using a modification of Dijkstra’s algorithm. To minimize overhead, each node only reports part of its source tree to neighbors. TBRPF uses a combination of periodic and differential updates to keep all neighbors informed of the reportable part of its source tree. TBRPF performs neighbor discovery using differential HELLO messages which report only changes in the status of neighbors, resulting in HELLO messages being a lot smaller than those of other routing protocols.

DSDV (Destination-Sequenced Distance Vector routing)

DSDV is an adaptation of a conventional routing protocol to ad hoc networks, based on the idea of the classical Bellman-Ford Routing (Distance Vector) Algorithm with certain improvements.

OLSR (Optimized Link State Routing)

OLSR is also based upon the traditional link state algorithm. Each node maintains topology information about the network by periodically exchanging link state messages. The optimization introduced by OLSR is that it minimizes the size of each control message and the number of nodes re-broadcasting a message by employing the multipoint relay strategy.

The local one hop and two hop neighbourhood is discovered through periodic exchange of HELLO messages. Thereafter each node selects some one hop neighbours to be its multi point relay in such a way that all two hop neighbours can be reached through at least one of the selected members of the MPR set. Nodes that are not MPRs can receive and process each control packet, but do not retransmit them and do not announce network topology to other nodes in the network. Nodes that are MPRs of at least one node will forward packets for the nodes that selected them as MPRs and announce all nodes that selected them as MPR by topology content packets to the entire network. Based on its one hop and two hop neighbourhoods and the topology information, each node calculates an optimal route (with regard to hop count) to every known destination in the network, and stores it in its routing table.

After taking into consideration these different types of ad-hoc routing protocols the proactive category was chosen and the OLSR protocol was used in the implementation part of this thesis and is therefore described in more detail in chapter 5 - .

2.2 - – Introducing PICA

In recent years, the number of different devices, used to connect to the network, is growing up quickly. It is sufficient to think about how many kinds of mobile phones, palms and lap-tops are available in the market. Each of these devices has a different programming environment, hence, in order to create a network-based application available for all kind of devices, it is necessary to build one for each of them.

Therefore, to reduce the development time and the testing cycle without affecting the overall performance it was essential to create an “Application programming interface” (API) that allows development of network-based applications hiding differences between platforms.

An ideal API should be simple enough to be easily learned and used by programmers but it should also be as complete as possible to offer a full set of services. This API should also handle all platform dependent issues for the application, allowing it to be moved to different systems without requiring any special porting procedures. The already available solutions to achieve multi-platform compatibility are not entirely satisfactory because they do not provide an API that is useful to protocol designers, especially due to the restricted nature of functions that such a designer needs.

The Java technology [10], for example, not only lacks of such an API, but it also slows down the overall performance of the application due to strong encapsulation. The Cygwin [11] project offers another possible solution but it does not provide some specific kernel calls, neither does it provide new functions in order to aid in protocol development tasks. The large size of its accompanying DLL can also be considered a drawback.

To solve this problem, GRC group created PICA, a library that provides a multi-platform intuitive API for communication protocol designers.

Developing a network-based application with PICA is quite simple, because instead of using system calls, the developer uses PICA calls, and the same code can run on different platforms avoiding extra effort to port it.

For example PICA was used to implements OLSRv1 making it available for Linux, Windows 2000, Windows XP, Windows CE 3.0 and Windows CE 5.0.

2.3 - – My objectives

My objectives are divided in two parts: the first concerned an update on PICA, while the second an improvement of OLSR.

PICA was developed in July 2003, it was available for Linux, winCE 3.0 and windows 2000. My objectives were to improve it adding necessary features and eliminating malfunctioning.

Moreover, in order to make PICA helpful, it was essential to verify its behaviour on different platforms, so it ascertain that its features acts in the same way on each platform. Also, it was needed write user and programmer manuals in order to make PICA easy to use and to give an help to anyone who wants to improve this library.

With the diffusion of more recent operating systems, it was necessary to update PICA for them. Another objective concerned to port PICA to Windows CE 5.0.

The second part of my work consists in analyzing OLSR version 2 (OLSRv2) suggestion and verifying if it was interesting update the current OLSR implementation based on PICA from version 1 with version 2.

That suggestion was published by IETF in “draft-ietf-manet-olsrv2” [12].

The research of interesting improvements was oriented both in improving in complexity time of algorithms and in added features to provide a better routing service.

3 - – Exploratory testing for the study of PICA

In order to study PICA, I worked on its code and the two articles published by GRC [1].

Due to lack of documentation and my knowing nothing about PICA and its code, I had to perform what is called “reverse engineering”. I tried to use its functionality creating new small applications, and this activity is quite like a test, so I decided to look for some testing strategy which might help me.

The best known testing strategies assert that testing activities must be planned early in the development process, and that testing concerns both static and dynamic activities.

Obviously, my work was not done in early phases of development, so I could not do a proper testing plan, but I needed testing to help me to investigate unknown code, so that testing allows me to learn and test it in same time. Therefore, I decided to use a testing strategy called “Exploratory testing” (ET).

3.1 - Used methodology: Exploratory testing

Different definitions are available, but I choose Swebok’s [14] one:

“Exploratory testing is defined as simultaneous learning, test design, and test execution; that is, the tests are not defined in advance in an established test plan, but are dynamically designed, executed, and modified. The effectiveness of exploratory testing relies on the software engineer’s knowledge, which can be derived from various sources: observed product behaviour during testing, familiarity with the application, the platform, the failure process, the type of possible faults and failures, the risk associated with a particular product, and so on.”

Exploratory testing was introduced by Kaner, Falk and Nguyen [13] as a revolutionary approach in opposition to waterfall testing model (this it the most diffuse and used model).

In Waterfall testing model, testing activities are planned early in the development process and executed later. This schedule does not bear in mind three important aspect of reality life:

consumer software products are developed quickly and relatively unstructured ways. Development and testing being before a full specification is complete, there may never be a full specification, and all aspects of the program are subject to change as market requirements change. There is no point releasing a program that can not compete with the features and design of a just-released competitor.

Even if tester has planed test activities already, he/she learns the most about the product and how to make it fail when he/she test it. Therefore, previous testing plan can turn out to be useless and/or ineffective.

Suppose tester receives a complete specification, written at the start of development. (This is when such things are written, under the waterfall method). Tester stars writing test plan in parallel with programming, so that he/she can start testing as soon as coding is finished. Unfortunately, during the next year of implementation the specification change significantly in response to technical problems and new market conditions. Everybody is aware of disaster along these lines: in one case, by the time the programming was complete and before any testing had started, the project entire test budget had been spent revising the test plan.

Due these problems, Kaner, Falk and Nguyen recommend exploratory test adoption that allows tester to design tests as he/she need them.

3.1.1 Exploratory testing general methodology

First of all, it is important to notice that E.T. is not a set of rules that assures a successfully test, but rather a methodology for accomplish ing tests.

By following this procedure it is possible to create a procedure getting like “general functionality and stability test Procedure” for certified for Microsoft Windows Logo.

The general methodology of ET is easy enough to describe. Over a period of time, a tester interacts with a product to fulfil a testing mission, and reports results. There you have the basic external elements of ET: time, tester, product, mission, and reporting. The mission is fulfilled through a continuous cycle of aligning ourselves to the mission, conceiving questions about the product that if answered would also allow us to satisfy our mission, designing tests to answer those questions, and executing tests to get the answers. Often tests do not fully answer the questions, so it is necessary to adjust the tests and keep trying (in other words, explore). We must be ready to report our status and results at any time.

Therefore. it is possible to say that testing documentation is done “on the fly”: since test activities are not planed, even their documentation can not be defined a priori.

In this chapter, I report Kaner, Falk and Nguyen’s approach and explain how I fit it to my work.

3.1.1.1 Initial development of test materials

Kaner, Falk and Nguyen’s approach requires parallel work on testing and on test plan. Never one must get far ahead of the other.

Kaner, Falk, Nguyen’s approach is defined on more levels, starting from the topmost, the most superficial one, and each lower level is a deeper exploration (of some particular area) of the upper one. The first step described by Kaner, Falk and Nguyen starts comparing program’s behaviour and documentation, creating a function list that includes every thing the program’s supposed to be able to do, and including a simple analysis of limitations.

During this step, I analyzed PICA general architecture and studied its interface. After this activity, I had general schema of PICA architecture in which are represented relationships between PICA’s files and provided functionalities.

Next step, in Kaner, Falk, Nguyen’s approach, concerns to choose a part of the program to test more meticulously. There is not a pattern to choose the part; it depends on testers’ instincts and knowledge, or on what portion and functionality seem more important, critical, complex, poorly documented….

It is important to point out that this in depth-exploration is not done on a stand alone program component, but concentrates on some aspects considered more relevant by testers.

These aspects are probably in one of the six areas listed here:

1) most likely error

2) most visible errors

3) most often used program areas

4) distinguished areas of the program

5) hardest area to fix

6) areas most understood by tester

The mechanism of adding depth to test plan consists in creation and expansion the various test plan components: list decision trees, function list, boundary charts, test metrics and so on.

It is important to point out that this in-depth exploration is not a test of stand alone program's component, but concentrates on some aspects considered more relevant by testers

In this second step, I focused on functions that I understood better.

Obviously, the approach, I was following, is general for all kinds of applications; therefore it provides charts for general purpose that could not be fit for PICA, like for example list of compatibility printer and software or charts about GUI interface.

Since PICA is a library that provided functionalities, I decided to adopt the "list of features and functions" chart and the matrix for relate function behaviour to operating systems.

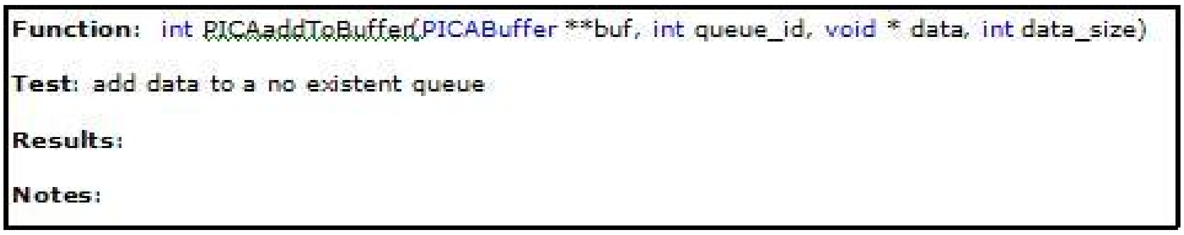
In point of fact, this matrix is an adaptation of one proposed by Kaner, Falk, Nguyen.

They propose hardware and feature compatibility matrix.

Since PICA was developed in order to operate on different platforms, it was not necessary to verify PICA compatibility with platforms, but rather, to verify whether for each platform the same function acts in the same way.

Therefore I adapted hardware and feature compatibility matrix to my needs.

The used chart for list function and functionalities consisted in a sample list of function in first time, and after more detailed study of each function I built charts as figure 3.1 shows:



Function: int PICAaddToBuffer(PICABuffer **buf, int queue_id, void * data, int data_size)

Test: add data to a no existent queue

Results:

Notes:

Figure 3-1: chart used to write down test results

In “test” section I wrote what test consists of. In case the test concerns of function behaviour test on different input, this section can be represented by a table this:

	Input 1	Input 2	……	Input n	behaviour / result
Test 1					
Test 2					

Table 1: used chart for test

In note field I usually wrote other possible tests to understand better the function. If an error occurred during test execution, I noted down it and its solution, or which other test I need to discovered its solution.

By using these charts, I could build the matrix chart in order to compare function behaviour, where in the first column I reported test activity and in the others the results or behaviour of tested function.

Test	Windows XP	WinCE 3.0	WinCE 5.0	Linux
test 1				

It is important to notice that I used test activity also to learn to use necessary tools for developing applications on different platform.

3.2 - used tools

During the initial phase of my work, I learned the necessary tools in order to operate with PICA on the different platforms, and how to develop applications on pocket pc.

The following table summarizes all tools I needed.

	Windws xp	WINCE 3.0	WINCE 5.0	Linux
Tools	Microsoft Visual Studio .Net Version:2003 and 2005	ActiveSync 4.1 EmbeddedVisul c++ 3.0	ActiveSync Visual studio .net 2005 Professional	Text editor

When I began studying PICA applying ET practices, I found out some malfunctioning or necessary improvements and therefore I updated

PICA. Hence I did also a more conventional testing phase, that is unit test and integration test on updated features.

Also, I did installation tests. Usually, installation test is underestimated, but it reveals if it is user-friendly enough to makes functionality provided by application under testing trivial. PICA, in addition to provide useful functionalities, is very simple to install and it is bound to only an other open source library only.

The next chapter explains results of my exploration get from previous explained steps.

4 - – PICA library

In order to study PICA I applied ET practises; I divided my work in four principal phases: first of all, I examined PICA architecture and I found out its principal features. The second phase concerned analysis of PICA's functions, while in the third one, I focused on testing some functions with different inputs and then studying their behaviour. In this step I pointed out that it was useless to test some PICA function alone, but it was more interesting studying them in an integration test. Therefore in the fourth step I created applications that let me do an integration test.

4.1 - PICA architecture

In this first phase, by studying articles published by GRC and PICA code I found out PICA architecture and PICA relationships with other libraries of different operating systems.

Moreover, I explored PICA to determine its internal structure.

4.1.1 Overall architecture

The PICA library is an adaptation layer between the programmers and the kernel space as shown in figure 4.1.

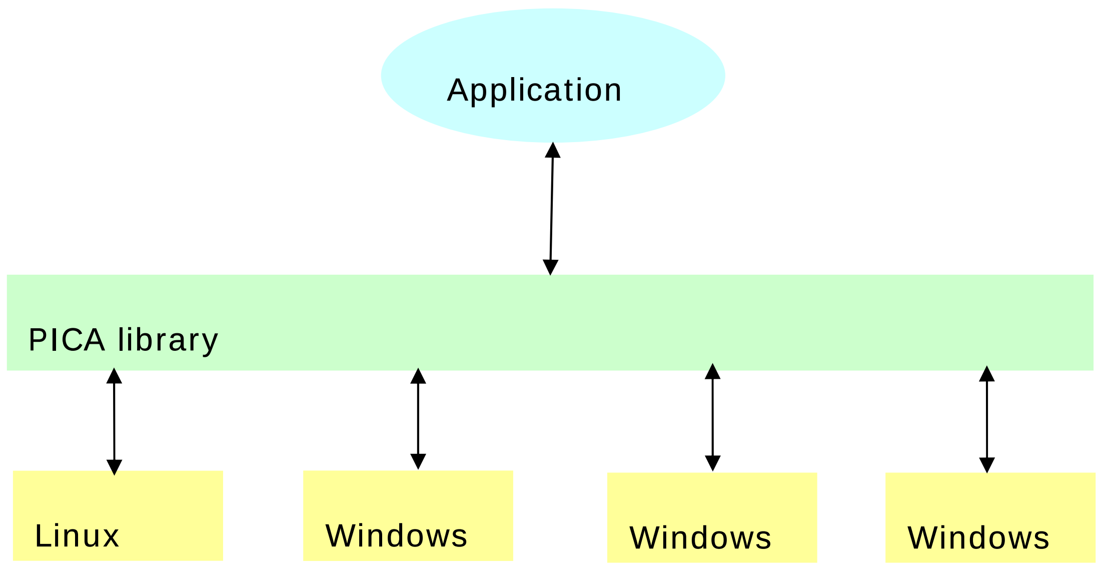


Figure 4-1:PICA overall architecture

Therefore, PICA makes possible the development of network-based applications using the same code for all platforms. Hence, PICA hides differences between operating systems, but without restricting their capabilities.

These differences consist in use of different resource representations, different syntax and sometimes even different semantic of functionalities.

For instance, PICA offers an integrated treatment of sockets and handles under Windows and Windows CE operating systems, simplifying the programmer’s tasks when both require a simultaneous processing.

The functionality of the different primitives available under each operating system is extended so as to achieve a similar level of complexity on both platforms. This requires augmenting the possibilities of the platform offering more limited functionality, in order to make it resemble the solution for the more complex platform.

PICA offers specialized functions to creating networking solutions: one of them can be a sniffer, that is an application that controls network traffic of low levels.

Therefore, PICA provides functionalities to send and receive data, operating as network adapter directly.

In order to accomplish this task, PICA must face up with a problem using Libpcap [15] library and its porting Winpcap and Packet32 [16] on Windows XP and windows CE respectively.

Libpcap and its porting consist of drivers, which extend the operating system to provide low-level network access, and a library that is used to easily access the low-level network layers.

Libpcap and its porting allow applications to capture and transmit network packets bypassing the protocol stack, and have additional useful features, including kernel-level packet filtering, a network statistics engine and support for remote packet capture.

PICA uses Libpcap functionalities concerning sending and receiving packets. If a developer needs to use also the other ones, he can call them directly in his code without losing portability.

However, it is important to notice that PICA, as shown in following figure, does not cover completely the kernel in order to leave developer free to choose if he wishes to use PICA or kernel functions. Obviously, by using kernel specific functions, the source code will loose its compatibility between platforms, requiring porting and extra effort on the developer side.

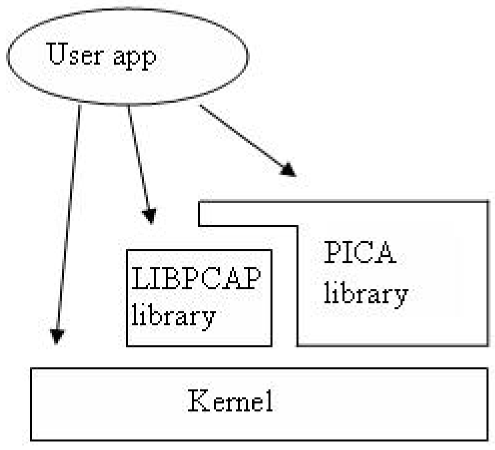


Figure 4-2: designer API choises

4.1.2 windows implementation of Libpcap: Winpcap

WinPcap is the industry-standard tool for link-layer network access in Windows environments: it allows applications to capture and transmit network packets bypassing the protocol stack, and has additional useful features, including kernel-level packet filtering, a network statistics engine and support for remote packet capture.

WinPcap consists of a driver that extends the operating system to provide low-level network access, and a library that is used to easily access the low-level network layers.

Winpcap structures result from Berkley Packet Filter (BPF) [17] developed by California University. BPF is the capture system used in BSD operating systems' family.

As shown in Figure 4-3: Winpcap and NPFFigure 4-3, Winpcap is composed by three modules.

The first is NetGroupPacketFilter (NPF), a device driver used in kernel layer and its principal role is capturing packets.

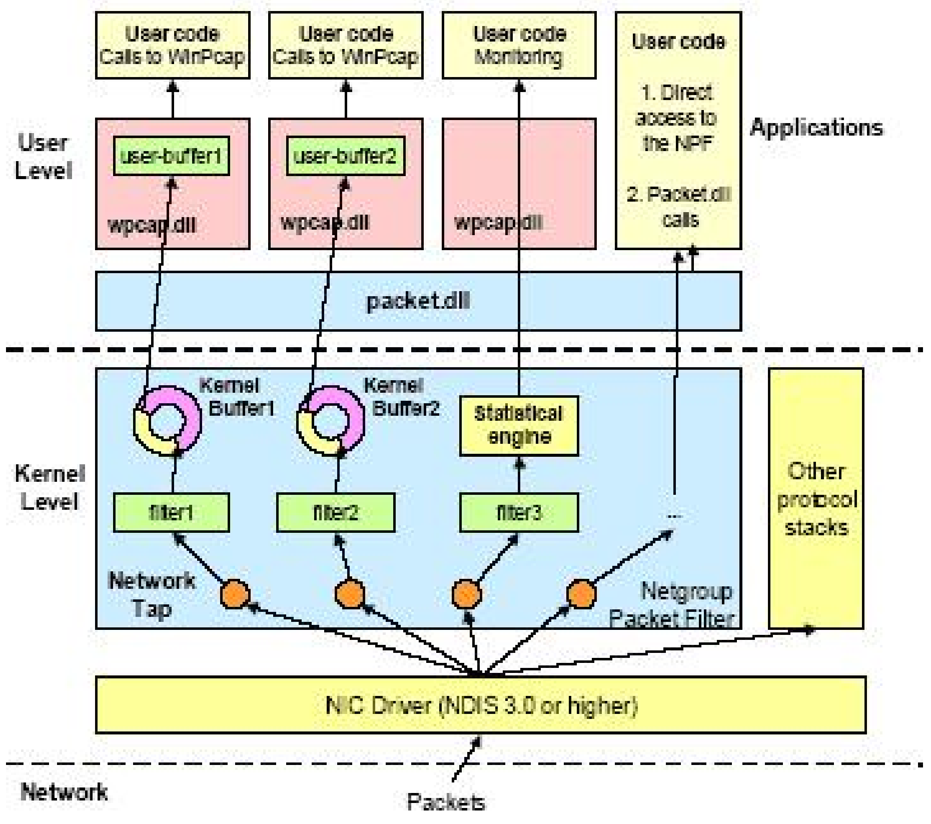


Figure 4-3: Winpcap and NPF

The second module is a dynamic library named packet.dll. it hides differences between different platforms providing the same interface to applications.

The last module is another dynamic library, wpcap.dll, that exports high level functions for capturing packets and analysing network traffic.

4.1.3 PICA internal structure

This paragraph shows PICA project internal structure for each platform on which PICA is available.

PICA project is constituted from eleven files of code and each of them includes a header file.

The Figure 4-4 shows the general internal structure of PICA project and the relationships between header files and used library.

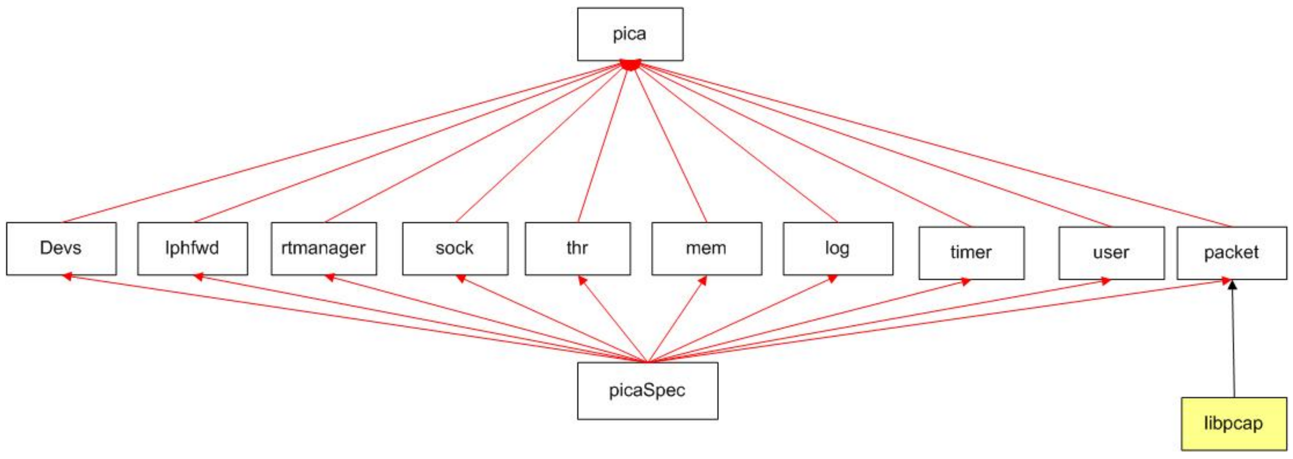
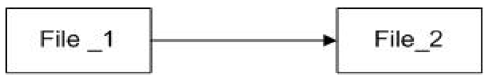


Figure 4-4: general internal structure of PICA

In this figure a white rectangle is a single PICA header file, while the yellow rectangle is used for libpcap porting header file.

Arrows are used as follows: for example when we draw



In this case File_2.h includes File_1.h with the directive #include“File_1.h”.

PICA Linux version internal structure is the same shown above, but without references to Libpcap.

Some functions provided by PICA need special libraries in order to interact with operating system and drivers; hence these libraries depend on operating system.

4.1.3.1 windows-based operating systems

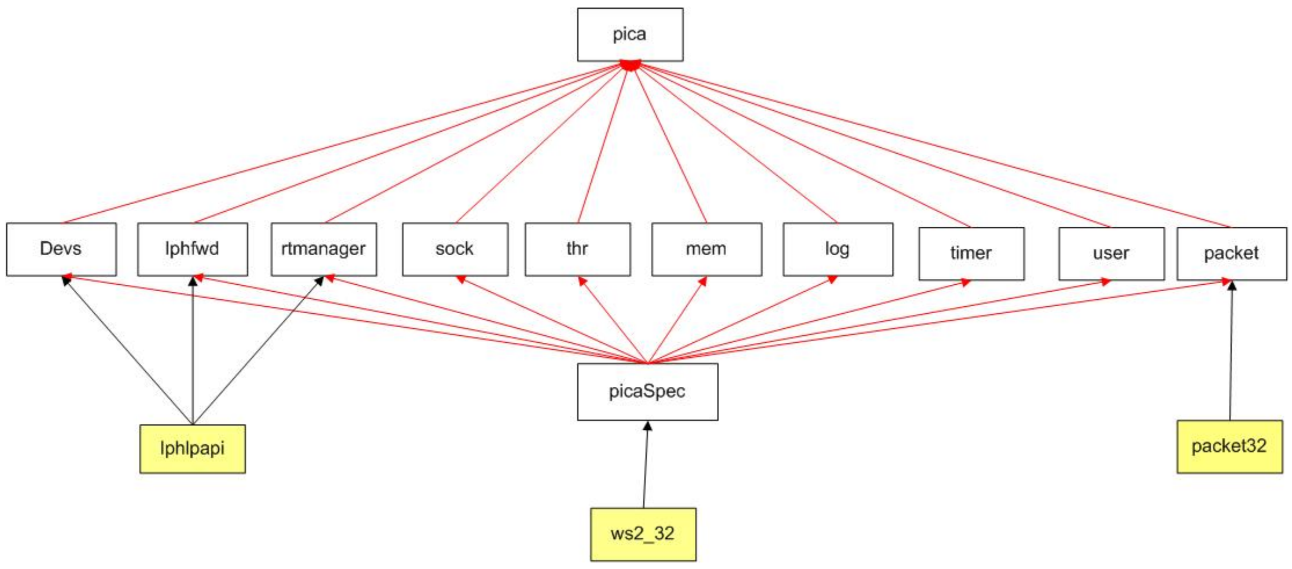


Figure 4-5: PICA internal structure for Windows operating systems

The Figure 4-5shows relevant libraries referred by PICA library for all operating systems based on windows.

The only difference between Windows XP or 2000 and Windows CE is the respectively porting of Libpcap: Winpcap for first type and Packet32 for the second.

Figure 4.5 reports only libraries that are not used ordinally.

Internet Protocol Helper API (Iphlpapi) [18] assists network administration of the local computer by enabling applications to retrieve information about the network configuration of the local computer, and to modify that configuration. IP Helper also provides notification mechanisms to ensure that an application is notified when certain aspects of the local computer network configuration change.

This library is used by PICA in order to get information about available network interfaces, to manage “time to live” (TTL) and forwarding settings, to send and to receive packets.

Ws2_32 Windows socket library (version 2) [19]**¡Error! No se encuentra el origen de la referencia.** uses the sockets paradigm that was first popularized by Berkeley Software Distribution (BSD) UNIX.

It was later adapted for Windows in Windows Sockets 1.1, with which Windows Sockets 2 applications are backward compatible.

4.1.3.2 Linux

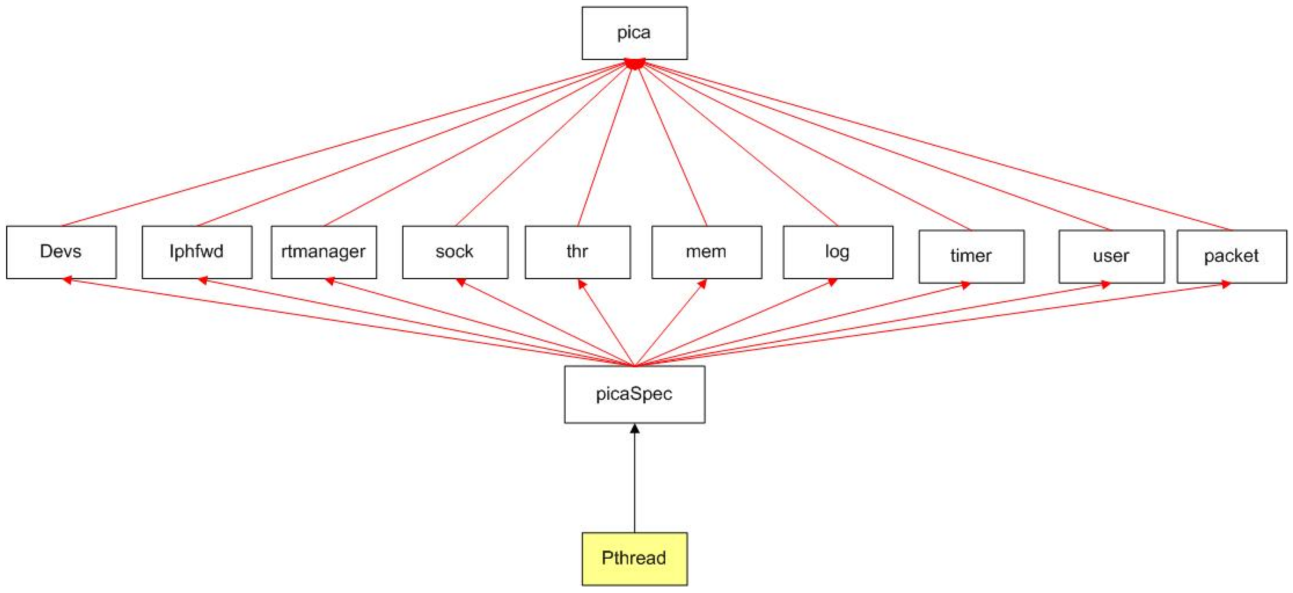


Figure 4-6:PICA internal structure for Linux

PICA Linux version's based on Pthread library.

Historically, hardware vendors have implemented their own proprietary versions of threads. These implementations differed substantially from each other making it difficult for programmers to develop portable threaded applications. In order to take full advantage of the capabilities provided by threads, a standardized programming interface was required. For UNIX systems, this interface has been specified by the IEEE POSIX 1003.1c standard (1995). Implementations which adhere to this standard are referred to as POSIX threads, or Pthreads. Pthreads are defined as a set of C language programming types and procedure calls, implemented with a pthread.h header/include file and a thread library - though this library may be part of another library, such as libc.

It is fundamental to notice that PICA Linux version does not need libpcap. In the first time, PICA was designed based on lipcap in order to use libpcap functionalities to interact with network adapter, but, subsequently, PICA designers opted for using socket feature in place of libpcap.

Sine socket Linux allows a direct interaction with network adapters, PICA designers prefer to use it, even if it is a little bit slower, instead of binding PICA to another library, augmenting code size and make it dependent on other code and its new released versions.

4.2 - PICA primitives

It is possible divide PICA functionalities in three logical groups basing on the principal task of network-based applications: process management, memory management and communication management.

It is important to notice that the differences between operating systems are not only in different systems calls, but also in different way to identify system resources. In Linux any resource is identified by an integer number, while windows uses integers only for sockets, while for other resources it uses HANDLE, a generic data type. To avoid this problem PICA uses its own wrapper types in order to hide these differences.

This chapter explains salient PICA functions characteristic, but for a detailed description see User manual and Programmer manual at appendix x.x and x.x respectively.

4.2.1 Process management primitives

The fork() call is of common use in Unix environments to manage processes. Windows systems, though, do not offer this function. We therefore adopted a combination of the threads approximation with the semaphore and mutex abstractions as an alternative to processes without generating too much extra code. Although the Posix standard [20] does not allow thread suspension and resuming, the PICA library allows the use of such functions in the Linux operating system by means of the SIGUSR1 and SIGUSR2 signals.

The select function is a well-known UNIX system call used to wait for events associated with any kind of resource. Instead, in Windows the select function is only available for sockets, while for other events we have to use a function of the WaitFor family. The PICA library obviates this problem by emulating the Linux behaviour. When invoking the PICAselect() function the current thread stops waiting for events using a WaitForMultipleObjects() call and a secondary thread is created. This thread is dedicated uniquely to socket handling. When an event occurs on a socket, the secondary thread informs the waiting thread, and through a system pipe it sends the information about which socket is active (see Figure 4-7).

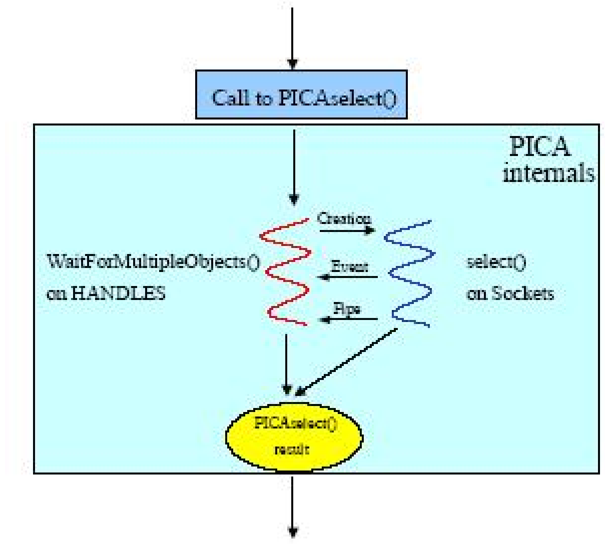


Figure 4-7:Architecture of PICAselect() function in windows

Handling timers is also of extreme relevance. Protocols and applications are often required to perform scheduled actions, e.g., to take into account network congestion status. Differently from Windows, the Linux architecture imposes a single timer per application. This “restriction” encouraged us to provide PICA with multiple timers, by means of a priority queue. In this queue all events make use of the same timer in a way that only the first to-happen event affected will be used to set the value of the timer. There is a thread that executes the code indicated by the user, allowing the event handling to be not only asynchronous, but also parallel. The solution provided allows high versatility due to the possibility of multiple call-back functions, each accepting an argument. This allows for a different approach to events, enabling the designer to group events into families, achieving more elegant solutions to common problems.

4.2.2 Memory management primitives

Relatively to memory management, the PICA architecture is based on offering the possibility to easily handle a queue data structure. The objective was to provide auxiliary functions which could be useful at implementation time. The programmer can create as many queues as desired, allowing different packet handling. The architecture chosen allows the use of multiple groups of queues, each group having a number of queues chosen by the user. In order to guarantee data coherence to multi-threaded applications, PICA uses a different mutex status variable to control the access to each queue so that, for example, distinct threads can read and write to different queues, even though these queues belong to the same group. It is relevant to point out that in order not to generate very long delays, these routines do not perform any kind of buffer duplication, having the sole task of managing pointers to the data.

Moreover, the pipe is available in PICA as a further mechanism for inter-process communication.

A UNIX pipe is a pair of file descriptors such that what's written on one shows up as available data on the other one. For its implementation in Windows, there is another problem: as well as differences between descriptor types, PICA faces up to different management of synchronization between in and out pipe: while in UNIX it is transparent, in windows programmers must use *waitforSingle* object to synchronize pipes. PICA hides this difference including in its code synchronization calls.

In network-based applications, like network daemons, log features are widely used; therefore, they are added to PICA.

The PICA logging functions offer straightforward methods to read from and write into files. The new file type created has the unique purpose of unifying the different data types, i.e., HANDLE in Windows and int in Linux, used as file descriptors.

4.2.3 Communication management primitives

The basic data structure abstraction is the “data packet”. PICA offers a small set of functions to create, send and receive (capture) packets.

The approaches taken in Linux and Windows are quite different.

PICA Windows version inherited the functions to send and receive packets offered by the PACKET.DLL library which is part of the winpcap distribution (the porting of Libpcap to windows).

On the other hand, Linux does not employ Libpcap library, because it provides direct read and write access to a network interface (NIC) by using the PF PACKET socket domain.

Even if the PF PACKET socket domain does not allow bypassing network stack, it is quick enough and avoids to bind PICA with Libpcap library.

The socket interface is also one of the most frequently used interfaces when developing protocols implementations.

PICA encapsulates this API using macros and eliminates the apparently slight differences between the various platforms. It should be noticed, though, that the Windows Sockets Version 2 (WinSock 2) API [19] does not provide useful functions such as the recvmesg and sendmsg whereas in Linux they are available.

WinSock 2 also lacks of some of the options that are available on Linux based implementations.

These factors should be taken into account by programmers, because they can compromise the compatibility between platforms.

Furthermore, PICA provides functions to handle the IP routing functionalities.

The PICA routing functions allow the designer to add and remove entries in the forwarding table.

Even if these functions are not frequently used, they are sometimes useful in protocol design to provide dynamic connectivity. See, for example, the AODV implementation [21]. Moreover some protocols need to check some of the entries, like the broadcast ones, and this can be easily accomplished using these functions.

PICA gives the possibility to get access to the current forwarding status and to the time-to-live (TTL) attributes. It should be noticed that the TTL value affected is the global system’s TTL. Lowering this value too much might cause loss of connectivity to other networks (e.g., Internet). For a per-socket definition of this value, the socket options available in most systems should be used.

Finally PICA provides functionalities to manage itself in order to initialize and remove all global variables used during its execution.

Some functions offered by PICA need to be executed in administrator mode; so PICA gives a function to test execution mode in order to notify to the user if those functions can operate correctly.

4.3 - - test PICA’s functions analyzing their inputs and limits

In this phase, I studied PICA functions’ behaviour testing them.

There are several ways to test a function; the choice of one depends on function itself.

Some PICA functions provide stand-alone functionalities and their behaviour depend on their parameters strictly. Their behaviour can be studied testing them on different inputs.

Other PICA functions encapsulate operating system ones simply; hence, I limited me to study them and verified that their behaviour is the same on each platform, because testing them it would have had to test operating system functionalities and it is not my objective.

Finally, in PICA there are functions whose behaviour can be analyzed in a certain context. Therefore, I studied this kind of functions in small applications and in OLSR implementation.

In this paragraph I explain the first PICA function type, that is, functions in mathematical meaning. I studied them analyzing their behaviour on different input.

Chosen input values were both belonging and not belonging to function domain in order to control if function can generate a dysfunctional behaviour.

Obviously, I did the same tests for all platforms.

Here I do not reports all performed tests, but I explicate the substantial problems I had in this phase and their solution.

During this activity, I used the chart explained in section 3.1.1.1 and that is reported in figure 3.1.

Reporting test results helped me to solve encountered problems during this activity.

The found problems were of various kinds; to determine their solution was a hard task because it was difficult determine where is the problem. The possibilities could be:

- I call PICA function in a wrong way in my application test

- PICA calls winpicap or o.s. procedures in a wrong way

- there is a problem inside PICA code

- winpcap or s.o system function doesn't work as specified in documentation

- there is a problem in interaction between hardware, driver and software.

The charts help me to record with solutions I have just tried and how they have solved the problem (only in part or completely); also, if the problem was in an over version of PICA, I can try to solve the problem with the same solution.

In addition, at implementation level, in order to face up these problems, I used debug tool if it was available or add code to PICA in order to put in output the variables value and to follow execution program flow.

The widely problem I found concerns of different behaviour of same function on various platforms.

To found out this, the matrix chart, explained in chapter x.x, wsa very useful, because let me compare result on each test performed on all platform.

This kind of problem could depend on different operating system behaviour in use of c-language base primitives, or an error in PICA code, or it is triggered by differences among used devices.

An example about of first kind of problem is this: I discovered that the same call of well-know function “malloc” on different platforms gives different results, because they have different ways of memory management.

I point out this problem when I tried to destroy PICAbuffer: windows xp throws an exception when I try to access a memory location allocated by PICA library.

The memory was allocated with malloc by PICA library when it created the buffer, but the application cannot to free that memory.

Therefore, I discovered that the problem concerns of a different operating system memory management.

So, by replacing malloc call with GlobalAlloc call I resolved the problem.

In another case, the different function behaviour depends on different interpretation of system function. I detected this problem in PICAwriteFile in append mode: while, in Linux, it is sufficient a call of write function in append mode to add text in the end of file, in windows, the programmer must move file pointer to the place where he wants to add text.

PICAwriteFile windows version wrote data always at beginning of file overwriting existent data. In this case, I add code in order to put file pointer at the ending of file before doing a write.

Hence with this change PICAwriteFile has the same behaviour in all platform.s

Function whose behaviour depends on platform is PICAreadFile. Among its required parameters there are a pointer to char and an integer number: the first is the pointer to data read and the seconds represents byte number of data to read.

In Windows it is necessary to allocate memory specifying the number of byte to read, while in Linux it is not necessary.

I tested this function by calling it with a number of byte bigger than size of allocated memory.

In Linux there was no problem, while in windows functions' behaviour is unpredictable, but the most widely retuned error concerns corruption of memory area near the pointer passed as parameter.

Therefore, I could choose one of these possibilities:

1) allocate necessary memory in the PICA functions; however in Widows XP I will have problem with memory management: I would have to allocate memory with GlobalAlloc function and leave effort to free that memory to user.

2) check in PICA function if allocated memory size is big enough to contain the number of bytes indicated as parameter. To perform this check I would have to use sizeof c-language function that computes array size; but this function requires array parameter is allocated statically. In this way I would have to force user to allocate memory statically and this is a too strong constrain.

Hence, I could not fix this problem and I limited me to notice it on user manual.

In other cases the different behaviour is triggered by differences among used devices: for example on PDA the functions used to get information on available network interface functions correctly only if network interface is on.

Hence in this case, I reported this constrain on user manual.

Moreover, PICA functions, concerned of setting TTL and forwarding proprieties, are available only after the system register "*HKEY_LOCAL_MACHINE\Comm\Tcpip\Parms\IpEnableRouter*" is been set.

Even in this case, I described how to set this register, without changing PICA code.

Since PICA was implemented to help network-based application, among them there are network traffic analyzers. This kind of applications needs to set network interface card (NIC) in promiscuous mode.

In an Ethernet local area network (LAN) and therefore WLAN also, promiscuous mode is a mode of operation in which every data packet transmitted can be received and read by a network adapter.

Therefore it would be interesting if PICA would let to open an adapter with promiscuous mode set by default.

But it was impossible, because some windows drivers do not allow setting that mode. Therefore, in order to make PICA the most flexible as possible, I added a parameter to specify in which mode to open the adapter.

PICA uses standard and elementary library, but I discovered that the famous strcpy function is not implemented on win ce and thefore I added it to PICA.

Even the IP routing functionalities depend on operating system. PICA provides procedures to add and remove entries in routing table: in Windows systems it is possible to add any address even if it does not belong to space network address, while Linux does not allow it. This is an intrinsic feature of operating system and adding a wrong address is a logical error. Therefore PICA windows version does not prevent it.

4.4 - 4.4 – Integration test

Some important PICA features are testable only with an integration test, that is they can be studied only in a certain context.

For example, PICA provides mutex and semaphore: it is more interesting to analyze their behaviour in a context switch than to test them on various inputs.

Therefore, I created small applications that let me study these features.

The same code was executed on all platforms obtaining the same performance and verifying that on all operating systems they are blocking: if the shared resource is occupied the thread that required it enters in a wait-state.

IP routing functionalities were analyzed by a test-application. This application is divided into sender code and receiver code. Each of this was executed on different devices: the first sends packet with Ethernet format to the second.

The choice of Ethernet version II format [24] was obligatory: the driver of wireless network interface interprets packets only with that format.

As figure 4.8 shows, Ethernet packet format is composed by:

- Preamble (7 bytes): This is a stream of bits used to allow the transmitter and receiver to synchronize their communication. The preamble is an alternating pattern of 56 binary ones and zeroes.

- Start Frame Delimiter [SFD] (1 byte): This is always 10101011 and is used to indicate the beginning of the frame information.

- Destination MAC address (6 bytes): This is the MAC address of the machine receiving data

- Source MAC address (6 bytes): This is the MAC address of the machine transmitting data

- Ethernet type/length (2 byte): if this field has value between 0 and 1500 then it indicates the use of the original Ethernet format with a length field, while values of 1536 decimal (0600 hexadecimal) and greater indicates the use of the new frame format with an EtherType sub-protocol identifier.

- Payload(46-1500 bytes) : The data is inserted here. This is where the IP header and data is placed if you are running IP over Ethernet.

- FSC: This field contains the Frame Check Sequence (FCS) which is calculated using a Cyclic Redundancy Check (CRC). The FCS allows Ethernet to detect errors in the Ethernet frame and reject the frame if it appears damaged.

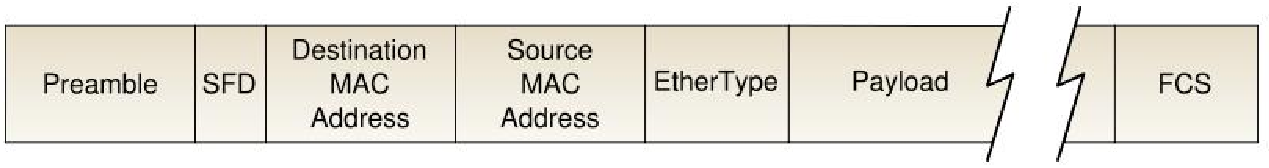


Figure 4-8: Ethernet II packet format

The first two and the last field are managed by Ethernet adapter only: they are removed before being passed on to the network protocol stack software.

A way to create an Ethernet packet in c-language is creating a data stream composed by MAC addresses, Ethernet type/length field and payload. Data size must be between 60 bytes and 1514. If data size is lesser then 46 byte it is obligatory to add padding data (usually all zero bits.)

Usually data stream is defined as unsigned char array. Each of its element has size of one bite and hexadecimal codification is used to set the right binary string because it is simpler to human user.

The first 12 array elements are reserved for MAC addresses, the next two code Ethernet type or length, while remaining ones contain payload.

PICA left packet creation to programmer. In many situations packet is created at run-time; this makes it impossible to write MAC address in dot-notation in the code. Therefore, now PICA provides a function that creates a Ethernet packet.

PICAselect and PICAtimer are other functions whose behaviour should be analyzed in a specific context. First function is used to wait for events associated with any kind of resource, while the second to managed one or more timer.

In order to examine PICAselect I created a test-application that calls PICAselect on each kind of resource and after it creates the respective event, while for the second I developed a small application that sets timers at different time with different function and I observed its behaviour.

Since PICA was created in order to make easier network-based application development, the best way to analyze and describe PICA is to insert it in a real application and to study how its functions work.

The chose real context is the INRIA implementation of OLSR [23]79

In this implementation systems calls were replaced by PICA ones; so PICA based OLSR implementation is offered for all platforms on which PICA is available.

This study is described in more detail in chapter 5 - .

5 - OLSRv1 with PICA

This chapter shows a concrete application of PICA and explains how PICA made easier OLSR implementation for Linux, Windows XP, Window CE 3.0 and Windows CE 5.0.

Since OLSR uses socket feature to access to the network, the used PICA library is the version without bind to Winpcap.

5.1 - *Used tools*

As *PICA user manual* (appendix x.x) explains, in order to build a PICA-based application it is necessary use the following tools:

	Win2000 - XP	WINCE 3.0	WINCE 5.0	Linux
Library	Winpcap	Packet32	-	-
Tools	Microsoft Visual Studio .Net 2003	ActiveSync 3.5 EmbeddedVisual c++ 3.0	ActiveSync 4.2 Visual Studio .Net 2005	

As matter of fact, I decided to use Microsoft visual studio 2005 in order to develop OLSR for windows operating systems because this tool let me operate with the three windows operating system, without using an extra tool.

Obviously, I needed Active Sync 4.1 to connect pocket pc with computer on which I was developing the application.

In Linux I used a simple text editor and make utility to compile the application in a fast way.

5.2 - OLSR project preview

This paragraph explains the principal execution flow of OLSR process.

First of all, the main process analyzes the possible input options; after, it enters in an initialization phase: in this phase it gets all information about the interface on which OLSR runs; if a filter is defined by a input option, it loads it also. Since OLSR uses many data structure in witch store all network information, the main process provides to initialize them; in addition it deletes all entries stored in kernel routing table by last execution in order to avoid mistake with the new one.

After this, the real OLSR daemon starts: it consists in an infinite loop that checks if one or more packets arrive on selected interface and if the interval time in which send HELLO message and TC message is elapsed.

After this, the real OLSR daemon starts: it consists in an infinite loop that performs two checks:

- verify if one or more packets are arrived on selected interface
- verify if HELLO-message and TC-message intervals time are elapsed.

In the first case, the arrived packet is processed fallowing draft instructions, while in the second HELLO and/or TC are created and are sent.

5.3 - PICA in OLSR

In this thesis is the OLSR implementation (for Linux only) referred to “draft-ietf-manet-OLSR-03.txt” [23] was used.

OLSR project is composed by twenty source code files. Many of them are dedicated to implement and manage all data structures to store information about messages and network information.

Other files are used to accomplish tasks of OLSR protocols.

In order to make OLSR available for all platforms, it was not necessary change all code, but only parts that use specific Linux primitives.

These parts are related to the following features:

- socket
- select function
- log file
- add and removing entry in routing table
- get information about available network interface.

The main task of OLSR is listened the media using socket feature and process each received packet.

By replacing Linux function for socket creation with PICAsocket function it is possible use the same code on any platform.

Moreover OLSR requires no-blocking socket and in order to set this feature on socket each operating system uses a own way. PICA allows to set socket operation mode at the moment its creation, by setting a parameter in PICAsocket function.

The original implementation uses Linux function “signal”: by using this function the application checks if timeouts for send HELLO or TC messages is expired or if the interface bind with socket has received a packet.

The PICA OLSR version uses PICAselect to accomplish this task. After creating a PICAsocket, its descriptor is added to PICAdescriptorList and then PICAselect is invocated passing as parameters the socket descriptor just created and setting the timeout for HELLO and TC messages. This timeout is calculated concurring with draft indications.

After PICAselection call, the application enters in “wait-state” until the socket receives a packet or timeout expires.

PICAselect is inside the forever loop of the main application; therefore when one of two event happened, PICAselect is re-call and put the application in a wait-state an other time.

As many important network-based applications, OLSR allows user to use logging features.

This task is realized writing on a file various information, like, for example, the received packets.

In order to write on a file, OLSR Linux implementation uses system primitives (open file, write something and after close it). By a simple replacing these primitives with PICA ones it is possible port the same code on any platform.

Since OLSR is a routing protocol, it needs to operate with system routing table by adding or removing entries. Therefore, the original implementation uses ioctl function to accomplish this task. This is a Linux primitives and therefore makes OLSR code un-portable on windows platform. By replacing that original code with PICAaddRoute and PICAdelRoute copes with this difference.

PICA allows these operations in transparent way: OLSR application has to calculate only necessary parameters like, destination address, metric, gateway address and PICA add or remove the entry independent platform way.

In the initialize phase, OLSR gets all information about interface on witch the protocol runs. The code necessary to accomplish this task depends on used operating system. Therefore, it is possible to cope this difference by replacing that call with PICA functions.

The original code uses ioctl function to get this information, while in windows it is not available. Hence that code is was changed in this way: first of all PICAgetAvailableDevices is call; after between this device the application search the interface required by user; when it founds the right one, calls PICAgetDeviceAttrs on it, so obtaining necessary information to go on.

Using PICA in OLSR code, other than, makes it available for all platform, makes it shorter and more understandable, because, for example, in order to add an entry in routing table, the original code needs more than one instruction, while using PICA it is necessary only a call to PICAaddroute.

6 - OLSR improvement

This chapter explains OLSR version 2 salient characteristics and differences and improvements compared to version 1.

Since OLSRv1 is already available for all platform, it was interesting find news in version 2 try to adapt them in OLSR version with PICA, so obtaining a good version of OLSR available for all platform.

This study was referred to “draft-ietf-manet-olsrv2-03” [12].

6.1 - Presenting OLSRv2.

Mobile Ad-hoc Networks Working Group of the IETF community published “draft-ietf-manet-olsrv2-03.txt” in February 2007. This is the third internet-draft of OLSR protocols version 2.

OLSRv2 retains the same basic mechanism and algorithms: it does not provided optimizations on algorithms and data structures both in time and space complexity. The differences presented by this version concerns in adding features and suggestion of standard to allow cooperation between MANET routers.

In order to provide a more flexible signalling framework, OLSRv2 adopts and suggests a general purpose multi-message packet format described in: draft-ietf-manet-packetbb-03 [24].

This document specifies only the syntax of this new packet format using regular expressions to facilitate generic, protocol independent, parsing logic.

The specification has been explicitly designed with extensibility property in mind: packets and messages defined by a protocol using this specification are extensible through defining new message types and new attributes and fields. Attributes and fields are represented in a generic way, so the parser must not understand the attribute. Full backward compatibility can be maintained.

Also, OLSRv2 suggest to use Data Link layer information to determinate the link quality between neighbourhood and therefore determinate if a neighbour is on a symmetric link or asymmetric or if the link is down.

Moreover, OLSRv2 provides a mechanism to manage and to disseminate information about other networks reachable by one or more nodes that act like gateways.

If a node of a ad-hoc network is a gateway for an other one, it can disseminates this information to other node in order to allow them to access to the other network or Internet.

Among this improvements, the last one was been chosen to add OLSR version with PICA, because inserting this feature allows nodes to access Internet if there is a node that act like gateway.

6.2 - *Adding feature: gateway and attached network*

In this chapter, gateway's interfaces on other network are called non-OLSRv2 interface, as defined in the draft.

In my work I used Microsoft Visual Studio 2005, because it can create executable file for all Windows systems, while in Linux I used a text editor.

In a MANET, if a node can act like gateway it is useful disseminate this information among other nodes in order to allow them to communicate with other networks.

For example, this feature permits to connect a MANET to Internet: if there is a node that can access to Internet, the other nodes can use it to reach Internet.

The adding feature makes possible to disseminate information about gateways through MANET.

Unlike wired networks, in a MANET existence of one or more gateway is not knows a priori; therefore in MANET it is necessary to discover gateways like as other nodes.

The draft say that is obligatory use TC-message to disseminate information about gateways and that is possible to add them in HELLO-message, also.

I chose disseminating this information by TC-message only: this kind of message are sent to all nodes using mpr mechanism, while adding this information in HELLO-message is only a repetition for the 1-hop-neighbours, because they are send in broadcast only to 1-hop-neighbours and they are not forwarded.

As explicate in the draft, when a node receives a TC-message within information about a gateway, it must store them in an appropriate data structure and update routing table by adding an entry containing network and gateway addresses. So it is possible to forward packets toward other network in the correct way, hence to make a connection between the node and the other network. Obviously, in order to disseminate this information through network, nodes, that are gateways or that have some information about them, must insert data about gateways in its TC-messages.

Since I must add a feature to an existent code (OLSRv1 based on PICA), I planed my work dividing it in small steps in order to verify that each my change did not compromise the entire program.

The steps are:

a) create data structure to store gateways information;

b) modify tc-message to suite gateways information and algorithms for creation and parsing tc-message;

c) modify algorithm for routing table creation.

d) insert a control to purging old information about non-olsrv2 interface.

6.2.1 Create data structure to store gateways information

The draft indicates the following set, called as “Attached network Set” to store information about gateways interface:

The draft indicates to store information about gateways in a set with the following structured:

(AN_net_addr, AN_gw_iface_addr, AN_dist, AN_seq_number, AN_time)

where:

AN_net_addr is the network address of an attached network, which may be reached via the node with the OLSRv2 interface address AN_gw_iface_addr;

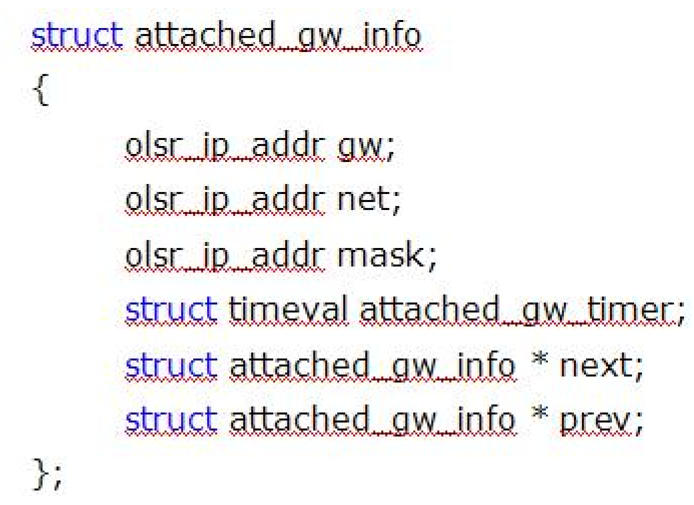
AN_gw_iface_addr is the address of an OLSRv2 interface of a node which can act as gateway to the network with address AN_net_addr;

AN_dist is the number of hops to the network with address AN_net_addr from the node with address AN_gw_iface_addr.

AN_seq_number is the highest received ANSN associated with the information contained in this Attached Network Tuple;

AN_time specifies the time at which this Tuple expires and must be removed.

I implemented this set as figure 7.1 shows:



```
struct attached_gw_info
{
        olsr_ip_addr gw;
        olsr_ip_addr net;
        olsr_ip_addr mask;
        struct timeval attached_gw_timer;
        struct attached_gw_info * next;
        struct attached_gw_info * prev;
};
```

Figure 6-1: definition of struct attached_gw_info

The type "olsr_ip_addr" is defined by OLSRv1 as a integer of 32 bits.

Since a network address id defined as an address and net mask, I inserted the second and the third field instead of a only one corresponded to "AN_net_addr".

I used a double link list to simplify insert and delete operations.

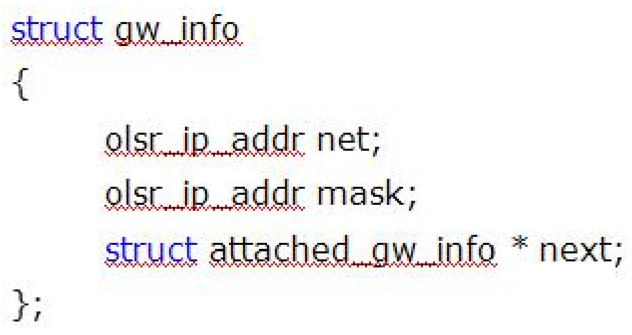
Each node has a global list called "attached_gw_list" to store information about all known gateways: when a received TC-message, has information about one or more gateways, the node adds an element to "attached_gw_list" for each gateway find in the message.

"attached_gw_list" is defined as following:

```
struct attached_gw_info  * attached_gw_list;
```

Obviously, if a node acts like gateway, it is necessary that the gateway-node stores information abouts its non-OLSRv2 interface in a structure, in order to put them in its TC-messages to send.

I defined the following structure to memorize them:



```
struct gw_info
{
        olsr_ip_addr net;
        olsr_ip_addr mask;
        struct attached_gw_info * next;
};
```

Figure 6-2: struct gw_info_definition

This structure is equal to "`struct attached_gw_info`", but without information about expired time.

A node can have more then one non-OLSRv2 interface; it can store information about its non-OLSRv2 interfaces in a global list named "`local_gw_list`" defined as:

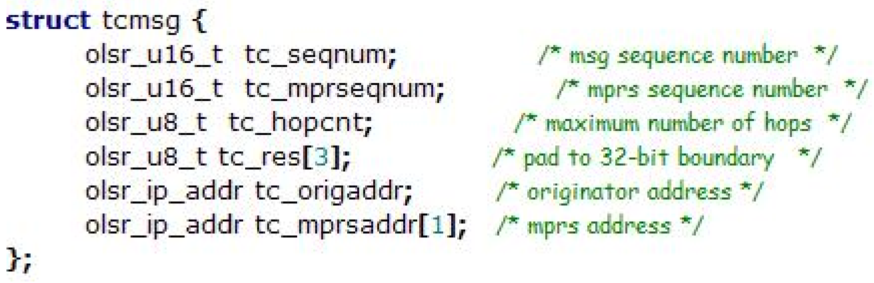
struct gw_info * local_gw_list;

A gateway-node initializes this list at the beginning of execution, during processing input option.

A gateway-node can make aware about its role of gateway, by executing OLSR demon with option "-g file.txt" and by reading networks addresses written on "file.txt".

6.2.2 modify tc-message to suite gateways information and algorithms for creation and parsing tc-message

OLSRv1 uses two types of tc-message: one defined by “struct tcmsg” and other by struct tc_message.



```
struct tcmsg {
        olsr_u16_t  tc_seqnum;              /* msg sequence number  */
        olsr_u16_t  tc_mprseqnum;             /* mprs sequence number  */
        olsr_u8_t  tc_hopcnt;              /* maximum number of hops  */
        olsr_u8_t tc_res[3];              /* pad to 32-bit boundary   */
        olsr_ip_addr tc_origaddr;         /* originator address */
        olsr_ip_addr tc_mprsaddr[1];   /* mprs address */
};
```

Figure 6-3: tcmsg structure definition

The first type represents the real sent TC-message, that is the TC-message will be transformed in a stream of bits by network interface; for this motivation it is important bit alignment of each tcmsg structure field. In fact, the field “tc_res” is the pad of 24 bit for tc_hopcnt field: both together have a word length (32 bits).

A TC-message with tcmsg structure can represented like in Figure 6-3.

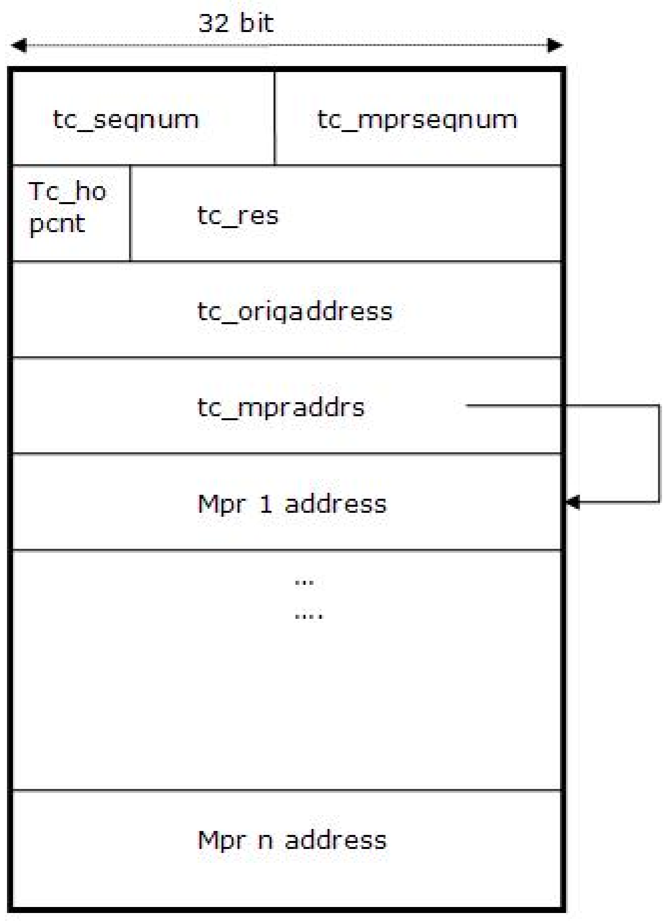


Figure 6-4: representation tcmsg structure

In Figure 6-4 addresses of mprs are represented sequentially after "tc_mpraddrs", but in reality it is not true; their memory location depends by operating system memory management.

I transformed these structure adding two fields:" tc_offset_gw" and "tc_res1" as Figure 6-4 shows. " tc_offset_gw" represents offset from address pointed by "tc_mpraddress" to first information about a gateway, while the second field is padding of " tc_offset_gw" for bits alignments.

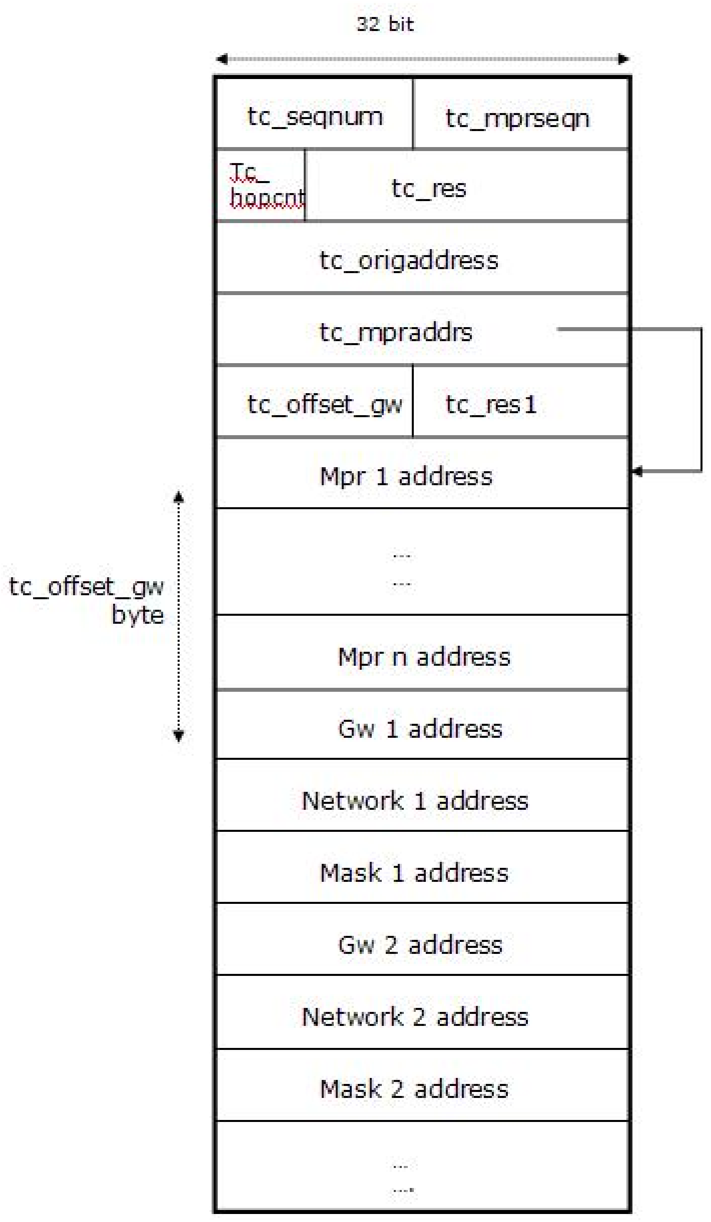


Figure 6-5: representation of new tcmsg structure

Therefore, it is possible identify where gateway information starts by summing “tc_offset gateway” to memory address pointed by “tc_mpraddrs”.

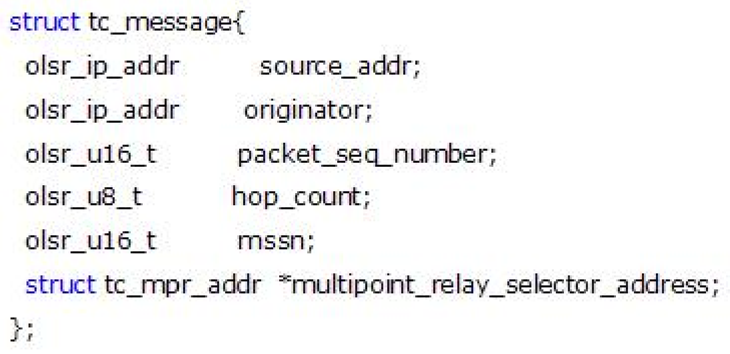
Also, a TC-message can contains only gateway information or only mpr adresses: in the first case “tc_offset_gw” value is zero, while the second it is equal to a defined constant in order to identified this case and to avoid to use un-allocated memory when a node are parsing a TC-message.

It is important to notice that one information about gateway is composed by three addresses: gateway, network and mask.

This fact is important in two situation:

- during creation of a new “tcmsg” to send: since it is possible to fragment information to send in more TC-messages, before adding data, the algorithm checks if there is space enough to insert the data. Therefore, I added a check in this algorithm, in order to check if in tcmsg there is space enough to add one information about gateway, that is, if there is space suitable for three system words: gateway address, network address and net-mask.
- During parsing a received “tcmsg”: in order to identify gateway information in is important to know that the first word represent gateway address, while the second the network address and the third the net-mask.

The other structure represented TC-message is defined as following:

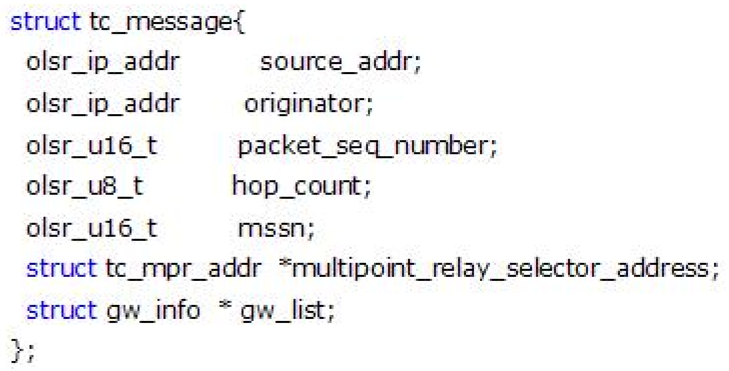


```
struct tc_message{
  olsr_ip_addr        source_addr;
  olsr_ip_addr       originator;
  olsr_u16_t        packet_seq_number;
  olsr_u8_t         hop_count;
  olsr_u16_t        mssn;
  struct tc_mpr_addr  *multipoint_relay_selector_address;
};
```

Figure 6-6: tc_message structure definition

Tc_message is the logical representation of tcmsg strcuture. It is used as intermediate representation. When interface receives a TC-message it will change form tcmsg structure to tc_message structure and after all information contained in the new structure are employed to update tables as described in the draft. When a node must send a TC-message, it creates a logical TC-message and fills it with necessary information. Afterwards, the logical message will be transformed in tcmsg structure and then will be sent by interface.

I transformed these structure as following:



```
struct tc_message{
  olsr_ip_addr        source_addr;
  olsr_ip_addr       originator;
  olsr_u16_t        packet_seq_number;
  olsr_u8_t         hop_count;
  olsr_u16_t        mssn;
  struct tc_mpr_addr  *multipoint_relay_selector_address;
  struct gw_info  * gw_list;
};
```

Figure 6-7: tc_message structure new definition

In this structure I added the ”gw_list” field: this is a pointer to a list of information about all gateways in the MANET.

6.2.3 modify algorithm for routing table creation.

The fundamental idea for this modification is very simple: after calculating routing table, add a new entry for each element of “attached_gw_list” list, if the route from the node to gateway is known. Suppose a node with ip 192.168.0.4 is a gateway to Internet and that node identified by 192.168.0.7 has this information.

Therefore 192.168.0.7 node has the element represented by figure 7.5 in its “attached_gw_list” list.

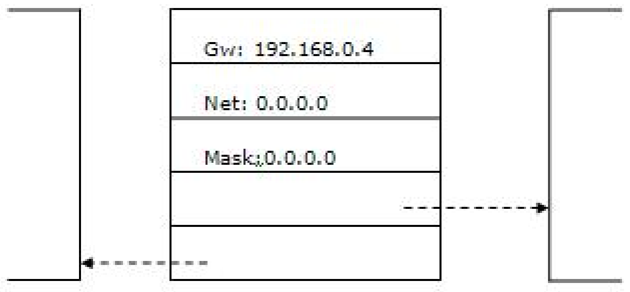
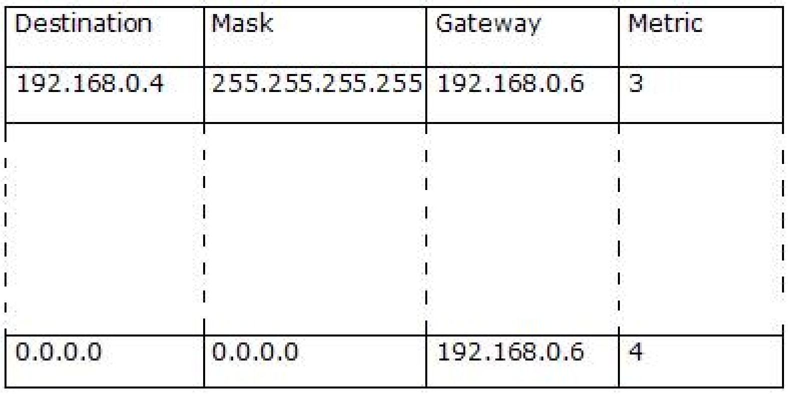


Figure 6-8: representation of an element of "attached_gw_list"

192.168.0.7 node can add an entry about Internet gateway only if in column of “Destination” of its routing table there is an entry with 192.168.0.4; this entry’s existence means that the node knows a route from itself to Internet gateway.

In this case 192.168.0.7 node adds an entry in its routing table like Figure 6-9 shows:



Destination	Mask	Gateway	Metric
192.168.0.4	255.255.255.255	192.168.0.6	3
0.0.0.0	0.0.0.0	192.168.0.6	4

Figure 6-9: representation routing table of node 192.168.0.7

It is important to notice that the address set as gateway of new entry is the address of gateway of entry that has Internet-gateway as destination.

In my algorithm I did not check if the entry exist already, because I used PICA primitive to perform this operation, so OLSR does not return error if the entry already exists.

6.2.4 insert a control to purging old information about non-olsrv2 interface.

Each information stored by OLSR has expiry time: if the information is not refresh before a certain interval time, they became old and then delete.

A node can refresh its information only if it finds such information in a new message.

Obviously, even gateways information has expiry time. Since they are diffused with TC-message, I used the same interval time of the other information contained in the TC-message, because I considered the gateway-node with the same mobility of other node. It is possible change this interval modifying value of constant that defined gateway information interval time.

6.3 - Results

In order to verify new adding feature I performed the following test.

I execute OLSRv2 simultaneously on four computer.

Communication among them is only arrows of Figure 6-10 indicate.

I obtained this using a firewall. For example, in the 192.168.0.7 node I set two security rules in order to block all traffic from nodes 192.168.0.5 and 192.168.0.4. In this rules I specified the correspondent MAC addresses in order to accept only packet coming from 192.168.0.6.

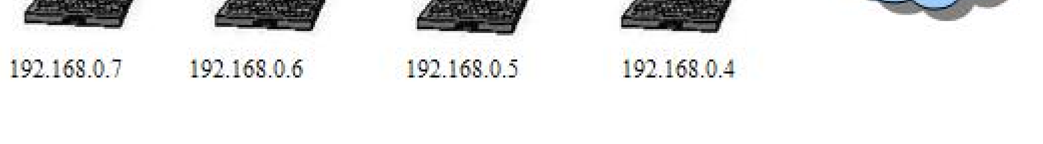


Figure 6-10: representation disposition of node during test

Before starting any test, I had deleted all entry routing table.

After I executed OLSR demon on each node using 192.168.0.4 like a gateway.

When 192.168.0.7 knows 192.168.0.4 existence, in the same time it knows that 192.168.0.4 is a gateway. In fact 192.168.0.7 adds a new entry in its routing table like figure 7.6 shows.

Sequentially, I tried to execute a ping from 192.168.0.7 towards Internet, and I had answers.

Time of answers from each node is the same: since the last is distant only 4 hop from Internet, it is impossible to establish if the packets for last node have a significant late for the user.

Even if it was not possible to do this kind test, the test performed the correct functioning even on node farer two-hops.

7 - Figure index

Figure 2-1: Left: Cellular mobile communication with infrastructure. ..12

Figure 2-2: direct Communication of wireless node 13

Figure 2-3: exchange of packets by forwarding 14

Figure 3-1: chart used to write down test results............................27

Figure 4-1:PICA overall architecture 31

Figure 4-2: designer API choises 33

Figure 4-3: Winpcap and NPF..34

Figure 4-4: general internal structure of PICA 35

Figure 4-5: PICA internal structure for Windows operating systems....36

Figure 4-6:PICA internal structure for Linux 38

Figure 4-7:Architecture of PICAselect() function in windows 40

Figure 4-8: Ethernet II packet format 52

Figure 6-1: definition of struct attached_gw_info 65

Figure 6-2: struct gw_info_definition 66

Figure 6-3: tcmsg structure definition 67

Figure 6-4: representation tcmsg structure.....................................68

Figure 6-5: representation of new tcmsg structure...........................69

Figure 6-6: tc_message structure definition 71

Figure 6-7: tc_message structure new definition..............................71

Figure 6-8: representation of an element of "attached_gw_list" 72

Figure 6-9: representation routing table of node 192.168.0.7............73

Figure 6-10: representation disposition of node during test 74

8 - References

[1] www.grc.upv.es

[2] Widens Project "wireless Deployable Network System For Future Public Safety" Thales Communication S.A. 2004 – www.widens.org

[3] FleetNet Project: "Inter- Vehicle Communication" – http://www.et2.tu-harburg.de/fleetnet/index.html (2003)

[4] Cartalk2000 Project "CarTALK 2000 – Safe and Comfortable Driving Based Upon Inter-Vehicle- Communication" - http://www.cartalk2000.net/(2004)

[5] Freinkfunk: "Freie Funknetze in deutschsprachigen Raum" http://www.freinkfunk.net/(2005)

[6] D. Maltz, D.Johnson, Y Hu: "The dynamic Source Routing Protocol for Mobile Ad Hoc Networks", internet-draft July 2004.

[7] C. Perkins, E. Belding-Royer, S. Das: "AODV RFC3561" experimental edition, July 2003.

[8] T. Clausen P.Jacquet "Optimized Link State Routing Protocol (OLSR)" Request-for-Comments:3626 October 2003 Available at: http://www.ietf.org/rfc/rfc3626.txt

[9] R. Ogier F.Templin M.Lewis Topology Dissemination Based on Reverse-Path Forwarding TBRPF (Request-For-Comments: 3684) February 2004. Available at: http://www.ietf.org/rfc/rfc3684.txt

[10] "Java(tm) 2 SDK, standard edition documentation", Sun Microsystem, Inc, 2002, available at http://java.sun.com/j2se/.

[11] Geoffry J. Noer, “Cygwin: A free win32 porting layer for UNIX Applications””, August 1998, Cygwin distribuition ia available at http://cygwin.com .

[12] T.Cluasen C.Dearlove P.Jaquet “The optimized Link State Routing Protocol version 2”, Internet-Draft “draft-ietf-manet-OLSRv2-03.txt”, MANET Working Group February 2007.

[13] “Testing computer software” Cem kaner, Jack Falk, Hung Quoc Nguyen

[14] http://www.swebok.org/ch5.html

[15] V. Jacobson, C. Leres and S. McCanne, “The libpcap packet capture library”, Lawrence Berkeley Laboratory, Berkely, Ca. Available at http:// www.tcpdump.org.

[16] www.winpcap.org

[17] http://www.winpcap.org/docs/WinPcap-AICA03.pdf

[18] http://msdn2.microsoft.com/en-us/library/aa366071.aspx

[19] Martin Hall and Dave Treadwel et al., “Windows sockets 2 application programming interface,” August 1997, Available at ftp://ftp.microsoft.com/.

[20] IEEE, Portable operating system interface (posix) – part 1: System application programming interface (api) [c language]” ISO/IEC 9945-1, 1996.

[21] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir R. Das, “Ad hoc on demand distance vector (AODV) routing”, Request For Comments: 3561, MANET Working Group, July 2003. available at: http://tools.ietf.org/html/rfc3561

[22] en.wikipedia.org/wiki/Ethernet_II_framing

[23] P.Jacquet, P. Muhlethaler, A. Qayyum, A. Laouiti, L.Viennot and T.Clausen "Optimized Link State Routing Protocol". Internet-Draft: “draft-ietf-manet-olsr-03.txt” MANET Working Group, November 2000.

[24] T.Clausen, C.Dearlove, J.Dean, C.Adjih “Generalized MANET packet/message Format”, Internet-draft: “draft-ietf-manet-packetbb-03,txt” Available at: http://tools.ietf.org