WATMIN (**W**ireless **ATM** **IN**terconnection)

Phase 1 Implementation Document

N. Tavier, C. Bonnet, D. Loisel
{tavier, bonnet, loisel} @ eurecom.fr

RR-97-038

Sophia, September 1997.

Institut Eurécom
Departments of Mobile Communications & Corporate Communications
2229, route des Crêtes
BP 193
06904 Sophia-Antipolis
FRANCE

Abstract

The purpose of this document is to introduce the WATMIN project, to describe its basic architecture and engineering concepts, and to describe the implementation of the Phase 1. The WATMIN project aims to provide seamless network connectivity to mobile computer users. It proposes to do this by employing the ATM technology as the backbone to interconnect multiple wireless networks.

Unlike some other research projects which attempt to provide wireless access to ATM by extending ATM over the wireless interface, WATMIN retains IP addressing and routing to the mobile computers, and uses ATM only to interconnect the wireless network equipment.

Mobile computing poses a number of specific technical challenges, such as mobility management, limited bandwidth of wireless links, address resolution protocol (ARP) problems, and handoffs for mobile hosts as they move. The paper details each of these problems and suggests tailored solutions for each.

WATMIN uses an innovative approach to extend the possibility of mobility between IP networks, by retaining the IP addressing to the mobile hosts. It also considers the decentralised architectures for communications between mobile hosts. It proposes adding intelligent processing to the access points, in addition to their usual function of packet forwarding between the wireless and fixed segments. This document presents the Address Resolution Protocols problems raised in such a context and the solutions we have proposed.

Introduction

ATM is becoming common place in the consumer market, with possible applications in all types of services and networks. ATM offers high speed transport, and can be used for both local and wide area networks supporting a range of applications including conventional data, and real time applications. Wireless networks allow computer users the freedom to roam over a local area, without the need to be physically interconnected with cumbersome cabling. The WATMIN (Wireless ATM INterconnection) project has proposed a solution dealing with both domains.

We consider a wireless network as consisting of a number of mobile hosts which are connected to the fixed network via an Access Point (AP). Wireless LANs may be based upon either a centralised or decentralised architecture. If the communication is centralised, connections between two mobile hosts are obliged to pass by the Access Point. For various reasons we have decided to use the decentralised one: when mobile hosts within the same cell can communicate directly, without passing via the Access Point.

This document presents the possibility of exploiting the benefits of ATM to provide mobility to users, by acting as a backbone for the interconnection of multiple wireless IP networks. The research has been carried out as part of the WATMIN (Wireless ATM INterconnection) project at the Institut Eurécom.

Table of Contents

List of Figures

1.0 Summary of the WATMIN Project

1.1 Outline

WATMIN will provide an ATM interconnection service between multiple wireless LANs. The ATM network will serve as a backbone for the transfer of data between individual wireless LANs. It is hoped to demonstrate the advantages of ATM over more traditional interconnection techniques with regards to mobility management. In our analysis of the problem, we first consider the possible scope of mobility that a mobile host may demonstrate as it moves over the ATM and IP networks The next section outlines the different types of mobility that may be exhibited by a mobile host.

1.2 Mobility Considerations in WATMIN

1.2.1 Mobility within a single radio cell

We consider a radio cell as the coverage area of an Access Point, thus it is centred around it. A host is mobile within the boundaries of this radio cell. Contact must be retained with the host as it roams over the Access Point coverage area. This is a purely radio path function performed by the wireless LAN protocol, and requires no additional functionality in the WATMIN project.

1.2.2 Mobility between neighbouring cells

A mobile host may migrate over the cell boundaries between adjacent Access Points. The individual radio path characteristics, and traffic flows define which of the access points has control of the mobile host at any particular time. Now we have the notion of handover between access points, whereby the control of the mobile host is transferred during the transmission of user data. In order to carry out the handover, and host location updates, we require a specific protocol operating between the access points, which is carried over the ATM backbone. In this scenario we consider the access points to be configured within the same Internet network.

1.2.3 Mobility between wireless LANs on different IP subnetworks

Here the mobile host passes over the boundary between two separate wireless LAN cells, which are no longer configured within the same IP subnetwork. This implies that the host is no longer contactable under the same IP network address. If we are to carry out effective handover between IP networks, we must be able to update the mobiles routing information and provide this information to the network. This requires a mechanism to perform location updating between IP networks, which once again must operate over the ATM backbone.

1.2.4 Mobility between different ATM routing domains

The mobile host may pass from one ATM routing domain to another. That is to say it moves between ATM subnetworks. This will call for some consideration to the differences in ATM addressing and signalling between different ATM subnetworks.

2.0 WATMIN Phase 1 Description

The WATMIN project has been divided into four logical phases. Each phase deals with a differing degree of host mobility. A description of the overall project stages for WATMIN is given in [1].

This section describes the system design for Phase 1 of the WATMIN project. It includes a detailed description of the system architecture and the new mechanisms.

2.1 Phase 1 objectives

The principle design aspects of phase 1 are:

- The interconnection of wireless and ATM networks by the Access Point within the same IP subnetwork.
- The design of Access Points that perform packet forwarding, interfacing and mobility management functions. (ATM / wireless cards, associated drivers and internal functionality).
- Programming / adapting drivers for ATM and WaveLAN networks.
- Definition of mobility management elements in both the Access Point (Association Agent daemon) and the Mobile Host (Wat_mobile daemon).
- Establishment of end to end connections between a Mobile Host and an ATM fixed host both in the same IP subnetwork.

2.2 Eurecom Testbed Architecture

In this section we intend to describe the Watmin Testbed Architecture used in EURECOM. Figure 1 presents our current configuration. Within the same IP subnetwork, we have 2 wireless cells interconnected with our campus ATM backbone. Note that we call a cell the coverage area of an Access Point..

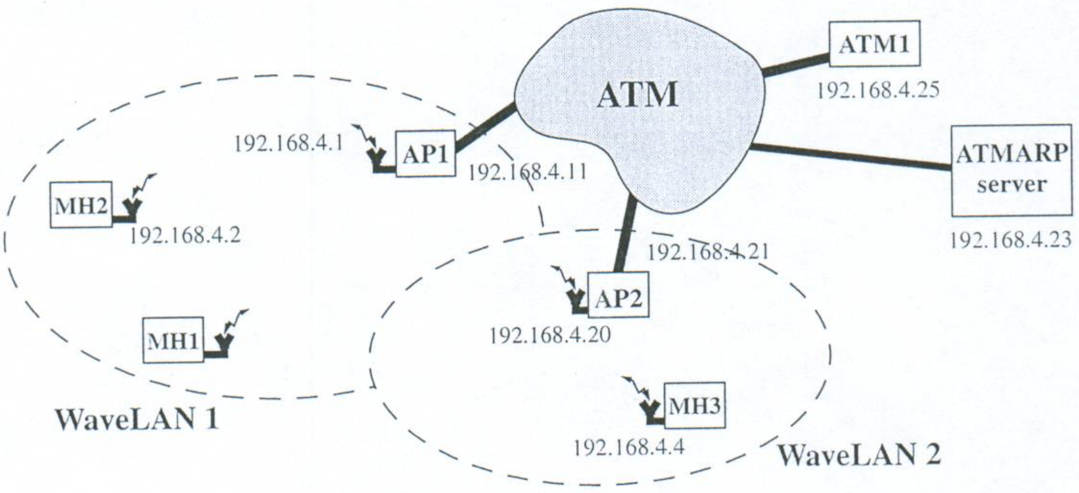


Figure 1 Proposed Architecture

2.2.1 System Components

- **Mobile Hosts** (MH):
    - 2 x Dell 486/T PCs running at 33 MHz. Equipped with 2 Mbps NCR WaveLAN-AT cards. Using the **LINUX** operating system.
    - **WaveLAN** Driver :The WaveLAN card is usually shipped with a MS-DOS, Windows, or OS/2 driver. However NCR also supply an experimental driver for WaveLAN under LINUX. Also D. Joseph of the Massachusetts Institute of Technology has made his LINUX driver for WaveLAN PCMCIA available on the public domain.

- **Access Point** (AP): The Access Point for the Mobile Hosts to the fixed network. It has to forward IP packets between the wireless and the wired networks.
    - Pentium PC running at 133MHz. Equipped with 2 Mbps NCR WaveLAN-AT card, (wireless interface), and an Efficient Networks EN155P adapters (ATM card). Using the **LINUX** operating system.
    - ATM Driver: Work is currently in progress at the EPFL, where they have developed and tested two device drivers and a suitable API for ATM on LINUX. The drivers are for the Zeitnet ZN1221 and Efficient Networks EN155P adapters. Both of these are PCI bus cards and run at 155Mbps over multimode fibre. The drivers support Classical IP (RFC 1577), and LAN Emulation. The device drivers are responsible for the segmentation and reassembling of IP packets for transportation over ATM as well as physical layer operations. The API contains daemons for establishing ATM sockets, (PVC or SVC), exchanging data, and closing the connections when the data transfer is complete.
    - **Each interface (WaveLAN and ATM) of the AP is configured with a distinct IP address within the same IP network.**
- **ATMARP Server:** (ATMARP). A dedicated process situated on an ATM host, responsible for address resolution, (IP --> ATM)
- ATM Switch --> *ForeRunner* ASX-200.

3.0 Watmin Protocol

With regards to the ATM configuration and operation, WATMIN is closely based upon Classical IP (marked **[C-IP]**). Additional functions have been added in order to provide support for the section of native LAN (WLAN), and for host mobility. Inspiration has been taken from Mobile IP **[M-IP]**, WaveLAN protocols **[WaveLAN/AROUND]**, and LAN Emulation **[LANE]**. Any additional functions that are not available under any of these protocols are marked **[WATMIN]**.

3.1 Access Point Start-up / Registration of all ATM Clients

NOTE: The AP registration process is the same as classical IP client registration.

- All ATM clients (i.e. hosts, routers, servers, and the access points), possess their ATM and IP addresses (pre-configured in memory). **[C-IP]**
- Also all ATM clients are pre-configured with the ATM address of the ATMARP server. **[C-IP]**
- Upon start-up, access points (AP) connect to the ATMARP using a point to point VC. **[C-IP]**
- The ATMARP sends an **[InATMARP-req]** to the AP requesting its IP address. **[C-IP]**
- The AP replies with an **[InATMARP-rep]** containing its IP address. **[C-IP]**
- The ATMARP server constructs an ATMARP table containing bindings of ATM to IP addresses for each client on the network. These values have a limited lifetime and must be updated at regular intervals. **[C-IP]**

3.2 Mobile Host Association to an Access Point

The scenario: A mobile host (MH) starts-up in a single cell (i.e. no overlap), of its home IP network[1]. The mobile host will have to register itself to the Access Point wich have the control of the cell. The registration procedure of a client to an Access Point is WATMIN specific. The procedure is described below:

- An Association_Agent (AA), is a daemon process situated on the Access Point, whose purpose is to allow Mobile Hosts to associate themselves to an Access Point. **[WATMIN]**
- A daemon WAT_mobile daemon equivalent is situated on each mobile host. **[WATMIN]**
- The AA broadcasts **Association-Agent-Advertisement** messages (lvl 3), periodically to all the mobile hosts in the cell coverage. **[WATMIN]**
- ARP[2] takes care of the IP to MAC address resolution, and the message is broadcast on the wireless interface.
- The client on the MH responds to this **Association-Agent-Advertisement** message with an **Association-Request** message (lvl 3), which it sends to the Association Agent daemon. **[WATMIN]**
- The **Association-Request** message contains, the IP address of the MH and other information which is used to select the mode of operation and authenticate the request.
- The AA opens a connection to the ATMARP server, which in turn sends an **[InATMARP-req]**. The AA responds to that request with an **[InATMARP-rep]** where it substitutes in the source IP address field its IP address with the MH's one. It leaves unchanged the source ATM address field (it contains the AP ATM address).

1. *All mobile hosts have a home IP network, where they are usually located i.e office or work place.*
2. *ARP is an integral part of IP equipped hosts, mobile or not.*

- Therefore the ATMARP server has a new entry in its ATMARP table. The association MHs IP address with the APs ATM address.
- To confirm the registration to the ATMARP server, the AP sends an **[ATMARP-req]** for the IP address of the MH.
- Once it receives the **[ATMARP-rep]** message the AP stores the IP address of the MH in its 'Watmin Association table',[3] and returns an **Association-Reply** to the mobile confirming that it is successfully associated to the AP. The IP table entry includes the interface on which the MH is found, (in the case of multiple port gateways). **[WATMIN]**
- The MH is now associated to the AP, and can send and receive messages. **[WATMIN]**

3.3 Connection between MH to an other Host, all in the same IP subnetwork

A mobile host wishing to send packet to an other host only has the destination IP address.

- In order to resolve the IP address to a usable MAC address it uses ARP **[IP].** The MH broadcasts an **[ARP-req]** to all hosts on its subnet (using the subnet mask), over the MAC broadcast address. **[IP]**
- One of two things may happen:

1. The destination is in the same cell coverage[4] as the MH and returns its own MAC address to the MH in an **[ARP-rep]**.
2. The destination is NOT in the same cell coverage, but beyond the Access Point.
    - Thus, the AP will perform an **[ARP-rep]** to the MH where it substitutes in the IP source field the IP address of the destination to its IP address. It fills the MAC source field with its MAC address. Thus the AP acts as a proxy **[WATMIN]** (ARP modifications).
    - NOTE: the AP maintains a table of IP addresses for all MHs currently in the cell coverage (the WATMIN Association Table). Thus if it sees an **[ARP-req]** with the IP address of a MH currently in the cell coverage -- it ignores the request in order to respect the decentralized model. **[WATMIN]**
    - Then, the AP performs an **[ATMARP-req]** to the ATMARP server, with the IP address of the destination. **[C-IP]**
    - If the ATMARP server has a value in its tables for the destination (i.e the host is registered), it will return the ATM address of the destination host to the Access Point. **[C-IP]**
    - The AP will store the IP: ATM binding in a local cache for future reference, and open an ATM connection to the destination. **[C-IP]**
    - The MH upon receiving an **[ARP-rep]** will store the binding [MAC:IP], in a local ARP cache for future reference. **[IP]**
    - The MH will address the frames, (level 2) to the MAC address of the AP, whilst retaining the IP address of the destination host (level 3). **[IP]**
    - The AP will then forward the packets, (level 3) to the destination host over the ATM circuit using encapsulation (RFC 1483). **[C-IP]**

---

3. *Association table lists IP addresses for all MH currently associated to a particular Access Point*
4. *A mobile node on the same WLAN network can enter into direct communication with the source mobile node WITHOUT passing via the Access Point.*

- NOTE: The fixed ATM host is unaware of the fact that the destination host is a mobile. The access point acts as a gateway, screening level 3 packets incoming to its cell coverage. The AP performs encapsulation of IP packets over the ATM network to the destination host.

3.4 Address Resolution Protocol mechanisms

We have just explained the design of the Watmin protocol and especially the modifications of the ARP mechanism required in the AP if want it to act as a kind of Proxy. Now we have to take care of sides effects regarding the Address Resolution Protocol in our special configuration. The overlapping of cells configured within the same IP subnetwork causes serious problems regarding the ARP mechanism. We have to underline the fact that problems are coming from our choice of having various cells and our ATM backbone all configured within the same IP subnetwork.

3.4.1 Address Resolution Protocol Problems

First we would like to describe the sides effects that could occur in our configuration.

- 1st Problem:

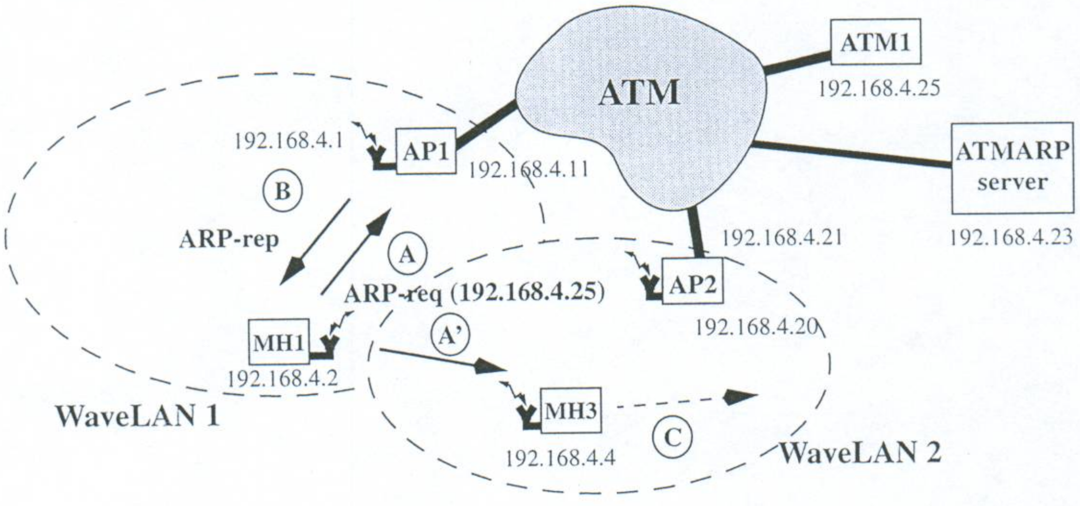


Figure 2 ARP problems

Let us consider the following configuration (Figure 2), the Mobile Host MH1, located close to the border of its cell (AP1 cell) issues an **[ARP-req]** (A) for the IP address of the ATM workstation ATM1 (192.168.4.25). As we have already underlined it, this request is broadcasted through the wireless Interface. As we have modified the ARP mechanism of its AP, AP1 will respond to that request (by sending an **[ARP-rep]** (B)) because in our configuration it acts as a proxy for the ATM workstation ATM1. But the MH3 in the neighbouring cell is able to receive that request (A') (because of the broacasting). Of course it will not respond to it, as is not concerned by the request. But with the normal implementation of the ARP mechanism under LINUX when an **[ARP-req]** is received, whether it has to respond or not, it has to check if it has already an entry in its ARP table regarding the source IP address of the request and if it has one, it has to update this entry with the source MAC address contained in the request. Note that if the request is not regarding one of its IP address, if it has not an entry for the source IP it won't create one ! Therefore if a MH (MH3) in the neighboring cells receives an **[ARP-req]** from a MH registered to another cells, if it has already an ARP entry for the IP address of the requester, it will thus update this entry. If these stations were not moving it would not have created any problems but as their name say they are Mobile. Therefore the MH could have updated its ARP entry. In our example MH3 could have updated its ARP entry for the IP address of MH1, if it moves (C) it could be in a position where it can no longer communicate directly with MH1, thus its ARP table will no longer be valid and the MH will not be able to recover from that situation. In this case, it will just believe that the MH1 is down. So we have to prevent the system from entering in such a situation, for that we have to guaranty the constant validity of the ARP table in each MH. We have to forbid direct communications between MHs that are not registered to the same cell. We propose to add some modifications to the ARP mechanism in all AP to avoid that kind of problems. Because in a first step we did not want to modify the kernel of each MH.

- 2nd Problem:

The other problem is almost the same but it is more comprehensible (see Figure 3). We take the same initial configuration that we have presented for the first problem. This time, we take the case where a MH is sending an **[ARP-req]** for an IP address of an MH registered to the neighbouring AP. In our figure, MH1 is requesting the resolution for the IP address of MH3. For that it broadcasts an **[ARP-req]** ((A) and (A')). Therefore, with the modifications added to the ARP mechanism of the AP (AP1), this one will "normally" responds (B) to that request because the only way to contact an host outside the group of mobiles registered to this AP should be to pass throuhg this AP. But in our case if the mobile is close to the fronteer of the cell where MH3 is located it will be able to receive the request (A'), thus it will update or create its ARP entry table for the IP address of the requester and will reply with an **[ARP-rep]** (B') containing its own MAC address. Thus, MH1 will receive two different responses. Depending upon the order in which it has received those replies, MH1 may have an invalid entry. It could establish direct "air"connection with a MH outside of its cell and this must be avoided. Because as we already described in the first problem, if one of the mobile moves off (C), the connection may no longer be possible and thus each mobile would have no way to recover from that situation. Thus we need to assure the constant validity of each MH's ARP cache.

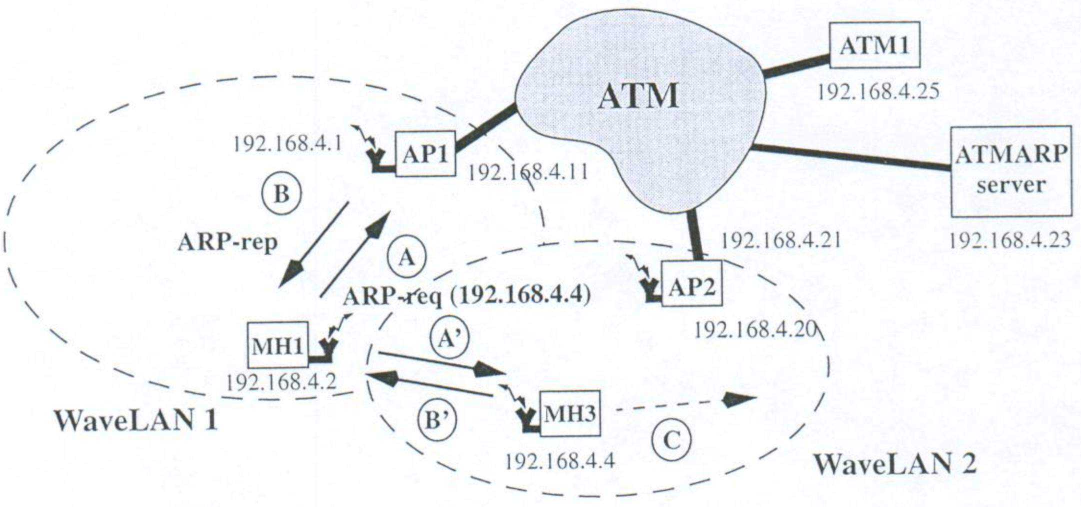


Figure 3 ARP problems

3.4.2 Solutions to ARP problems

To respect the WATMIN requirements (i.e. no modifications of the protocol stack of the MH), the solution we proposed is to add other modifications in the ARP mechanism of the AP and to exchange messages between neighbouring APs (see Figure 4).

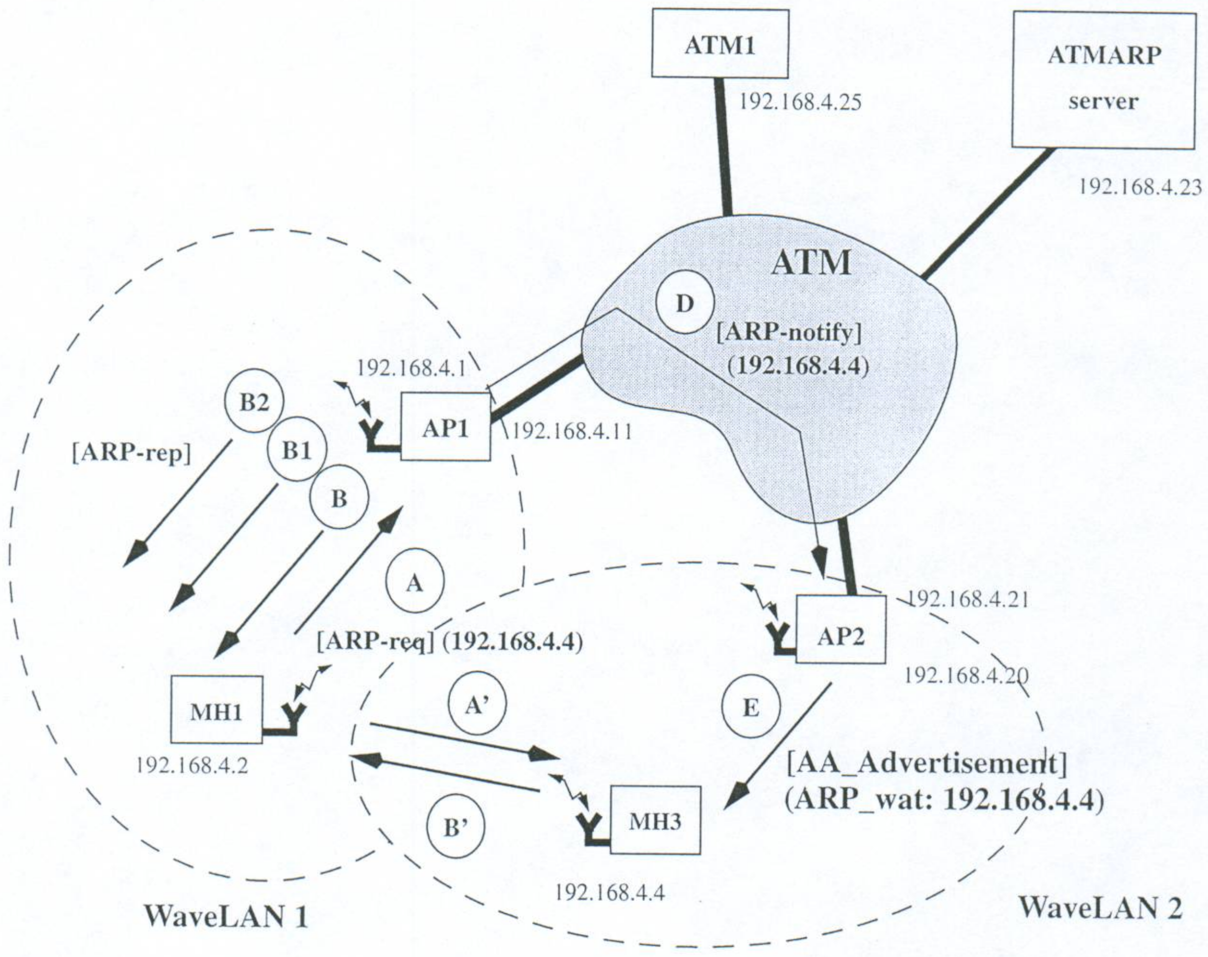


Figure 4 ARP Solutions

In the AP, when it receives an **[ARP-req]** (A) sent by a MH in its cell, the ARP function will systematically inform the Association Agent that an MH (with IP address: IP@MH) has issued an **[ARP-req]** (The aim of that procedure is to inform the AA that it has to notify its neighbouring AP that their MHs may have wrong ARP information about the requesting MH). On the other hand, in order to force the requester to get the right information in its ARP table, we will re-transmit 2 more identical **[ARP-rep]** (B1)(B2) after a certain delay. The AA will then send messages (D) to the APs of its neighbouring cells to inform them that their MHs may have invalid ARP entry for the IP address of the MH that has issued the **[ARP-req]**. Each AP will make use of the **[AA_Advertisement]** (E) messages that are periodically broadcasted to inform the Wat_Mobile daemon of each MH of its cell, that they have to update or to create the ARP entry with the right association: MAC address of their AP with the IP address of the MH that has issued the

[ARP_req]. For that purpose we have to create a new field ARP_wat in the **[AA_Advertisement]** to carry this information. Thus the AA fills the ARP_wat field of the AA_advertisement with the IP address of the requester. As the Wat_Mobile daemon in each MH receives the **[AA_Advertisement],** it thus updates or creates an entry even if it has not required it. Therefore with that procedure, we are trying to prevent MHs from issuing to much **[ARP-req]** and we assure that the ARP cache of each mobile is valid in the sense that no direct communication is possible between two mobiles that are not registered to the same Access Point. The details of the implementation are explained a little bit further.

4.0 Configuration validation

After having achieved the design phase and having chosen the cards we will use for the Access Point and for all mobile hosts, we started to try to send packets from a mobile host to an ATM station and in the reverse direction as well. This was the first step to validate our choice of cards.

For this test, we decided to have two subnets: one for the WaveLan (192.168.4.0) and one for the ATM subnet (192.168.3.0). The Access Point will act as a gateway between the 2 subnets. The fact that we have chosen 2 subnets was led by the simplicity of such a test where no WATMIN specific modifications was required. The configuration we have tested can be seen in the figure below. The test consists in sending packets from "nelke" to "purcell" and in the reverse direction.

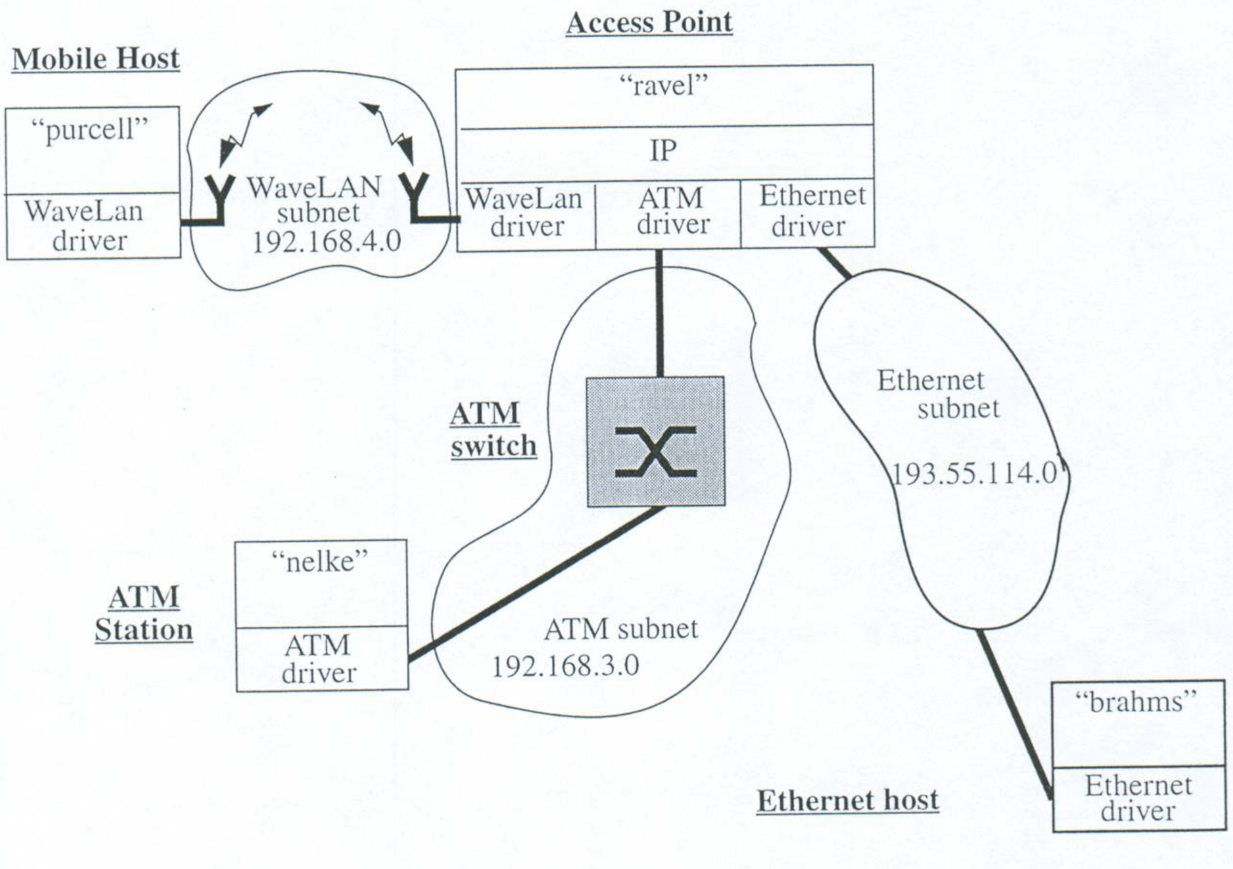


Figure 5 Configuration for validation

4.1 Ping Test

In the test we will try to ping purcell from nelke. When pinging host B from host A, indeed we send an ICMP message from host A to host B, and the host B send back an ICMP message. Thus with ping we test the connection between host A and B.

Unfortunately the test did not work. Hence, in a first step we tested different configuration that were using the cards we had chosen:

- from nelke (ATM station): >>ping brahms (Ethernet host). The test was valid.
- from purcell (Mobile host): >>ping brahms (Ethernet host). The test was valid.

At this step, we were sure that the problem was coming from the drivers in the AP and specifically in the case of a ping between an ATM station and a Mobile host. But we did not know at this step, if the problem of forwarding a packet in the AP was in both directions or only in one and in this case in which direction the AP was working well.

4.2 Debugging

In order to precise where the problem was located, we had to make the various functions print many debugging informations. The functions was the ones used when the forwarding of packet is required. This is almost the only way of debugging the kernel: by using the function *printk* that prints messages in the file */var/adm/messages* (see [1] page x). As you can imagine, we had to analyse a huge amount of debug messages in order to find out where the problem lied. We quickly concluded that there was only one direction for which the AP was unable to forward the packet: from a Mobile host to an ATM host.

When analysing more accurately the debugging messages, the only hint we had was this two messages:

1. >> eni(itf 0): VCI 80 has mis-aligned TX data

2. >> put_dma: unaligned addr (0xdb8426)

This two messages are coming from the ATM driver. The packet coming from the WaveLAN's side is passing through the WaveLAN driver without any fatal error, through the IP layer without any trouble and it's when is passing through the ATM driver that the problem occurs. And we get this two messages. The first one comes from the function *do_tx()* in *linux/drivers/atm/eni.c* where there is the test:

```
if ((unsigned long) skb->data & 3)
printk(KERN_ERR DEV_LABEL "(itf %d): VCI %d has mis-aligned TX data\n",
vcc->dev->number,vcc->vci);
```

The second comes from the function *put_dma()* in *linux/drivers/atm/eni.c* where we can find the test:

```
if (paddr & 3)
printk(KERN_ERR "put_dma: unaligned addr (0x%lx)\n",paddr);
```

As the messages were coming from the ATM driver, we asked the author [2] of this driver if he had an idea from where the problem comes from. Werner Almesberger (DI-LRC, EPFL, Lausanne) explained to us that the ATM driver requires an alignment to a four-byte boundary for reasons of performances of the system. He said that because mis-aligned memory accesses also tend to be a lot more expensive than aligned ones, so the overall performance of the systems using that driver is improved by making sure the alignment is right. The problem here is that the byte-shifter in the ENI board that is supposed to take care of re-adjusting mis-aligned data does not seem to work very well. So he suggested that we should look into fixing the alignment of packets received by the WaveLAN board.

So we decided to concentrate our effort on the value of *skb->data* and the assignment of this value. As we have said previously, the only way of debugging the kernel is to make use of the function *printk()* to print debugging informations. So we print the value of skb->data all along its trip through the different functions of the WaveLAN driver and through the IP layer's functions.

It turned out that the problem was only coming from the assignment of *skb->data* and what is done with this value within the wavelan driver, especially in the function *void wavelan_receive(device *dev)* in file *linux/drivers/net/wavelan.c*. We concluded that the problem was located here because apart from the scope of this function the value of *skb->data* is never modified until it is passed to the ATM driver.

We show you what is done in the function *wavelan_receive(device *dev)* regarding *skb->data*:

```
1     sksize = pkt_len;
2     if ((skb = dev_alloc_skb(sksize)) == (struct sk_buff *)0)
3     {
4             printk("%s: could not alloc_skb(%d, GFP_ATOMIC).\n", dev->name, sksize);
5             lp->stats.rx_dropped++;
6     }
7     else
8     {
9             skb->dev = dev;
10            obram_read(ioaddr, rbd.rbd_bufl, skb_put(skb,pkt_len), pkt_len);
11            skb->protocol=eth_type_trans(skb,dev);
12            netif_rx(skb);
13            .........
14    }
```

The function *wavelan_receive()* as its name says is the one called when a packet is received on the WaveLAN card. Before this piece of code, some checking are done but they are not regarding our matter. What we show you, here, is where the *skb->data* is allocated in order to store the data received on the card. At line 2, an skb structure is allocated with respect to the packet length *pkt_len*. Here the value of *skb->data* is aligned on four-byte. The four-byte alignment means that *(skb->data & 3) = 0* so that in hexadecimal format *skb->data* ends by 0,4,8 or A. At line 10, the packet received on the card is read and written in the *skb* structure thanks to the function *obram_read()*. This step does not modify the value of *skb->data* so the alignment is kept. The next step is the one responsible for our problem. The function *eth_type_trans()* adds ETH_HLEN= 14 to *skb->data*, but as you can see 14 is not a multiple of 4, hence at this step we loose the four-byte alignment. And at line 12, the structure *skb* is passed to the function *netif_rx()* that put the *skb* in a queue that IP layer's functions will process. As we have said previously the IP layer's function will not modify the value of the *skb->data*. So we can say that we have located the problem.

4.3 Bug fix

After having located where the problem lays, we have to find a kind of glue in order to fix it. What we want is that, before calling the function *netif_rx()*, *skb->data* must be four-bytes aligned. We proposed the following modifications of *wavelan_receive()*:

```
1    sksize = pkt_len + 2;
2    if ((skb = dev_alloc_skb(sksize)) == (struct sk_buff *)0)
3    {
4            printk("%s: could not alloc_skb(%d, GFP_ATOMIC).\n", dev->name, sksize);
5            lp->stats.rx_dropped++;
6    }
7    else
8    {
9            skb->dev = dev;
10           skb_reserve(skb,2);
11           obram_read(ioaddr, rbd.rbd_bufl, skb_put(skb,pkt_len), pkt_len);
12           skb->protocol=eth_type_trans(skb,dev);
13           netif_rx(skb);
14           .........
15   }
16
```

Hence we proposed to add 2 to the value of *skb_size* in order to reserve 2 more bytes for the *skb* structure, this is done at line 1. And then at line 10, we use the function *skb_reserve()* from *linux/net/core/skbuff.c* that will add 2 to *skb->data*, updates other values of the structure *sbk* and makes some checking on this structure. Thus, from the step of the allocation of *skb->data* to the call of *netif_rx*, we have added 14+2=16 to *skb->data* therefore we have kept the four-bytes alignment of *skb->data*.

Now, the test ping purcell from nelke is working. Therefore we have a configuration where the AP is able to forward packet between a mobile host to an ATM station in both directions. Therefore we have fixed the problem and we can say that the configuration we are working with, is ready for the WATMIN specific modifications. The validation phase of the configuration is now achieved.

5.0 Access Point Architecture

In this section, we will describe the current architecture of our AP. We present the various modules that has been implemented and all the modifications done in the kernel. We will first introduce our approach to carry out the AP functionalities. And then we try to give to the reader an accurate description of the implementation.

5.1 AP Design Approaches

As we have already described it, in section 2.0, we have chosen to have 2 wireless cells interconnected with our ATM backbone, the important characteristic is that they are all configured within the same IP subnetwork. Thus, the AP should act as a kind of bridge as it is interconnecting segments which are configured within the same subnetwork. We also could use the term of "proxy" to qualify its action. In a first step, we were thinking about modifying the IP layer (see figure 6) and more precisely the forwarding function (linux/net/ipv4/ip_forward.c). We had imagined that the *ip_forward()* function (see figure 7) would have use an internal (kernel) table (containing all the IP address of the MH associated with this AP) to forward the packet to the right driver. But modifying this was not enough, because not only the *ip_forward()* function need to know how to route those packets but functions like *ip_rcv()* in *linux/net/ipv4/ip_input* and much more. Indeed if we add kept this idea, we would have re-written another kind of routing table.

Therefore we have dropped this idea and we have decided to make use of the routing table. This idea was much more better in the sense that we did not have to make big modification in the Kernel, thus keeping the Watmin idea to make a light weight implementation in term of modifications in the Kernel. The other good aspect is that we do not have to take care about the validity of a new routing mechanisms implemented in the Kernel. By using the routing table already implemented we are almost sure that things will work well. But the idea of having a "Watmin Association Table" in the Kernel is kept in order to solve ARP problems.

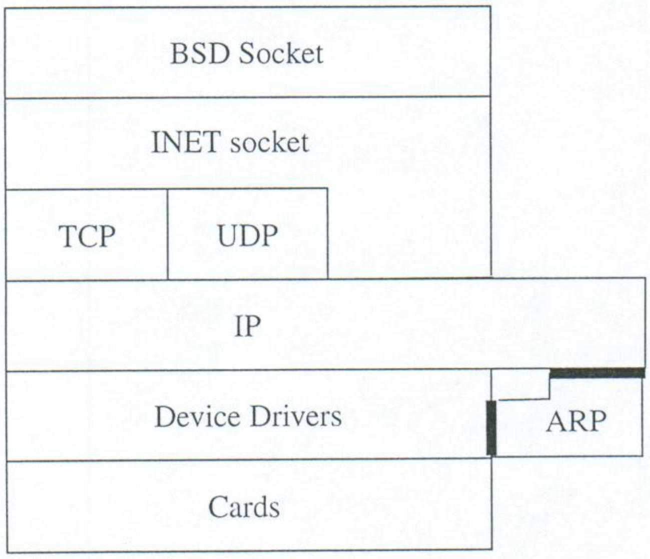


Figure 6 The layer structure of a network.

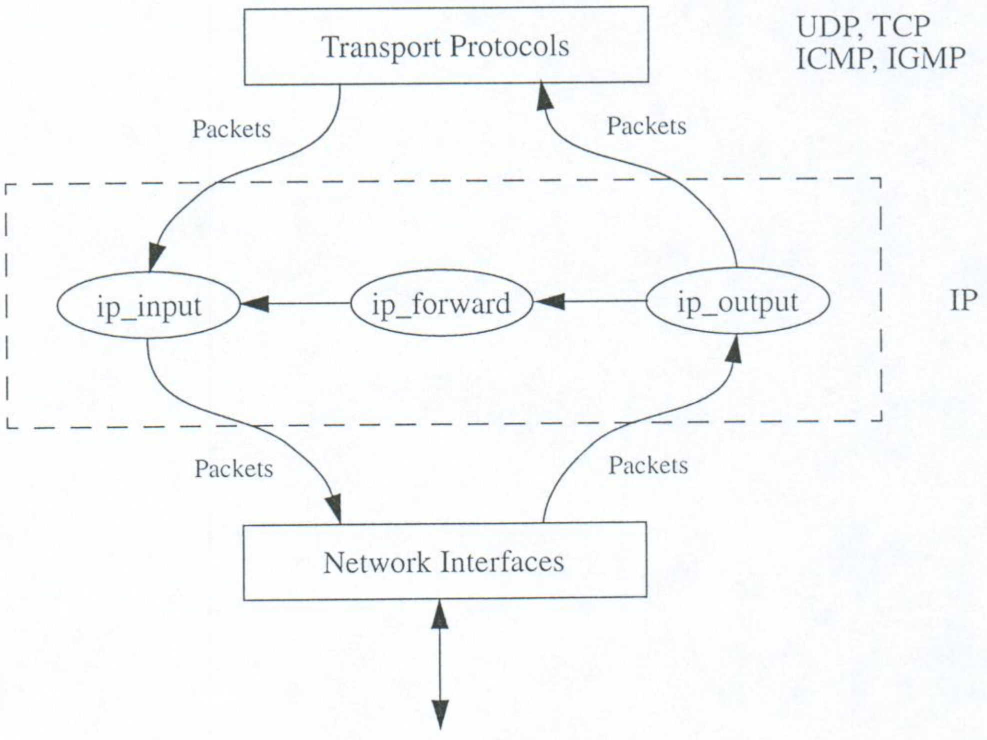
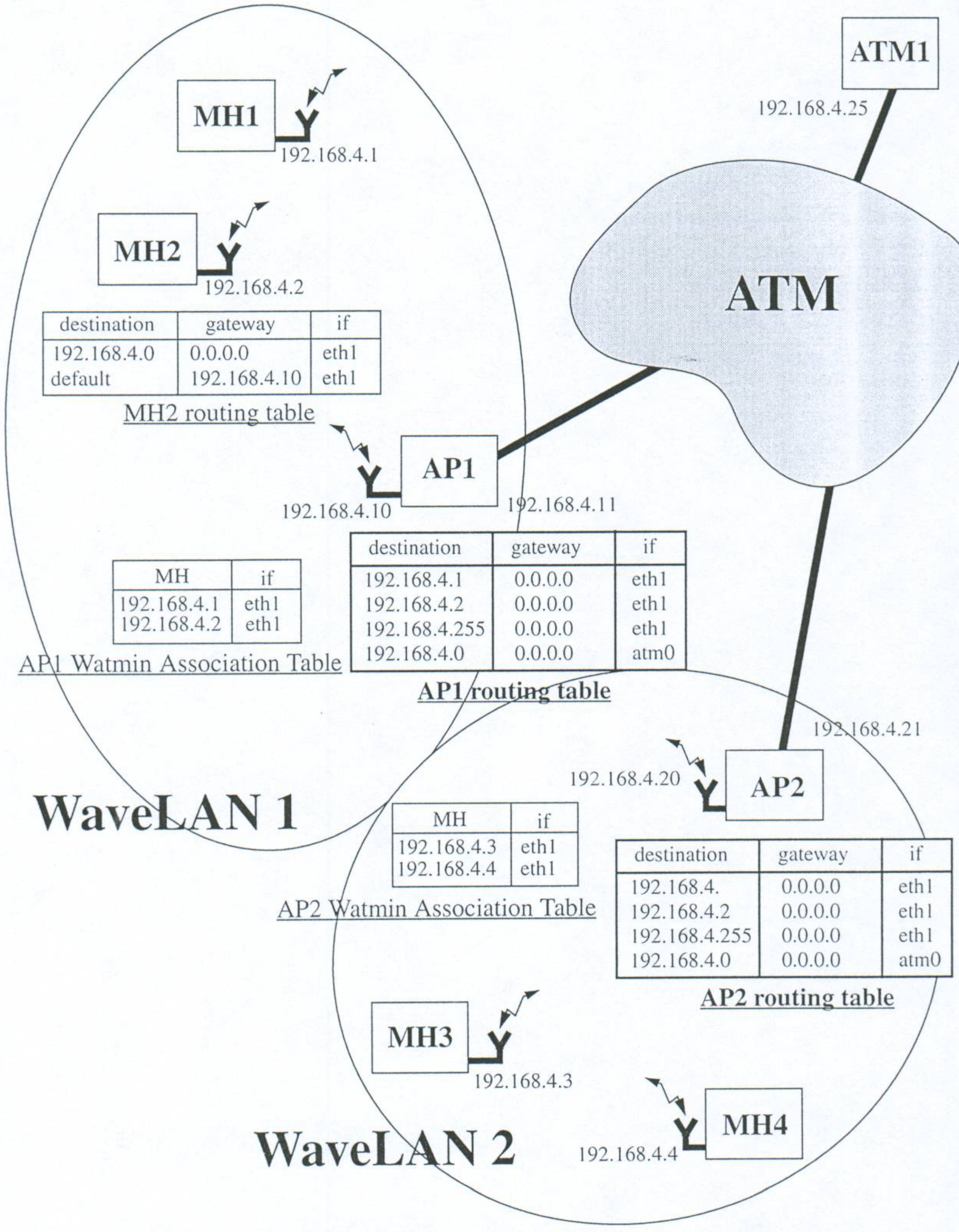


Figure 7 IP layer processing.

5.2 IP Routing in the AP

To understand how we will use the routing table in the AP, we prefer to present the testbed configuration. In figure 8, we represent a typical situation of WATMIN where we have 2 WaveLans with their respective Access Point (AP1 and AP2) connected to the ATM backbone. We show the case where there are 2 mobile hosts in each WaveLAN Cell. What is interesting in the figure is that we represent the state of several routing tables as well as the "Watmin Association table" in each AP. We have to underline the fact that in this configuration all the mobiles hosts are already registered with an AP. Our aim is to show how packet will be routed.



destination	gateway	if
192.168.4.0	0.0.0.0	eth1
default	192.168.4.10	eth1

MH2 routing table

MH	if
192.168.4.1	eth1
192.168.4.2	eth1

AP1 Watmin Association Table

destination	gateway	if
192.168.4.1	0.0.0.0	eth1
192.168.4.2	0.0.0.0	eth1
192.168.4.255	0.0.0.0	eth1
192.168.4.0	0.0.0.0	atm0

AP1 routing table

MH	if
192.168.4.3	eth1
192.168.4.4	eth1

AP2 Watmin Association Table

destination	gateway	if
192.168.4.	0.0.0.0	eth1
192.168.4.2	0.0.0.0	eth1
192.168.4.255	0.0.0.0	eth1
192.168.4.0	0.0.0.0	atm0

AP2 routing table

Figure 8 Typical WATMIN Configuration.Access Point Architecture

The situation is not so easy because we are working within the same IP subnetwork. This is why the reader may be somewhat surprised by the content of the routing tables. In an AP, the routing entry regarding the subnet 192.168.4.0 to which it belongs says that it can be reached through the ATM interface (atm0). Thus, this table entry routes all the packet destinated to our subnet, to the ATM interface. You could ask how the MHs could be reached in such a situation. When a MH is registered to the AP, we had a new routing entry that says that the IP address of the MH can be reached by sending the packet through the WaveLAN interface (eth1). Thus, for each MH asociated to the AP, we add a new entry in the routing table. Those new entries have a flag set that means that this routing entry is regarding a host. This is very relevant because someone could ask if there was no conflict with the routing entry regarding the IP subnet. There is no problem because when trying to find a routing entry, the ip_rt_route() function (net/ipv4/route.c) first looks for host entry, therefore the MH routing entries will be taken into account before the subnet entry. An other special entry need to be set up at the begining, the entry for broadcasting on the subnet we need this one because the Association_Agent has to broadcast the **[AA_Advertisement]** message on the wireless cell. For that we add the entry 192.168.4.255 passing through the Wireless interface (eth1). The figure 9 represents how the routing table of the Access Point AP1 looks like when there is 2MHs already registered to it. Figure 8 gives a summary of the topic and represents the routing table of a Mobile Host registered to an Access Point.

Just a word about the routing table of the MH, the only modification for it is that when the Wat_Mobile daemon receives the confirmation of its association with the AP, it modifies only one routing entry by setting the IP Address of the AP as the default route.

destination	gateway	if
192.168.4.1	0.0.0.0	eth1
192.168.4.2	0.0.0.0	eth1
192.168.4.255	0.0.0.0	eth1
192.168.4.0	0.0.0.0	atm0

Figure 9 Routing table of the Access Point (AP1) with two MHs registered.

At this step, let us give you a summary of the situation in order to understand the use of these dynamic modifications of the routing tables.

- When an ATM workstation wants to send a packet to a MH:

  As we have described it in section 3.3, when the MH is registered to an Access Point, the Association procedure has added a new binding to the ATMARP server (IP address of the MH : ATM address of the AP), thus, when an ATM WS wants to send a packet to the MH, it fisrt ask the ATM server to resolve the IP address to the ATM address. The ATM WS sends the packets to the AP. The packet received on the ATM interface of the AP goes up to the IP where the routing tables are consulted. As it has an entry for the IP of the MH the packet is thus forwarded on the Wireless interface. And finally it reaches its final destination.

- In the reverse direction, when a packet is sent from a MH to an ATM workstation:

  When the MH wants to send an IP packet to an ATM WS station, the first time it needs to resolve the IP address to the MAC address by sending an [ARP-req] (as they are within the same IP subnetwork). As we have modified the ARP mechanism of the AP, this one will respond for the ATM WS. Thus, the MH will send its packet to the AP. With the routing entry that says that the subnetwork can be reached through the ATM interface the IP layer passes the packet to the Classical IP module that sends the packet to the final destination: the ATM WS.

5.3 Architecture of the implementation of the Access Point

In this section we present the Access Point Architecture. We described wich modules has been created to carry out the Watmin functionnalities and wich modules has been slightly modified, in the Kernel and in the user space.

In order to carry out the watmin functionnalities, we have decided to implement most of them in a daemon in the user mode. We did not want ot overload the kernel if we were able to do it.

Figure 10 gives a good overview of the architecture of the Access Point.

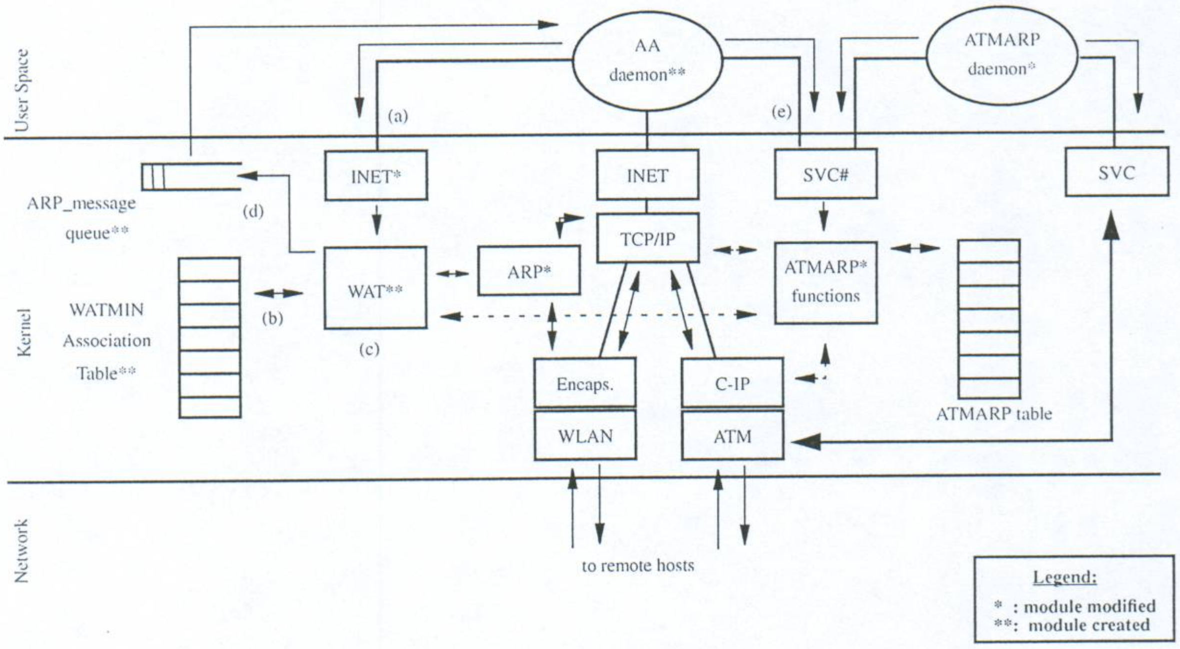


Figure 10 Architecture of the Access Point

- **Association _Agent daemon:**

  This module is the daemon that implements most of the functionalities of the Watmin Protocol. It is implemented in the user space. You can find this module in the directory /usr/src/watmin under the name AA_daemon.c.

  When started, it makes use of a new system call that will initiate the various functionalities of the WAT module in the Kernel and creates a message queue (ARP_message queue) in order to receive message coming from this modules. When started the AA_daemon needs to know which AP are its neighbours. This is required for the procedure that solves the ARP problems.

  It is responsible for broacasting the **[AA_Advertisement]** messages. It waits also for the **[Association-Request]** from the MHs that would like to be associated with it, in turns it makes a new system call in order to inform the WAT module in the kernel that a new mobile wants to be registered to the AP. This system call is carrying the IP address of the MH, it is essentially used by the WAT modules to updates the WATMIN Association table. In order to register the new binding to the ATMARP server we have to call the ATMARP client modules that will be responsible for that. Thus, we have to use the system calls already implemented by the solution of Werner Almersberger. We have to define new flags for those system calls in order to implement our specific call to a new functionality of the ATMARP modules. When the AA_daemon receives the confirmation from the ATMARP modules saying that the ATMARP server has register the new binding, it acknowledges the Association of the MH by sending the **[Association_Reply]** to it. At this step the handover of the MH is not implemented!

The AA_daemon is responsible for listening to the message sent in the ARP_message queue by the ARP module. In turn, it will notify to its neighbouring APs that a MH in its cell has issued an **[ARP-req]** by sending an **[ARP-notify]** message containing the IP address of the MH. Then the other AA_daemon will add the IP address of the MH that has issued the **[ARP-req]**, in the wat_arp field of the **[AA_Advertisement]** broadcasted periodically. Thus, each mobile will be able to update or to create an ARP entry containing the binding IP address of the MH: MAC address of their AP.

For further details on the implementation please refer to the code in the file /usr/src/watmin/AA_daemon.c on the PC "ravel".

- **ARP module**

  The ARP modules has been modified with respect to the solutions we have proposed in section 3.3 and 3.4.2. The details of the modifications of the ARP mechanism are given in section 5.4. The modifications are regarding the file /usr/src/linux/net/ipv4/arp.c

- **WAT module**

  At this step of the implementation, what we call the WAT module is implemented in the file /usr/src/linux/net/ipv4/arp.c, it regroups the functions that treat the various system calls made by the AA_daemon and the functions that handle the Warmin Association table.

- **ATMARP module**

  We call it a module but this represents the various function that process the system calls used by the ATMARP daemon in the user space (for further information please refer to [2], it explains more about the implementation of classical IP under linux. The aim here is to modify the implementation in order to accept other system call. These one will be responsible for the procedure that will register a new binding to the ATMARP server. In the section future work we will present the concept of the procedure and explain where the modifications should be made.

5.4 Address Resolution Protocol (ARP) Modifications.

As we have already said: the routing tables are a little bit modified as well as the way they are consulted. In section 3.3 we have described the modifications of the ARP mechanism in the AP required if want it to act as a PROXY. In section 3.4 we have underlined ARP problems raised in such a context and we have proposed solutions to overcome them. Thus in this section we present the modifications implemented in the ARP mechanism.

5.4.1 ARP Mechanism Flow Chart in Linux

We will, first, describe how the ARP mechanism works when an **[ARP-req]** or an **[ARP-rep]** is received. We have to underline the fact the ARP implementation is a little bit different from the UNIX one. Figure 11 shows how an ARP packet is processed in Linux. All the ARP packets received are processed by the function *arp_rcv() (linux/net/ipv4/arp.c)*.

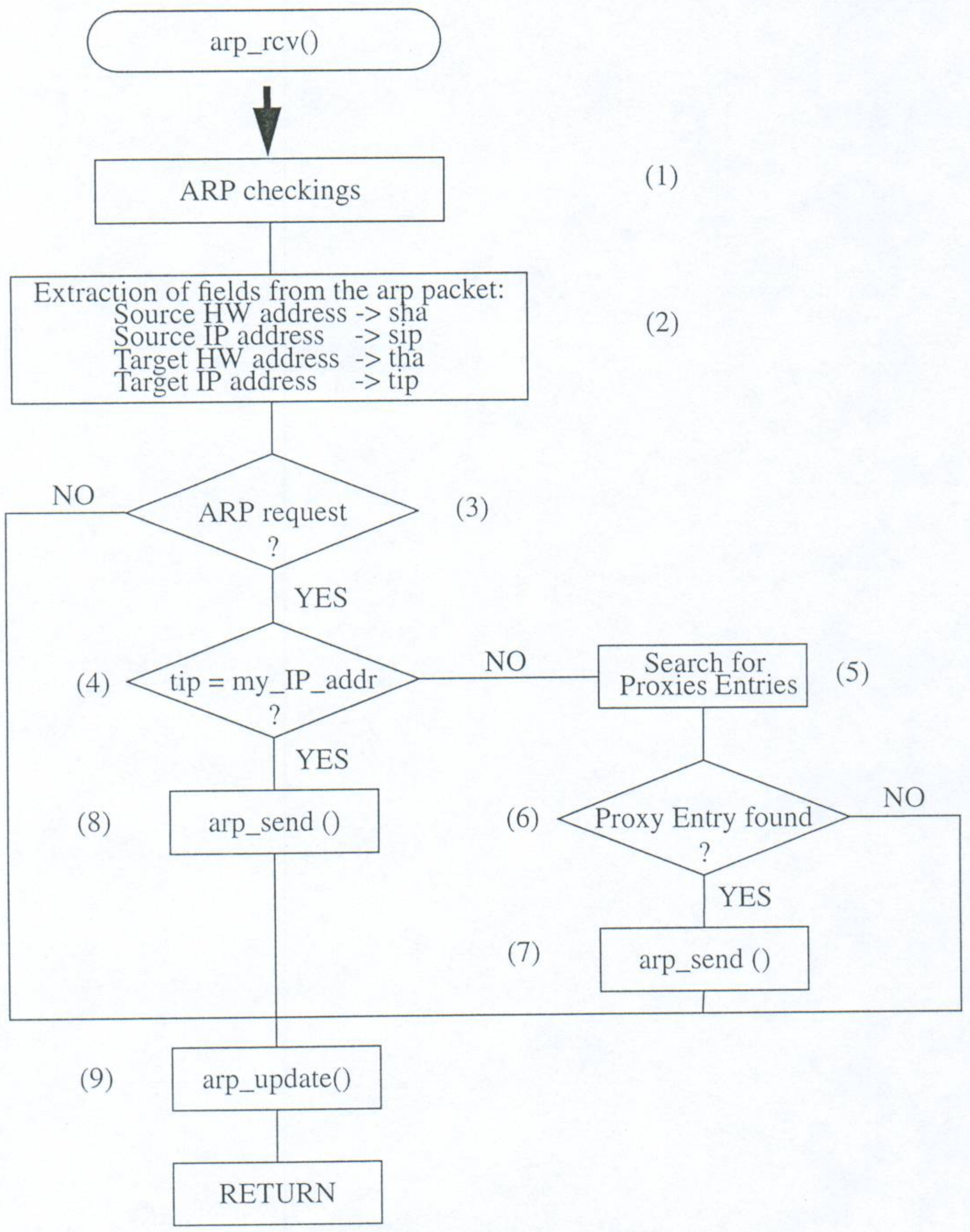


Figure 11 Flow Chart of arp_rcv() function.

Description :

When the arp_rcv() is called the fisrt things done are different tests (1) on the compatibility of protocols and on the various types of the packet. After, there is an extraction of the fields contained in the packets (2). Then, the function tests wheher it is an **[ARP-req]** or not (3). If it is an **[ARP-req]**, it has to check if the request is for us (i.e. target IP = my_IP_address) (4). If it is the case we have to send an **[ARP-rep]** to the requester(8), with our hardware address (i.e MAC address). If it is not the case we have to check if the request is regarding an IP address for which we have a Proxy ARP Entry(5), if it is the case we have to send an [ARP-rep] to the requester(7) containing our MAC address. Finally, we have to update the ARP cache: the function arp_update is called (9). With respect to the parameters given, the function will do the following:

- If the target IP address received matches one of the IP address of the host, the arp_update() function will either create a new entry or update the entry regarding the IP address of the sender(source IP of the ARP message).

- If the target IP address received does not match one of the IP address of the host, the arp_update function will be allowed to update an entry matching with the source IP address. But the function will not be allowed to create a new entry in that case.

5.4.2 Watmin Modifications in the ARP mechanism

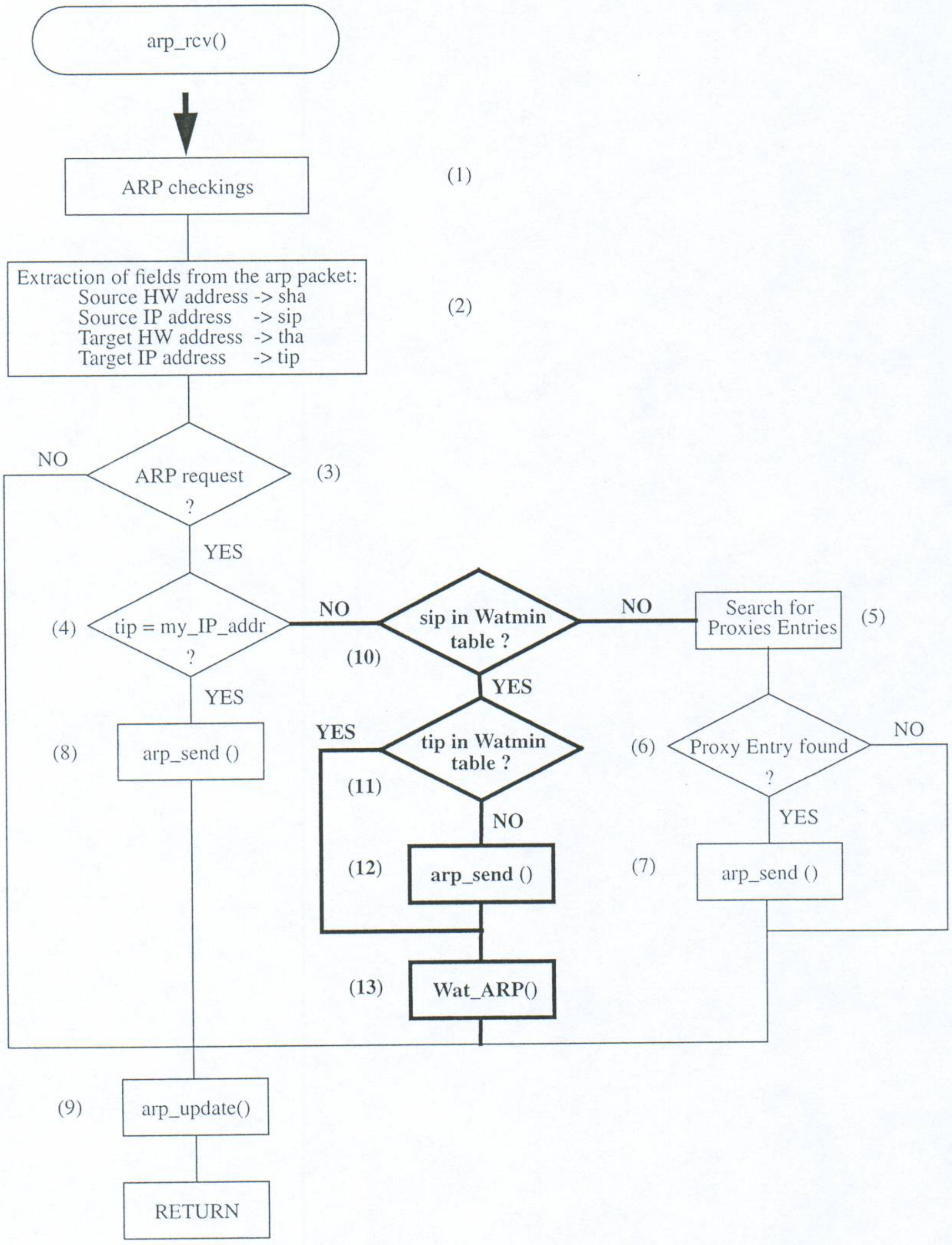


Figure 12 Flow Chart of arp_rcv() function Watmin Modified.

Description of the Watmin Modifications in the AP ARP mechanism:

In order to achieve the Watmin requirements, we had to make the modifications showed in bold in figure 12. If the **[ARP-req]** is not for the AP, we fisrt check if the requester is one of the MHs in the AP's cell overage, for that matter we search if the sip (source IP address) is present in the Watmin Association table. If it is not the case, it is the normal case where we look for proxy Entries. Otherwise we are in the Watmin exception case. If the target address is not in the Watmin Association table, we have to send a **[ARP-rep]** because the AP is acting as the forwarder for this type of destination. This **[ARP-rep]** will be exactly the same as if it was a request for us (same as (8)) excepted that the source IP field contains the target IP address of the **[ARP-req]** received. If the target is not present in the Watmin Association table we will not do anything because the MH ragarded will respond to that request as it is located in the same cell as the requester (we use a decentralized model). After this, we have to call the Watmin specific function called: Wat_arp(). The Wat_arp() function informs the AA_daemon that a MH has issued an [ARP-req] in the cell and therefore that it has to notify this fact to its neighbouring AP. The Wat_arp() function inform the AA_daemon by sending a message in the ARP_message queue.

5.5 Modification of the ATMARP mechanism in the AP

In this section, we will give you all the indications in order to implement the modifications required to register the IP address of the MH to the ATMARP server.

In the registration phase of a MH to an AP (section 3.0), we described the fact that the AP needs to register the MH to the ATMARP server. In our configuration, the AP will act as an ATM proxy for its MHs.

To achieve this registration phase, we propose to modify the ATMARP client (developed by W. Almersberger [x]). By opening a special socket SVC* (Figure 10,(e)) and using new command flags, the AA communicates to the ATMARP client that it intends to register a new MH. To implement the opening of the socket, we advise the reader to take the ideas from the ATMARP daemon (source /usr/src/atm/arpd/atmarpd.c and atm.c). New flags must be added to process this Watmin request (in file /usr/src/linux/include/linux/sockios.h). As we have done for the implementation of the system call regarding the Watmin Association table other modifications must be done in the kernel in order to process the new flags. Then, in the file /usr/src/linux/atm/atmarp.c you have to process this new flags, in fact you just have to do exactly the same as what is done for the other flags: it calls a function that send the request to the ATMARP daemon in the user space. Then in the daemon, you need to have another case in the switch() of the function arp_ioctl() to treat the case. What should be done is described thereafter.

The principle of the registration makes use of the selector field of the ATM address of the AP. Thus, it will associate an unused selector to the IP address of the MH (information stored in the WATMIN Association table), this step could be done when the flags is first treated in the kernel (file /usr/src/linux/atm/atmarp.c). Then the ATMARP daemons opens a new connection (with the selector field chosen) to the ATMARP server. The later when the connection is opened, will issue an InATM_request to the caller. On the reception of this type of message, the ATMARP client (AP) will systematically checks if the selector of the ATM address used for the connection is one associated with the IP address of an MH under its control. In this case, it will substitute in the InATMARP_reply (field source IP address) the IP address of the AP by the MH's one. Those modifications will be done in the function inarp_reply() (file /usr/src/atm/arpd/arp.c). Then the WATMIN Association entry used is validated and the registration of the MH to the ATMARP server is confirmed to the AA.

In the InATMARP_reply and the InATMARP_request sent by the AP, the source IP address will be systematically adapted according to the selector of the ATM address used by the connection. With that method, we think that the ATMARP client of the AP will be consistent with the information stored in the ATMARP server. To implement this new mechanism other modifications must be done in the function inarp_request() (file /usr/src/atm/arpd/arp.c).

At this step we think that we have found a way to register the MH to the ATMARP server without having to modify the ATMARP server. We hope that what we have proposed will work correctly.

Conclusion

This report has presented the current status of the WATMIN project. A MH, roaming in a cell coverage is able to communicate with an ATM host in the same IP subnetwork. It makes use of a light weight process in the user space of the MH and the implementation of new mechanisms in the AP. The implementation of the modifications required in the ATMARP client mechanism is still incomplete, but a description of what should be done is provided. Our implementation of the AP has enabled us to leave the MH's protocol stack unchanged.

Thus, the fisrt phase is almost complete! The design of the second phase (handover between cell in the same IP subnetwork) has been started but it is still incomplete.