Load Distribution In IEEE 802.11 Cells

Víctor Aleo

KTH, Royal Institute of Technology

Department of Microelectronics and Information Technology

Stockholm, March 2003

Master of Science Thesis
Performed at Center for Wireless Systems, KTH
In collaboration with Telia Research

Examiner: **Prof. Gunnar Karlsson**

Advisor: **Héctor Velayos**

Industrial advisor (Telia): **George Koudouridis**

Abstract

A key issue in Wireless LANs (WLANs) is the management of user congestion at popular zones called "hot-spots". At these sites, there are several access points (APs) with overlapped coverage and throughput is usually unevenly distributed among them. The reason is that the current IEEE 802.11 standard does not support a mechanism to distribute stations, thus they select APs based exclusively on the received signal quality. In addition, when the number of users per AP increases, the throughput per user decreases. As a result, the total network throughput is reduced producing under utilisation of the network resources.

Several approaches have been suggested to solve this problem. Some of them are based on the modification or enhancement of the MAC layer, hence changes to the physical layer are required. This would imply that all deployed stations should be changed. Other approaches are based on adding Quality of Service (QoS) support to the standard. These solutions require that stations and APs cooperate, which makes its deployment difficult in existing WLANs. Recently, some vendors of WLAN devices have incorporated load-balancing capabilities within their products. Nevertheless, they also require cooperation between stations and APs. Another limitation of these load-balancing schemes is that they simply balance the number of associated users across APs.

From the analysis of related work, we have identified in this thesis two groups of common issues that any load distribution scheme should deal with: architectural and algorithmic issues. Architectural issues deal with key points such as the cooperation between APs and stations, centralized versus distributed control or the most efficient load metrics to be used. Algorithmic issues refer to the four policies that a load distribution algorithm should include: *transfer*, which defines when an AP is suitable to participate in the load distribution; *selection*, which selects the user to transfer; *location*, which finds a suitable AP for the user and *information*, which specifies when, from where and what information is to be collected.

To address these issues, we propose and evaluate a new group of mechanisms, called Load Distribution System (LDS), the goal of which is to provide higher utilization of the overall network resources. This is achieved by means of dynamically transferring users among APs. We consider as a load metric the throughput per AP and not only the number of associated users per AP. Each AP determines whether the network is balanced or not, calculating the *balance index* (β). This index, bounded between 0 and 1, indicates any slight change in the load of the APs and quantifies the fairness of the network. The LDS runs at each AP in a distributed manner; it does not require the modification of the standard and it is transparent to stations. Furthermore, our proposed LDS can be applied to any type of IEEE 802.11 networks (a, b or g) since they share the same architecture and MAC protocol.

We evaluate the effectiveness of our LDS, building an experimental prototype. We perform three initial tests to set the necessary parameters of the LDS: the handover delay, the sampling time to monitor the traffic and the reactivity of the algorithms. After initial parameters are determined, we experimentally test the performance of the LDS. The results show that average packet delay per user can be decreased and total throughput in the network can be increased in comparison with a WLAN without our LDS. We also show that the LDS is stable and that it only transfers a station if this increases overall performance. Based on these results, we conclude that current WLANs will benefit from applying our LDS.

Acknowledgments

First and foremost I would like to thank my family, who always supported me along this work. I also would like to thank both, Telia Research AB and Center for Wireless Systems for providing the means to develop this thesis. My thanks also to my advisor, Héctor Velayos for his technical advices and valuable comments, and to my examiner Gunnar Karlsson for offering me the possibility to carry out this thesis at LCN. Finally, my thanks to my friends for being there when I need it.

Table of contents

List of Figures

List of Tables

1. Introduction

Wireless local area networks (WLANs) provide higher bandwidth than any other cellular technology. The most widely used WLAN standard (IEEE 802.11b[1]) [1] provides a maximum bit rate of 11 Mbps while wireless cellular networks, such as General Packet Radio Service (GPRS) offers a data rate up to 172 kbps and the third generation (3G) systems up to 2 Mbps.

However, there are still some problems with IEEE 802.11b such as radio interference from other devices and networks [2], and security concerns [3]. Furthermore, there are some key features that are not defined in the standard such as Quality of Service[2] (QoS) and Load Distribution (LD). The latter is the goal of this thesis. Because the IEEE 802.11 standard does not specify a mechanism to distribute traffic load, a mobile terminal typically selects the access point (AP) that provides the best radio signal quality when there are several available and this might not be the best option. The reason is that in currently deployed WLANs, the Distributed Coordination Function (DCF) is used as the mechanism to access the medium. It has been shown that the performance of this mechanism strongly depends on the number of competing users [4]. As a result, when the number of users competing for the channel increases, the throughput per user decreases resulting in a lower performance. Therefore, when a station selects the AP based only on the received signal quality and it discards a less loaded AP (e.g., in terms of throughput) it contributes to decrease the utilization of the network.

This problem is a challenge in areas with a high concentration of users called “hot-spots”, where user service demands are very dynamic in terms of both time delay and location [5]. At these areas, throughput distribution across APs is highly uneven and does not directly correlate with the number of users at each AP. Thus, load distribution solutions exclusively based on the number of associated users, such as call admission control in cellular networks [6], perform poorly [7]. Although different solutions have been proposed to address this problem [8, 9, 10, 11, 12, 13], none of them have considered designing a system transparent to the users and without modifying the standard. These two characteristics are essential because WLAN operators would be able to use in the best way the deployed resources.

In this thesis we design and experimentally test a new load distribution system (LDS) that distributes the total throughput in the network among APs with overlapped coverage. It is transparent to the users and it does not require any modification to the standard. The aim of this thesis is to investigate a new approach where each AP, in a distributed manner, transfers its users and rejects new ones if there is another AP less loaded. As a proof of concept, we implement our LDS in software in order to test it within an experimental WLAN prototype. The guarantee of any kind of service level to users is not considered in this thesis.

This thesis is organised as follows: we present an overview of the IEEE 802.11 standard, the related work and concerned issues with load distribution in section *2*. The proposed load distribution solution is explained in section *3*. The solution is implemented and experimentally tested in a prototype described in section *4*. The final results can be found in section *5*. The general conclusions are presented in section *6* and we provide some hints for future work in section *7*.

[1] We will use WLAN as synonym of IEEE 802.11b.

[2] There is an upcoming extension to the standard, 802.11e, which supports QoS [14].

2. Background

In this section, we summarize the IEEE 802.11 standard describing its essential features. We also identify issues that are common to load distribution systems. First, in subsection *2.1* we look at key points of the standard such as the association and handover procedures. In the second section, *2.2*, we present two main points: the concept of load balancing and related work that has been performed in this area applied to WLANs. Finally, in the third section, *2.3*, we describe two types of design issues, architectural and algorithmic, which form part of any load distribution scheme.

2.1. Overview of the IEEE 802.11 standard

2.1.1. Generalities

The standard IEEE 802.11, 1999 edition [1] is a part of a family of standards for local and metropolitan area networks. The scope of the standard is limited to Physical and Data Link layers (see Figure 1) as defined by the International Organization for Standardization (ISO) Open Systems Interconnection (OSI). The first version of the standard, 802.11, was approved in July 1997 and in September 1999 there were ready new extensions: 802.11b and 802.11a [15]. The extension 802.11b includes two new data rates, 5.5 Mbps and 11 Mbps for the 2.4 GHz band. The 802.11a extension operates in the 5 GHz frequency band and achieves a maximum data rate of 54 Mbps.

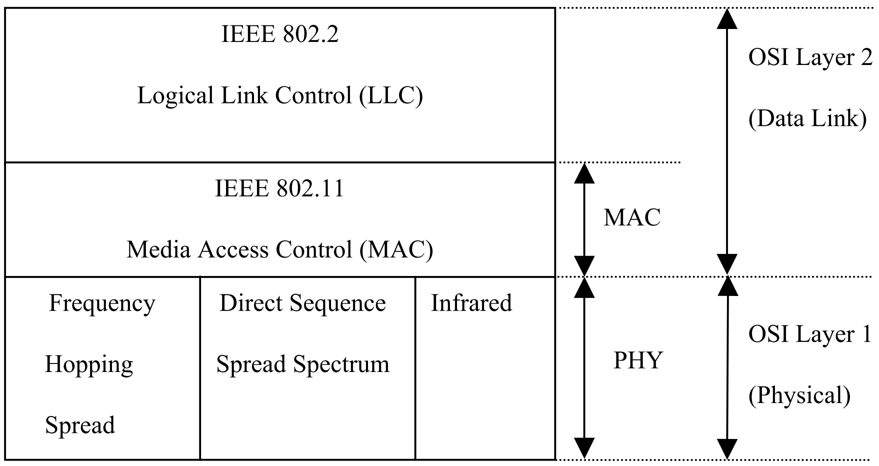


Figure 1: IEEE 802.11 standards mapped to the OSI reference model

The purpose of the standard is to provide wireless communications to fixed, portable and moving stations within a local area. It also defines standardized access to the unlicensed frequency band called instrument, scientific and medical (ISM) band in the range of 2.4 to 2.483 GHz. Each channel is 22 MHz wide so there are 14 channels in total, of which only 13 are available in Europe and 11 in USA. Thus, there are only 3 non-interference channels (1, 6 and 11) [2]. Specifically, the 802.11 standard addresses (source: [1], section "*1.2 Purpose*"):

- Functions required for an 802.11 compliant device to operate either in a peer-to-peer fashion or integrated with an existing wired LAN.
- Operation of the 802.11 devices within possibly overlapping 802.11 wireless LANs and the mobility of these devices between multiple wireless LANs.
- MAC procedures to support asynchronous MAC service data unit (MSDU) delivery services.

- Several physical layer signalling techniques and interfaces.
- Privacy and security of user data being transferred over the wireless media.

The standard specifies two mechanisms to access the medium, the *Distributed Coordination Function* (DCF) and the *Point Coordination Function* (PCF). DCF uses a Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA) and binary exponential back-off. On the other hand, PCF is a polling based media access that may be used to create a contention-free access method [1]. However, most common WLAN devices do not support PCF. As a result, the DCF is usually used as the access method.

WLANs typically cover small areas of a few hundred meters (typical indoor range of 802.11b is 30-46 m. at 11 Mbps, 40-46 m. at 5.5 Mbps and 76-106 m. at 2 Mbps [16]), whereas 3G networks support cell radius up to ten kilometres with reliable coverage [17]. Therefore, WLAN faces the future as a complementary option to 3G systems indoors and outdoors where the goal is to provide high bandwidth to end users instead of extensive coverage as cellular wireless networks (such as 3G) aim to.

One of the factors that have increased the popularity of WLAN today is the low cost of 802.11b equipment, much lower than 3G networks because of the simple architecture of the network. Furthermore, the competition among WLAN vendors and the operation in the 2.4 GHz ISM band have increased its usage. Additionally, the WECA[3] members have collaborated to encourage 802.11 interoperability and the consortium's "wireless fidelity" (Wi-Fi) certification program has been a key factor in the standard's widespread acceptance [16].

Typical deployments of WLAN include indoor and outdoor environments, such as airport and railway terminals, hotels, business parks, office buildings and university campus like the one located at the Royal Institute of Technology (KTH) at Kista[4]. This will not be an isolated example, due to the growth of mobile terminals (such as laptops and PDA devices). It is expected that by 2006, over 20 million people in Europe will use WLAN services in more than 90.000 confined "hot-spots" [18].

2.1.2. Network architecture

This subsection describes different WLAN architectures specified by the standard. The goal is to identify and describe our WLAN scenario, its main components and the relations between them.

The standard IEEE 802.11 defines two modes of operation: the *infrastructure* network and the *ad hoc* network or independent Basic service set (IBSS). The *ad hoc* network (see Figure 2) is the most basic WLAN topology composed by a set of stations, which have recognized each other and are connected via the wireless media in a peer-to-peer fashion.

In this thesis, the working WLAN scenario uses the *infrastructure* network mode that is a set of stations controlled by a single coordination point, called Access Point (AP). The area covered by an AP is called a Basic Service Set (BSS). The group of BSSs where APs communicate among themselves to forward traffic from one BSS to another is called an Extended Service Set (ESS) (see Figure 3). The AP provides a local relay function for the

[3] WECA stands for Wireless Ethernet Compatibility Alliance and includes a group of companies such as Cisco, 3Com, Enterasys, Lucent and many other wireless networking companies.

[4] See http://www.stockholmopen.net

BSS. All stations in the BSS communicate with the AP and no longer directly. In our scenario, several APs should have partial overlapped BSSs. This is a common configuration generally used to arrange contiguous coverage in a given area.

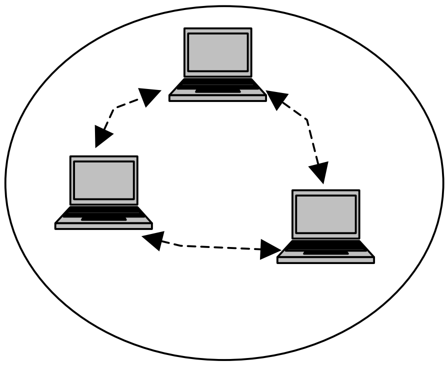


Figure 2: IBSS or *ad hoc* network

Another architectural component that appears in Figure 3 is the Distribution System (DS). It is used to interconnect multiple BSSs. In this way, an AP communicates with another AP to exchange frames for stations in their respective BSSs and forward frames to follow mobile stations as they move from one BSS to another. IEEE 802.11 does not specify on purpose the implementation of the DS to allow for the possibility that the DS may not be identical to an existing wired LAN. In fact, the DS may be created from many different technologies and it is not constrained to be either data link or network layer based. In the same way, the IEEE 802.11 does not constrain a DS to be either centralized or distributed. The cost of this freedom to implement the DS is that different vendors of APs are unlikely to interoperate across a DS [19]. In the scope of this project, it is assumed that there is a DS that provides mechanisms to perform the communication between APs. The type of the DS does not affect the design of the distribution mechanisms and consequently it is independent from it. In Table 1 there is a summary of four possible DS implementations [20]. According to the conclusions of the article [20], the best option to implement a DS with an Ethernet backbone (as it would be in our case) is using the MAC layer addressing with combined APs and portals (option 2 in Table 1).

Finally, the last logical architectural component that appears in Figure 3 is the *portal*. A portal provides logical integration between the IEEE 802.11 architecture and existing wired LANs (such as 802.xLAN). It is possible for one device to offer both, the functions of an AP and a portal. The portal connects between the DS and the LAN that is to be integrated. All the data from non-WLANs enters the IEEE 802.11 network via the portal. The network protocol stacks (physical and link layers) of the components of our WLAN scenario are shown in Figure 4.

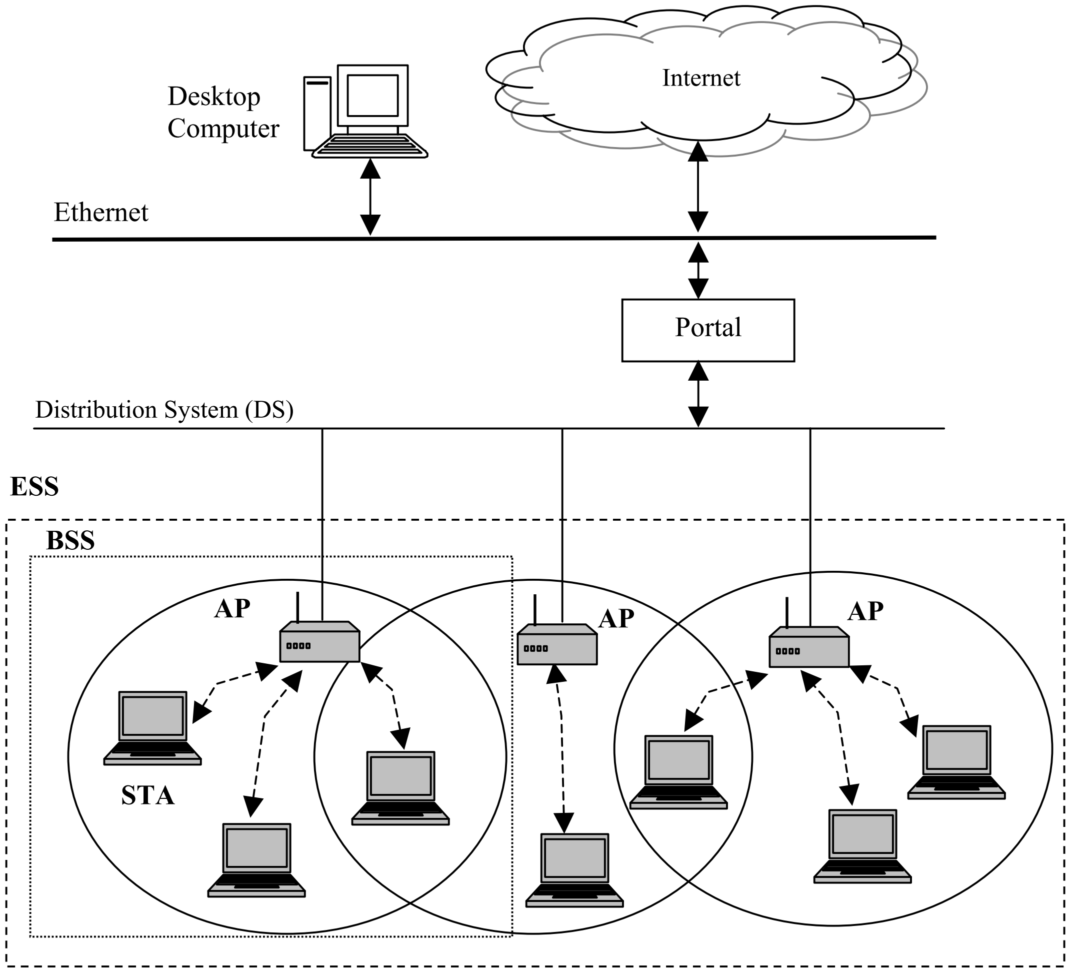


Figure 3: Our IEEE 802.11b network scenario using the infrastructure mode

Distribution System option	Description
1. MAC layer addressing (Separated DS and wired LAN)	This option selects Ethernet to implement the DS. The DS is a broadcast medium where every AP and portal can receive every message.
2. MAC layer addressing (Combined DS and wired LAN)	In this option, the portal function is included in each AP. This implies that the DS uses the same physical network as the wired LAN.
3. MAC layer addressing (MAC Bridge)	In this case, each AP is a filtering bridge between the BSS and the wired Ethernet LAN. This is a very simple solution and can only be implemented if we reduce the size of an MSDU in the 802.11 network to 1476 bytes.
4. Network layer addressing	It uses network layer addressing within the DS that permits to run the location management and forwarding protocols over an IP network composed of several LANs interconnected by routers.

Table 1: Distribution System (DS) implementation options (source: [20])

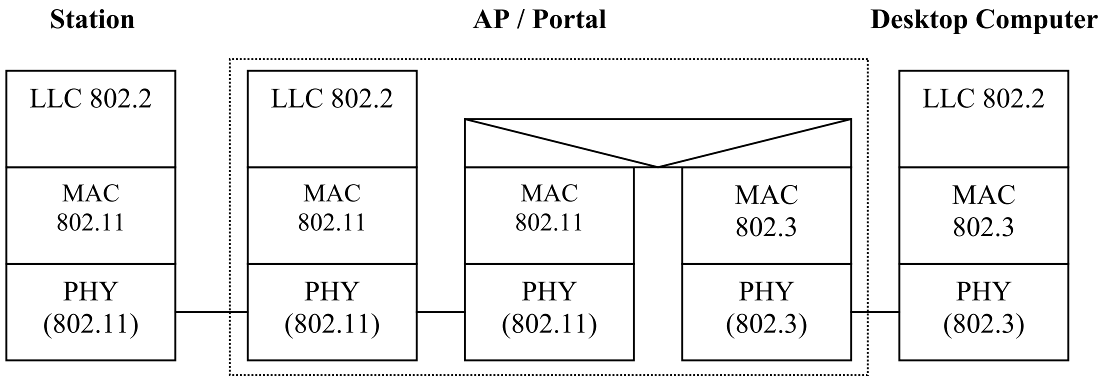


Figure 4: Protocol stack of the network components considered in our scenario

2.1.3. Association in WLAN

This subsection describes in detail the association procedure in WLAN. It is essential to understand this procedure because it specifies the way a station discovers APs in range and the criteria for selecting a particular AP when several are available.

A station must be associated with an AP in order to send or receive data frames[5]. The association procedure is always initiated by the station (mobile-controlled handover) and a station can only be associated with one AP. In the considered scenario, (see Figure 3) when a station powers on, it must discover which APs are present and then requests to establish an association with a particular AP. Thus, first the station initiates a *scanning* process that can be either active or passive:

1. *Passive scanning*: in this case the station waits to receive a beacon frame from the AP. The beacon frame is a frame sent by the AP periodically (with a typical period of 100 ms) with synchronization information. The beacon contains information corresponding to the BSS such as ESS ID, beacon interval, capabilities and traffic indication map (TIM).

2. *Active scanning*: the station tries to find an AP by transmitting Probe Request Frames, and waiting for Probe Response from the APs.

These two methods are valid, and either one can be chosen according to the power consumption/performance trade-off. Once the *scanning* process has finished, the station has an updated list of APs in range. This information is used by the station to associate with the AP that provided a higher Signal-to-Noise Ratio (SNR).

At this point, the station sends an *Authentication Request* to the selected AP (assuming that the default association method, *Open System Authentication*, is used). Upon the reception of this notification, the AP answers sending an *Authentication Response* to the station. If the status value of this response is “successful”, the station is now authenticated with the AP and sends an *Association Request* message to it. Upon the reception of this message, the AP sends an *Association Response* to the station. If this second response was also successful (the response could be negative if, for example, the particular MAC address of that station was not allowed to communicate through that AP), the station is authenticated and associated with the AP.

[5] Only data frames with frame control (FC) bits “To DS” and “From DS” both false can be send when a station is unauthenticated and unassociated (source: [1], “*5.5 Relationships between services*” section).

2.1.4. Handover in WLAN

This subsection describes the handover procedure in WLAN. This is an important issue because load distribution mechanisms may disassociate a station to distribute the load. Therefore, it is important to understand the way a station reassociates with a new AP.

The 802.11 standard specifies the handover procedure as follows. When the SNR becomes lower than a certain threshold, the station starts to search for new neighbouring APs in range triggering the scanning process. In this process, called *Reassociation*, the station transmits a *Reassociation Request* to the selected AP. If the station receives a *Reassociation Response* with a successful status value from the AP, then the station is now associated with the new AP. According to the standard (see [1], "*11.3.2 AP association procedures*" section), the AP shall inform the DS of this new association sending a *reassociation notification*. The station always initiates the *Reassociation* process. As an indication, the layer-2 handover delay has been measured in [21]. The results show that the handover incurred an additional peak delay of 157 ms.

2.1.5. Management in WLAN

Any load distribution scheme needs to gather information about the state of the network (number of stations associated with an AP, signal strength of a link, etc.) and to set specific parameters to perform a particular action (i.e., control the power management, disassociate a station, etc.). In this subsection, we present the management capabilities that the standard provides because it can be a useful mechanism for load distribution mechanisms.

The IEEE 802.11 standard specifies two management entities, included in the MAC and physical layers, called MAC sub layer management entity (MLME) and PHY layer management (PLME) entity. These entities provide the layer management service interfaces through which layer management functions may be invoked (see Figure 5). Another management entity is the Station Management Entity (SME) that is a layer independent entity. Its functions are not specified in the standard but they would be gathering layer-dependent status from the various layer management entities and setting the value of layer-specific parameters. The standard also defines the interactions within these entities via a Service Access Point (SAP), across which the defined primitives are exchanged.

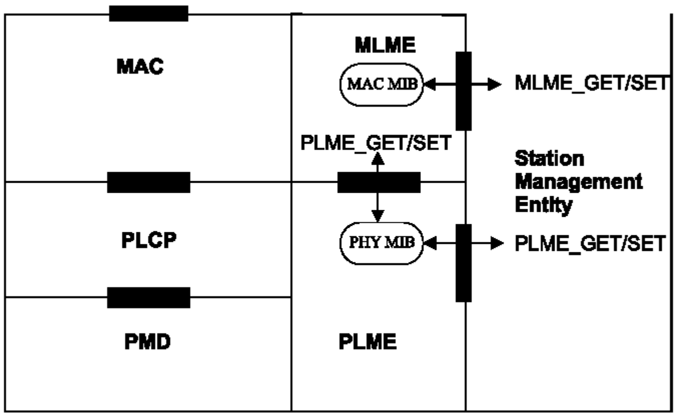


Figure 5: Relationship among management entities (source: [1])

The management information specific to each layer is represented in a management information base (MIB) for that layer. Both MLME and PLME contain the MIB for the corresponding layer. The SAP user-entity can either GET the value of a MIB attribute, or SET the value of a MIB attribute. These services provided by the MLME to the SME (MLME SAP interface) are described in abstract way and do not imply any particular implementation or exposed interface. The services are: power management, scan to determine the characteristics of the available BSSs, synchronization, authentication, association, reassociation, disassociation, reset, and start (to create a new BSS).

The standard offers a management possibility based on the Simple Network Management Protocol (SNMP) [22]. Since it was developed in 1988, SNMP has become "de facto" standard for network management. The use of SNMP to access the MIBs specified in the standard (Annex D) has been used previously in [23] and it is extensively analysed in [24]. The 802.11 MIB has a tree structure and it is expressed in Abstract Syntax Notation 1 (ASN.1). The root is: .iso.member-body.us.ieee802dot11 (.1.2.840.10036). Four main branches compose the MIB: Station Management (SMT) attributes, MAC attributes, Resource type ID and PHY attributes. The SMT is the term used to describe the global configuration parameters that are not part of the MAC itself. Figure 6 shows the six sub-trees that form the SMT.

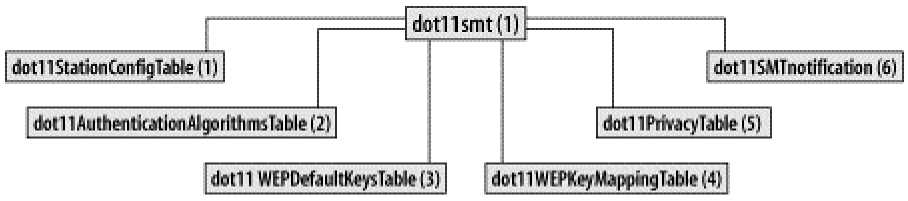

dot11smt (1)
dot11StationConfigTable (1)
dot11AuthenticationAlgorithmsTable (2)
dot11 WEPDefaultKeysTable (3)
dot11WEPKeyMappingTable (4)
dot11PrivacyTable (5)
dot11SMTnotification (6)

Figure 6: Main branches of the station management tree (SMT) (source: "802.11® Wireless Networks: The Definitive Guide")

2.2. Load balancing solutions

Load balancing algorithms enter into play when overlapped coverage areas of different APs exist and stations can attach to more than one AP (see Figure 7). This problem has been previously studied for cellular networks that are based on a fixed channel assignment [6]. In these systems, whenever a station can attach to more than one base station (BS), the purpose is to direct the new call to the BS with the greatest number of available channels. It has been proved that this idea reduces the probability to block future incoming calls (newly generated or from a handover) because of lack of channels. One common technique to implement this concept is call admission control (CAC) [6]. Where some channels are reserved for handover calls.

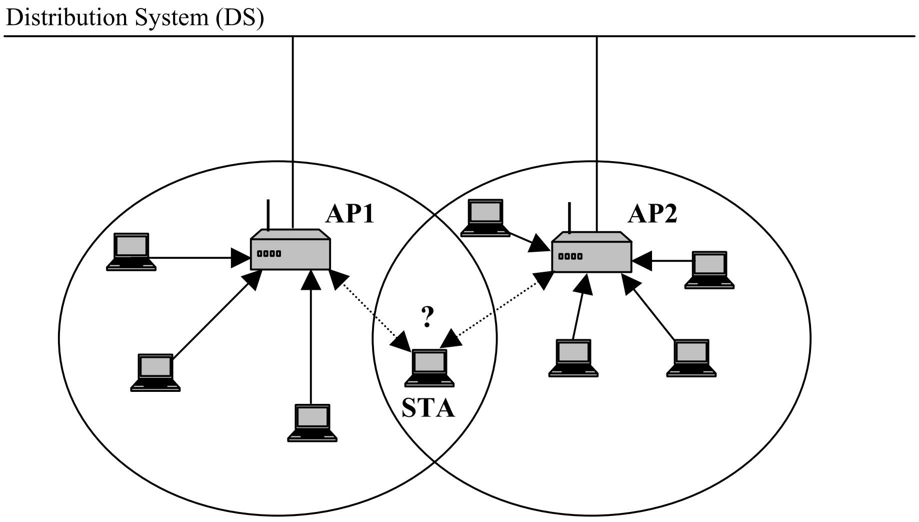


Figure 7: Load distribution problem

When the term load balancing is used, the load refers to the number of active calls per cell and balancing to the mechanisms that tend to assign the same number of active calls per cell. In this thesis, we deal with wireless packet networks such as WLAN. Therefore, the concept of load balancing as defined for cellular networks is not appropriate because in WLAN the load is not only related with the number of active calls per cell. As an experimental study of WLANs concludes [7], load-balancing algorithms that attempt to balance AP load according to the number of users alone can perform poorly. This study also states that these algorithms would benefit of considering balancing the users across APs according to their actual bandwidth requirements. Therefore, the load is also related with the "packet" level information, such as the retransmission error probabilities, bandwidth that every station is using at a specific moment, etc.

The concept of using packet level information in WLAN was published in [8] where it compares two different design criteria to implement a load-balancing algorithm. The next example will illustrate these two criteria. In Figure 7, a station placed in an area covered by two APs must choose one to associate with considering two possible types of information:

1. **Call level information**: the algorithm would only take into account the currently associated stations. Therefore, it will decide to associate with AP1 because there is one less station associated than in AP2. However, although now the cells achieve a balanced situation (the same number of associations), this could lead to an inefficiency situation. Since the stations in AP1 are placed at more distance than those associated with AP2, they will likely suffer a worse channel conditions and consequently a greater packet error probability. This will generate extra load generated by the packet retransmission and a degradation of the link performance for the attached stations. Furthermore, since the amount of traffic generated per station

is unknown and could be higher in AP1 than in AP2, the association with AP1 could decrease more throughput per user than if the association was with AP2.

2. **Packet level information**: the algorithm may assign the station to AP2 if this results in a better performance (for example, in terms of throughput per user, average delay, etc.), despite the fact that this would generate a load unbalancing at the call level. In [8], this decision is done by the algorithm taking into account two novel quality metrics that allow the station to select the less loaded AP at the packet level. The first metric is based on the computation of the average number of packet transmissions within a cell. The second metric attempts to directly estimate the packet loss performance, which in turns represents an indirect measure of the packet load. The station selects the AP as the one that minimizes the selected metric (there is no combination), computed including the contribution of the incoming station. This novel approach is compared against traditional schemes such as Minimum Distance (MD) and Nominal Load (NL). In MD, the AP selected is the closest one (no load balancing is implemented) while in Nominal Load, the one that accommodates the lowest number of connections is selected among the APs that can admit the user.

The simulated results from [8] show that the packet level approach has been proved superior to traditional load balancing schemes. Therefore, load distribution algorithms that use packet level information perform better than those that only use call level in WLANs.

A recent article [9] also addresses the load balancing issue in WLANs taking into account packet level information. The authors propose that both, the network and its users should explicitly and cooperatively adapt themselves to changing load conditions depending on their geographic location within the network. The simulated results show that the algorithms improve the degree of load balance in the system by over 30%. In order to achieve this performance, two methods are used to balance the load: *Explicit Channel Switching* and *Network-Directed Roaming*. Explicit channel switching is used when the network can distribute the load (according to the user requirements) among neighbouring cells. In this case, the algorithm trades off signal strength with load by forcing the user to switch from an overloaded cell containing the AP with a stronger signal to a neighbouring lightly loaded cell where the signal to the AP may possibly be weaker. Network-Directed roaming is used when the neighbouring APs cannot handle user admission request using explicit channel switching. In this case, the network can instead provide feedback suggesting potential locations to which users can roam to get the desired level of service. Network-Directed roaming strongly depends upon the ability of the network to determine a user's location and the ability to direct the user to locations with available capacity.

Another approach to load balancing in WLAN has been done from the point of view of QoS and mobility [10, 11]. In this case, load balancing acts as a mechanism to provide appropriate QoS in WLAN. According to [10], there are three facts that must be taken into account in order to provide QoS mechanisms with mobility support:

1. The number of stations allowed to use the channel must be limited: because the available bandwidth of the WLAN link depends strongly on the number of active stations and their traffic.

2. The geographical area in which stations communicate should be limited so that all stations use the same high bit rate: the reason is that most popular WLAN products degrade the bit rate when repeated frame drops are detected (due to signal fading, interference, etc.). However, as the channel access probability is equal for all stations, stations that send at low rates penalize stations that use high rates.

3. Traffic sources should be constrained by configuring traffic shapers in stations to obtain desired QoS effects.

In [11] the problem of load balancing is considered to achieve service differentiation in WLANs. The scenario considered in this paper is similar to ours (several APs in a multicell environment), and the mechanism to distribute the load is based on a distributed admission control algorithm. The novel approaches in this paper are the Virtual MAC (VMAC) and Virtual Source (VS) algorithms and a modification of the MAC layer. The VMAC passively monitors the radio channel and estimates locally achievable service levels, obtaining MAC level statistics related to service quality such as delay, delay variation, packet collision and packet loss. The VS utilizes the VMAC to estimate application-level service quality. These algorithms are running in all APs independently and continuously monitor the radio channel. Two types of traffic are considered (TCP and voice traffic), but admission control is only applied to delay sensitive voice sessions. More precisely, when the estimated delay exceeds 10 ms, new voice sessions were rejected from the service. There was no admission control applied to Web traffic. The results show that this developed system can maintain a globally stable state in WLANs even if cell areas overlap and the radio channel is shared.

Finally, various vendors of WLAN devices have implemented their own load balancing solutions [12, 13]. This is the case of Cisco Aironet 350 AP series where the APs include its load on the beacons and probe responses that are broadcasted in the cell. In this way, the stations receive this information from the APs in range and associate with the least loaded AP. On the other hand, the Proxim ORiNOCO AP-1000 series include a load balancing mechanism based in evenly distributing the stations over available APs.

2.3. Load distribution design issues

After the review of related work, we classify the issues that any load distribution solution should deal with into two different groups: *architectural* and *algorithmic issues*. By *architectural issues*, we understand these topics related with the load distribution architecture such as the type of control (centralized versus distributed) or suitable load metrics. *Algorithmic issues* deal with points specifically related to algorithm behaviour (transfer, selection, location and information policies).

2.3.1. Architectural issues

In this subsection, we describe six architectural issues. For each one, several design alternatives are presented. We analyse each alternative pointing out its advantages and drawbacks. The six issues are:

1. Entities participating in the load distribution

2. Load distribution control

3. Load metric

4. Network traffic flows

5. Load distribution scope

6. Mechanisms to force a handover by AP

2.3.1.1. Entities participating in the load distribution

There are two entities in WLAN that can cooperate or not to distribute the load: the stations and the APs. Thus, there are two possible options: *no cooperation between APs and stations* and *cooperation between APs and stations*.

1. **No cooperation between APs and stations**: in this case, the APs (with either centralized or distributed architecture) take all the decisions regarding load distribution. The main advantage is that load distribution decisions are transparent to the stations, which ease the deployment of the solution in existing WLANs.

2. **Cooperation between APs and stations**: in this case, the stations may negotiate some quality parameters with the APs (such as desired bandwidth) and therefore explicitly cooperate with them to perform load distribution. Typically, the stations request service from the APs in an overloaded region and the APs try to adapt themselves to handle the station service request by readjusting the load across the network [9]. As a drawback, this option is not transparent to the stations.

2.3.1.2. Load distribution control

There are two types of load distribution control that can be selected: *centralized* or *distributed*.

1. **Centralized**: by centralized control, we mean that load distribution mechanisms run at a single node or entity within the WLAN. As a main advantage, it does not require any modification to the station or to the AP. As an example, in [23], the architecture and components of a Wireless Access Server (WAS) are described to achieve QoS and location based access control in WLANs. The WAS is a centralized entity that consists of two components: 1) "Wireless Gateway" (WG), which sits between the wired and the wireless network, and 2) "Gateway Controller" (GC) that can reside anywhere on the wired network. The WG acts like a bridge with filtering capabilities at the IP and TCP/UDP layers and the GC is responsible for controlling the behaviour of the WG. This WAS is a centralized option that solves the inter-operability with multi-vendor APs and it does not require any changes neither to the stations nor to the APs software or hardware. Although the first results shown are preliminary, (and more experimentation is needed) the authors stated that the performance is satisfactory. On the other hand, selecting a centralized control implies introducing a new architectural component, which is not defined by the standard and decreases the scalability of the system. Furthermore, centralized controls are less reliable since the failure of a central component may cause the entire system to fail.

2. **Distributed**: by distributed control, we mean that the load distribution mechanisms are running at each AP in a distributed way. This means that each AP takes its own distribution decisions based on the information provided by its own state and the state of the other APs. There are several advantages using distributed controls. First, it is tolerant to failures. Second, it is easier to implement because it does not require defining a new entity as the centralized case. Among the drawbacks, we can mention that a distributed control limits the ease of deployment, since each involved node has to support the load distribution mechanisms. Moreover, it requires coordination between APs. For instance, APs have to communicate between them in order to exchange load metrics.

2.3.1.3. Load metric

A key issue in the design of load distributing algorithms is identifying a suitable load metric. Therefore, it is necessary to define what we understand by "load" in WLANs. The definition of load can vary for different type of wireless technologies, such as cellular telephony and WLANs. For instance, we can define load in cellular telephone networks as the number of active calls per cell. However, we have argued in this thesis that taking into account packet level information (such as throughput per AP, packet error probability, etc.) is necessary because it leads to a better performance. We present in this subsection load metrics, some of them related with packet level information, that we have found in the literature.

1. **Gross Load (GL)**: it defines load as the number of stations per AP and the retransmission probability (based on the physical position of the station obtained from the SNR of the link) [8]. GL considers packet load information but since the retransmission probability is computed from the station side, it is the station that chooses the "best" AP. Thus, it requires the modification of the station side.

2. **Packet Loss (PL)**: the Packet Loss metric is motivated by the observation that the best possible load balancing metric is to select the target cell as the one that minimizes the expected packet loss percentage after the addition of a new station [8]. The main advantage is that it considers packet load information, but PL is based on the Gross Load metric, so it is also computed by the station side.

3. **Traffic (bytes/second) coursed per AP**: it is a quantitative measure of the total traffic cursed at the AP. The AP can compute it and therefore it does not need cooperation with the stations. Moreover, it considers packet load information.

4. **Number of associated stations (N)**: it only takes into account the number of associated stations with the AP. Thus, exploits the fact that when the number of stations associated with an AP increases, the throughput per station decreases. On the other hand, it does not take into account whether the stations are "competing" for the channel. Moreover, it does not take into account traffic load. Therefore, it should be used jointly with another load metric (such as *Traffic coursed per AP*) to take into account the traffic load per AP.

5. **Number of competing stations (n)**: this metric takes into account only competing stations, i.e., the ones that are actually in the process of transmitting packets, number that can defer from the number of associated stations [4]. This information cannot be retrieved directly from the protocol operation and the AP only knows the number of associated stations (N). The estimation is based on a numerically accurate closed form expression that relates n with the probability of a collision seen by a packet being transmitted on the channel by a selected station. By independently monitoring the transmissions eventually occurring within each slot-time, each station is in the condition to estimate n. The simulations have been applied to two different network conditions: 1) saturated and 2) non-saturated. In 1), where all stations are assumed to always have a packet to transmit in their transmission buffer, the numerical results show that the proposed estimation technique is devised. In 2), a more realistic scenario is simulated where the packets arrive to each station according to a Poisson process. In this case, the estimated number of competing stations shows large and fast fluctuations but now the estimation target becomes the average number of competing stations (rather than the total number of stations in saturation conditions). This proposed model has two main characteristics: it allows computing the load metric (number of competing stations) from the AP side and the time response depends on the number of competing stations (for instance, if such a number is lower than 10 stations few milliseconds are sufficient to guarantee numerical convergence). On the other hand, it does not take into account traffic load. Therefore, it should be used jointly with another load metric (such as *Traffic coursed per AP*) to take into account the traffic load per AP.

2.3.1.4. Network traffic flows

In order to measure load metrics related with the traffic, it is necessary to consider the direction of the flows. There are three different options for measuring the flow direction of the traffic: 1) uplink (from the station to the AP), 2) downlink (from the AP to the station), 3) both (uplink and downlink). It is important, in order to decide which is the best option, to note some observations about traffic types: TCP involves the bulk of non-real time user traffic (Telnet, Email, HTTP, etc.) while UDP traffic only constitutes a small fraction of the total traffic (DNS queries, SNMP traffic, etc.) [25]. Until a couple of years ago, the bulk of non-real time traffic was from servers (such as WWW or FTP) to clients. Thus, the air-link was utilized mostly in the direction "from the server to the client" (option 2, downlink).

However, this appreciation does not consider that the current panorama has changed by new popular Peer-to-peer (P2P) programs such as Napster, Gnutella or FreeNet. Compared to the traditional client-server model, in P2P applications files are served in a distributed manner and replicated among the network on demand. With the wide deployment of P2P applications, the P2P traffic is becoming a growing portion of the Internet traffic [26, 27]. Moreover, a study of public WLANs [5] shows that while downlink traffic dominates over uplink, the opposite tends to be true during periods of peak throughput.

2.3.1.5. Load distribution scope

One important issue is to find out which is the scope of application of our load distribution scheme within an ESS. There are two basic scopes a load distribution scheme may consider: 1) *wide scope*, which takes into account all the APs in the WLAN and 2) *local scope*, which only takes into account APs with overlapped coverage areas. The main difficulty with the wide scope option is that typically not all APs in the ESS have overlapped coverage areas. This means that a station located at a point where only can hear one AP should not be transferred (otherwise it will not be able to reassociate with another AP). Therefore, it is necessary for the APs to detect if a selected station to transfer can hear at least another AP. We propose three different mechanisms to detect this situation: *SNMP polling*, *Pre-authentication recommendation* and *Active scanning*. On the other hand, this problem is overcome limiting the scope of the load distribution to only those APs with overlapped coverage areas (*local scope*).

1. **Mechanism based on SNMP polling**: the AP polls (by means of SNMP) the selected station to request information about the APs that the station can hear. The main disadvantage of this mechanism is that requires cooperation between the station and the AP. Moreover, it increases the load on the radio side (due to the SNMP traffic).

2. **Mechanism based on pre-authentication recommendation**: in this case, the stations follow the pre-authentication recommendation, described in the standard ([1], subsection *5.4.3.1.1*), that recommends stations to pre-authenticate with all the APs in range to reduce the handover time. Thus, if every AP broadcasts the received successful authentications to the DS, the other APs can store the places where the stations are authenticated. As an advantage compared with *SNMP polling*, it reduces the network overload on the radio side because the information is obtained through the DS. On the other hand, the stations must feature pre-authentication and this is only a recommendation. Therefore, it may happen that not all the stations have this option implemented.

3. **Mechanism based on active scanning**: using this mechanism, the stations have to use active scanning. The active scanning procedure specifies that each station scans the channels according to its ChannelList. For each channel, the station broadcasts a Probe request frame. APs can store the IEEE MAC of the station that has sent this Probe frame and therefore exchange this information to find out if a station can be transferred. For example, consider 2 APs and 1 station using active scanning placed at an overlapped coverage area. Both APs will receive the Probe frame from the station and both will answer with a Probe response frame. Then, AP1 and AP2 will store the address of this station. Let's say that the station associates with AP1. Now,

if AP1 is overloaded it will check if the station can be transferred. To check this, it will query AP2 to find out if it received a Probe frame from this station in the past. If positive, AP1 will disassociate the station, otherwise not. As and advantage, no modification of the stations is required. However, the stations have to use active scanning. A drawback is that the station does not scan actively (sending Probe requests) constantly but only when it switches on or when it performs a reassociation. Thus, if the transmission conditions on the radio side change it may happen that the APs in range vary and invalidate this mechanism.

2.3.1.6. Mechanisms to force a handover by AP

In order to distribute the load, APs need a way to disassociate the current associated stations. We describe three mechanisms to force a handover by the AP: *Disassociation notification*, *Power Control* and *Avoiding replying ACK frames*.

1. **Disassociation notification**: the AP sends a disassociation message (or notification) to the selected station. This message is defined by the standard, invoked whenever an existing association is to be terminated and cannot be refused by either party to the association. This mechanism does not suppose any implementation problem since all the certified Wi-Fi APs must be able to send disassociation notifications. Moreover, there is no possibility for the station to reject this notification, which implies that the handover is produced as soon as the station receives the message. As a drawback, APs have to use the radio air-link.

2. **Power control**: the AP can modify the transmitted power per packet to force a handover. With this method, the number of failure packets for the selected station will increment and the station will lower its nominal bit rate progressively. At the end, when the received quality of the radio signal will be below a threshold, the station will trigger the handover procedure. This method uses more effectively the radio air-link than 1) since the AP does not send special messages to force the handover. On the other hand, the response is slower because the handover is not effective until the dropped packets achieve a minimum number. Moreover, it may affect the performance of other stations because the selected station reduces its bit rate to the lowest one.

3. **Avoiding replying ACK frames**: the AP does not reply the incoming packets of the selected station with ACK frames. Thus, if the station does not receive an ACK within a specified ACK_Timeout it will reschedule the packet transmission according to the back-off rules. As it happens in 2), when the number of failure packets increases, the station will lower its bit rate progressively. At the end, it will reach a maximum limit and will trigger the handover procedure. This method has the same advantage than in 2), thus, it is not necessary to send a disassociation message from the AP to the station. On the other hand, it is also a slow process because the handover is not effective until the dropped packets achieve a minimum number. Moreover, it may affect the performance of other stations because the selected station reduces its bit rate to the lowest one.

2.3.2. Algorithmic issues

Load distribution algorithms have been extensively studied in the area of distributed computing [28, 29]. In this area, load distribution improves performance by transferring computer tasks from heavily loaded computers (called nodes), where service is poor, to lightly loaded computers. In this way, load distribution can minimize the average response time of tasks. Although it is not the same to distribute computer tasks than to distribute stations, the load distribution algorithm has the same components. In this subsection, we describe its components as well as some design trade-offs. In particular, we describe two different initiation types (sender-initiated and receiver-initiated) and four main components of a load distribution algorithm: a transfer policy, a selection policy, a location policy and an information policy.

2.3.2.1. Algorithm initiation types

Typically, it is possible to classify load distribution algorithms by its initiation methods [29, 30]. There are two common initiation methods: *Sender-initiated* and *Receiver-initiated*.

1. **Sender-initiated**: under sender-initiated algorithms, load distribution activity is initiated by an overloaded node (sender) trying to send a task to an under loaded node (receiver). While distributing computer tasks in this way does not suppose a problem, in WLANs is not possible to "send" a station to a particular destination AP (receiver) because is the station that selects the destination AP.

2. **Receiver-initiated**: in receiver-initiated algorithms, load distributing activity is initiated from an under loaded node (receiver) which tries to get a task from an overloaded node (sender). In our case, this algorithm has the same problem than the *Sender-initiated* since it is not possible to assign a station to a selected destination AP. Moreover, it is more complex and slower than the first option because the receiver AP needs to communicate with the sender in order to decide to transfer a station.

2.3.2.2. Transfer policy

A transfer policy determines whether a node is in a suitable state to participate in a task transfer, either as a *sender* or as a *receiver*. There are two groups of policies: *Threshold* and *Relative transfer policies* [28].

1. **Threshold policy**: threshold policy decides that an AP is a sender if the load at that AP exceeds a threshold T1. If the load falls below T2, the transfer policy decides that the AP can be a receiver. This policy will only work if the load follows a static pattern and it can be bounded.

2. **Relative transfer policy**: in this case, the load of AP is considered in relation to load of other APs. For instance, a relative policy might consider an AP to be a suitable *receiver* if its load is lower than that of some other APs by at least some fixed amount ρ.

2.3.2.3. Selection policy

The selection policy selects a station to transfer after the transfer policy has decided that an AP is a sender. We propose two selection policies: *Random selection* and *Best candidate*.

1. **Random selection**: the simplest approach is to select randomly a station that is associated with the AP. Although this policy is very simple and it does not require computing time for the algorithm, it may not achieve the equilibrium as fast as possible. The reason is that this policy does not take into account the traffic generated by the selected station.

2. **Best candidate**: we propose another selection policy where the goal is to select a station taking into account three traffic metrics: the traffic generated by the station, the own AP traffic and the average network traffic. First, the algorithm computes the difference between the traffic of the AP and the average network traffic. Then, the selected station is the one whose traffic is closer to that difference. In this simple way, the number of decisions to distribute the load is reduced.

2.3.2.4. Location policy

The location policy's responsibility is to find a suitable AP for a station, after the transfer policy has decided that the AP is a sender. We propose three different location policies:

1. **Polling**: an AP polls another to find out whether it is suitable for load sharing. APs can be polled either serially or in parallel. An AP can be selected for polling on a random basis, on the basis of the information collected during the previous polls, or on a nearest neighbour basis. The main drawback is that it requires coordination among the APs.

2. **Broadcast a query**: an alternative to polling is to broadcast a query seeking any AP available for load sharing. Although the coordination with this mechanism is lower than in the polling case, the APs have to communicate the queries.

3. **Receiver enforcement**: we propose a new policy where the sender APs do not accept new associations until they become receivers. Since in WLAN the station selects the AP, it will always reassociate with a receiver AP because the senders APs reject its association request. The main advantage of this policy is its simplicity: the APs do not have to communicate or coordinate between them in order to select a destination AP for the station.

2.3.2.5. Information policy

The information policy decides when information about other APs in the system is to be collected, from where it is to be collected, and what information is collected. There are three types of information policies: *Demand driven policies*, *Periodic policies* and *State change driven policies*.

1. **Demand driven policies**: with this distributed policy an AP collects the state of other APs only when it becomes either a sender or a receiver, making it suitable to initiate load sharing. This policy is inherently dynamic and its actions depend on the system state. Using a sender-initiated algorithm and selecting a demand driven policy implies that when an AP becomes a sender it starts to poll the receivers APs to get their load state. Therefore, the main drawback of this policy is that the sender AP cannot take a load distribution decision immediately because it needs time to find out the load state from the other APs.

2. **Periodic policies**: these policies collect information periodically and can be either centralized or distributed. Periodic information policies generally do not adapt their rate of activity to the system state. A drawback of periodic policies is the overhead due to periodic information collection that may increase the network load on the wired side.

3. **State change driven policies**: with these policies, APs propagate information about their states whenever their states change by a certain degree. A state change driven policy differs from a demand driven policy in that it propagates information about the state of an AP, rather than collecting information about other APs.

3. Load distribution system design

In this section, we describe our design of a Load Distribution System (LDS) for WLANs. First, we enumerate and describe the assumptions that affect the scope of this thesis in subsection *3.1.1*. Second, we make a design decision for each design issue we presented in section *2.* The decisions are made by weighting the advantages and drawbacks for both architectural and algorithmic issues (see subsections *3.1.2* and *3.1.3*). Finally, in subsection *3.1.4* a table summarizes these decisions. Once all design issues have been decided, we describe in detail our LDS in section *3.2*. Specifically, the architecture (subsection *3.2.1*) and its functionality (subsection *3.2.2*) are presented in this section.

3.1. Solutions to design issues

3.1.1. Assumptions

We enumerate in this subsection a list of assumptions that apply to this thesis. The aim of most of them is to reduce the deployment and implementation complexity.

1) The design will be limited to only one operator, thus the load distribution mechanisms can only be applied within APs of one ESS. The reason is that the standard does not specify an inter-AP communication protocol between different APs vendors[6].

2) The stations will not be modified because it is easier to deploy a load distribution system where only the APs are modified. In this way, load distribution is transparent to the stations.

3) The Distribution System (DS) is already implemented and provides the necessary mechanisms to enable the communication among the APs. Furthermore, the solution will be independent from the particular DS implementation.

4) The design of the load distribution will be valid for any IEEE 802.11 network (a, b or g) as well. The reason is that the different IEEE 802.11 standards mostly differ in the physical layer while the architecture is common.

3.1.2. Architectural issues

In this subsection, we evaluate the advantages and disadvantages for every architectural issue. Then, we chose a specific option for each issue:

1. **Entities participating in the load distribution**: we have chosen *No cooperation between APs and stations* since one of our assumptions is to avoid modifications in stations.

2. **Load distribution control**: we have chosen a *Distributed* control for three reasons: first, it follows the philosophy of the 802.11 standard. Second, it is tolerant to failures and it is scalable. Third, it eliminates previous configuration work. Therefore, load distribution activity will take place at the APs within the ESS.

3. **Load metric**: in order to choose the adequate load metric, it is necessary to decide about the goal of the load distribution. In our case, the distribution of the load dynamically transfers stations to improve overall network utilization. Thus, load distribution tends to increase the total throughput of the network. Load metrics that

[6] The IEEE 802.11 has defined a draft of an inter-AP protocol [19].

are only based on the number of stations, associated with the AP (N) or competing for the channel (n) [4], do not take into account this goal. On the other hand, metrics such as GL or PL [8] have to be computed at each station, which is not feasible given our assumptions. Therefore, we have selected *Traffic coursed per AP* (bytes/second) as the load metric because is directly related with the traffic at the APs. Moreover, this metric provides an indication of the current utilization of the network resources.

4. **Network traffic flows**: we have chosen *Measuring both uplink and downlink*. First, the bulk of the non-real time traffic is from servers (such as WWW or FTP) to clients (downlink traffic). Second, new services such as VoIP and P2P applications have grown and it is necessary to consider them as sources of traffic for uplink traffic. Third, a study [5] about network traffic in public WLANs shows that while downlink traffic dominates over uplink the opposite tends to be true during periods of peak throughput.

5. **Load distribution scope**: we have chosen *local scope*. First of all, it is not possible to employ the mechanism based on SNMP polling without modifying the station. The reason is that the results from the scanning process are only available at stations. Hence, it is necessary to modify the stations to communicate this information to APs. The second mechanism, based on pre-authentication, requires that all stations must follow a recommended advice, which is not supported by the majority of the current deployed Wi-Fi devices. The main drawback of the third mechanism is that the station does not use active scanning constantly. As a result, this mechanism can fail. Moreover, there may be some stations only using passive scanning instead of active so load distribution will not work for these stations.

6. **Mechanisms to force a handover by AP**: we have chosen *Disassociation notification* mechanism since is the method specified by the standard to terminate an existing association. The reason is that modifying the transmitted power or avoiding replying ACK frames will affect all the stations competing for the channel in the same cell. Moreover, the handover will be slower than sending the *Disassociation notification*.

3.1.3. Algorithmic issues

In this subsection, we evaluate the advantages and disadvantages for every algorithmic issue. Then, we chose a specific option for each issue:

1. **Algorithm initiation types**: since in WLANs is not possible to assure that a station will associate with a selected destination AP (receiver), both initiation methods are very similar. We have chosen *Sender-initiated* because it is easier to implement compared to the *Receiver-initiated*. Moreover, *Sender-initiated* type is faster: the AP that is overloaded initiates the load distribution activity without the need to communicate with another AP to execute this decision.

2. **Transfer policy**: we have chosen *Relative transfer* because the load of the AP (its traffic) is dynamic and is not predictable. Therefore, an AP is overloaded in relation with other APs and not in relation with static thresholds.

3. **Selection policy**: we have chosen *Best candidate* since it selects the station that will distribute more evenly the traffic among APs. On the other hand, *Random selection* does not take into account the traffic per station. Therefore, it does not tend to reduce the number of decisions to distribute the load.

4. **Location policy**: we have chosen *Receiver enforcement* because avoids communication among the APs. Therefore, it simplifies the load distribution mechanisms.

5. **Information policy**: we have chosen a *State change driven* because in this way all APs can take distribution decisions without the need to request load state information from other APs as it is done using *Demand driven* policies. On the other hand, the main disadvantage of *Periodic policies* is overhead due to periodic gathering of load metrics. Moreover, it is not necessary to periodically broadcast the load since load distribution is only needed whenever the state of the network changes.

3.1.4. Summary of proposed design issues

In Table 2, we summarize the selected decisions for architectural and algorithmic issues. These decisions conform the basis of our proposed design.

	Issue	**Decision**
Architectural design issues	Entities participating in load distribution	No cooperation between APs and stations
	Load distribution control	Distributed control
	Load metric	Traffic coursed by AP
	Network traffic flows	Both downlink and uplink
	Load distribution scope	Overlapped coverage areas (local scope)
	Mechanisms to force a handover by AP	Disassociation notification
Algorithmic design issues	Algorithm initiation types	Sender-initiated algorithm
	Transfer policy	Relative transfer
	Selection policy	Best candidate
	Location policy	Receiver enforcement
	Information policy	State change driven

Table 2: Final decisions to each issue

3.2. Solution description

3.2.1. Architectural description

The suggested load distribution solution is based on a distributed architecture where each AP has its own **Load Distribution System** (LDS) running locally. The LDS is the group of modules (load distribution algorithm, metric monitor, state information storage, etc.) which goal is to evenly distribute the traffic among APs that have overlapped coverage areas. Each module is in charge of performing a determined function (see Figure 8).

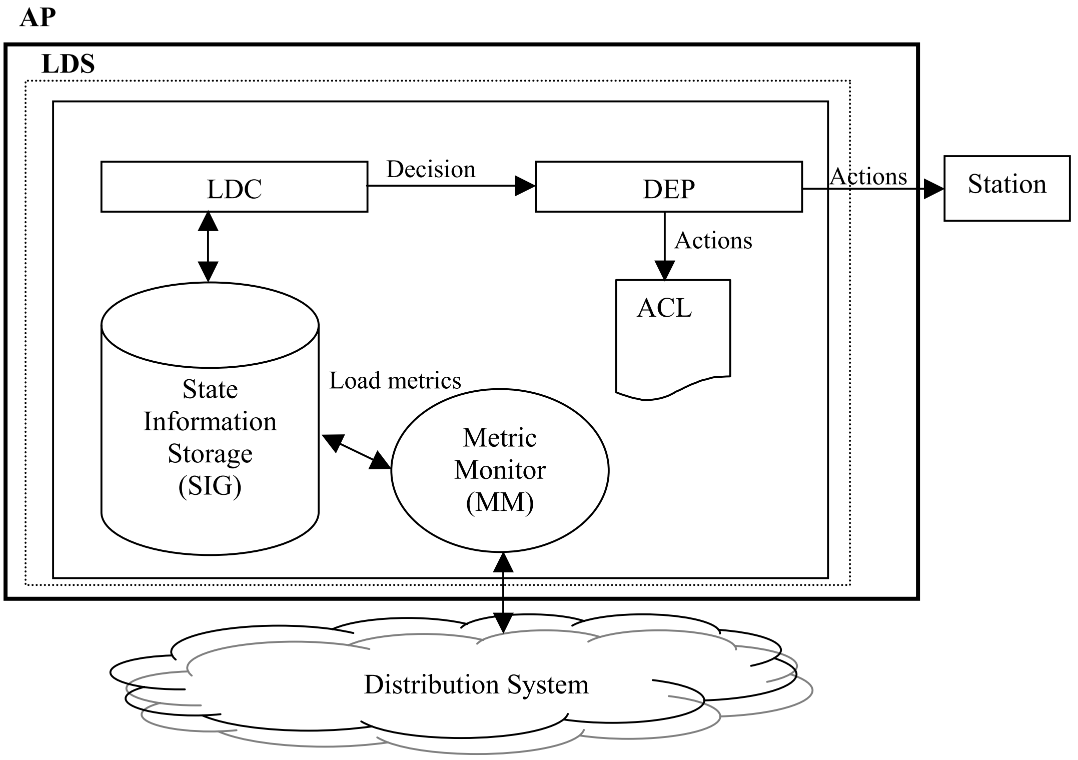


Figure 8: Load Distribution System components

We have divided the LDS into four modules: the *Load Distribution Controller* (LDC), the *Decision Enforcement Point* (DEP), the *Metric Monitor* (MM), and the *State Information Storage* (SIG). We briefly describe the meaning of each module:

1. **Load Distribution Controller (LDC)**: this is the most important module because it runs the load distribution algorithm (LDA). The algorithm takes all decisions related with load distribution based on load metrics that are obtained from the State Information Storage (SIG).

2. **Decision Enforcement Point (DEP)**: this module is in charge of converting the decisions from the LDC into actions. These actions manage the Access Control List (ACL), which is a MAC filter that allows or denies the association with the AP to the selected stations. It is also in charge of sending Disassociation notifications to transfer stations.

3. **Metric Monitor (MM)**: this module is in charge of monitoring the traffic of the AP and of getting necessary information from stations in order to obtain the load metrics, which are stored in the SIG. It is also in charge of broadcasting AP's traffic and of receiving the traffic from other APs.

4. **State Information Storage (SIG)**: it stores all information related with load metrics that the LDC needs in order to take distribution decisions.

In the next subsections, we describe the details of each module as well as relations that it maintains with others.

3.2.1.1. Load Distribution Controller (LDC)

The LDC is the module that runs the load distribution algorithm (LDA). Therefore, it is in charge of taking load distribution decisions based on metrics available in the SIG. The LDA block diagram is shown in Figure 9. The operation of the algorithm is as follows. First of all, two initiation requirements are checked before proceeding: the AP must have more than one station associated and last distributed station has to be reassociated. The first requirement is clear: an AP cannot distribute its load with only one station associated. The second requirement is necessary in order to avoid wrong decisions made by the LDA. While the station is reassociating, the load that it was generating into the system *disappears* in the sense that average network load decreases during handover time. This is the effect that may induce the LDA of the local AP, and other APs, to take wrong decisions. To overcome this problem, the LDA waits until a notification (from another AP) informs that the station has been reassociated successfully. Setting a timeout that has a value slightly higher than handover time solves the case where the disassociated station does not reassociate with another AP. In this way, the LDA would be executed again even if the message was never received.

Once both requirements are fulfilled, the LDA gets the traffic measurements of the AP, the stations associated with the AP and of the other APs from the SIG. The traffic of the local AP and of the other APs is used to find out if the WLAN is balanced. The traffic of the stations is used by the *Selection policy* to choose a candidate station to transfer. The LDA determines if there is an unbalanced situation computing the *balance index* or β. The balance index appeared for the first time in [30] and it is used in [9] as a performance measure. The balance index reflects the used capacity in each AP. Let Ti be the total traffic of the AP i. Then, the balance index β is:

$$\beta = \left(\sum T_i\right)^2 \Big/ \left(n * \sum T_i^{\,2}\right)$$

Where *n* is the number of cells over which the load is being redistributed. In our case, *n* is the number of cells with overlapped coverage area. The balance index has the property that it is 1 when all APs have exactly the same traffic and it gets closer to 1/*n* when APs are heavily unbalanced.

Therefore, the LDA checks if load in the WLAN is balanced (β≈1 and AP's state is OK) or if it is unbalanced (β<1). In the last case, the *Transfer policy* decides if the AP is overloaded (sender) or under loaded (receiver). A relative transfer policy is used, which means that the load of the AP is considered in relation to the average network load. If the load of the AP is higher than a certain δ (where δ is a percentage of the current average network load) the AP is a sender. Otherwise, it is a receiver and the LDA starts from the beginning again. Once the *Transfer policy* determines that the AP is a sender, it performs two actions. First, it denies new associations applying *Receiver enforcement* as the *Location policy*. Second, it executes the *Selection policy* that selects a candidate station to transfer. This selection is based on the *Best candidate policy*.

Finally, the *Distribution policy* decides whether it is worth to distribute the selected station or not. In order to discern about this point, the *Distribution policy* re-computes an estimated balance index as follows. First, the traffic generated by the selected station is subtracted from the traffic of the local AP. Then, the LDA computes as many estimated balance indexes as APs participating in the load distribution. Each estimated balance index is obtained by adding the selected station traffic to each AP traffic. If there is at least one estimated balance index higher than the current balance index, then it is worth to distribute the load and the station is disassociated. Otherwise, it is not to worth to distribute the load and the LDA starts again. Thus, the goal of the *Distribution policy* is to avoid costly handovers when they are not needed.

As we have seen before, the LDA must wait until the last disassociated station has been reassociated. The reason is that during the handover time, the average network load decreases and the local AP and other APs may take wrong distribution decisions. This procedure prevents the local AP to take a wrong decision based on temporary change of the average network load. However, it does not take into account the other APs so their decisions will be based on a lower average network load. To overcome this problem, the local AP broadcasts the *same* load than before the station left so the other APs perceive that the average network load remains without changes. The local AP starts to broadcast its current load upon the arrival of the reassociation notification.

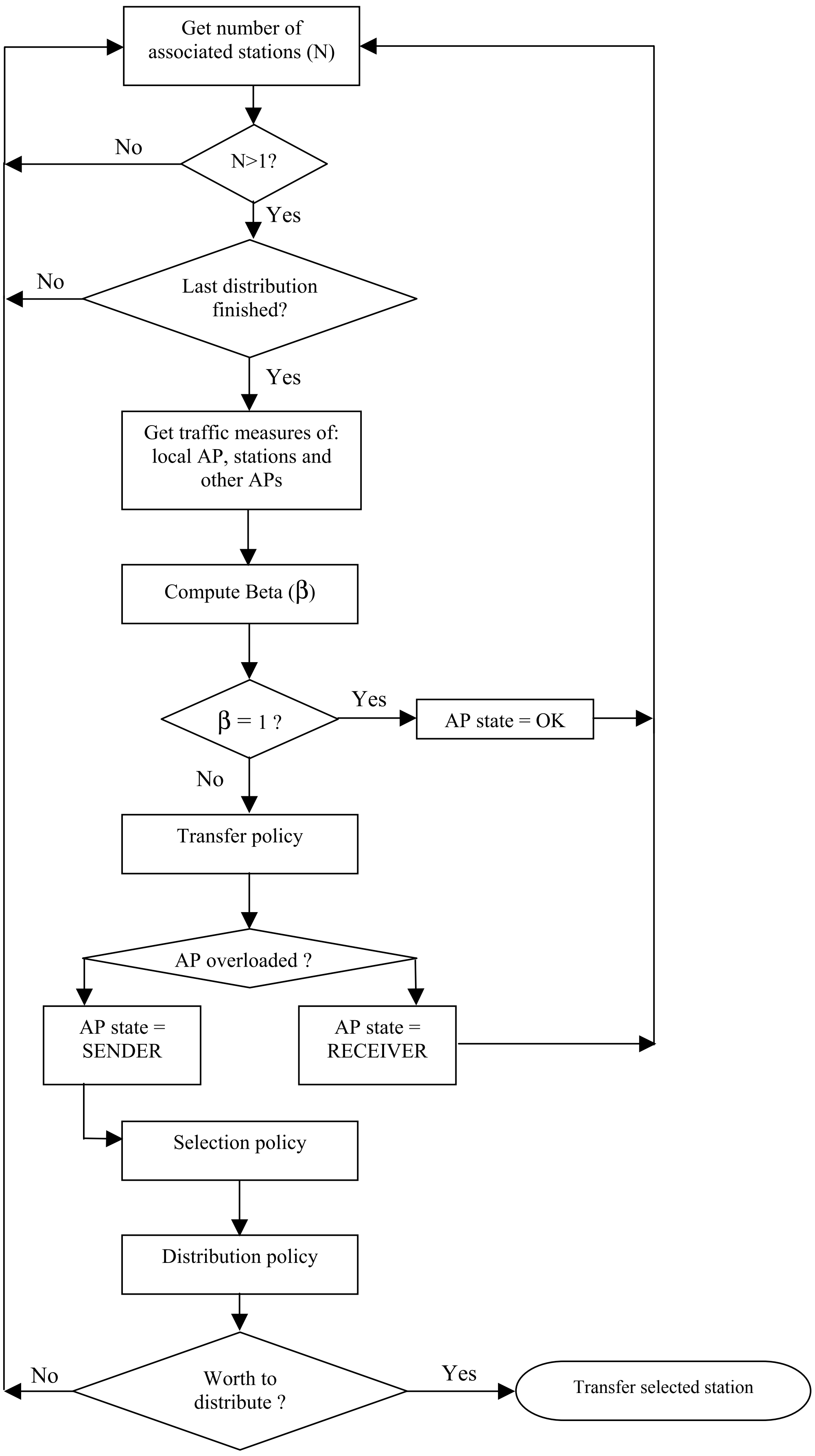


Figure 9: Block diagram of the load distribution algorithm (LDA)

3.2.1.2. Decision Enforcement Point (DEP)

The DEP is the module in charge of converting the decisions taken by the LDC into actions that lead to distribute the load among the APs. It is a two-step process. First, it involves the management of the Access Control List (ACL). Second, it sends a disassociation message to the selected station. We describe these two processes:

1. **ACL management**: this action involves updating the ACL of the AP. When a distribution decision is taken, DEP deletes the selected station MAC address from the ACL. Because the ACL is a MAC filter configured with an *allow policy* (only stations included in this filter can be associated with the AP), this means that the station will not be able to reassociate with the AP from where it was rejected. This action is necessary since after receiving the Disassociation message, the station tries to reassociate again with the same AP. In this way, the AP will deny the reassociation request so the station reassociates with another AP. Upon the arrival of the reassociation notification, the DEP adds the MAC of the selected station to the ACL. Therefore, the station will be able to reassociate in the future with the old AP.

2. **Disassociation message**: after the selected station is deleted from the ACL, the DEP sends a disassociation message to the station. The station cannot reject it, so it starts the reassociation immediately.

3.2.1.3. Metric Monitor (MM)

The Metric Monitor (MM) module is in charge of two functionalities related to load metrics: *measurement* and *broadcast*. It also takes care of *reassociation notifications* that APs exchange between them to find out if the last distributed station has been reassociated.

The *measurement* functionality measures the traffic coursed by the local AP and by the stations associated with it. As it has been decided, the traffic is measured in bytes/second for both APs and stations. One important issue here is the necessary time to obtain reliable measurements. Thus, it is a parameter that has to be obtained directly from the tests with the prototype.

The second functionality of the MM is to *broadcast* the average load state of the AP to the DS. Thus, the AP uses the wired side of the network in order to exchange traffic information so the state of the AP is known by other APs. This is done by broadcasting Ethernet frames where the source address is the Ethernet MAC address of the AP and the data field is its own traffic (bytes/second).

In addition, the MM stores the last distributed station MAC address with its last load metric when the LDC decides to distribute a station. It also updates, deletes the station and its load from the SIG, this information when a reassociation notification is received from the DS. In case the station does not reassociate again with another AP (the station switches off, for instance) the MM detects this situation and deletes the station MAC address and its load from the SIG after a time slightly higher than the handover time.

3.2.1.4. State Information Storage (SIG)

The State Information Storage (SIG) is the module where resides all state information that LDS needs. This information is the local AP traffic, other APs traffic, associated stations traffic and last distributed station traffic. The unit for the traffic is bytes/second. The SIG needs 6 bytes to store the MAC address of a station and 3 bytes more for its load. Therefore, the SIG needs 9 bytes per station to store the station’s information.

The SIG is accessed by two entities: the MM and the LDC. The MM has read and write access while the LDC has only read access. The information is stored in the SIG using the MAC address of the AP or the station as the index and with the load metric value as data.

3.2.2. Functional description

The LDS operates in a distributed way where each AP takes its own distribution decisions. The goal is to evenly distribute the traffic among all APs in a WLAN with overlapped coverage areas. We will illustrate the functionality of the LDS with the aid of the next example. Let's take a WLAN with two APs (AP1 and AP2) with overlapped coverage areas where there are four stations (see Figure 10): STA1, STA2, STA3 and STA4. Let's assume, for the sake of simplicity, that the generated traffic of all stations is static and equals to 3, 2, 0.7 and 0.3 Mbps respectively. The total AP1 traffic is 5 Mbps and 1 Mbps for AP2, approximately.

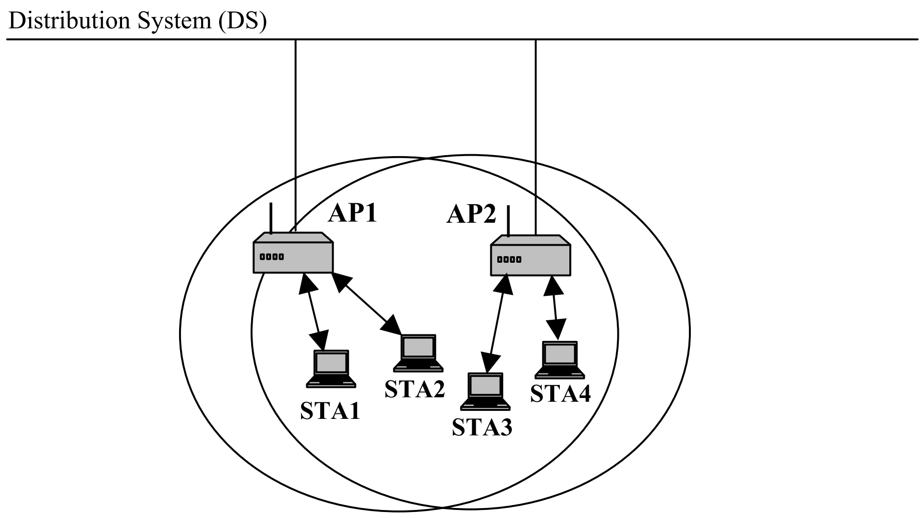


Figure 10: Load distribution example

Both APs know the traffic of each other because the MM has broadcasted the local AP load. Therefore, based on this information, the LDC of each AP is able to find out whether the load within the WLAN is balanced or not. Thus, according to the LDC, the balance index (β) in this particular situation is equal to:

$$\beta = \left(\sum T_i\right)^2 \Big/ \left(n * \sum T_i^{\ 2}\right) = \frac{(5+1)^2}{2\times\left(5^2+1^2\right)} = 0.69$$

The value of β below one indicates that the system is unbalanced and that a load distribution decision may increase the utilization of the network. Then, both APs execute its Transfer policy to find out which one is in situation of distributing the load:

1. Average network load (ANL) = 3 Mbps
2. Relative transfer policy: δ = 10 % of ANL + ANL = 3.3 Mbps
3. Transfer policy $\begin{cases} T_i \geq \delta \Rightarrow \text{Sender} \\ T_i < \delta \Rightarrow \text{Receiver} \end{cases}$

Therefore, the Transfer policy determines that AP1 (T_1 = 5 Mbps > δ) is a sender while AP2 (T_2 = 1 Mbps < δ) is a receiver. Note that δ specifies the level of unbalance in the network. If a higher δ is set (i.e., 30% instead of 10%), some APs will be more loaded than others. On the other hand, when δ is set to be equal to the ANL the LDS will tend to achieve a completely balanced scenario ($\beta \approx 1$).

The next step for AP1 is to select a candidate station to disassociate, task that is done by the Selection policy as follows:

1. First, the difference from the ANL is computed:
   - Difference from ANL = T_1 – ANL = 5 – 3 = 2 Mbps
2. Second, for each station the best candidate station metric (BCSM) is computed:
   - STA1: $BCSM_1 = |T_{STA1}$ - Difference from ANL$| = |3 – 2| = 1$
   - STA2: $BCSM_2 = |T_{STA2}$ - Difference from ANL$| = |2 – 2| = 0$
3. The station that minimizes the BCSM (STA2 in our example) is selected to be disassociated.

Once the Selection policy of AP1 has selected STA2 to be distributed, the Distribution policy computes the new estimated balance index ($\beta_{estimated}$) to decide if the distribution of the station will produce a better balanced scenario, i.e., a higher β:

1. First, it subtracts from the local AP (AP1) the traffic generated by the STA2:
   - $T_{1\_estimated}$ = 5 – 2 = 3 Mbps
2. Second, for each AP (except the current one) it computes a new $\beta_{estimated}$ adding the traffic of the STA2 only to that particular AP:
   - $T_{2\_estimated}$ = 1 + 2 = 3 Mbps
3. Finally, it computes the $\beta_{estimated}$ for every AP:
   - $\beta_{estimated} = \left(\sum T_i\right)^2 \Big/ \left(n * \sum T_i^{\,2}\right) = \dfrac{(3+3)^2}{2 \times (3^2 + 3^2)} = 1 > 0.69$

Thus, the $\beta_{estimated}$ is 1 that is higher than 0.69. Therefore, the LDC in AP1 takes the decision to transfer STA2. Note that the Distribution policy is not a mere confirmation of the Selection policy but a policy to avoid possible unstable situations and costly handovers. For instance, let's say that both APs have the same load and suddenly a new station associates with AP1. As in the previous example, the Transfer policy decides that AP1 is a sender and AP2 a receiver. Then, the Selection policy chooses a station to be distributed. Without the Distribution policy, AP1 will disassociate the station but AP2 will do the same since both APs are equally loaded.

Following with the first example, when the DEP receives the decision from the LDC it performs two actions: first, it denies future access to STA2 (deleting STA2 MAC address from the ACL) and second, it sends a disassociation message to the same station. Once STA2 has received the disassociation message, it starts the reassociation procedure with another AP, i.e., AP2. During the handover time, AP1 cannot take additional distribution decisions until AP2 broadcasts via DS the reassociation notification informing that STA2 has been reassociated with it. It is important to note here that while the AP1 does not receive this notification it broadcasts its old traffic level, thus 5 Mbps instead of the new one that is 3 Mbps. In this way, the average network load remains without changes, at 3 Mbps, which avoids possible wrong distribution decisions from other APs. When the reassociation process is finished, the DEP in AP1 allows again the access to STA2 (adding STA2 MAC address to the ACL) and the MM deletes the information in the SIG about the last distributed station. At this moment, the MM of AP1 starts to broadcast the new AP1 load level, 3 Mbps instead of 5 Mbps.

At this point, the state of both APs is *ok* since are equally loaded, with 3 Mbps each one, and the LDS does not take further decisions given that the load remains static. As a result of the load distribution, one station is associated with AP1 (STA1) and three with AP2 (STA2, STA3 and STA4). Thus, the load distribution does not take into account the number of users per AP but the traffic per AP.

4. Load distribution system implementation

As a proof of concept we have implemented and experimentally tested the designed LDS. In this section, we describe our prototype. First, in subsection *4.1*, hardware and software used in the prototype is described, including a description of the LDS code. Then, in subsection *4.2* we run three initial tests in order to determine the necessary parameters of the LDS: the handover time of our prototype, the sampling time of the metric monitor and the cycle time of the LDA.

4.1. Implementation

4.1.1. Hardware components

Three hardware components compose the prototype (see Figure 11): the *wireless cards*, the *computers* and the *switch*. There are six wireless cards D-Link 520: an IEEE 802.11b wireless PCI adapter that operates within the 2.4 GHz using Direct Sequence Spread Spectrum (DSSS). These cards are based on the Prism2.5 chipset made by Intersil.

Three computers compose the prototype (Computer 1, Computer 2 and Computer 3) and each one has two WLAN cards installed (wlan0 and wlan1 interfaces). The computers, Dell PowerEdge 1650, have a single Pentium III processor at 1.13 GHz and 512 MB of 133 MHz ECC SDRAM. The storage is composed by two hard disks of 20 GB configured as RAID 1. Additionally, they have an integrated dual Intel Pro/1000 XT (Gigabit device).

The third hardware component, the *switch*, allows the computers to access the Internet and communicate between each other. The computers use one of the Gigabit interfaces to connect with the switch. Because the switch does not have Gigabit ports, we have limited the speed of the Gigabit interface of the computers to 100 Mbps. Therefore, we use Gigabit Ethernet over Category 5 copper cabling [31]. The complete network architecture is shown in Figure 11.

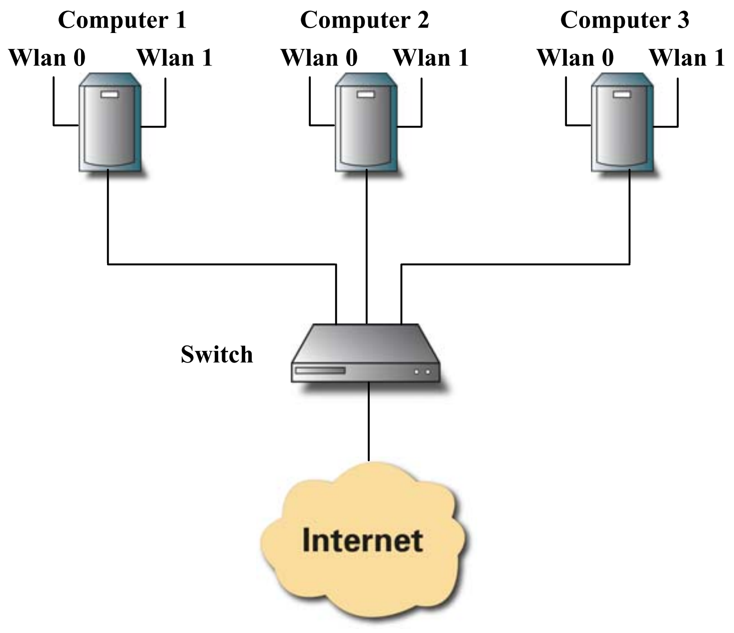


Figure 11: Prototype network architecture

4.1.2. Selection of the Linux driver for the WLAN card

The wireless cards are based on the Prism2.5 chipset by Intersil. This is a key feature since they can be converted into APs using the appropriate Linux drivers. Currently, there are three WLAN drivers available in Linux: *OpenAP*[7], *HostAP*[8] and *Linux-wlan-ng*[9]. We describe in this subsection the characteristics of each one in order to select the most suitable driver for our prototype.

1. **OpenAP**: OpenAP is not only a driver but also a complete distribution of open-source software that supports the configuration of special WLAN hardware as 802.11b APs. It only supports Eumitcom WL11000SA-N board cards based on the Prism2 chipset by Intersil.

2. **HostAP**: HostAP is a Linux driver for wireless LAN cards based on Intersil's Prism2/2.5/3 chipset. The driver supports a mode called Host AP, i.e., it takes care of IEEE 802.11 management functions in the host computer and acts as an AP. This does not require any special firmware for the wireless LAN card. In addition to this, it has support for normal station operations in BSS and also in IBSS. The driver has also various features for development debugging and for researching IEEE 802.11 environments such as access to hardware configuration records, I/O registers, and frames with 802.11 headers.

3. **Linux-wlan-ng**: the linux-wlan-ng driver also supports the Intersil Prism2/2.5/3 chipsets. A drawback of this driver is that it only supports AP operation by using a special tertiary firmware. However, this firmware is not generally available and it is necessary to contact Intersil to buy it.

We have selected *HostAP* driver because is the only one that is able to support AP mode in WLAN cards based on Prism2/2.5/3 chipsets. The first option, *OpenAP,* is intended to flashing Linux on APs with a particular hardware platform (Eumitcom WL11000SA-N board cards) and therefore it does not fit our requirements. Finally, the third option, *Linux-wlan-ng*, does not support AP operation.

The version of HostAP used is the 2002-05-01. HostAP driver supports extensively the Wireless Tools[10], which allows manipulating wireless interfaces. Additionally to the HostAP driver, a kernel module that features the link layer update for wireless LANs is used[11]. Using HostAP driver is possible to configure the wireless devices on each computer (wlan0 and wlan1) either as an AP (Master mode) or as a station (Managed mode) depending on the test. When configured as an AP, it is necessary to bridge the wireless device with the Ethernet device. HostAP driver does not provide bridging between wireless and wired networks but Linux 2.4.x kernels have the Ethernet bridging code built-in. We use the 2.4.18-3 kernel version, which is included in Red Hat Linux 7.3. Therefore, we can bridge the computer's network interface with a wireless interface.

[7] See http://opensource.instant802.com/home.php
[8] See http://hostap.epitest.fi
[9] See http://www.linux-wlan.com/linux-wlan
[10] See http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html
[11] See http://www.it.kth.se/~hvelayos/software.shtml

4.1.3. Description of the LDS code

The LDS is implemented purely in software and runs locally at each AP. As it can be seen in Figure 12, the LDS uses the wireless tools (WT) to perform some functions such as rejecting a station from the AP or the management of the ACL (adding and removing stations from it).

The code has been developed as a user space program in C. It is divided in three main parts that correspond to the entities described in section *3*: load distribution algorithm (LDA), metric monitor (MM) and decision enforcement point (DEP). Except the DEP, all the blocks run concurrently, i.e., while the MM gets new measurements from other APs (as well as from the local AP) the LDA takes the decisions to distribute the associated stations.

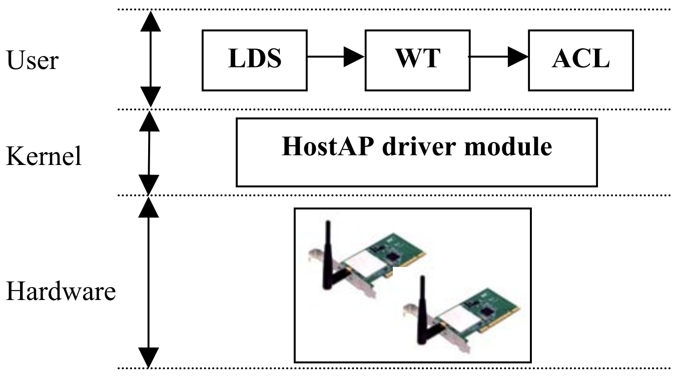


Figure 12: Position of the LDS in the prototype

4.1.4. System parameters

We describe in this subsection system parameters related with the prototype, such as specific wireless cards parameters. Then, we describe software we have used to perform the tests, to interpret the collected data and to configure the network.

1) Prototype configuration

The wireless LAN cards were installed, configured and tested before starting the experimental tests. The APs were configured to use non-interference channels such as 1, 6 and 13. Each AP operates at a data rate of 11 Mbps and at a maximum power of 100 mW (20 dBm). The particular configuration of the prototype is explained in every test subsection.

Additionally to Managed mode (station) or Master mode (AP), a WLAN interface can be configured in Monitor mode. In this mode, the WLAN card can switch to promiscuous mode and it is able to get the frames on the radio side on the configured channel.

2) Traffic generator

The *MGEN*[12] tool is used to generate UDP traffic streams. MGEN also time-stamps the packet and if clocks of the source and sink are synchronized, packet delay can be measured. The transmitted packets could be analysed by using the log files, which are generated at the destination node. These are some of the features of MGEN:

- Traffic type: Unicast / Multicast

[12] See http://manimac.itd.nrl.navy.mil/MGEN

- Traffic pattern: Periodic / Poisson
    - Periodic: this pattern type generates messages of a fixed size (in bytes) at a regular rate (in messages/second).
    - Poisson: this pattern type generates messages of a fixed size (in bytes) at statistically varying intervals at an average rate (in messages/second).

3) Data analysis and plots

To get information (throughput, latency, delay, etc.) from MGEN output, the program *TRPR* (TRace Plot Real-time)[13] is used. The program *Gnuplot*[14] is used to represent graphically this data. Additionally, the program *Ethereal*[15] is used as a network analyser and allows us to examine data from a live network or from a capture file on disk.

4) Synchronization considerations

In order to perform the tests it will be necessary to have all the computers synchronized. The Network Time Protocol (NTP)[16] distributed protocol is used to synchronize the computers.

5) Data analysis and plots

An IP address is assigned to both, APs and stations, by means of a Dynamic Host Configuration Protocol (DHCP) client. Therefore, a DHCP client running on each computer is in charge of assigning IP addresses.

4.2. Algorithm parameters: initial tests

We describe in this subsection three initial tests to adjust some parameters in our prototype. First, we measure handover time because is an essential parameter for the LDA module since a new load distribution decision cannot be taken until the last transferred station has been reassociated. Second, it is necessary to adjust the sampling time (Ts) of the metric monitor to obtain reliable traffic measurements from the AP and its associated stations. Finally, we adjust the periodicity of execution of the LDA to find out if it affects the capacity of the LDS to balance the load.

4.2.1. Handover time transition measurement

4.2.1.1. Description

The goal of this test is to measure the handover time. As it has been explained, the AP cannot take a load distribution decision during this time in order to prevent possible wrong decisions. Therefore, this time provides an indication about how fast a distributed scenario can be achieved. The handover time is measured using two different initiation methods:

1. Switching off the radio transmitter of the AP.

2. Sending a Disassociation message from the AP to the station.

In this test, we force a handover of a station using of the above methods. The goal is to measure the reassociation time. The measurement of the time is done by analysing the timestamps of the transmitted radio frames on the channel. To do this, it is necessary to

[13] See htttp://proteantools.pf.itd.nrl.navy.mil/trpr.html
[14] See http://www.gnuplot.info
[15] See http://www.ethereal.com
[16] See http://www.halley.cc/ed/linux/howto/ntp.html

configure another wireless device in Monitor mode. Then, frames obtained with this wireless device are analysed with Ethereal and only management frames related with the reassociation procedure are studied. In this way, it is possible to follow in detail the reassociation procedure and to measure precisely the handover time.

4.2.1.2. Test-bed configuration

The configuration of the test-bed is shown in Figure 13:

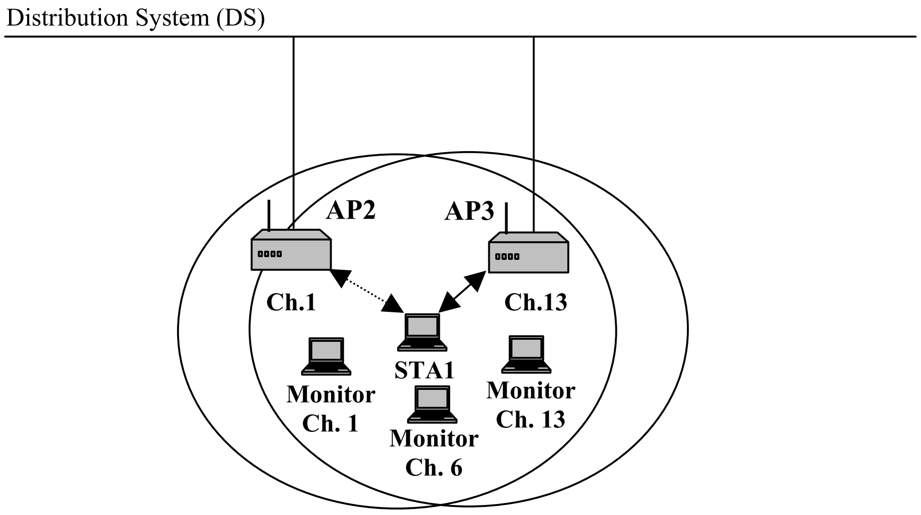


Figure 13: Test-bed configuration to measure handover time

The station is initially associated with AP3 (solid line in Figure 13). The station generates UDP traffic with a bit rate of 25.6 Kbps (50 messages/s, 64 bytes/message) and its nominal bit rate is 11 Mbps. In all tests, the data flow is periodic and the source is the station (uplink traffic).

4.2.1.3. Results

First of all, the results show that the station uses active scanning. Thus, once the reassociation procedure is initiated, the station starts to send *Probe requests* in order to get *Probe responses* from the APs in range. We study the particular reassociation procedure for both methods:

1) Switching off the radio transmitter of the AP

The sequence of the handover with the this method was the next (see Figure 14):

1. Last acknowledgment packet (ACK) received from the AP
2. First Probe request
3. Last Probe request
4. Authentication request from the station to the new AP (AP2)
5. Authentication response from AP2 to the station
6. Reassociation request from the station to the AP2
7. Reassociation response from AP2 to the station

The duration of the periods for each step of the sequence is detailed in Table 3. Figure 14 graphically shows the sequence.

	Period duration	
	Method 1	Method 2
Δt1	2.245 (s)	1.057 (s)
Δt2	189 (ms)	752 (ms)
Δt3	60 (ms)	73 (ms)
Δt4	3 (ms)	3 (ms)
Total	**2.496 (s)**	**1.885 (s)**

Table 3: Handover time comparison using method 1 and 2

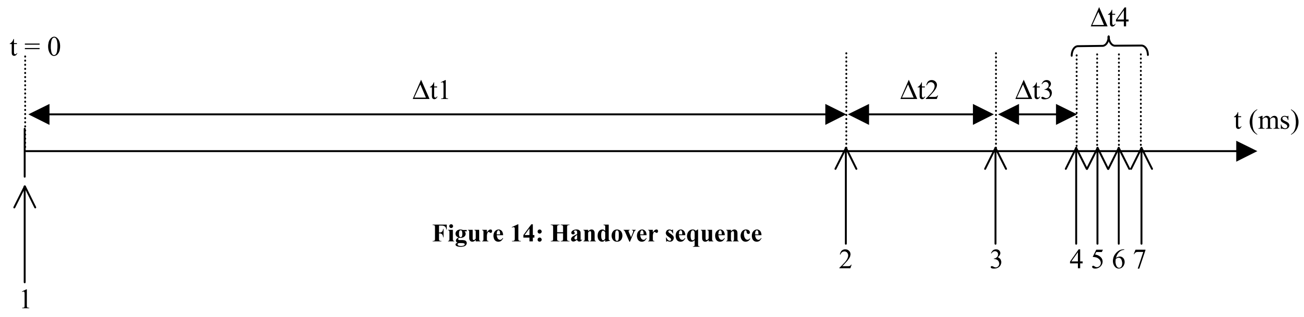
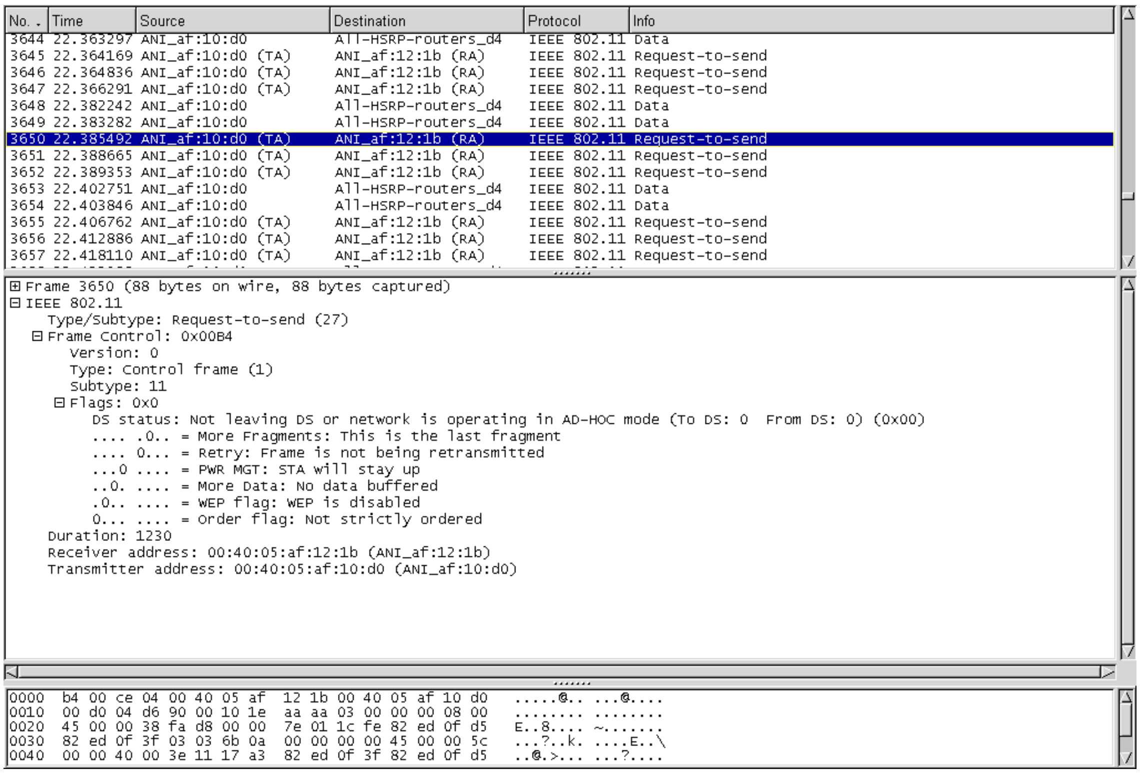


Figure 14: Handover sequence

The resulting handover time is almost 2.5 s, which is an unexpectedly high value. The expected value was between 200 and 300 ms [21]. The main contribution to this high value is Δt1, which is equal to 2.245 s. Thus, the problem is that the station takes 2.245 s to be aware that cannot reach AP3 anymore. During Δt1, the station sends Request-to-send frames to AP3 in order to start the transmission again (see Figure 15). Once the station realizes that AP3 is not reachable, it starts to send Probe requests and to wait Probe responses from AP2 during period Δt2. Finally, during period's Δt3 and Δt4 the station authenticates and reassociates with AP2 (see Figure 16). Therefore, from period Δt2 to period Δt4 only takes 243 ms.

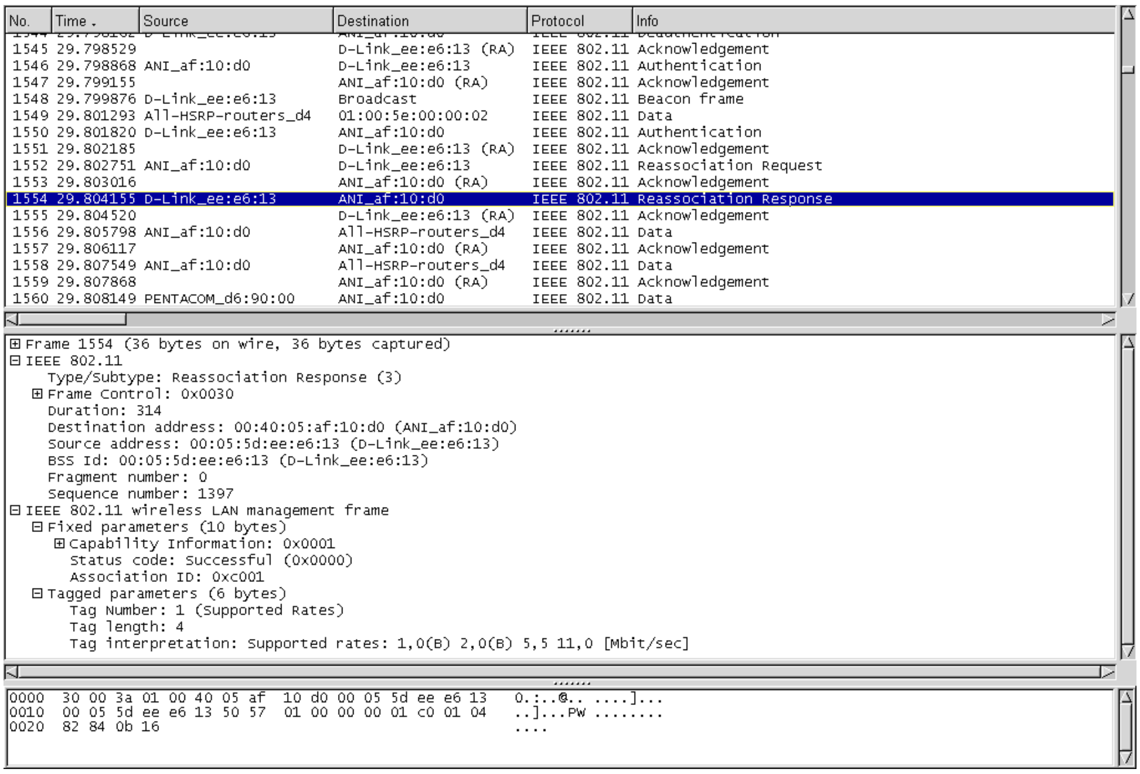


```
No. . Time      Source              Destination          Protocol    Info
3644 22.363297 ANI_af:10:d0        All-HSRP-routers_d4  IEEE 802.11 Data
3645 22.364169 ANI_af:10:d0 (TA)   ANI_af:12:1b (RA)    IEEE 802.11 Request-to-send
3646 22.364836 ANI_af:10:d0 (TA)   ANI_af:12:1b (RA)    IEEE 802.11 Request-to-send
3647 22.366291 ANI_af:10:d0 (TA)   ANI_af:12:1b (RA)    IEEE 802.11 Request-to-send
3648 22.382242 ANI_af:10:d0        All-HSRP-routers_d4  IEEE 802.11 Data
3649 22.383282 ANI_af:10:d0        All-HSRP-routers_d4  IEEE 802.11 Data
3650 22.385492 ANI_af:10:d0 (TA)   ANI_af:12:1b (RA)    IEEE 802.11 Request-to-send
3651 22.388665 ANI_af:10:d0 (TA)   ANI_af:12:1b (RA)    IEEE 802.11 Request-to-send
3652 22.389353 ANI_af:10:d0 (TA)   ANI_af:12:1b (RA)    IEEE 802.11 Request-to-send
3653 22.402751 ANI_af:10:d0        All-HSRP-routers_d4  IEEE 802.11 Data
3654 22.403846 ANI_af:10:d0        All-HSRP-routers_d4  IEEE 802.11 Data
3655 22.406762 ANI_af:10:d0 (TA)   ANI_af:12:1b (RA)    IEEE 802.11 Request-to-send
3656 22.412886 ANI_af:10:d0 (TA)   ANI_af:12:1b (RA)    IEEE 802.11 Request-to-send
3657 22.418110 ANI_af:10:d0 (TA)   ANI_af:12:1b (RA)    IEEE 802.11 Request-to-send

⊞ Frame 3650 (88 bytes on wire, 88 bytes captured)
⊟ IEEE 802.11
    Type/Subtype: Request-to-send (27)
  ⊟ Frame Control: 0x00B4
      Version: 0
      Type: Control frame (1)
      Subtype: 11
    ⊟ Flags: 0x0
        DS status: Not leaving DS or network is operating in AD-HOC mode (To DS: 0  From DS: 0) (0x00)
        .... .0.. = More Fragments: This is the last fragment
        .... 0... = Retry: Frame is not being retransmitted
        ...0 .... = PWR MGT: STA will stay up
        ..0. .... = More Data: No data buffered
        .0.. .... = WEP flag: WEP is disabled
        0... .... = Order flag: Not strictly ordered
    Duration: 1230
    Receiver address: 00:40:05:af:12:1b (ANI_af:12:1b)
    Transmitter address: 00:40:05:af:10:d0 (ANI_af:10:d0)

0000  b4 00 ce 04 00 40 05 af  12 1b 00 40 05 af 10 d0   .....@.. ...@....
0010  00 d0 04 d6 90 00 10 1e  aa aa 03 00 00 00 08 00   ........ ........
0020  45 00 00 38 fa d8 00 00  7e 01 1c fe 82 ed 0f d5   E..8.... ~.......
0030  82 ed 0f 3f 03 03 6b 0a  00 00 00 00 45 00 00 5c   ...?..k. ....E..\
0040  00 00 40 00 3e 11 17 a3  82 ed 0f 3f 82 ed 0f d5   ..@.>... ...?....
```

Figure 15: The station is not aware that AP3 has switched off and starts to transmit Request-to-send frames to AP3



```
No.  Time .    Source               Destination          Protocol    Info
1545 29.798529                      D-Link_ee:e6:13 (RA) IEEE 802.11 Acknowledgement
1546 29.798868 ANI_af:10:d0         D-Link_ee:e6:13      IEEE 802.11 Authentication
1547 29.799155                      ANI_af:10:d0 (RA)    IEEE 802.11 Acknowledgement
1548 29.799876 D-Link_ee:e6:13      Broadcast            IEEE 802.11 Beacon frame
1549 29.801293 All-HSRP-routers_d4  01:00:5e:00:00:02    IEEE 802.11 Data
1550 29.801820 D-Link_ee:e6:13      ANI_af:10:d0         IEEE 802.11 Authentication
1551 29.802185                      D-Link_ee:e6:13 (RA) IEEE 802.11 Acknowledgement
1552 29.802751 ANI_af:10:d0         D-Link_ee:e6:13      IEEE 802.11 Reassociation Request
1553 29.803016                      ANI_af:10:d0 (RA)    IEEE 802.11 Acknowledgement
1554 29.804155 D-Link_ee:e6:13      ANI_af:10:d0         IEEE 802.11 Reassociation Response
1555 29.804520                      D-Link_ee:e6:13 (RA) IEEE 802.11 Acknowledgement
1556 29.805798 ANI_af:10:d0         All-HSRP-routers_d4  IEEE 802.11 Data
1557 29.806117                      ANI_af:10:d0 (RA)    IEEE 802.11 Acknowledgement
1558 29.807549 ANI_af:10:d0         All-HSRP-routers_d4  IEEE 802.11 Data
1559 29.807868                      ANI_af:10:d0 (RA)    IEEE 802.11 Acknowledgement
1560 29.808149 PENTACOM_d6:90:00    ANI_af:10:d0         IEEE 802.11 Data

⊞ Frame 1554 (36 bytes on wire, 36 bytes captured)
⊟ IEEE 802.11
    Type/Subtype: Reassociation Response (3)
  ⊞ Frame Control: 0x0030
    Duration: 314
    Destination address: 00:40:05:af:10:d0 (ANI_af:10:d0)
    Source address: 00:05:5d:ee:e6:13 (D-Link_ee:e6:13)
    BSS Id: 00:05:5d:ee:e6:13 (D-Link_ee:e6:13)
    Fragment number: 0
    Sequence number: 1397
⊟ IEEE 802.11 wireless LAN management frame
  ⊟ Fixed parameters (10 bytes)
    ⊞ Capability Information: 0x0001
      Status code: Successful (0x0000)
      Association ID: 0xc001
  ⊟ Tagged parameters (6 bytes)
      Tag Number: 1 (Supported Rates)
      Tag length: 4
      Tag interpretation: Supported rates: 1,0(B) 2,0(B) 5,5 11,0 [Mbit/sec]

0000  30 00 3a 01 00 40 05 af  10 d0 00 05 5d ee e6 13   0.:..@.. ....]...
0010  00 05 5d ee e6 13 50 57  01 00 00 00 01 c0 01 04   ..]...PW ........
0020  82 84 0b 16                                        ....
```

Figure 16: Authentication and reassociation responses from AP2 to the station

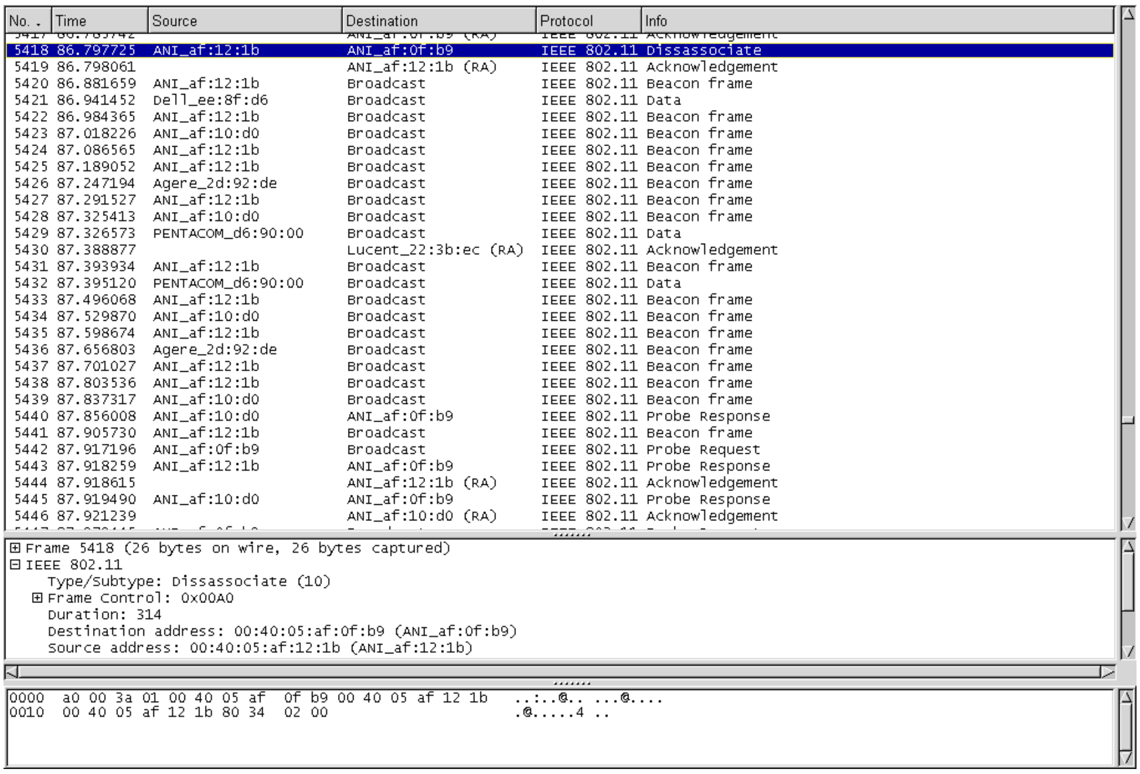
2) Sending a Disassociation message from the AP to the station

The sequence of the handover with this method was the next:

1. Disassociation message from AP3 to the station
2. First Probe request
3. Last Probe request
4. Authentication request from the station to the new AP (AP2)
5. Authentication response from AP2 to the station
6. Association request from the station to AP2
7. Association response from AP2 to the station

The duration of the periods for each step of the sequence is detailed in Table 3. Figure 14, the same figure as in the previous method, graphically shows the sequence.

The total handover time has decreased around 0.7 s (1.885 s instead of 2.496 s) sending a disassociation message from AP3 to the station (see Figure 17). Again, the longest period is Δt1 with 1.057 s (see Table 3). Although this value is shorter, now the period Δt2 (752 ms) is longer than with the first method. One reason to explain this may be that now the station is receiving Probe responses from both APs (AP2 and AP3) and therefore it takes more time (563 ms more) to choose the target AP. Another important difference respect to the first method is that now there is an association instead of a reassociation message.


```
No. .  Time       Source              Destination          Protocol     Info
5418 86.797725  ANI_af:12:1b        ANI_af:0f:b9         IEEE 802.11 Dissassociate
5419 86.798061                      ANI_af:12:1b (RA)    IEEE 802.11 Acknowledgement
5420 86.881659  ANI_af:12:1b        Broadcast            IEEE 802.11 Beacon frame
5421 86.941452  Dell_ee:8f:d6       Broadcast            IEEE 802.11 Data
5422 86.984365  ANI_af:12:1b        Broadcast            IEEE 802.11 Beacon frame
5423 87.018226  ANI_af:10:d0        Broadcast            IEEE 802.11 Beacon frame
5424 87.086565  ANI_af:12:1b        Broadcast            IEEE 802.11 Beacon frame
5425 87.189052  ANI_af:12:1b        Broadcast            IEEE 802.11 Beacon frame
5426 87.247194  Agere_2d:92:de      Broadcast            IEEE 802.11 Beacon frame
5427 87.291527  ANI_af:12:1b        Broadcast            IEEE 802.11 Beacon frame
5428 87.325413  ANI_af:10:d0        Broadcast            IEEE 802.11 Beacon frame
5429 87.326573  PENTACOM_d6:90:00   Broadcast            IEEE 802.11 Data
5430 87.388877                      Lucent_22:3b:ec (RA) IEEE 802.11 Acknowledgement
5431 87.393934  ANI_af:12:1b        Broadcast            IEEE 802.11 Beacon frame
5432 87.395120  PENTACOM_d6:90:00   Broadcast            IEEE 802.11 Data
5433 87.496068  ANI_af:12:1b        Broadcast            IEEE 802.11 Beacon frame
5434 87.529870  ANI_af:10:d0        Broadcast            IEEE 802.11 Beacon frame
5435 87.598674  ANI_af:12:1b        Broadcast            IEEE 802.11 Beacon frame
5436 87.656803  Agere_2d:92:de      Broadcast            IEEE 802.11 Beacon frame
5437 87.701027  ANI_af:12:1b        Broadcast            IEEE 802.11 Beacon frame
5438 87.803536  ANI_af:12:1b        Broadcast            IEEE 802.11 Beacon frame
5439 87.837317  ANI_af:10:d0        Broadcast            IEEE 802.11 Beacon frame
5440 87.856008  ANI_af:10:d0        ANI_af:0f:b9         IEEE 802.11 Probe Response
5441 87.905730  ANI_af:12:1b        Broadcast            IEEE 802.11 Beacon frame
5442 87.917196  ANI_af:0f:b9        Broadcast            IEEE 802.11 Probe Request
5443 87.918259  ANI_af:12:1b        ANI_af:0f:b9         IEEE 802.11 Probe Response
5444 87.918615                      ANI_af:12:1b (RA)    IEEE 802.11 Acknowledgement
5445 87.919490  ANI_af:10:d0        ANI_af:0f:b9         IEEE 802.11 Probe Response
5446 87.921239                      ANI_af:10:d0 (RA)    IEEE 802.11 Acknowledgement

⊞ Frame 5418 (26 bytes on wire, 26 bytes captured)
⊟ IEEE 802.11
     Type/Subtype: Dissassociate (10)
   ⊞ Frame Control: 0x00A0
     Duration: 314
     Destination address: 00:40:05:af:0f:b9 (ANI_af:0f:b9)
     Source address: 00:40:05:af:12:1b (ANI_af:12:1b)

0000  a0 00 3a 01 00 40 05 af  0f b9 00 40 05 af 12 1b   ..:..@.. ...@....
0010  00 40 05 af 12 1b 80 34  02 00                     .@.....4 ..
```

Figure 17: AP3 sends a disassociation message to the station, which starts to send Probe requests

4.2.1.4. Conclusions

The handover time is a key parameter in our load distribution scheme since when an AP disassociate a station, the LDA cannot take further decisions until the last transferred station has been reassociated. The results show two facts: first, the stations use active scanning and second, the handover time sending a disassociation message (around 1.8 s) is much higher than the expected value (between 200 and 300 ms).

The main consequence of this high handover delay, from the user side, is that there is a noticeable interruption of the communications. Therefore, this will produce higher delays in packet delivery as well as packet retransmissions and throughput decrement during the handover. However, this high value will not affect the load distribution behaviour but it will make it react slowly. Note than from now on, a handover value of 2 s should be used to interpret the results of the tests in the next subsections.

Finally, it is important to note that this is not an exhaustive analysis of the handover process in WLANs. Instead, the target was to calculate the handover time in our prototype since is a key parameter to interpret the results from the load distribution tests.

4.2.2. Metric monitor sampling time

4.2.2.1. Description

The goal of this test is to determine the sampling time (Ts) of the MM. This time is an important parameter because it determines the frequency that traffic of APs and stations is updated. We also have tested if the MM correctly computes the throughput of the AP and of the associated stations. We compare the results with a traffic generator (MGEN).

4.2.2.2. Test-bed configuration

The configuration of the test-bed is shown in Figure 18:

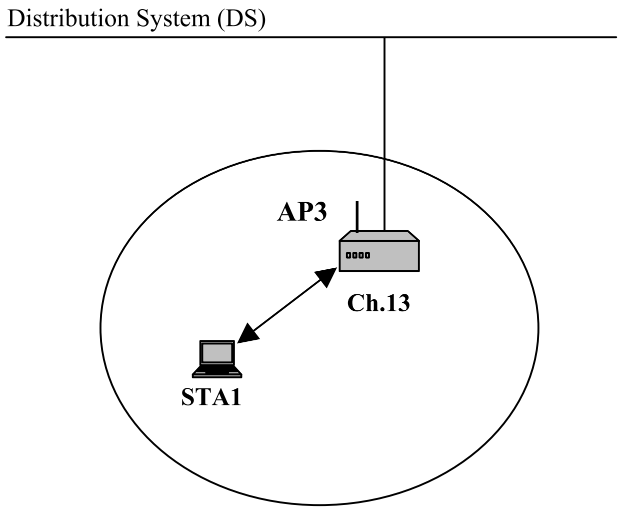


Figure 18: Test-bed configuration to test the sampling time of the metric monitor

The station, associated with AP3, generates UDP traffic using MGEN. Its nominal bit rate is 11 Mbps. In all tests, the data flow is periodic and the source is the station (uplink traffic).

4.2.2.3. Results

First, we select the sampling time (Ts) update the traffic of APs and stations. In order to measure the traffic, the MM samples at certain times the packets that are counted by the kernel. Specifically, HostAP driver maintains traffic statistics of the associated stations with the AP. Received and transmitted bytes per station are updated on a packet basis. Thus, HostAP increments its respective counters every time a packet is sent to the station or it is received from it. This implies that Ts should be at least equal or higher than the time needed to transmit a message of a minimum size at the maximum nominal rate, which is 11 Mbps. According to [5], over 70% of messages in a WLAN were smaller than 200 bytes. Thus, if we consider the transmission of a 228 bytes message at 11 Mbps, the theoretical average delay (minimum average Ts) is equal to 1.035 ms [32]. Therefore, Ts should be at least equal or longer than 1 ms. Once we have lower bounded Ts, we have to select a specific Ts. There are several trade-offs selecting Ts. For instance, selecting a sampling time close to the minimum, i.e., 2 ms, implies to increase traffic load on the wired side. In addition, it also will increase the computational load on the AP. Moreover, when the number of stations associated with an AP increases, the minimum average delay will also increase and it will be much longer than 1 ms. On the other hand, selecting shorter sampling times implies that load distribution algorithms will react faster to changes in the load. Therefore, we have selected a Ts equal to 100 ms since is a period shorter enough to detect changes in the load and it will decrease computational and traffic load on the wired side compared with the minimum Ts.

Second, we study the behaviour of the MM measuring the received and transmitted bytes of the STA1. We compare these results (see Figure 19) with the throughput obtained with the traffic generator (MGEN). The station transmits at 100 Kbps (25 messages/s, 500 bytes/message), which is exactly the throughput measured by MGEN. As it was expected, the throughput with the MM is higher because it also includes some additional headers (for instance, IP and UDP headers sums 28 bytes). In particular, for each message sent by MGEN, the MM increments the number of transmitted bytes in 536 bytes. Therefore, only considering the received bytes, the MM obtains a throughput of 107.2 Kbps (536 bytes x 25 messages/second x 8 bits) instead of 100 Kbps. If we consider both, the received and the transmitted bytes, the throughput obtained by the MM increases up to 112 Kbps. The reason is that now the MM also counts the acknowledgment messages (ACKs) that the AP sends to the station.

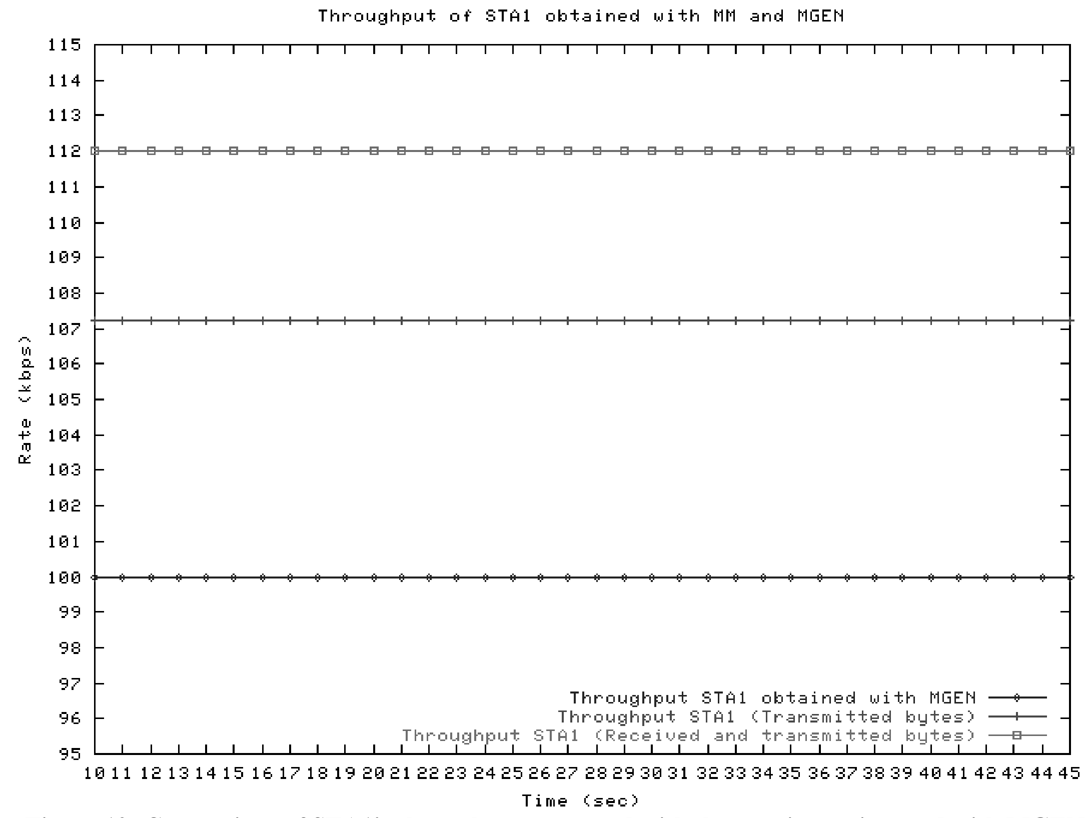


Figure 19: Comparison of STA1's throughput measured with the metric monitor and with MGEN

4.2.2.4. Conclusions

First, we have lower bound Ts considering the transmission of a packet of 228 bytes transmitted at 11 Mbps. The minimum value of Ts is 1.035 ms. However, there are several trade-offs selecting Ts, such as computational load on the AP and traffic load on the wired side. Based on these trade-offs, we have selected a Ts equal to 100 ms. Second, we have shown that MM correctly computes the throughput per station and per AP. The differences in the results between MGEN and the MM are due to overheads (such as UDP/IP).

4.2.3. Reactivity of the load distribution algorithm

4.2.3.1. Description

We understand by reactivity of the load distribution algorithm (LDA) its periodicity of execution. The goal of the last initial test is to find out how the reactivity of the LDA can influence the load distribution performance, i.e., the balance index parameter. Thus, the LDA is executed periodically and this test studies the effects of this period (called cycle time or CT) on the performance of the system.

Therefore, we test different CT values for the LDA with a simple WLAN configuration (two APs and one station). The load of AP2 is dynamic, while the load of the AP3 and the station is constant. In this way, the station must be distributed when the load of AP2 decreases or increases. The load of both APs is simulated within the LDA and its goal is to overload an AP while another is low loaded creating an unbalanced scenario.

4.2.3.2. Test-bed configuration

The configuration of the test-bed is shown in Figure 20:

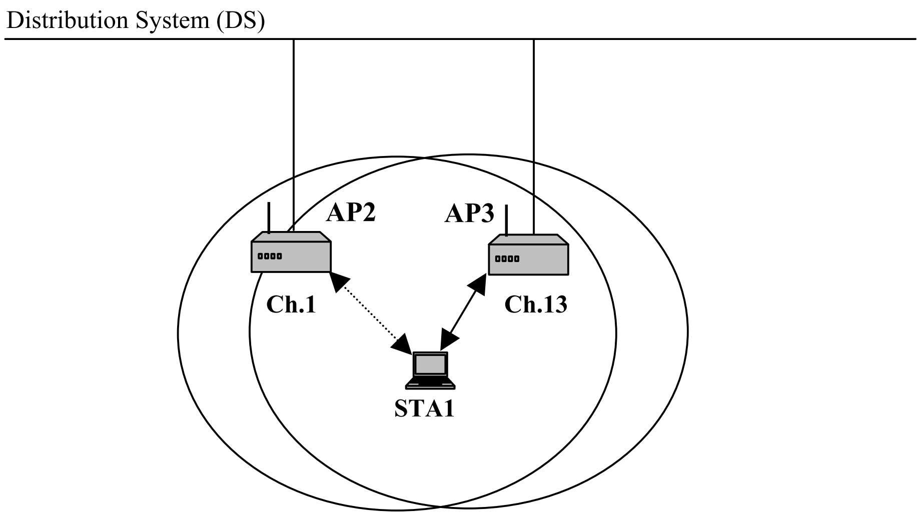


Figure 20: Test-bed configuration to test the reactivity of the LDA

The station is initially associated with AP3 (solid line in Figure 20). The station generates UDP traffic with a bit rate of 0.5 Mbps (125 messages/s, 500 bytes/message) and its nominal bit rate is 11 Mbps. In all tests, the station data flow is periodic and the source is the station (uplink traffic). A simulated load is generated in both APs in order to produce the distribution of STA1. In AP3, the simulated load is 0.5 Mbps and is constant. In AP2, the load is dynamic and it varies periodically every 10 s between 5 and 0.1 Mbps. When AP2 has a load of 0.1 Mbps, AP3 distributes the station so it reassociates with AP2 (dotted line in Figure 20). Therefore, the station is being distributed continuously while the test is running.

4.2.3.3. Results

We have obtained, for every CT value, the average balance index (β_{avg}) and its variance. These results are shown in Figure 21 and Figure 22. Note that for both figures, the CT value equal to 0 corresponds to the case without load distribution. The measured CT values are market with a dot.

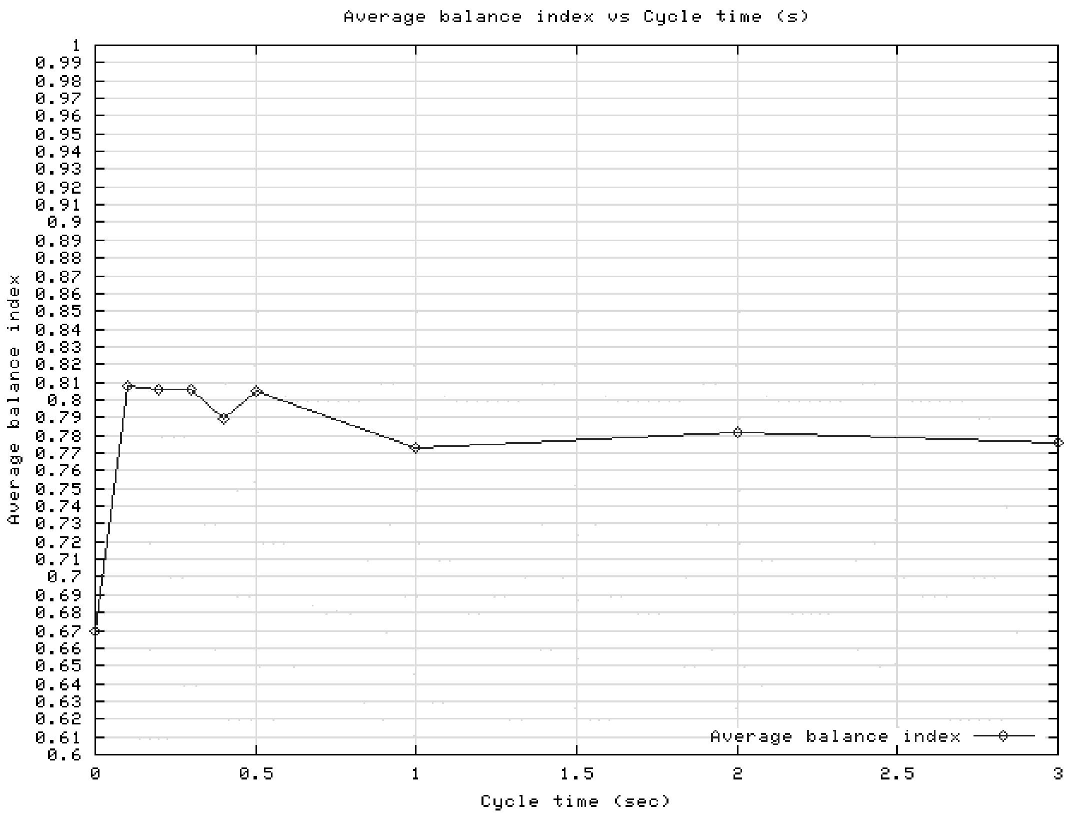


Figure 21: Average balance index (β_{avg}) for values of the CT = 0 (without LDS), 0.1 s, 0.2 s, 0.3 s, 0.4 s, 0.5 s, 1 s, 2 s and 3 s

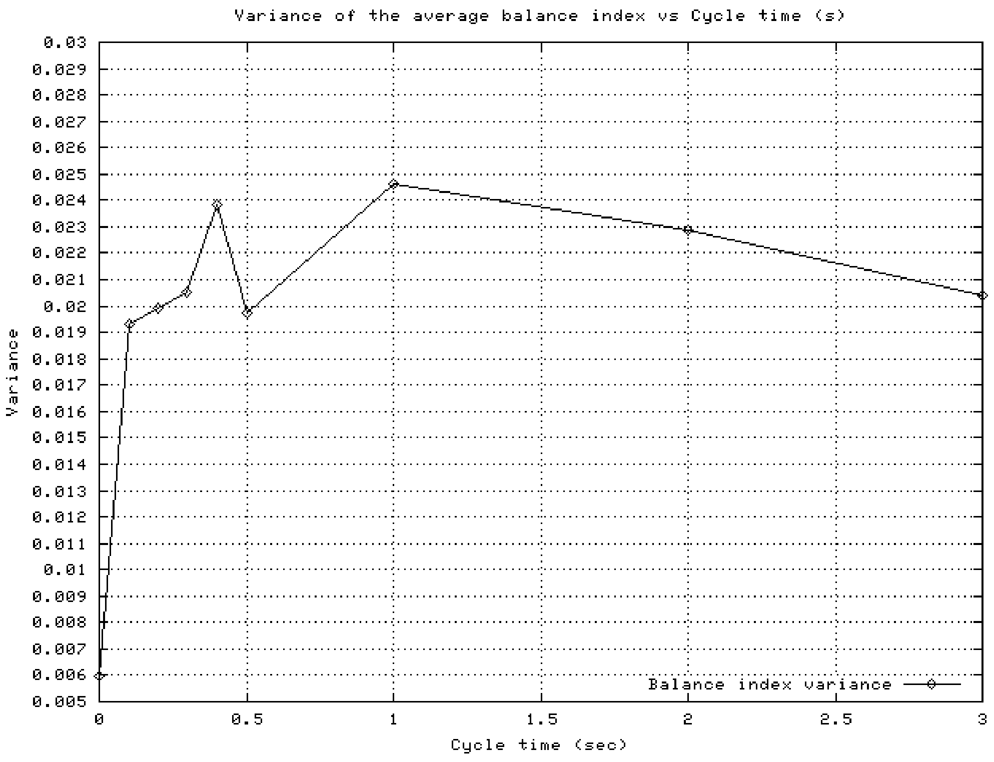


Figure 22: Variance of the average balance index for values of the CT = 0 (without LDS), 0.1 s, 0.2 s, 0.3 s, 0.4 s, 0.5 s, 1 s, 2 s and 3 s

4.2.3.4. Conclusions

The results from Figure 21 show that the average of the balance index tends to decrease when the CT value increases. The best β_{avg} values are obtained for the CT values 0.1 s, 0.2 s, and 0.3 s. For instance, the gain using a CT of 0.1 s instead of 1 s is around 4%. The variance of β_{avg} is also lower for these CT values than for the rest (see Figure 22). Without load distribution (see dot corresponding to 0 in Figure 21), β_{avg} decreases by 14% compared to the case using a CT equal to 0.1 s. Therefore, we conclude that the CT affects the performance of the LDA. Moreover, values of the CT close to the sampling time of the metric monitor, such as 0.1 s, obtain better performance than higher values. Selecting a lower CT value also implies that the LDA is able to react faster to changes in the load. However, it is not possible to react faster than the frequency the traffic is updated. This frequency is given by Ts, which is 100 ms. Therefore, we have selected a CT of 0.1 s for the LDA.

5. Analysis

In this section, we present the results of the tests we have conducted with the prototype. The goal is to experimentally determine the performance and the advantages that the LDS provides in comparison with a WLAN without it.

The first test, *Behaviour of the LDS*, shows the influence of the load distribution on the balance index. In the second test, *Packet delay*, we analyse the effect of load distribution on average packet delay and total throughput. The third test, *Location policy performance*, analyses two different location strategies and their effect on the station's throughput. Finally, the last test, *Distribution time measurement*, shows the necessary time to achieve a balanced scheme after a traffic variation.

5.1. Behaviour of the LDS

5.1.1. Description

These tests show how the LDS behaves with dynamic load. Thus, we study how the balance index varies versus time. In this way, we check that the LDS works correctly.

5.1.2. Test-bed configuration

The configuration of the test-bed is shown in Figure 23:

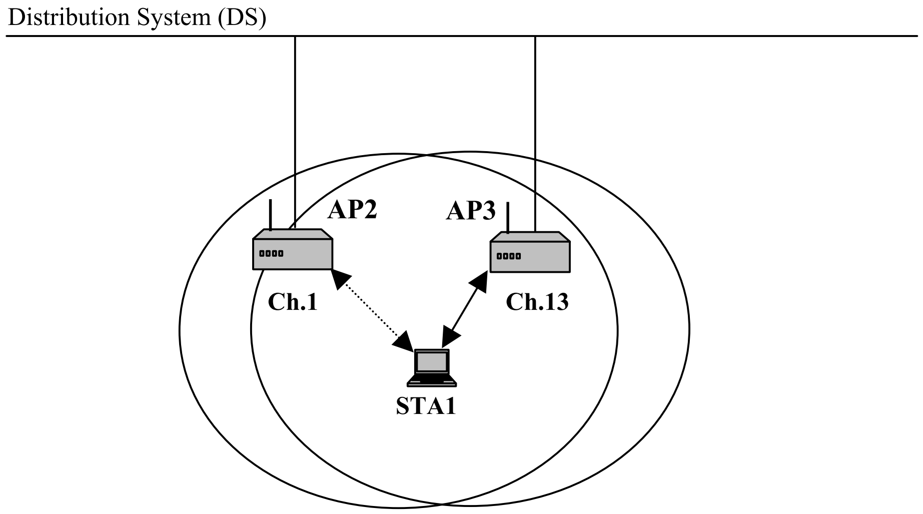


Figure 23: Test-bed configuration to test the behaviour of the LDS

The station is initially associated with AP3 (solid line in Figure 23). The station generates UDP traffic with a bit rate of 0.5 Mbps (125 messages/s, 500 bytes/message) and its nominal bit rate is 11 Mbps. In all tests, the station's traffic pattern is periodic and the source is the station (uplink traffic). A simulated load is generated in both APs in order to produce the distribution of STA1. In AP3, the simulated load is 0.5 Mbps and is constant. In AP2, the load is dynamic and it varies periodically every 10 s between 5 and 0.1 Mbps. In this way, when the load in AP2 is 5 Mbps and 0.5 Mbps in AP3, the station is associated with AP3. On the other hand, when the load in AP2 is 0.1 Mbps, AP3 transfers the station so it reassociates with AP2 (dotted line in Figure 23). Therefore, the station is being distributed continuously while the test is running.

5.1.3. Results

We have generated three different figures that describe the behaviour of our LDS. The first one (see Figure 24), shows the instant balance index value versus the time. The second one (see Figure 25), indicates with which AP the STA1 is associated and the handover transitions. Finally, the third one (see Figure 26) shows the effects of the load distribution on the throughput per AP in comparison with a WLAN without our LDS.

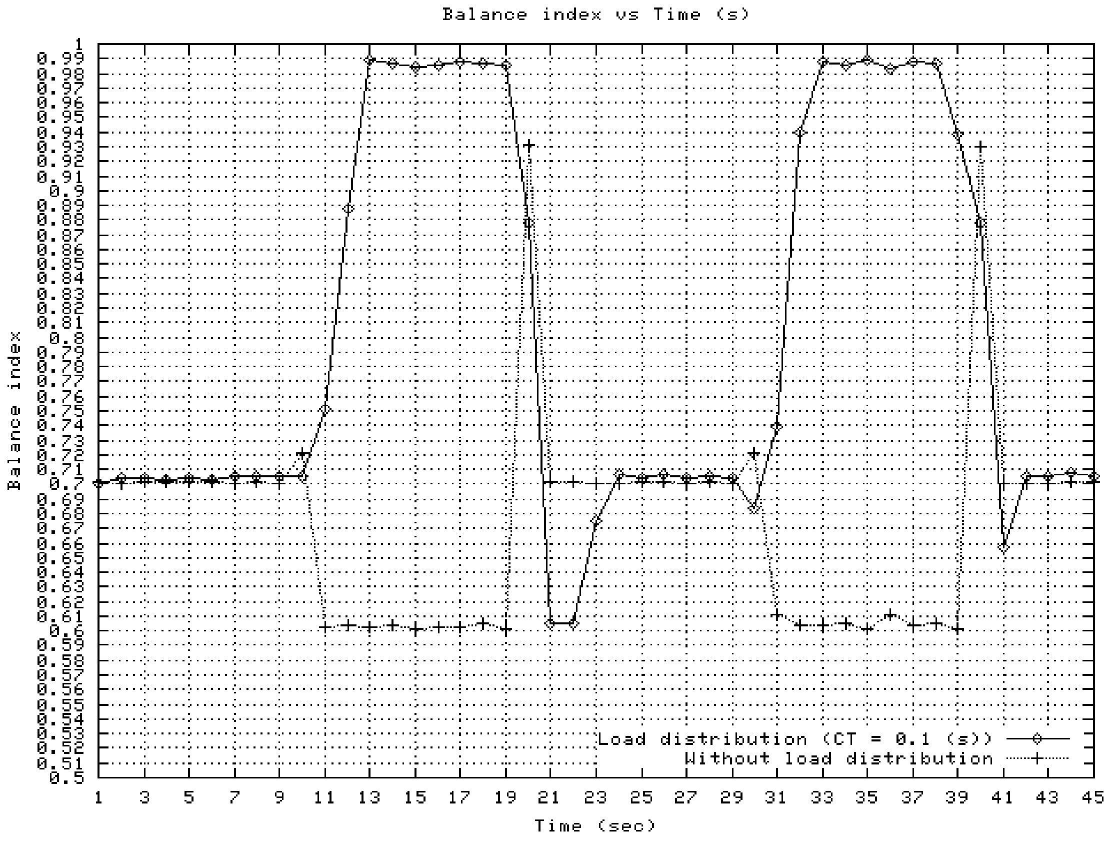


Figure 24: Balance index vs. time (s)

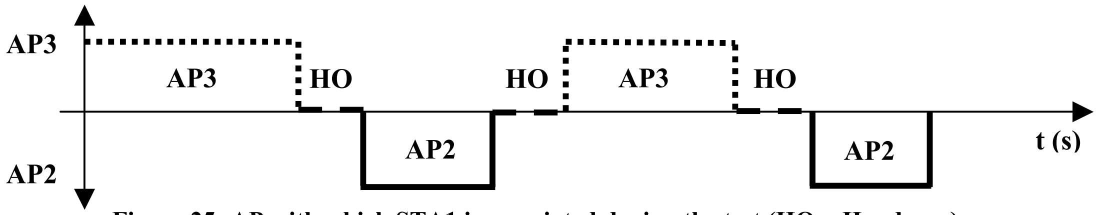


Figure 25: AP with which STA1 is associated during the test (HO = Handover)

According to Figure 24 and Figure 25, during the first ten seconds both systems (with and without load distribution) have the same performance (β=0.7). This is correct since the (simulated) traffic of AP2 is 5 Mbps until t=10 s and AP3's traffic is 1 Mbps. Because the station is associated with AP3, the LDS does not take any decision. From t=10 s to t=20 s, the load in the network varies: AP2's traffic decreases to 0.1 Mbps. Therefore, the LDS in AP3 decides to disassociate the station, that reassociates with AP2 balancing the load among the APs. The result of this distribution decision is that the balance index increases almost to the maximum (β=0.99). Thus, in comparison without load distribution (β=0.6) the gain is around 40%. Finally, it is also important to note that it takes roughly 3 s to increase the balance index from 0.7 to 0.99. This delay is mainly due to the handover time.

The third figure, Figure 26, shows the throughput per AP versus time with and without LDS. As it can be seen, the throughput of AP2 is increased by the station traffic every time AP3 takes the decision to disassociate the station. Note also that the system is stable since the station does not handover back to AP3 (see Figure 26, from t=13 to t=19) because the LDS determines that this decision will not improve the overall performance.

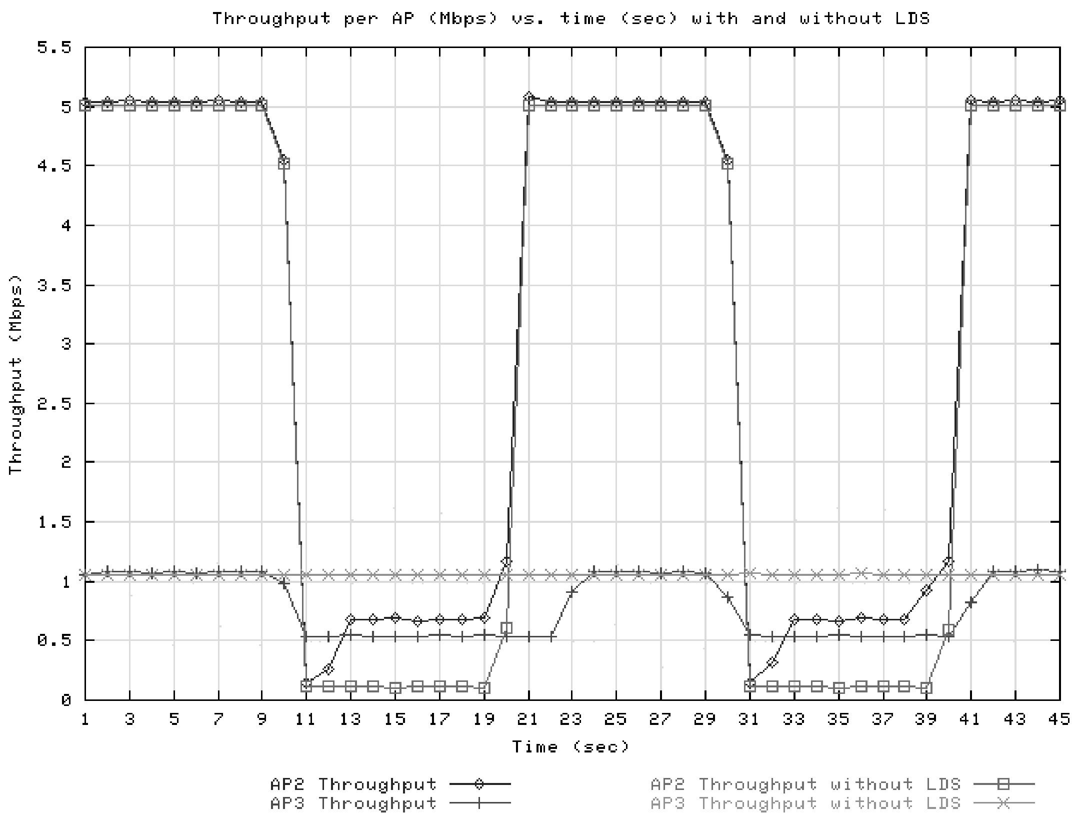


Figure 26: Throughput of AP2 and AP3 with and without load distribution

5.1.4. Conclusions

The LDS balances the load among APs only when it is necessary, i.e., when AP2's traffic decreases. Moreover, the LDS increases the balance index of the system although its gain depends on the particular network state. In the test, the maximum achieved gain is around 40% while the average gain is 14%. The tests have also showed that the load distribution is stable in the sense that STA1 always remains associated with the lower loaded AP.

Finally, it is possible to see that the station takes between 2 and 3 s in order to transmit data once is disassociated. The main contribution to this delay is the handover time, which affects the number of distribution decisions per minute but not the behaviour of the LDS.

5.2. Packet delay measurement

5.2.1. Description

This test demonstrates that the LDS can decrease the average packet delay of the stations. The test consists of two APs and two stations that are initially associated with the same AP (AP3). We compare the packet delay in a WLAN without load distribution (both stations remain associated to the same AP) with a WLAN with our LDS. In the last case, the LDS distributes one station in order to balance the load among the APs.

5.2.2. Test-bed configuration

The configuration of the test-bed is shown in Figure 27:

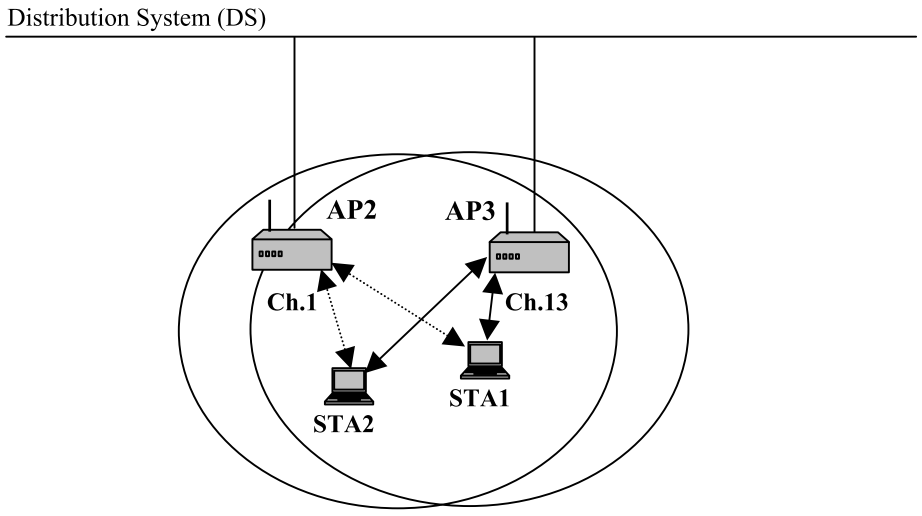


Figure 27: Test-bed configuration to measure packet delay

Initially, STA1 and STA2 are associated with AP3. The stations generate UDP traffic with the same bit rate of 2 Mbps (500 messages/s, 500 bytes/message) and their nominal bit rate is 11 Mbps. In all tests, the station's traffic pattern is periodic and the source is the station (uplink traffic).

5.2.3. Results

The results of this test are displayed in two figures: the first figure (see Figure 28), shows the packet delay for STA1 without and with load distribution. The second figure (see Figure 29), shows STA1's throughput without and with load distribution.

From Figure 28, we notice that packet delay without load distribution (i.e., both stations transmitting via AP3) is 0.45 s in average and it remains constant. This high latency value can be explained knowing that the bit rate of both stations was decreasing (from 11 Mbps to 5.5 and then to 2 Mbps) and increasing continuously during the test. It is possible to distinguish three different phases in Figure 28 when the LDS is applied. The first phase starts when AP3 transfers STA1 (t=13.6 s). Then, the station starts the reassociation procedure until it reassociates with AP2 (t=17 s). The consequence of this handover time is the big peak showed in Figure 28. During the second phase, that goes from t=17 s up to t=21 s, the packet delay decreases considerably (from 0.45 s to 0.15 s in average) but without achieving instantaneously the minimum average value. The reason is two folded: first, the station does not transmit at the maximum rate when it is just reassociated and second, some packets were buffered by the kernel since the wireless interface was not ready to accept more packets during the handover process. Finally, in the third phase (from t=21 s) the packet delay is reduced to 8 ms in average.

The effects of load distribution on the STA1's throughput can be seen in Figure 29. According to this figure, without load distribution STA1 only achieves a throughput of 1.5 Mbps. On the other hand, when load distribution is applied its throughput is increased up to 2 Mbps when STA1 reassociates with AP2.

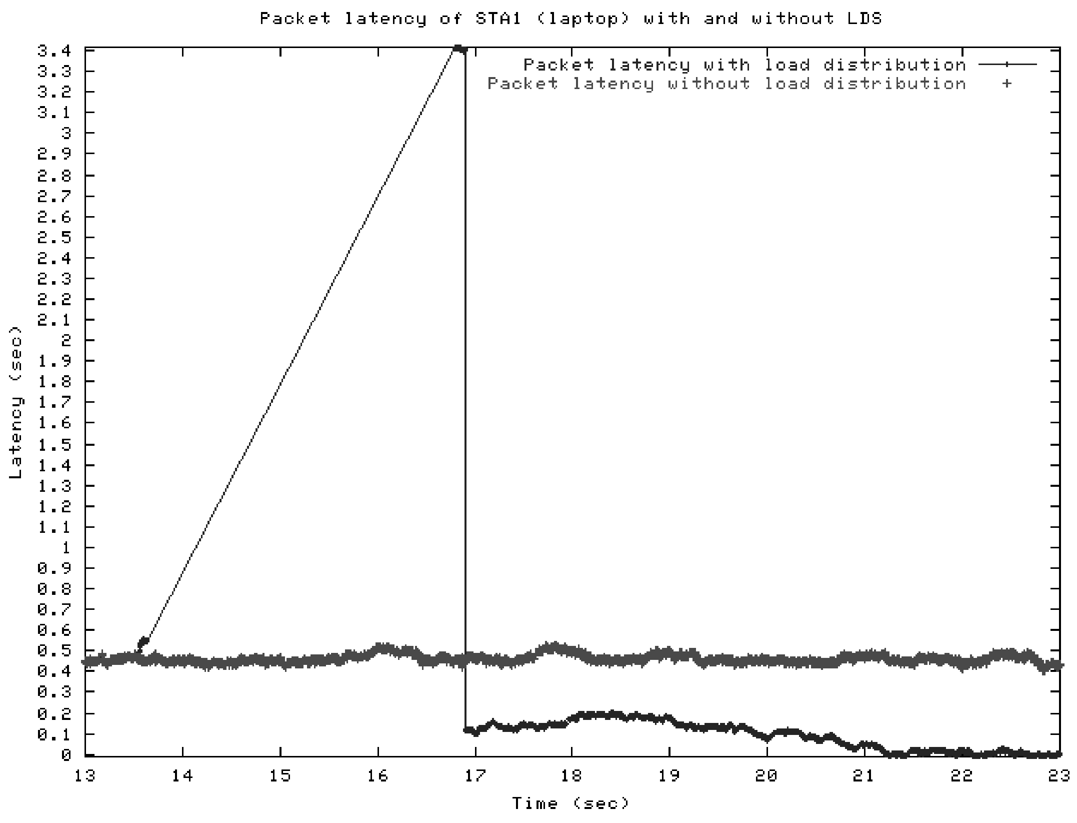


Figure 28: Packet latency of STA1 with and without LDS

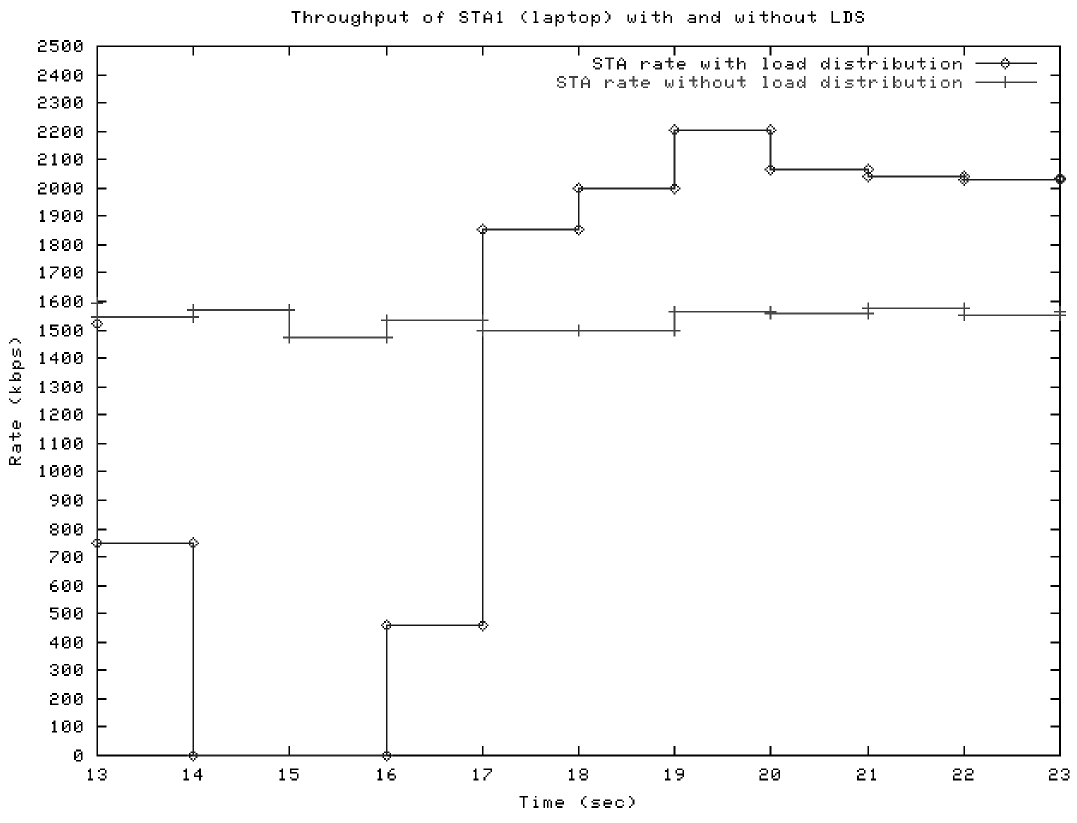


Figure 29: Throughput of STA1 with and without LDS

5.2.4. Conclusions

We can derive two important results from this test. First, distributing a station from an overloaded AP to a lower loaded one can decrease the average packet delay of the stations. Second, this distribution can also produce an increment of the throughput of the stations compared with a system without LDS. In the test, STA1 has increased its total throughput in 500 Kbps compared with a system without LDS. Thus, the LDS has increased the total throughput in the WLAN from 3 Mbps (since both stations transmit at 1.5 Mbps) without LDS to 4 Mbps (both stations transmitting at 2 Mbps) with LDS.

5.3. Location policy performance

5.3.1. Description

In this test, we study how the performance of the system is affected when there are two or more APs that are able to simultaneously distribute its stations. In this situation, a *location policy* can be applied to force stations to reassociate with receiver APs. To determine whether a location policy is needed or not, we study the behaviour of the LDS without applying a location policy, thus leaving the stations the possibility to reassociate with sender APs. For this test, we set up a WLAN with three APs and two stations. Thus, we create a scenario where two APs are senders and there is only one receiver AP.

5.3.2. Test-bed configuration

The configuration of the test-bed is shown in Figure 30:

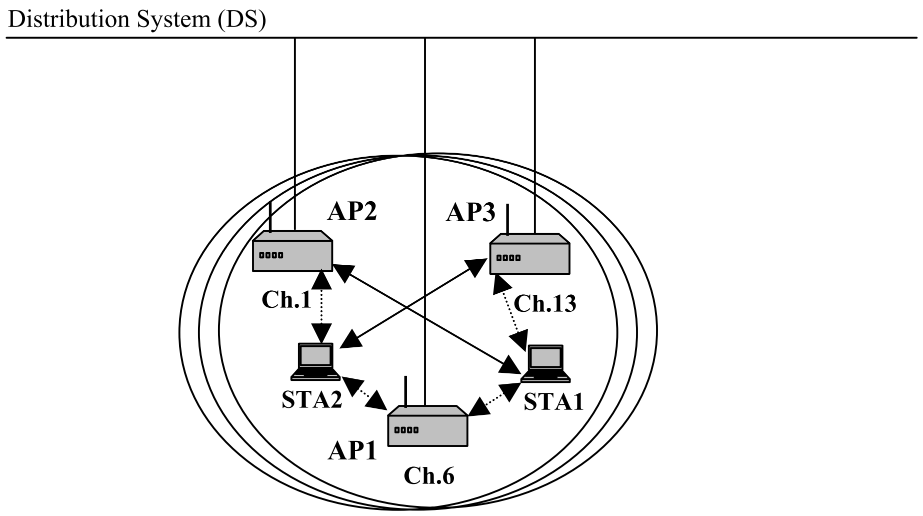


Figure 30: Test-bed configuration to test the transfer policy performance

Initially, STA1 is associated with AP2 and STA2 with AP3. The stations generate UDP traffic with the same bit rate of 0.1 Mbps (25 messages/s, 500 bytes/message) and their nominal bit rate is 11 Mbps. In all tests, the station's traffic pattern is periodic and the source is the station (uplink traffic). A simulated load is generated in all APs in order to produce the distribution of the stations. In AP3 and AP2, the simulated load is 2 Mbps and is constant. In AP1, the load is dynamic and varies periodically every 20 s between 5 and 0.5 Mbps. In this way, when the load in AP1 is 0.5 Mbps (AP1 is a receiver), AP2 and AP3 (both senders) will distribute the stations to achieve a state where STA1 and STA2 are associated with AP1. On the other hand, when traffic in AP1 changes to 5 Mbps it becomes a sender while the others, AP2 and AP3, become receivers. Thus, AP1 will disassociate both stations to balance the traffic.

5.3.3. Results

The results of this test are displayed in two figures. In both figures, we compare the performance of the system with and without location policy. The first figure, Figure 31, shows the throughput of STA1. The second one, Figure 32, shows the total throughput in the network.

Figure 31 shows the implications of the location policy on STA1's throughput. When the load changes in AP1 (from 5 to 0.5 Mbps), it becomes a receiver while AP2 and AP3 become senders. Therefore, AP2 and AP3 disassociate its stations (STA1 and STA2) to balance the load. Without location policy, STA1 can choose to reassociate with AP1 (receiver), AP3 (sender) or AP2 (sender). In this test, STA1 chose to reassociate only with AP2 and AP3 but not with AP1. As a result, AP2 and AP3 continuously disassociated STA1 since both were senders at that time. The effect of successive rejections can be seen in Figure 31, from t=25 to t=38 s. Thus, during 13 seconds STA1 is being rejected by AP2 and AP3. As a consequence, STA1 reduces its throughput until successfully reassociates with AP1 at t=38. On the other hand, when location policy is applied STA1 reassociates with AP1 after the handover delay. Thus, STA1's throughput has only been interrupted by the handover delay.

Figure 32 shows how the total throughput increases compared to the case without applying the location policy. In average, from t=22 to t=39 s, the total throughput is 4.639 Mbps without location policy and 4.689 Mbps with it. Therefore, total throughput has increased 0.05 Mbps during this period applying our proposed location policy.

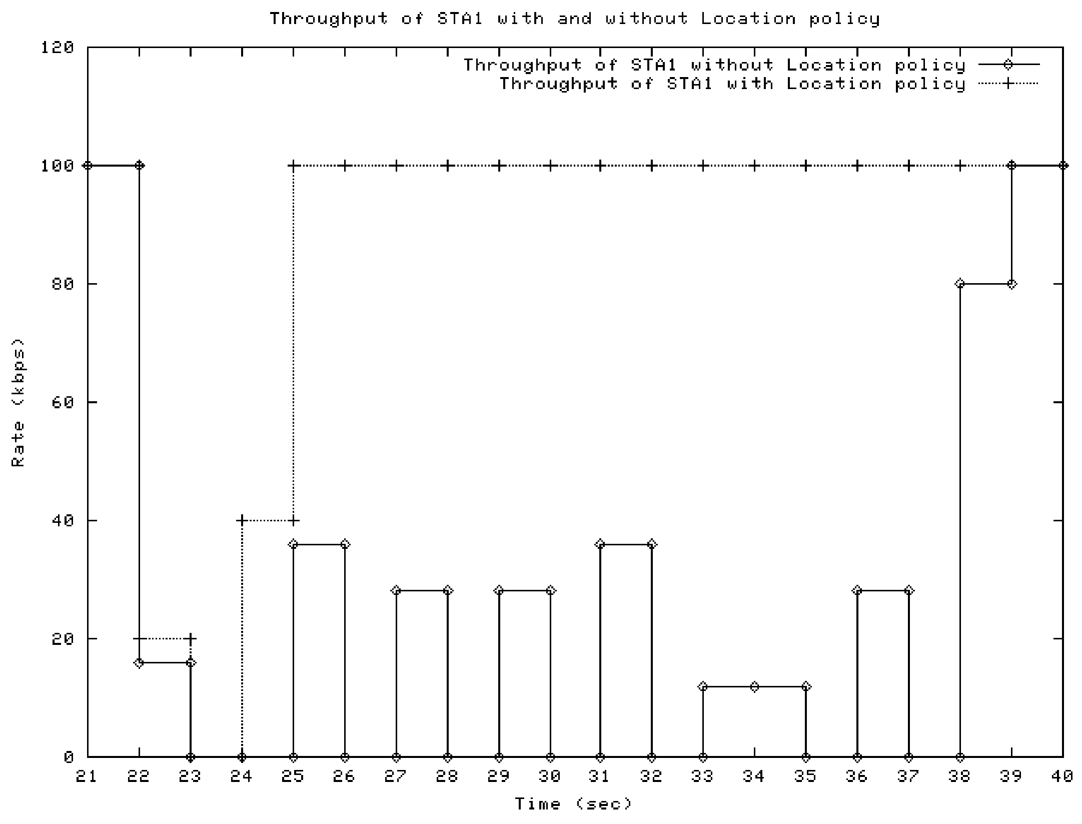


Figure 31: Throughput of STA1 with and without location policy

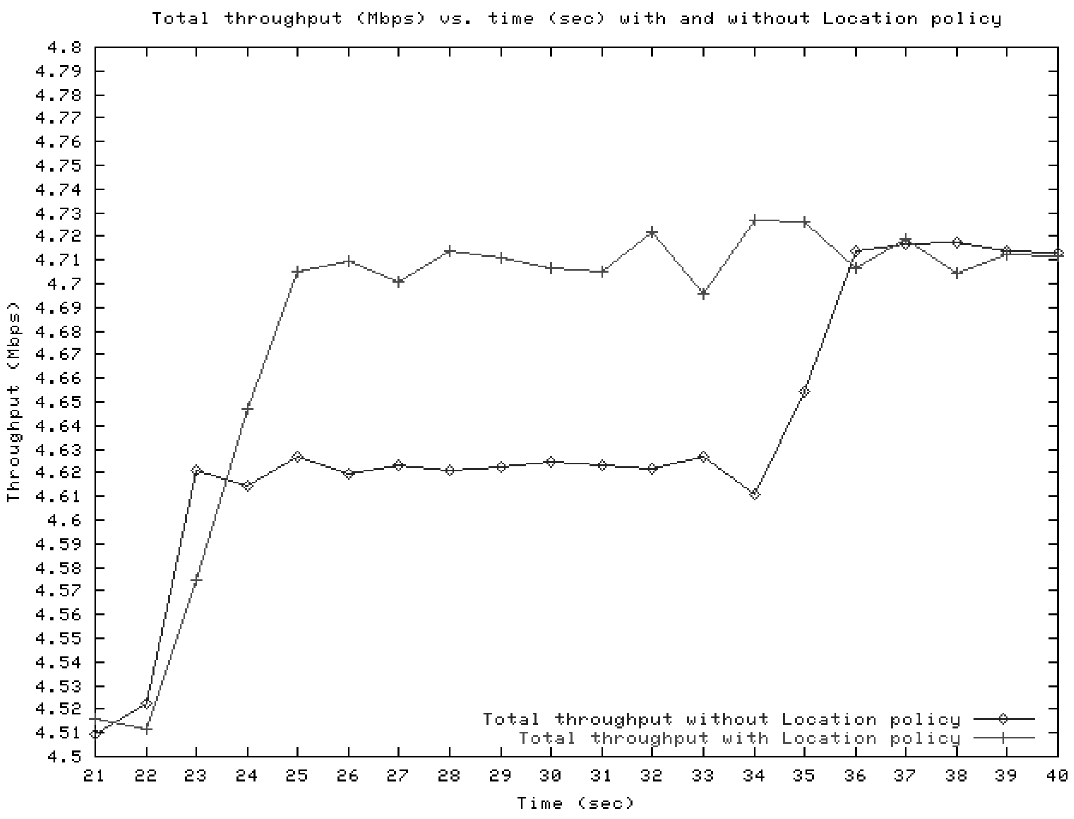


Figure 32: Total throughput with and without location policy

5.3.4. Conclusions

These results draw some interesting points. First, the importance of a location policy so the stations reassociate, at the first time, with APs that are receivers. Without a location policy, in the worst case, a particular station may never choose a receiver AP producing consequently a decrement of the total throughput in the network. Applying a location policy increases the throughput of the distributed station and therefore the total throughput in the network. Thus, the need of a location policy where the senders APs reject associations of new stations has been proved.

5.4. Distribution time measurement

5.4.1. Description

One important issue is the amount of time the LDS needs to distribute the load among APs after a traffic variation. This time, called distribution time, depends on the handover delay and on the computational time of the LDS. In this test, we determine the contribution of each one of these factors to the distribution time.

5.4.2. Test-bed configuration

The configuration of the test-bed is shown in Figure 33:

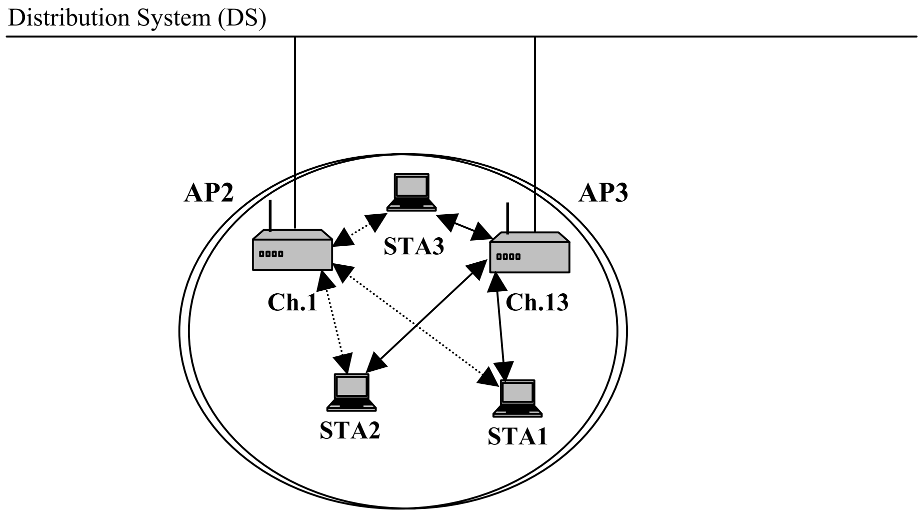


Figure 33: Test-bed configuration to determine the distribution time

Initially, all stations are associated with AP3. The stations generate UDP traffic at the same bit rate of 0.1 Mbps (25 messages/s, 500 bytes/message) and their nominal bit rate is 11 Mbps. In all tests, the station's traffic pattern is periodic and the source is the station (uplink traffic). A simulated load is generated in both APs in order to produce the distribution of the stations. In AP3, the simulated load is 1 Mbps and is constant. In AP2, the load is dynamic and varies periodically every 10 s between 1 and 0.4 Mbps. With this configuration, the LDS will distribute all stations from AP3 to AP2 in order to balance the load.

5.4.3. Results

First, we show in Figure 34 the balance index versus the time. This figure illustrates the total distribution time after a load variation. At time t=10 s, the load in AP2 decreases, from 1 Mbps to 0.4 Mbps, so AP2 starts to transfer its associated stations to balance the load. As it can be seen, the total time needed to completely balance the network (i.e., to achieve β=1) is roughly 6 s. Second, we calculate the computational time of the LDS. It comprises the cycle time (CT) plus the amount of time the LDA needs to take a distribution decision. The CT in our prototype is 100 ms and we have measured experimentally that LDA takes 0.4 ms to take a decision. The longest computational time is produced when last distribution decision is taken after a traffic variation. Thus, in the worst case the computational time it will be equal to 100.4 ms, which is much shorter than handover delay. This result demonstrates that handover time (around 2 s per station) is the main contribution to distribution time.

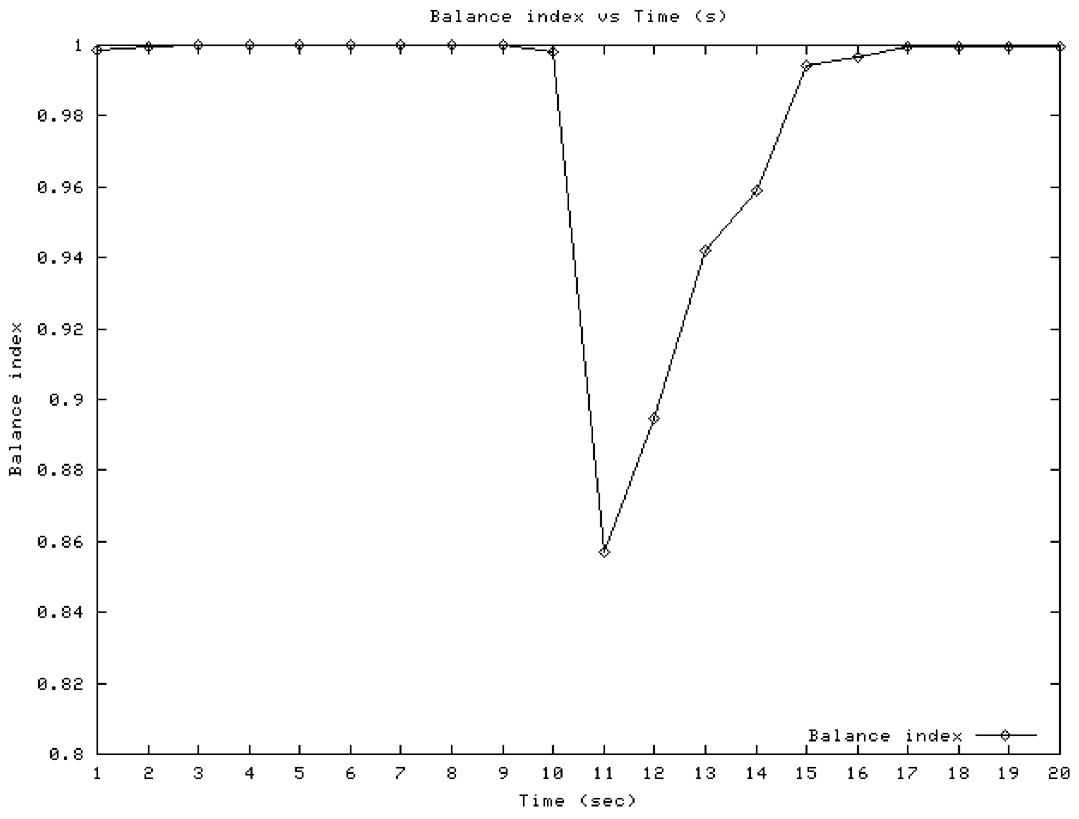


Figure 34: Balance index vs. time that shows the distribution time

5.4.4. Conclusions

The results show how the distribution time depends on the handover time and on the computational time of the LDS. As it has been shown, the handover time (around 2 s) is the main component of the distribution time since it is much shorter than the computational time, which is equal to 100.4 ms in the worst case.

6. Conclusions

6.1. Summary

In this thesis, we designed and evaluated a new group of mechanisms, called Load Distribution System (LDS) that distribute load among APs with overlapped coverage. An essential point in this work is that overall network utilization can be increased based on adding load distribution mechanisms to the IEEE 802.11 standard. These mechanisms consider throughput measurements at each AP (and from its associated users) rather than solely the number of users per AP. The main reason is that in WLAN user's traffic pattern is dynamic. Therefore, there is no correlation between the number of users associated per AP and its traffic.

From the related work, we identified two groups of issues that any load distribution scheme should deal with: architectural and algorithmic issues. Architectural issues consider aspects related with the type of control, load metrics to be used, etc. Algorithmic issues specify the behaviour of algorithms. To address these issues, we designed a new scheme, following a top-down approach, to distribute the load in WLANs. It runs locally at each AP, it is transparent to the stations and it does not require any modification to the standard.

To study the performance of the LDS, we implemented and tested it in an experimental prototype. First, three initial tests were done to set important parameters of the algorithms: the handover time, the sampling time to get traffic measurements and the reactivity of the algorithms. Then, we proceed with the tests that showed the capabilities of the LDS to distribute throughput among neighbour APs. In these tests, we showed the capacity of the LDS to balance load, packet delay and total throughput performance, location policy performance and required distribution time to achieve a balanced scenario.

6.2. Discussion of the results

We can conclude, based on the experimental results, that WLANs would benefit from applying our proposed mechanisms to distribute load among APs. When our LDS is applied to a WLAN in which the traffic pattern is typically dynamic, the results show that the balance index increases. It has also been showed that load distribution is stable in the sense that a transferred station is not continuously distributed. This is achieved by analysing costly handovers to determine if a new distribution will increase the performance of the network.

The experimental results show that the LDS can decrease average packet delay and increase total throughput. First, the LDS decreases average packet delay of a station when it is transferred from an overloaded AP to another less loaded. In this case, the delay decreased by 98% since the competition for the channel was lower in the less loaded AP. Second, the distribution of the station increased by 25% its throughput. As a consequence, total throughput increased by 25%. Therefore, an important effect of load distribution is the increment of total throughput.

In order to achieve a stable scenario, we have tested two different strategies: a LDS without and with location policy. Without location policy, a distributed station is not forced to associate with a receiver AP. The experimental tests with the prototype show that most of the times stations do not select a receiver AP but a sender one. This produces an unstable situation where a station is being continuously distributed until it reassociates with a receiver AP. During this time, station's throughput is drastically reduced compared to a system with location policy. This policy forces the distributed station to reassociate with a receiver AP avoiding sender APs. The results show that with this location policy, the station always reassociates with a receiver AP without decreasing its throughput.

Finally, we have studied which factors affect the time LDS needs to achieve a balanced scenario (called distribution time). This time is composed of two parts: the amount of time the LDS spends to take a decision plus the amount of time a station needs to reassociate. The results show that the main contribution to distribution time is due to handover delay (which is 2 s approximately) since the computation time of the LDS only adds, in the worst case, 100.4 ms.

7. Future work

This thesis has investigated most of the important points related to load distribution in WLANs. Nevertheless, there are still some interesting issues that may be worth to investigate in future work:

- Increase the scope of our load distribution (from local to wide scope) in order to distribute load among APs that have partial overlapped coverage areas. It may be very interesting to implement and test the proposed mechanism, based on active scanning, to detect if a station can be safely disassociated. Thus, to detect from the AP if a station can associate with more than one AP at any time.

- Study alternative load metrics, such as the number of competing stations that access the channel. It would be interesting to compare different load metrics to look at its influence on the distribution performance.

- Implement and study a centralize control architecture where a central node collects the state information of the APs (via SNMP) and takes distribution decisions based on this information. It would be interesting to compare the performance of this central control with a distributed one such as the designed in this thesis.

8. References

[1] *IEEE 802.11 Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999 edition.

[2] J. Lansford, A. Stephens, R. Nevo, "Wi-Fi (802.11b) and Bluetooth: enabling coexistence," *IEEE Network*, vol. 15, pp. 20-27, Sep/Oct 2001.

[3] J. Williams, "The IEEE 802.11b security problem. I," *IT Professional*, pp. 95-96, Nov/Dec 2001.

[4] G. Bianchi, I. Tinnirello, "Kalman Filter Estimation of the Number of Competing Terminals in an IEEE 802.11 network," *IEEE INFOCOM 2003*, April 2003.

[5] D. Tang, M. Baker, "Analysis of a Local-Area Wireless Network," *Proc. ACM MobiCom'00*, pp. 1-10, August 2000.

[6] Y. Fang, Y. Zhang, "Call Admission Control Schemes and Performance Analysis in Wireless Mobile Networks," *IEEE Transactions on Vehicular Technology*, vol. 51, no. 2, March 2002.

[7] A. Balachandran, G. M. Voelker, P. Bahl, P.V enkat Rangan "Characterizing User Behaviour and Network Performance in a Public Wireless LAN," *Proc. ACM SIGMETRICS'02*, Marina Del Rey, June 2002.

[8] G. Bianchi, I. Tinnirello, "Improving Load Balancing mechanisms in Wireless Packet Networks," *IEEE International Conference on Communications*, vol. 2, pp. 891-895, April 2002.

[9] A. Balachandran, P. Bahl, G. M. Voelker, "Hot-Spot Congestion Relief in Public-area Wireless Networks," Proceedings Fourth IEEE Workshop on *Mobile Computing Systems and Applications*, pp. 70-80, June 2002.

[10] J. Antonio García-Macías, F. Rousseau, G. Berger-Sabbatel, L. Toumi, A. Duda, "Quality of Service and Mobility for the Wireless Internet," *Proceedings of the first workshop on Wireless mobile internet*, pp. 34-42, 2001.

[11] A. Veres, A. T. Campbell, M. Barry, L. Sun, "Supporting Service Differentiation in Wireless Packet Networks Using Distributed Control," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 10, October 2001.

[12] Cisco Systems Inc. Data Sheet for Cisco Aironet 350 Series Access Points, June 2001.

[13] Proxim Inc. Data Sheet for ORiNOCO AP-1000 Access Point, 2001.

[14] IEEE 802.11 WG, "*Draft Supplement to standard for Telecommunications and Information Exchange Between Systems- LAN/MAN Specific Requirements - Part 11: Wireless Medium Access Control (MAC) and physical layer (PHY) specifications: Medium Access Control (MAC) Enhancements for Quality of Service (QoS)*," IEEE 802.11e/D2.0, Nov. 2001.

[15] IEEE 802.11a-1999, "Supplement to IEEE Standard for Information technology. Telecommunications and information exchange between systems. Local and metropolitan area networks. Specific requirements," 1999 edition.

[16] Kapp, S., "802.11: Leaving the Wire Behind", *IEEE Internet Computing*, vol. 6, pp. 82-85, Jan.-Feb. 2002.

[17] Clark, M.V., Leung, K.K., McNair, B., Kostic, Z., "Outdoor IEEE 802.11 cellular networks: radio link performance," *IEEE International Conference on Communications*, vol. 1, pp. 512-516, May 2002.

[18] R. Pow, “Public Wireless LAN Access: Market Forecasts,” *Analysis*, November 2001.

[19] IEEE Std. 802.11f/D3, “*Draft Recommended Practice for Multi-Vendor Access Point Interoperability via an Inter-Access Point Protocol Across Distribution Systems Supporting IEEE 802.11 Operation*,” January, 2002, work in progress.

[20] A. El-Hoiydi, “Implementation Options for the Distribution System in the 802.11 Wireless LAN Infrastructure Network,” *IEEE International Conference on Communications*, vol. 1, pp. 164-169, June 2000.

[21] T. Kanter, G. Q. Maguire Jr., A. Escudero-Pascual, “802.11b Handover Measurement,” November 2001.

[22] J.D. Case, M. Fedor, M.L. Schoffstall, C. Davin. “Simple Network Management Protocol (SNMP),” RFC 1098. April 1989.

[23] S. Garg, M. Kappes, M. Mani, “Wireless Access Server for Quality of Service and Location Based Access Control in 802.11 Networks,” *Proceedings of the Seventh International Symposium on Computers and Communications (ISCC’02)*, pp. 819 – 824, July 2002.

[24] Kantorovitch, Z. Shelby, T. Saarinen, P. Mähönen, "Wireless SNMP", *INET 2001*, June 5 - 8, Stockholm, Sweden, CD-ROM

[25] M. Murray, “Measuring the Immeasurable: Global Internet Measurement Infrastructure,” *Cooperative Association for Internet Data Analysis – CAIDA*, 2001.

[26] Balachander K., Jia W., Yinglian X, “Early measurements of a cluster-based architecture for P2P systems,” *Proceedings of the First ACM SIGGCOMM Workshop on Internet Measurement Workshop 2001*, San Francisco, California, USA.

[27] Matei, R. Iamnitchi, A. Foster, P., “Mapping the Gnutella network,” *IEEE Internet Computing*, pp. 50-57, vol. 6, Jan.-Feb. 2002.

[28] N. G. Shivaratri, P. Krueger, M. Singhal, “Load Distributing for Locally Distributed Systems,” *Computer*, pp. 33-44, vol. 25, Dec 1992.

[29] R. Riedl, L. Richter, “Classification of load distribution algorithms,” *Proceedings of the Fourth Euromicro Workshop on Parallel and Distributed Processing*. PDP '96. Jan 1996.

[30] Chiu, Dah-Ming, Jain, “Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks,” *Journal of Computer Networks and ISDN*, vol. 17, pp. 1-14, June 1989.

[31] “1000BASE-T: Gigabit Ethernet over Category 5 Copper Cabling” Technical Fundamentals, *By Philippe Ginier-Gillet, 3Com Corporation, and Christopher T. Di Minico, Cable Design. Technologies Corporation. 3COM.*

[32] M. G. Arranz, R. Agüero, “Behaviour of UDP-Based Applications over IEEE 802.11 Wireless Networks,” *12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pp. F-72 -F-77, vol.2, 2001.

9. Appendices

9.1. Appendix – A: Acronyms and abbreviations

3G	Third Generation wireless networks
AP	Access Point
BS	Base Station
BSS	Basic Service Set
CAC	Call Admission Control
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
DCF	Distributed Coordination Function
DEP	Decision Enforcement Point
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Server
DS	Distribution System
DSSS	Direct Sequence Spread Spectrum
ESS	Extended Service Set
FTP	File Transfer Protocol
GPRS	General Packet Radio Service
HTTP	Hypertext Transport Protocol
IAPP	Inter-Access Point Protocol
IBSS	Independent Basic Service Set
IP	Internet Protocol
ISO	International Organization for Standardization
LD	Load Distribution
LDA	Load Distribution Algorithm
LDC	Load Distribution Controller
LDS	Load Distribution System
LLC	Link Layer Control
MAC	Medium Access Control
MIB	Management Information Base
MLME	MAC sublayer Management Entity
MM	Metric Monitor
MSDU	MAC Service Data Unit
OSI	Open Systems Interconnection
P2P	Peer-to-Peer
PCF	Point Coordination Function
PHY	Physical Layer
PLME	Physical Layer Management Entity
QoS	Quality of Service
SAP	Service Access Point
SIG	State Information Storage
SME	Station Management Entity
SMT	Station Management
SNMP	Simple Network Management Protocol
SNR	Signal to Noise Ratio
SP	Selection Policy
STA	Mobile station
TCP	Transport Control Protocol
TIM	Traffic Indication Map
TP	Transfer Policy
UDP	User Data Protocol
VoIP	Voice over IP
WECA	Wireless Ethernet Compatibility Alliance
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network

9.2. Appendix – B: Code of the load distribution system

```
/*Program: LDC.c
*Description: Load Distribution Controller (LDC) is the module in charge of taking all the decisions related with
*load distribution, based on load metrics that are obtained from the State Information Storage (SIG) module. It also
*includes the functions from the Decision Enforcement Point (DEP).
*/

#include "../include/ldc.h"

/* Comparison functions */
int compare_int(int a, int b)
{
  int temp = a - b;
  if(temp > 0) return 1;
  else if (temp < 0) return -1;
  else return 0;
}
int metric_compare(const struct load_metric *metric_sta1,const struct load_metric *metric_sta2)
{
  return(compare_int(metric_sta1->metricvalue,metric_sta2->metricvalue));
}
/**** Get the number of associated stations from the SI **********************/

/*-----------------------------------------------------------------------------------------------*/
int get_sta_number(void)
{
  FILE *lm;
  int N=0;
  char *lm_name = "lm4", char buffer[100];

  if((lm = fopen(lm_name,"r"))== NULL) perror(lm_name);
  while(fgets(buffer,sizeof(buffer),lm) != NULL){N++;}
  fclose(lm);
  return(N);
}
/******************** Get load metrics from the SIG **************************/

/*--------------------------------------------------------------------------------------------*/
int get_load_metric(struct load_metric *metric_ap_or_sta, int N, int metric_type)
{
  FILE *lm, *lm_other; /* FILE structure to open load metric file */
  char buffer[100], hwa[128], *lm_name, *other_ap_metric;
  int lm_values[MAX_NUMBER_AP], ap_or_sta, lm_sum, lm_value, i, count;
  struct load_metric metric2[N];

  switch(metric_type)
   {
   case LM1:
    {
          lm_name = "lm1";
          other_ap_metric = "lm_others";
          break;
    }
   case LM4:
    {
          lm_name = "lm4";
          break;
    }
   }
  /* Open file descriptor to get the load metric of the AP */
  if((lm = fopen(lm_name,"r"))== NULL) perror(lm_name);
  if(metric_type == LM1)
    {
     /* Open file descriptor to get the load metrics from the other APs in the ESS */
     if((lm_other = fopen(other_ap_metric,"r"))== NULL) perror(other_ap_metric);
      /* Read the load metric from the local metric */
      fgets(buffer,sizeof(buffer),lm);
      sscanf(buffer,"%s %d",&metric_ap_or_sta[ap_or_sta].ieee_addr,&metric_ap_or_sta[ap_or_sta].metricvalue);
      /* Increment counter */
      ap_or_sta ++;
      /* Read load metric from other APs in the ESS */
      while(fgets(buffer,sizeof(buffer),lm_other) != NULL)
            {
              sscanf(buffer,"%s
%d",&metric_ap_or_sta[ap_or_sta].ieee_addr,&metric_ap_or_sta[ap_or_sta].metricvalue);
              ap_or_sta++;
             }
      if(fclose(lm_other) != 0) perror("Error closing file");
   }
  if(metric_type == LM4)
   {
     /* Read load metric from stations associated with the AP */
     while(fgets(buffer,sizeof(buffer),lm) != NULL)
            {
             sscanf(buffer,"%s
%d",&metric_ap_or_sta[ap_or_sta].ieee_addr,&metric_ap_or_sta[ap_or_sta].metricvalue);
             sscanf(buffer,"%s %d",&metric2[ap_or_sta].ieee_addr,&metric2[ap_or_sta].metricvalue);
             ap_or_sta++;
            }
     count = sizeof(metric2) / sizeof(struct load_metric);
     /* Order the stations from min. to max. metric value */
     qsort(metric2,count,sizeof(struct load_metric),metric_compare);
     for(i=0;i<ap_or_sta;i++)
            {
             sprintf(buffer,"%s",metric2[i].ieee_addr);
             sscanf(buffer,"%s",&metric_ap_or_sta[i].ieee_addr);
             sprintf(buffer,"%d",metric2[i].metricvalue);
             sscanf(buffer,"%d",&metric_ap_or_sta[i].metricvalue);
            }
   }
  /* Close file descriptors */
  if(fclose(lm) != 0) perror("Error closing file");
  return(ap_or_sta);
}
/*** Time function to wait a determined amount of time with us precision *****/

/*-----------------------------------------------------------------------------------------------------*/
int usecsleep(int secs, int usecs)
{
  struct timeval tv;
  tv.tv_sec = secs;
  tv.tv_usec = usecs;
  return select(0, NULL, NULL, NULL, &tv);
}
/******************************* compute_beta ********************************/

/*------------------------------------------------------------------------------------------------*/
float compute_beta(struct load_metric *metric_ap, int number_ap_ESS)
{
  float metric_ap_kbytes, sum_beta = 0, sum_beta_square = 0, beta;
  int i;
  FILE *betafd; /* FILE structure to save the beta values of the AP */

  /* Open file descriptor to write Beta */
  if((betafd = fopen("beta","a"))== NULL)
   {
     perror("beta");
     exit(-1);
  }
  syslog(LOG_INFO,"compute_beta: COMPUTING BETA VALUE...");
  for(i=0;i<number_ap_ESS;i++)
   {
     metric_ap_kbytes = ((float )(metric_ap[i].metricvalue))/(1000);
     sum_beta = sum_beta + metric_ap_kbytes;
     sum_beta_square = sum_beta_square + (metric_ap_kbytes*metric_ap_kbytes);
   }
  if(sum_beta_square == 0 || number_ap_ESS == 0) beta = 0;
  beta = (sum_beta*sum_beta)/(number_ap_ESS*(sum_beta_square));
  fprintf(betafd,"%f",beta);
  fputs("\n",betafd);
  fclose(betafd);
  return(beta);
}
/******************************* Transfer_policy ********************************/

/*------------------------------------------------------------------------------------------------*/
int transfer_policy(struct load_metric *metric_ap, int number_ap_ESS, float delta)
{
  float load_average, load_sum = 0, trigger_load_sum, min_load_level_to_distribute;
  int load_local_ap, ap;

  syslog(LOG_INFO,"transfer_policy: TRANSFER POLICIY DECISION...");
  /* Get load metric from the local AP: it is always the first one */
  load_local_ap = metric_ap[0].metricvalue;
  /* First, compute the Average Network Load (ANL) */
  for(ap=0;ap<number_ap_ESS;ap++){
    load_sum = load_sum + metric_ap[ap].metricvalue;
  }
  load_average = load_sum/number_ap_ESS;
  /* Second, we compute the minimum load that is necessary to initiate the load distribution */
  trigger_load_sum = (delta/100)*(load_average);
  min_load_level_to_distribute = trigger_load_sum + load_average;
  /* Third, we compare the load of the local AP with min_load_level_to_distribute */
  if(load_local_ap>min_load_level_to_distribute){
    /* Achieved condition to distriute the load */
    return (SENDER);
  }
  else{
    return (RECEIVER);
  }
}
/******************************** distribution_decision ********************************/

/*-----------------------------------------------------------------------------------------------------*/
void distribution_decision(struct distribution_decision *dist_decision, struct load_metric *metric_ap, struct
load_metric *metric_sta, int number_ap_ESS, int N, float beta)
{
  float estimated_beta = 0;
  int ap;

  dist_decision->decision = DO_NOT_DISTRIBUTE;
  syslog(LOG_INFO,"distribution_decision: DISTRIBUTION DECISION...");
  /* First, compute the estimated load of the local AP without the load of the STA */
  metric_ap[0].metricvalue = metric_ap[0].metricvalue - metric_sta[dist_decision->selected_sta_index].metricvalue;
  if( metric_ap[0].metricvalue < 0) metric_ap[0].metricvalue = 0;
  for(ap=1;ap<number_ap_ESS;ap++)
    {
      metric_ap[ap].metricvalue = metric_ap[ap].metricvalue + metric_sta[dist_decision-
>selected_sta_index].metricvalue;
      estimated_beta = compute_beta(metric_ap,number_ap_ESS);
      if(estimated_beta > beta)
           {
             dist_decision->decision = DISTRIBUTE;
             break;
           }
    }
}
/******************************** Selection_and_Distribution_policy ************************************************/

/*-------------------------------------------------------------------------------------------------------------------------------------*/
void selection_and_distribution_policy(struct distribution_decision *dist_decision,struct load_metric *metric_ap,
struct load_metric *metric_sta, int number_ap_ESS, int N, float beta)
{
  float load_average, load_sum = 0, dfa, load_difference[N], smallest;
  int load_local_ap, ap, sta, candidate_sta, smallest_index;

  syslog(LOG_INFO,"selection_and_distribution_policy: SELECTION POLICY...");

  /* Get load metric from the local AP: it is always the first one */
  load_local_ap = metric_ap[0].metricvalue;
  /* First: compute the Average Network Load (ANL) */
  for(ap=0;ap<number_ap_ESS;ap++){
    load_sum = load_sum + metric_ap[ap].metricvalue;
  }
  load_average = load_sum/number_ap_ESS;
  /* Second: compute the distance from the average */
  dfa = load_local_ap - load_average;
  /* Selection policy: select the STA which has the minimum difference from the dfa */
  for(sta=0;sta<N;sta++) load_difference[sta] = abs(dfa - metric_sta[sta].metricvalue);
  smallest = load_difference[0];
  smallest_index = 0;
  /* Get the smallest value */
  for(sta=0;sta<N;sta++)
```

```
    {
      if(smallest > load_difference[sta])
              {
                smallest = load_difference[sta];
                smallest_index = sta;
              }
    }
  /* Selection policy decision */
  dist_decision->selected_sta_index = smallest_index;
  /* Distribution policy: decides whether it is worth or not to distribute the load */
  distribution_decision(dist_decision,metric_ap,metric_sta,number_ap_ESS,N,beta);
}
/******************************* disassociate_station ********************************/

/*-----------------------------------------------------------------------------------------------------*/
void disassociate_station(char *dev, int skfd, char *ieee_addr)
{
  char *priv_command[2]; /* Arguments to the DEP functions */

  /* Private command: Delete mac from the ACL */
  priv_command[0] = "delmac";
  priv_command[1] = ieee_addr;
  set_private(skfd,priv_command,2,dev);

  /* Disassociate selected station */
  priv_command[0] = "kickmac";
  priv_command[1] = ieee_addr;
  set_private(skfd,priv_command,2,dev);
}
/******************************* check_activity_last_decision() *********************/

/*-------------------------------------------------------------------------------------------------------*/
int check_activity_last_decision(void)
{
  FILE *stafd; /* FILE structure to read the file */
  char buffer[3000];
  int active_distrib = 0;

  if((stafd = fopen("last_sta_disassociated","r"))== NULL)
  {
    perror("last_sta_disassociated");
    exit(-1);
  }
  while(fgets(buffer,sizeof(buffer),stafd) != NULL)
  {
    /* If there is a line there is the last STA that has not been yet reassociated */
    active_distrib ++;
  }
  fclose(stafd);
  return(active_distrib);
}
/********************* update_last_sta_disassociation *****************/

/*-----------------------------------------------------------------------------------------*/
void update_last_sta_disassociation(char *sta_ieee_addr, int sta_metricvalue)
{
  FILE *stafd; /* FILE structure to read the file */
  char buffer[3000];

  if((stafd = fopen("last_sta_disassociated","w"))== NULL)
  {
    perror("last_sta_disassociated");
    exit(-1);
  }
  fprintf(stafd,"%s %d",sta_ieee_addr,sta_metricvalue);
  fclose(stafd);
}
/***** LDA: Load Distribution Algorithm ******************************/

/*-----------------------------------------------------------------------------------*/

void lda_A(int skfd, char *dev, float delta, int num_samples, int measurement_time)
{
  int number_ap_ESS, N, ap_state, i, j, k, sta_old, active_distrib = 0;
  float beta;
  struct load_metric metric_ap[MAX_NUMBER_AP], sampled_metric_ap[MAX_NUMBER_AP],
metric_sta[MAX_NUM_STA], sampled_metric_sta[MAX_NUM_STA];
  struct sockaddr hwa_stas[IW_MAX_SPY]; /* IEEE addresses of the STAs */
  struct distribution_decision dist_decision;

  /* Get old number of stations associated with the AP */
  sta_old = get_sta_number();

  /* Get APs with ESS in the ESS and N to know if it is possible to distribute the load in this AP*/
  N = get_sta_number();
  number_ap_ESS = get_load_metric(metric_ap,N,LM1);

  /* Check if the last distribution decision is finished */
  active_distrib = check_activity_last_decision();

  if(active_distrib == 0)    {
     syslog(LOG_INFO, "Executing LDA...");
     if(N > 0 && number_ap_ESS > 1)
          {
           if(num_samples > 1)
            {
             N = get_sta_number();
             syslog(LOG_INFO, "Number of STAs associated with the AP = %d",N);

             /* Init AP and STA vector */
             for(k=0;k<N;k++)
                    {
                     sampled_metric_sta[k].metricvalue = 0;
                    }
             for(i=0;i<number_ap_ESS;i++)
                    {
                     sampled_metric_ap[i].metricvalue = 0;
                    }
             /* Take num_samples for the average AP and STA load */
             for(i=0;i<num_samples;i++)
                    {
                     usecsleep(0,measurement_time);
                     number_ap_ESS = get_load_metric(metric_ap,N,LM1);
                     get_load_metric(metric_sta,N,LM4);

                     for(j=0;j<number_ap_ESS;j++)
                       {
                        sampled_metric_ap[j].metricvalue = sampled_metric_ap[j].metricvalue +
metric_ap[j].metricvalue;
                       }
                     for(k=0;k<N;k++)
                       {
                        sampled_metric_sta[k].metricvalue = sampled_metric_sta[k].metricvalue +
metric_sta[k].metricvalue;
                       }
                    }

             /* Compute the average load for each AP */
             for(j=0;j<number_ap_ESS;j++)
                    {
                     metric_ap[j].metricvalue =  sampled_metric_ap[j].metricvalue / num_samples;
                    }
             if(N == sta_old)
                    {
                     /* Compute the average load for each STA associated */
                     for(j=0;j<N;j++)
                       {
                        metric_sta[j].metricvalue =  sampled_metric_sta[j].metricvalue / num_samples;

                       }
                    }
             else
                    {
                     /* Check if there were more or less associated STAs meanwhile */
                     N = get_sta_number();
                     get_load_metric(metric_sta,N,LM4);
                    }
            }
           else
            {
```

```
          /* Get the number of stations associated to the AP from the SI */
          N = get_sta_number();
          /*Get station metrics from the SI*/
          get_load_metric(metric_sta,N,LM4);
          /* Get the number of APs with the LDS and its load level */
          number_ap_ESS = get_load_metric(metric_ap,N,LM1);
        }
      /* 1. Compute beta and write it to a file */
      beta = compute_beta(metric_ap,number_ap_ESS);
      if(beta < 1)
        {
          /* 2. Transfer Policy */
          ap_state = transfer_policy(metric_ap,number_ap_ESS,delta);
          if(ap_state == SENDER)
               {
                 /* 3. Selection Policy: selects the candidate station */

selection_and_distribution_policy(&dist_decision,metric_ap,metric_sta,number_ap_ESS,N,beta);

                 if(dist_decision.decision == DISTRIBUTE)
                  {
                    /* Disassociate the selected station from the AP */
                    disassociate_station(dev,skfd,metric_sta[dist_decision.selected_sta_index].ieee_addr);
                    /* Write STA hw address and load to "last_sta_disassociated" */

update_last_sta_disassociation(metric_sta[dist_decision.selected_sta_index].ieee_addr,metric_sta[dist_decision.sel
ected_sta_index].metricvalue);
                  }
                 else
                  {
                    syslog(LOG_INFO, "Distribution decision: IT IS NOT WORTH TO DISTRIBUTE THE
LOAD");
                  }
               }
          else syslog(LOG_INFO, "State of the AP: RECEIVER");
        }
      else
        {
          ap_state = OK;
          syslog(LOG_INFO, "State of the AP: OK");
        }
     }
 }
}
/****************************** MAIN *******************************/

/*-----------------------------------------------------------------------------*/
int main(int argc, char  *argv[]){

  int skfd, num_samples, measurement_time;
  char *dev = "wlan0"; /* WLAN device name */
  float delta; /* Value (in %) the load of the local AP can be higher than the average network load (ANL) in the ESS */
  int cycle_time, sampling_time, time_until_cycle_time, time_to_wait_until_lda;

  /* Check input parameters */
  switch(argc)
   {
    case 4:
     {
          /* Get input parameters */
          delta = atof(argv[1]);
          num_samples = atoi(argv[2]);
          cycle_time = atoi(argv[3]);
          break;
     }
    default:
     {
          printf("Usage: ldc_lda <delta> <num_samples> <cycle_time>\n");
          exit(-1);
     }
   }
  /* Prepare logger */
  openlog("ldc_lda_A", LOG_CONS, LOG_DAEMON);
  syslog(LOG_INFO, "Starting LDC block with LDA...");
  /* Create a channel to the wireless interface wlan0 */
  if((skfd = iw_sockets_open()) < 0)
   {
     perror("socket");
     exit(-1);
   }
  syslog(LOG_INFO, "Using wireless interface %s", dev);
  /* Set measurement time in us*/
  measurement_time = 50000;
  /* Log initial parameters related with the LDA */
  syslog(LOG_INFO, "Parameters(LDA_A).Measurement time = %d (us)",measurement_time);
  syslog(LOG_INFO, "Parameters(LDA_A).Delta (%) = %f",delta);
  syslog(LOG_INFO, "Parameters(LDA_A).Cycle time(us) = %d = %f(s)",cycle_time,cycle_time/1e6);
  /* Compute necessary time to wait */
  sampling_time = num_samples * measurement_time;
  time_to_wait_until_lda = cycle_time - sampling_time;
  if(num_samples == 1) time_to_wait_until_lda = cycle_time;
  /* Infinite loop to execute the LDA */
  for(;;)
   {
     /* Wait cycle_time seconds before executing LDA */
     usecsleep(0,time_to_wait_until_lda);
     /* Execute the LDA */
     lda_A(skfd,dev,delta,num_samples,measurement_time);
   }
}
```

```
/* Program: MM.c
*Description: Metric Monitor (MM) module is in charge of two functionalities related to load metrics: measurement
*and broadcast. It also takes care of reassociation notifications that APs exchange between them to find out if the
*last distributed station has been reassociated.
*/

#include "../include/ldc.h"
#include "../include/iwlib.h"
#include "../include/wireless.h"

/* Get AP address */
void get_ap_address(int skfd,char *ifname,struct sockaddr *hwa)
{
  struct iwreq wrq;
  char buffer[sizeof(struct sockaddr)],temp[128];

  if(iw_get_ext(skfd,ifname,SIOCGIWAP,&wrq) < 0)
    {
      fprintf(stderr, "%-8.8s  Interface doesn't support wireless statistic collection\n\n", ifname);
      return;
    }
  memcpy(hwa,&(wrq.u.ap_addr), sizeof (sockaddr));
}
/* Store the given load metric in the SIG */
void store_load_metric(int lm_value,char *hw_address)
{
  FILE *lm; /* FILE structure to open load metric file */
  char temp[128],sta_addr[20], value[100], ap_addr[20];
  int i;

  if((lm = fopen("lm1","w"))== NULL)
    {
      perror("lm1");
      exit(-1);
    }
  /* Update to new value */
  fprintf(lm,"%s %d",hw_address,lm_value);
  fclose(lm);
}
/* Store the load metric values for every station associated with the AP */
void store_load_metric_stas(int metric, struct sta_info *sta, int N)
{
  FILE *lm; /* FILE structure to open load metric file */
  char sta_addr[20];
  int i;

  switch(metric)
    {
      case LM4:
      {
              if((lm = fopen("lm4","w"))== NULL)
                {
                  perror("lm4");
                  exit(-1);
                }
              for(i=0;i<N;i++)
                {
                  if(sta[i].total_tx_rx >= 0)
                    {
                          fprintf(lm,"%s %d",sta[i].ieee_address_sta,sta[i].total_tx_rx);
                          fputs("\n",lm);
                    }
                }
              fclose(lm);
              break;
      }
    }
}
/*This function returns the hw address of stations associated with the AP and
 *its number (N)
 */
int get_sta_hw_address(int skfd,char *ifname,struct sockaddr *hwa)
{
  struct iwreq wrq;
  char                buffer[(sizeof(struct iw_quality) +
                              sizeof(struct sockaddr)) * IW_MAX_SPY];
  char                 temp[128];
  struct iw_quality    qual[IW_MAX_SPY];
  iwrange range;
  int                  has_range = 0;
  int                  n;
  int                  i;

  /* Collect stats */
  wrq.u.data.pointer = (caddr_t) buffer;
  wrq.u.data.length = IW_MAX_SPY;
  wrq.u.data.flags = 0;
  if(iw_get_ext(skfd, ifname, SIOCGIWSPY, &wrq) < 0)
    {
      fprintf(stderr, "%-8.8s  Interface doesn't support wireless statistic collection\n\n", ifname);
      return;
    }
  /* Number of addresses */
  n = wrq.u.data.length;
  memcpy(hwa, buffer, n * sizeof(struct sockaddr));
  return(n);
}
/* Get the AP traffic */
void get_ap_traffic(struct tx_rx_bytes *traffic)
{
  FILE *fd; /* FILE structure to open /proc/net/prism2/wlan0/stats */
  char buffer[3000], dev_name[10];
  int line=0;
  int rx_packets, rx_errs, rx_drop, rx_fifo, rx_frame, rx_compressed, rx_multicast;
  int tx_packets, tx_errs, tx_drop, tx_fifo, tx_frame, tx_compressed, tx_multicast;
  int rx_bytes, tx_bytes;

  /* Get traffic statistics from the driver */
  /* Open /proc/net/prism2/wlan0/stats in read mode */
  if((fd = fopen("/proc/net/dev","r"))== NULL)
    {
      perror("/proc/net/dev");
      exit(-1);
    }
  while(fgets(buffer,sizeof(buffer),fd) != NULL)
    {
      sscanf(buffer,"%5s",&dev_name);
      if(strcmp(dev_name,"wlan0") == 0)
           {
             sscanf(buffer," wlan0:%d %d %d %d %d %d %d %d %d",&rx_bytes,&rx_packets,&rx_errs,
                       &rx_drop,&rx_fifo,&rx_frame,&rx_compressed,&rx_multicast,&tx_bytes);
             break;
           }
    }
  fclose(fd);
  traffic->tx_unicast = tx_bytes;
  traffic->rx_unicast = rx_bytes;
}
/* Function to wait a period of time (in usecs) to take a new measure */
int usecsleep(int secs, int usecs)
{
  struct timeval tv;
  tv.tv_sec = secs;
  tv.tv_usec = usecs;
  return select(0, NULL, NULL, NULL, &tv);
}
/* Get the traffic from the associated stations with the AP */
int get_lm4_value(char *address_sta)
{
  FILE *stats; /* FILE structure to open /proc/net/prism2/wlan0/stats */
  char sta_complete_file_name[41]="/proc/net/prism2/wlan0/",temp[128],buffer[50];
  int line=0,rx_bytes_sta=0,tx_bytes_sta=0,int i,sta_left = 0;

  /* Append the file name to the station file path:/proc/net/prism2/wlan0/ */
  strcat(sta_complete_file_name,address_sta);
  /* Convert to lowercase the station file name*/
  for(i=0;i<18;i++)
    {
      sta_complete_file_name[23+i]=tolower(sta_complete_file_name[23+i]);
    }
  /* Open /proc/net/prism2/wlan0/ieee_address_station in read mode */
  if((stats=fopen(sta_complete_file_name,"r")) == NULL)
    {
      printf("STA %s has left the AP!\n",address_sta);
      sta_left = 1;
    }
  if(sta_left == 0)
    {
      while(fgets(buffer,sizeof(buffer),stats) != NULL)
           {
             line++;
             switch(line)
               {
               case 15:
                 {
                       sscanf(buffer,"rx_bytes=%d",&rx_bytes_sta);
                       break;
                 }
               case 16:
                 {
                       sscanf(buffer,"tx_bytes=%d",&tx_bytes_sta);
                       break;
                 }
               }
           }
      fclose(stats);
      return(rx_bytes_sta+tx_bytes_sta);
    }
}
/* Check if the last distribution is finished */
int check_and_read(void)
{
  FILE *stafd;
  char buffer[3000], sta_disassociated[3000];
  int lines = 0, sta_load = 0;

  if((stafd = fopen("last_sta_disassociated","r"))== NULL)
  {
    perror("last_sta_disassociated");
    exit(-1);
  }
  /* Check if there is a STA address stored in the SIG */
   while(fgets(buffer,sizeof(buffer),stafd) != NULL)
    {
      /* Scan only the hw address */
      sscanf(buffer,"%s %d",&sta_disassociated,&sta_load);
      lines ++ ;
    }
   fclose(stafd);
   return(sta_load);
}
/* Update information about last distributed station */
void update_last_disassociated_sta(void)
{
  FILE *stafd;

  if((stafd = fopen("last_sta_disassociated","w"))== NULL)
  {
    perror("last_sta_disassociated");
    exit(-1);
  }
  fclose(stafd);
}
/* Measurement of AP and stations traffic. It also broadcasts AP's traffic on the DS */
void get_and_broadcast_traffic(int usecs,struct sockaddr *hwa_stas,int skfd,char *dev, dsd_t dsd, char *ap_addr, int
background_traffic_high, int background_traffic_low, int cycle_high, int cycle_low, int handoff_time)
{
  FILE *stats; /* FILE structure to open /proc/net/prism2/wlan0/stats */
  FILE *traffic_ap; /* FILE structure to open /proc/net/prism2/wlan0/stats */
  int secs = 0, sta_number, N = 0;
  float lm4_value=0;
  struct sta_info sta[MAX_NUM_STA],sta_old[MAX_NUM_STA],sta_new[MAX_NUM_STA];
  char temp[128];
  /* AP traffic variables */
  struct tx_rx_bytes traffic_old,traffic_new;
```

```
 int traffic_new_tx_unicast, traffic_new_rx_unicast, total_traffic_ap;
 /* Monitor stas variables */
 int num_new_sta, num_old_sta, i, j, found;
 struct sockaddr hwa_stas_old[IW_MAX_SPY];
 /* Background traffic */
 int background_traffic = background_traffic_high;
 int cycle = 0;
 int high_low_state = 0; /* 0: high load; 1: low load */
 /* AP traffic statistics */
 int cycle2 = 0, target_cycle = 1000000 / usecs; /* For 100000 us, target_cycle will be 10 */
 int total_traffic_ap2 = 0, total_traffic_ap3 = 0;
 /* Load from a previous disassociated STA */
 int sta_disasssociated_load, timeout_sta_load = handoff_time / usecs, cycle_sta_load = 0;

 num_old_sta = get_sta_hw_address(skfd,dev,hwa_stas_old);
 for(;;)
  {
    /* Time configuration for the simulated traffic */
    /* Time in the high state */
    if(high_low_state == 0)
         {
           cycle = cycle + 1;
           if(cycle == cycle_high)
             {
               cycle = 0;
               if(background_traffic_low != 0)
                      {
                        high_low_state = 1;
                        background_traffic = background_traffic_low;
                      }
             }
         }
    /* Time in the low state */
    else if(high_low_state == 1)
         {
           cycle = cycle + 1;
           if(cycle == cycle_low)
             {
               cycle = 0;
               high_low_state = 0;
               background_traffic = background_traffic_high;
             }
         }
    /* Get the number of associated stations */
    N = get_sta_hw_address(skfd,dev,hwa_stas);
    num_new_sta = N;

    /* Check if there is any associated station */
    if(N!=0)
         {
           /* Get old AP traffic */
           get_ap_traffic(&traffic_old);
           for(sta_number=0;sta_number<num_old_sta;sta_number++)
             {
               /* Get the station hardware address in a readable format */
               sscanf(iw_pr_ether(temp,hwa_stas_old[sta_number].sa_data),"%s",
                       &(sta_old[sta_number].ieee_address_sta));
             }
           for(sta_number=0;sta_number<N;sta_number++)
             {
               /* Get the station hardware address in a readable format */

sscanf(iw_pr_ether(temp,hwa_stas[sta_number].sa_data),"%s",&(sta[sta_number].ieee_address_sta));

               /* Get current tx and rx bytes by the station */
               sta[sta_number].traffic_old = get_lm4_value(sta[sta_number].ieee_address_sta);
             }
           /* Wait usecs to get a new measure */
           usecsleep(secs,usecs);
           /* Get new AP traffic */
           get_ap_traffic(&traffic_new);
           traffic_new_tx_unicast = traffic_new.tx_unicast - traffic_old.tx_unicast;
           traffic_new_rx_unicast = traffic_new.rx_unicast - traffic_old.rx_unicast;
           total_traffic_ap = traffic_new_tx_unicast + traffic_new_rx_unicast;
           /* Check if there is the need to add some load from a previous disassociated STA */
```

```
          sta_disasssociated_load = check_and_read();
          if(sta_disasssociated_load != 0)
           {
             cycle_sta_load ++;
             if(cycle_sta_load == timeout_sta_load)
                   {
                     /* Update the SIG since the timeout has expired */
                     update_last_disassociated_sta();
                     cycle_sta_load = 0;
                   }
           }
           else cycle_sta_load = 0;
          /* Add background traffic and traffic from a previous disassociated STA to the total traffic of the AP */
          total_traffic_ap = total_traffic_ap + background_traffic;
          /* Only take into account the real (included the background) traffic of the AP to compute Beta */
          total_traffic_ap3 = total_traffic_ap;
          /* Add traffic a previous disassociated STA to the total traffic of the AP */
          total_traffic_ap = total_traffic_ap + sta_disasssociated_load;
          /* Store number of bytes AP */
          store_load_metric(total_traffic_ap,ap_addr);
          /* Broadcast load metric */
          ds_notify_metric(&dsd,dsd.own_ll_addr,ap_addr,total_traffic_ap,LM1);

          /* Get the number of associated stations */
          for(sta_number=0;sta_number<N;sta_number++)
           {
             sta[sta_number].traffic_new = get_lm4_value(sta[sta_number].ieee_address_sta);
             sta[sta_number].total_tx_rx = sta[sta_number].traffic_new - sta[sta_number].traffic_old;

             /* Copy the traffic to a new structure to ensure the date is coherent */
             if(sta[sta_number].total_tx_rx >= 0)
                   {
                     sta_new[sta_number].total_tx_rx = sta[sta_number].total_tx_rx;
                     strcpy(sta_new[sta_number].ieee_address_sta,sta[sta_number].ieee_address_sta);
                   }
           }
          store_load_metric_stas(LM4,sta_new,N);
          /* Only monitor STA traffic when there are no new stations associated with the AP */
          if(num_new_sta != num_old_sta)
           {
             if(num_new_sta > num_old_sta)
                   {
                     /* Check for the new associated station */
                     found = 0;
                     for(i=0;i<num_new_sta;i++)
                      {
                        for(j=0;j<num_old_sta;j++)
                              {
                                if(strcmp(sta_old[j].ieee_address_sta,sta[i].ieee_address_sta) == 0)
                                 {
                                   found = 1;
                                   break;
                                 }
                              }
                        if(found == 0)
                              {
                                sprintf(temp,iw_pr_ether(temp, hwa_stas[i].sa_data));
                                /* Broadcast hwaddr of the new STA on the DS to update the ACL list of the source
AP */
                                ds_update_acl(&dsd,dsd.own_ll_addr,temp,ap_addr);
                                break;
                              }
                      }
                   }
             num_old_sta = get_sta_hw_address(skfd,dev,hwa_stas_old);
           }
         }
   else
         {
          /* Only monitor STA traffic when there are no new stations associated with the AP */
          if(num_new_sta != num_old_sta)
           {
             if(num_new_sta > num_old_sta)
                   {
                     /* Check for the new associated station */
```

```
                 found = 0;
                 for(i=0;i<num_new_sta;i++)
                  {
                   for(j=0;j<num_old_sta;j++)
                        {
                         if(strcmp(sta_old[j].ieee_address_sta,sta[i].ieee_address_sta) == 0)
                          {
                           found = 1;
                           break;
                          }
                        }
                   if(found == 0)
                        {
                         sprintf(temp,iw_pr_ether(temp, hwa_stas[i].sa_data));
                         /* Broadcast hwaddr of the new STA on the DS to update the ACL list of the source
AP */
                         ds_update_acl(&dsd,dsd.own_ll_addr,temp,ap_addr);
                         break;
                        }
                  }
                 }
           num_old_sta = get_sta_hw_address(skfd,dev,hwa_stas_old);
          }
         /* Get old AP traffic */
         get_ap_traffic(&traffic_old);
         /* Wait usecs to get a new measure */
         usecsleep(secs,usecs);
         /* Get new AP traffic */
         get_ap_traffic(&traffic_new);
         traffic_new_tx_unicast = traffic_new.tx_unicast - traffic_old.tx_unicast;
         traffic_new_rx_unicast = traffic_new.rx_unicast - traffic_old.rx_unicast;
         total_traffic_ap = traffic_new_tx_unicast + traffic_new_rx_unicast;
         /* Check if there is the need to add some load from a previous disassociated STA */
         sta_disasssociated_load = check_and_read();
         if(sta_disasssociated_load != 0)
          {
           cycle_sta_load ++;
           if(cycle_sta_load == timeout_sta_load)
                {
                 /* Update the SI since the timeout has expired */
                 update_last_disassociated_sta();
                 cycle_sta_load = 0;
                }
          }
         else cycle_sta_load = 0;

         /* Add background traffic and traffic from a previous disassociated STA to the total traffic of the AP */
         total_traffic_ap = total_traffic_ap + background_traffic;
         /* Only take into account the real (included the background) traffic of the AP to compute Beta */
         total_traffic_ap3 = total_traffic_ap;
         /* Add traffic a previous disassociated STA to the total traffic of the AP */
         total_traffic_ap = total_traffic_ap + sta_disasssociated_load;
         /* Store number of bytes AP */
         store_load_metric(total_traffic_ap,ap_addr);
         /* Broadcast load metric */
         ds_notify_metric(&dsd,dsd.own_ll_addr,ap_addr,total_traffic_ap,LM1);
         /* Delete old stations from the SIG */
         /* Open /proc/net/prism2/wlan0/ieee_address_station in write mode */
         if((stats=fopen("lm4","w")) == NULL)
          {
           perror("Error opening file");
           exit(-1);
          }
         fclose(stats);
        }
   /* Store AP traffic per second */
   total_traffic_ap2 = total_traffic_ap2 + total_traffic_ap3;
   cycle2 = cycle2 + 1;
   if(cycle2 == target_cycle)
        {
         /* Open file to save AP traffic statistics */
         if((traffic_ap = fopen("traffic_ap","a"))== NULL)
          {
           perror("traffic_ap");
           exit(-1);
```

```
              }
             fprintf(traffic_ap,"%d",total_traffic_ap2);
             fputs("\n",traffic_ap);
             cycle2 = 0;
             total_traffic_ap2 = 0;
             fclose(traffic_ap);
            }
    }
}
/****************************** COMMUNICATION FUNCTIONS WITH THE DS ***********************************/
/* static int ds_init(dsd_t *dsd, char* if_name)
/* static int ds_notify_metric(dsd_t *dsd, u8 *addr,char *ap_addr,float metric_value,int metric_type)
/*----------------------------------------------------------------------------------------------------------------------------------------------*/
static int ds_init(dsd_t *dsd, char* if_name)
{
        struct ifreq ifr;
                struct sockaddr_ll addr;
                int returnCode = 0;

        memset(dsd, 0, sizeof(dsd));
                dsd->sock = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
                if (dsd->sock < 0) {
                                perror("socket");
                                exit(1);
                }
        memset(&ifr, 0, sizeof(ifr));
        snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), "%s", if_name);
        if (ioctl(dsd->sock, SIOCGIFINDEX, &ifr) != 0) {
                                perror("ioctl(SIOCGIFINDEX)");
                close(dsd->sock);
                exit(1);
        }
                memset(&addr, 0, sizeof(addr));
                addr.sll_family = AF_PACKET;
                addr.sll_ifindex = ifr.ifr_ifindex;
                syslog(LOG_INFO, "Opening raw packet socket for ifindex %d\n", addr.sll_ifindex);
                if (bind(dsd->sock, (struct sockaddr *) &addr, sizeof(addr)) < 0) {
                                perror("bind");
                                close(dsd->sock);
                                exit(1);
                }
        memset(&ifr, 0, sizeof(ifr));
        snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), "%s", if_name);
        if (ioctl(dsd->sock, SIOCGIFHWADDR, &ifr) != 0) {
                                perror("ioctl(SIOCGIFHWADDR)");
                close(dsd->sock);
                exit(1);
        }
                if (ifr.ifr_hwaddr.sa_family != ARPHRD_ETHER) {
                                printf("Invalid HW-addr family 0x%04x\n",
                                       ifr.ifr_hwaddr.sa_family);
                                close(dsd->sock);
                                exit(1);
                }
                memcpy(dsd->own_ll_addr, ifr.ifr_hwaddr.sa_data, ETH_ALEN);
                syslog(LOG_INFO,"Using interface %s with hwaddr " MACSTR "\n",
                                        if_name, MAC2STR(dsd->own_ll_addr));

                return returnCode;
}

int ds_notify_metric(dsd_t *dsd, u8 *addr,char *ap_addr,int metric_value,int metric_type)
{
                /* build and send a minimum IEEE802.3 frame */
                unsigned char pkt[ETH_FRAME_LEN];
                unsigned char metric[50];
                size_t pkt_len;
                struct ethhdr *eth_hdr;

                memset(&pkt, 0, ETH_FRAME_LEN);
                eth_hdr = (struct ethhdr *) pkt;
                /* Fill Ethernet header structure */
                memset(eth_hdr->h_dest, 255, ETH_ALEN);
                memcpy(eth_hdr->h_source, addr, ETH_ALEN);
                eth_hdr->h_proto = htons(ETH_ZLEN); /* size instead of type, see IEEE802.3 */
        /* Fill with protocol data: AP source address and load metric */
        memcpy(pkt,addr,ETH_ALEN);
        sprintf(metric,"%d %s %d",metric_type,ap_addr,metric_value);
        pkt_len = ETH_ZLEN;
        if (send(dsd->sock, metric, pkt_len,0) < 0) perror("ds_notify_metric");
        return 0;
}
/******************************* MAIN *******************************/

/*------------------------------------------------------------------------------*/
main(int argc, char  *argv[]){
  int skfd, N, usecs, load_metric = 1, ap_traffic, control = 1, pid, background_traffic_high, background_traffic_low,
cycle_high, cycle_low, handoff_time;
  char *dev="wlan0", temp[128], ieee_address_ap[18];
  struct sockaddr hwa_ap[1]; /* IEEE address of the AP */
  struct sockaddr hwa_stas[IW_MAX_SPY]; /* IEEE addresses of the STAs */
  dsd_t dsd;

  /* Check input parameters */
  switch(argc)
   {
   case 7:
     {
        /* Get input parameters */
        usecs = atoi(argv[1]);
        background_traffic_high = atoi(argv[2]);
        background_traffic_low = atoi(argv[3]);
        cycle_high = atoi(argv[4]);
        cycle_low = atoi(argv[5]);
        handoff_time = atoi(argv[6]);
        break;
     }
   default:
     {
        printf("Usage: metric_monitor usecs background_traffic_high(bytes/0.1s)
background_traffic_low(bytes/0.1s) cycle_high cycle_low handoff_time(us)\n");
        exit(-1);
     }
   }
  /* Prepare logger */
  openlog("metric_monitor", LOG_CONS, LOG_DAEMON);
  syslog(LOG_INFO, "Starting Metric Monitor block...");
  /* Init communication variables */
  ds_init(&dsd, "eth0");
  /* Create a channel to the wireless interface */
  if((skfd = iw_sockets_open()) < 0)
   {
     perror("socket");
     exit(-1);
   }
  /* Get the AP wireless IEEE addr */
  get_ap_address(skfd,dev,hwa_ap);
  sscanf(iw_pr_ether(temp,hwa_ap[0].sa_data),"%s",&ieee_address_ap);
  /* Start to get load metrics from others APs in the ESS */
  if((pid = fork()) == -1) perror("Error in fork call");
  else if(pid == 0)
   {
     execlp("../bin/com_ds_server","../bin/com_ds_server","eth0",ieee_address_ap,NULL);
     perror("GET_LOAD_OTHERS_AP execution has failed!\n");
   }
  for(;;)
   {
     /* Gets AP and STA traffic (rx and tx bytes), broadcasts AP traffic on the DS and stores it in the SI */
     get_and_broadcast_traffic(usecs,hwa_stas,skfd,dev,dsd,ieee_address_ap,background_traffic_high,
                                        background_traffic_low,cycle_high,cycle_low,handoff_time);
   }
}
```

```
/*
* Program: ldc.h
* Description: these are the header functions and data structures for the functions used in LDC.c and MM.c
*/

#if !defined(_LDC_H_)
#define _LDC_H_

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <netinet/in.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <time.h>
#include <sys/wait.h>
#include <syslog.h>

/* Load metrics(LM):
* LM1 = AP throghput (rx and tx bytes)
* LM4 = Stations throughput (rx and tx bytes)
*/
#define LM1 1
#define LM4 4

/* AP definitions for the LDC */
#define OK 90
#define RECEIVER 91
#define SENDER 92
#define DISTRIBUTE 5
#define DO_NOT_DISTRIBUTE 6
#define MAX_NUMBER_AP 50 /* Maximum number of APs in the ESS */
#define MAX_NUM_STA 100 /* Maximum number of STAs associated to one AP */

#define MAC2STR(a) (a)[0], (a)[1], (a)[2], (a)[3], (a)[4], (a)[5]
#define MACSTR "%02x:%02x:%02x:%02x:%02x:%02x"

typedef unsigned char u8;
struct tx_rx_bytes {
  int tx_unicast;
  int rx_unicast;
};
struct sta_info {
  char ieee_address_sta[18];
  int traffic_old;
  int traffic_new;
  int total_tx_rx;
};
typedef struct ds_data {
        int sock; /*raw packet socket for driver access towards DS*/
        u8 own_ll_addr[6]; /* interface link layer address */
} dsd_t;
struct load_metric{
  char ieee_addr[18];
  int metricvalue;
};
struct distribution_decision{
  int decision;
  int selected_sta_index;
};
#endif
```